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

IIC3413 — Implementación de Sistemas de Base de Datos — 1' 2020

LABORATORIO 1

Laboratorio:	Almacenamiento de datos.
Publicación:	Viernes 27 de marzo.
Ayudantía:	Viernes 27 de marzo.
Entrega:	**Viernes 10 de abril a las 23:59 horas**.

Descripción del laboratorio

El objetivo de este primer laboratorio es conocer y familiarizarse con el almacenamiento de IIC3413-DB [1]. Para que el alumno conozca el sistema de almacenamiento e identifique los componentes clave del mismo, para este laboratorio usted deberá extender la clase `ObjectFile` para manejar strings con un estructura de HeapFile.

Buffer manager y Page

IIC3413-DB ya cuenta con un sistema de buffer manager y páginas. Esta pieza de software se encuentra implementado en el módulo de `storage`, en los archivos `buffer_manager.h` y `page.h` respectivamente. La clase `Page` implementa el acceso "plano" a una página de datos. Cada página tiene tamaño de 4096 bytes. Para acceder estos datos, `Page` entrega el método `char* get_bytes()` que entrega un puntero donde empieza la secuencia de bytes. La clase `BufferManager` implementa el buffer manager del sistema del tipo pin/unpin. Para solicitar una página a BufferManager, este provee el método:

```
Page& get_page(FileId file_id, uint_fast32_t page_number)
```

que dado el identificador de un archivo (implementado por la clase `FileId`) y la dirección de una página (almacenado como un `uint_fast32_t`), entrega la referencia a la página correspondiente.

Es importante notar que `BufferManager` no provee métodos pin y unpin. En cambio, cuando uno solicita una página con el método `get_page`, es el buffer manager el encargado de hacer pin de la página. En otras palabras, `get_page` es lo mismo que el método pin. Para hacer unpin de la página, la clase `Page` provee el método `void unpin()` y `void make_dirty()` para hacer unpin de la misma página o marcar la página como "sucia". Esto es, el proceso que usa la página es el encargado de hacer unpin de la página, directamente sobre el objeto. Notar que, si un proceso desea hacer unpin de la página y la página fue modifica, primero debe llamar al método `make_dirty()` para después llamar al método `unpin()`.

Almacenamientos de strings

En IIC3413-DB, que es una base de datos de grafos con modelo de datos de *property graphs*, todos los datos como nodos, aristas, keys y values, son representados y manejados como identificadores. Incluso un string

[1] https://github.com/PUC-IIC3141/IIC3413-DB

cualquiera de la base de datos, es manejado con un identificador, que no es más que un int de 64 bits. Para almacenar y recuperar estos strings dado su identificador, IIC3413-DB provee la clase `ObjectFile`. Para acceder y almacenar un string, `ObjectFile` provee los métodos:

```
std::unique_ptr<std::vector<unsigned char>> read(uint64_t id)
uint64_t write(std::vector<unsigned char>& bytes)
```

El método `read` recibe el ID de un string, y retorna el string (representado como un vector de chars). El método `write` recibe un string (como una secuencia de chars), y retorna su identificador.

En la versión actual de IIC3413-DB, `ObjectFile` no hace uso del `BufferManager` y accede directamente el archivo donde se almacenan los strings consecutivamente. Específicamente, cada ID de 64 bits es una posición donde empieza el string en el archivo y cuyos primeros 4 bytes codifican el largo del string, que empieza después de estos primeros 4 bytes. Como uno puede ver, esto implica varios problemas al momento de acceder cada string. Por ejemplo, el manejo del buffer es dejado al sistema operativo y cada string puede estar en dos páginas, obligando a traer 2 páginas en vez de una. Por otro lado, si uno desea modificar o borrar un string, esto no es posible realizar de manera eficiente, dado que requiere modificar el ID de todos los strings siguientes.

Implementación de HeapFile para almacenamiento de strings

Para este laboratorio, su tarea es crear una nueva clase, llamada `HeapFile`, que será una alternativa a la clase `ObjectFile` para almacenar strings utilizando el buffer manager del sistema y los pages. Específicamente, su clase HeapFile debe implementar los mismos métodos `read` y `write` de `ObjectFile`, pero almacenando los strings en páginas, implementado la solución de "records de tamaño variable" vista en clases, y utilizando el buffer manager para administrar las páginas.

