TP Bases de Datos II

Parte 2

(ENTREGA 2)
En esta segunda parte del trabajo práctico nos enfocaremos en la implementación de las dos funcionalidades pendientes de la primera entrega: insert y select (proyección de todos los elementos de la tabla), y un metacomandos: .table-metadata
Límite de tiempo: 1 semana

Requerimientos para la entrega

1. Insert statement funcionando
2. Select statement funcionando
3. Meta-comando .table-metadata

Introducción

Vamos a empezar con los siguientes requerimientos:

- Soporte para solo dos operaciones:
    - insertar un registro
    - obtener e imprimir todos los registros
- La base residirá solamente en memoria (no hay persistencia a disco aún)
- Tendrá una única tabla sin nombre con una lista fija de columnas

La estructura de dicha tabla permitirá alojar datos de usuarios y será la siguiente

columna	tipo
id	integer
usuario	varchar(32)
email	varchar(255)

- Guardaremos los registros en bloques de memoria de 4096 bytes llamados **páginas**
- Cada página guardará cuantos registros quepan en la de página
- Los registros serán **serializados** en un bloque de 291 bytes cuyo formato se especifica a continuación
- Nuevas páginas serán ubicadas a medida que se necesiten
- Mantendremos un array de tamaño fijo con punteros a las páginas

Registros

Un registro se representará como una secuencia de bytes construida a partir de 3 bloques de tamaño fijo que van a contener los valores de cada columna **serializados**, es decir, codificados como una secuencia de bytes:

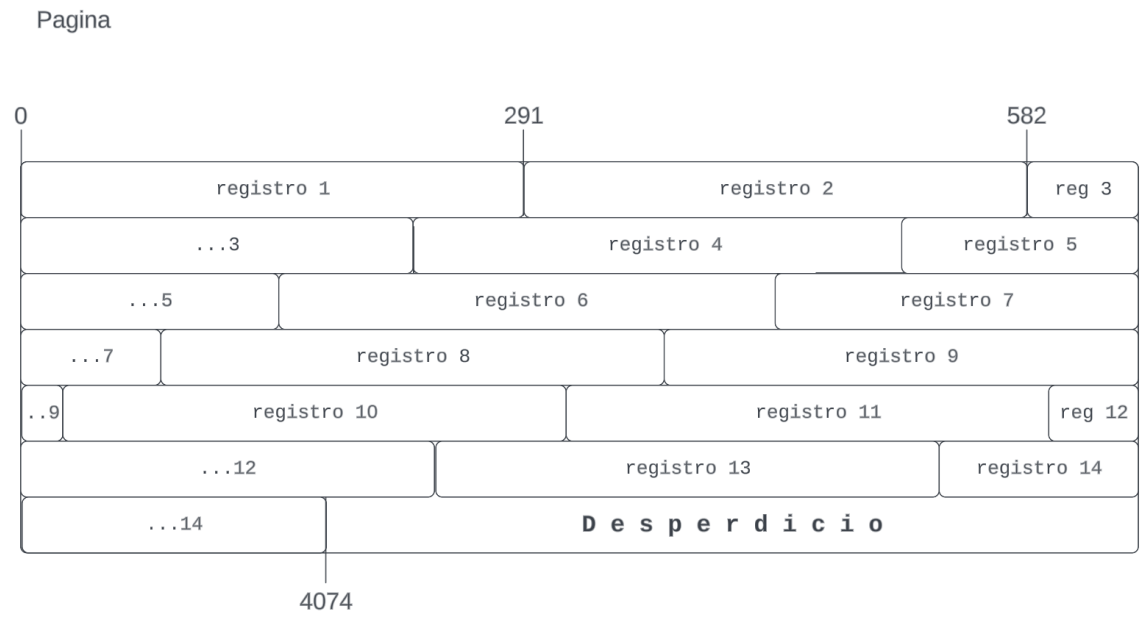
- Los campos de tipo entero como el ID, se codificarán con formato big endian (el byte más significativo a la izquierda)
- Las columnas usuario y email se codificarán como ASCII.