A groso modo, el laboratorio se divide en los siguientes dos items.

Formato y directorio de cada página. Cada página del HeapFile debe tener la estructura de directorio similar que para manejo de "records de tamaño variable", pero en vez de records guardara strings de tamaño variable. Cada página (page) deberá tener el siguiente formato:

El header debe ser de tamaño fijo, 10 bytes, y esta compuesto de tres campos: `pageno`, `dirsize` y `freespace`. El orden de aparición de estos campos en el header es el mismo (de izquierda a derecha). Específicamente, los campos son los siguientes:

- `pageno`: esta compuesto por 6 bytes y codifica el número de página.
- `dirsize`: esta compuesto por 2 bytes y codifica el tamaño del directorio, en otras palabras, el número de entradas en el directorio (ver definición de las entradas del directorio más abajo).
- `freespace`: esta compuesto por 2 bytes y codifica el tamaño de espacio libre, número de bytes libres en la página. El espacio libre se define como la cantidad de bytes sin usar entre la última entrada del directorio y el último string (`string`$_n$), esto es, la zona gris del diagrama de arriba.

El directorio consta de una secuencia de pares (`pointer`, `size`) donde:

- `pointer`: son 2 bytes que representan la posición donde se almacena el string. Esto es, la distancia desde el byte 0 en la página hasta donde empieza el string.
- `size`: son 2 bytes que representan el largo del string, esto es, número de bytes.

Por ejemplo, si el directorio tiene 3 entradas, entonces el directorio tendrá largo de 12 bytes, 4 bytes por cada entrada. Por último viene la sección de la página donde se almacena el contenido de los strings, que se almacenan de derecha a izquierda para dejar espacio para insertar nuevos strings y entradas al directorio.

Se recomienda fuertemente crear una clase `HeapPage` que se encargue de toda la lógica de formatear la página, crear el header y el directorio, y encargarse de hacer la inserción y manejo de los strings.

Es importante que su implementación siga las instrucciones anteriores, esto es el formato de la página, al "pie de la letra". Para la corrección se harán test de inserción y lectura, y las páginas del `HeapFile` deben estar exactamente como se describen anteriormente. Durante la primera semana del laboratorio se publicarán algunos test para que puedan comprobar que el formato implementado por ustedes es el correcto.

Formato y manejo del HeapFile. Como se mencionó anteriormente, su clase `HeapFile` debe implementar los mismos métodos `read` y `write` de `ObjectFile`, pero ahora almacenando los strings por páginas y ocupando el formato de páginas descrito anteriormente.

`HeapFile` continuará usando `uint64_t` como IDs para strings donde cada ID utilizará los primeros 48 bits (6 bytes) para representar el número de la página (partiendo desde 0) y los siguientes 16 bits (2 bytes) para representar la entrada en el directorio (partiendo desde el 0). Por ejemplo, el siguiente ID de 64 bits:

$$\underbrace{\texttt{00000000 00000000 00000000 00000000 00000000 0000100}}_{\#\text{ de página}}\underbrace{\texttt{00000000 00000010}}_{\#\text{ entrada en el directorio}}$$

representa al string que se encuentra en la página 4, en la entrada 2 de su directorio. Notar que está página se encuentra a partir del byte 16384 (esto es, 4×4096) del archivo correspondiente.

Para los método de `read` y `write`, su implementación deberá hacer lo siguiente:

- `std::unique_ptr<std::vector<unsigned char>> read(uint64_t id)`: Leer desde la página y directorio el string apuntado por ID, según el formato de dirección descrito anteriormente. Para esto, usted debe solicitar la página al buffer manager, extraer el string de la página, copiarlo en la memoria y entregarlo como resultado.
- `uint64_t write(std::vector<unsigned char>& bytes)`: Escribir el string en el HeapFile. Su implementación debe buscar linealmente (desde la primera página hasta la última), una página que tenga espacio libre suficiente para almacenar el string. Para esto, puede usar la entrada `freespace` en el header para verificar si hay espacio suficiente en la página para almacenar el string. Usted debe insertar el string en la primera página que encuentre con espacio suficiente para almacenar el string. En caso de no encontrar ninguna página con espacio suficiente, usted debe agregar una página al final del archivo, para almacenar el string.

  Para simplificar la solución de este laboratorio, usted puede asumir que el tamaño de los strings siempre serán menor al espacio máximo libre de una página y, por lo tanto, no es necesario hacer *spanned* de los strings. Es importante que para insertar un string en una página, el espacio libre tiene que ser suficiente como para almacenar el string y una nueva entrada en el directorio de la página.

Es importante destacar que, para la implementación de ambos métodos, su clase `HeapFile` **debe usar el buffer manager como también la clase page**.

Al igual que el item anterior, la lectura, inserción y el uso del buffer manager debe ser realizado "al pie de la letra" como sale descrito anteriormente. Se realizarán test automáticos para la corrección, y de no

cumplir con la descripción anterior (o sea, su archivo no cumple con el resultado esperado), todo el test estará incorrecto.

Por último, usted puede modificar el buffer manager, page o cualquier archivo del código que usted estime conveniente.

Bonus: Método `remove` (opcional)

La implementación del `HeapFile` no tiene ventaja frente al `ObjectFile` si no tiene la posibilidad de eliminar strings. Para este bonus usted debe implementa adicionalmente el método `void remove(uint64_t id)` que permita eliminar un string dado su id. Para esto usted debe ir a la página respectiva, "eliminar" la entrada en el directorio del string y `compactar` la página, esto es, eliminar la fragmentación interna. Dado que la eliminación de una entrada del directorio, cambiaría el ID de muchos strings, usted debe marcar la entrada del directorio como (0,0), o sea, `pointer` y `size` de la entrada serán 0. Mas aún, el contenido del string y fragmentación resultante debe ser removida de la página, compactándola. En otras palabras, su implementación debe mover todos los strings que están al lado izquierdo del string eliminado hacia la derecha, ocupando el espacio vacío. También debe actualizar las entradas de los directorios correctamente. Por último, es importante que compactar un página se debe hacer cada vez que se elimina un string.

El bonus es opcional y supondrá **1 punto adicional** en la nota final del Laboratorio 1.

Bonus por bugs encontrados/solucionados en IIC3413-DB

Se darán los siguientes bonus por encontrar o solucionar un bug de IIC3413-DB:

1. **Bug encontrado**: se darán 2 décimas en la nota final del laboratorio si encuentran un bug. Para ser efectivo este bonus, deben publicar una issue[2] del repositorio del proyecto sobre el bug, esto es, la descripción del bug, junto con el input que se le está entregando al sistema más el output "errado" que entrega.
2. **Bug solucionado**: se darán 2 décimas adicionales por entregar el código que soluciona este bug en el sistema (no necesariamente tiene que ser el mismo estudiante). Para esto deben dar una explicación de como solucionar el problema en el mismo issue y hacer un pull request con el código que soluciona el problema (no importa si es una línea o muchas líneas). Aparte de tener las dos décimas, su usuario GitHub quedará como contribuidor al proyecto.

Una vez publicado el bug o solución, los ingenieros del proyecto revisarán la publicación y determinarán si el bonus corresponde o no. Por último, dado un bug o solución se dará el bonus al primer estudiante que lo publiqué en las issues del repositorio.

Evaluación y entrega

El día límite para la entrega de esta tarea será el Viernes 10 de abril a las 23:59 horas. Para ello se utilizará el repositorio privado en GitHub que fue proporcionado por el curso. Para la entrega el alumno deberá crear una **nueva rama** en git a partir de la rama principal del proyecto, para que así pueda seguir desarrollando sus próximos laboratorios sin considerar las modificaciones del actual. La evaluación se realizará en base a test que debe pasar su código.

[2] https://github.com/PUC-IIC3141/IIC3413-DB/issues

Ayudantía y preguntas

El día Viernes 27 de marzo se realizará una ayudantía donde se darán más detalles sobre IIC3413-DB y el laboratorio. Para preguntas se pide usar el foro del curso o enviar un correo a `iic3413@inc.puc.cl`. De preferencia, se sugiere escribir al foro del curso para que todos los estudiantes estén al tanto de las dudas.