La estructura de un registro serializado sería la siguiente:

Para los campos de tipo VARCHAR, al ocupar un tamaño fijo, debemos agregar un carácter NUL (cod 0 de ASCII) como delimitador al final del valor serializado, que determine donde termina el valor que guardamos y empieza la "basura" de la memoria. Es decir, si tenemos un bloque de 32 bytes reservado para el campo usuario, pero el valor a guardar solo ocupa 4 bytes (o caracteres asumiendo un byte por caracter), en el 5to byte debemos guardar un carácter NUL. Así cuando leamos el bloque de datos para des-serializar el valor del campo, sabremos que solo debemos leer hasta la posición de este delimitador.

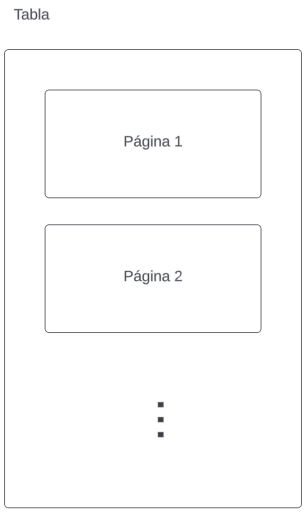
Páginas

Una página se representará como un bloque de máximo de 4096 bytes (4 kb) que puede contener 1 a N registros, donde N será el cociente del tamaño en bytes de la página, sobre el tamaño en bytes del registro, en este caso 4096/291 = ~14.
El sobrante de bytes de una página al haberse alcanzado el límite de registros completos que esta puede alojar, será desperdiciado, los registros no se alojarán parcialmente en dos páginas diferentes.



Tablas

Si bien los detalles de representación dependen del lenguaje y el criterio elegidos, una tabla deberá llevar registro de todas las páginas alojadas que la componen y el número total de registros insertados.



1- Insert Statement

Dado que la base de datos tiene una única tabla, y las columnas están predefinidas, la sintaxis para el INSERT será un poco más simple que en una base de datos real. Con esto simplificamos el parseo de statements que no es el foco de estudio.
Una sentencia de INSERT deberá contener los valores de las 3 columnas en el órden dado por la tabla más arriba (id, usuario, email), separados entre sí por un espacio. Los valores para las columnas de tipo VARCHAR no admitirán espacios como parte del valor. Las comillas no se usan para delimitar un string, sino que se interpretarán como parte del valor a insertar.
Ejemplos:

```
sql>insert 1 miguel miguel@gmail.com
     Válido

sql>insert 1 "miguel" miguel@gmail.com
     Válido, pero el valor de la columna usuario será
"miguel" incluyendo las comillas

sql>insert 1 miguel
     Inválido: Falta el valor de la tercera columna

sql>insert 1 "miguel perez" miguel@gmail.com
     Inválido: "miguel se interpretaría como el valor de la
columna usuario, perez" como el de la columna email y
miguel@gmail.com ya no sería aceptado por la gramática
```

Al ejecutar un INSERT exitosamente el programa debería imprimir al stdout "INSERT exitoso".
Ej:

```
sql>insert 1 miguel miguel@gmail.com
INSERT exitoso
sql>
```

Si el input es inválido se imprimirá al stdout "Operación inválida"
Ej:

```
sql>insert 1 "miguel perez" miguel@gmail.com
Operación inválida
sql>
```

A su vez, recuerden que estaremos trabajando con caracteres ASCII solamente para los strings, de manera que cada carácter pueda codificarse en un byte. No pierdan tiempo intentando soportar otros encodings.

2- Select Statement

De la misma manera que con el INSERT, simplificamos la sintaxis del SELECT.
SELECT no aceptará ningún tipo de argumentos, y mostrará todos los registros de la tabla, en el órden en que fueron insertados.
Ejemplos

```
sql>SELECT
```
Válido

```
sql>SELECT *
```
Inválido

```
sql>SELECT WHERE id=1
```
Inválido

Al ejecutar un SELECT exitosamente el programa debería imprimir al stdout el listado de todos los registros, donde cada registro se imprimirá en una nueva línea, y el valor de cada columna estará separado por un espacio. El órden de las columnas será el mismo que para la inserción. No deben imprimirse espacios adicionales al final de cada línea.
Ej:

```
sql>insert 1 marta marta@gmail.com
INSERT exitoso
sql>insert 2 marcos marcos@gmail.com
INSERT exitoso
sql>SELECT
1 marta marta@gmail.com
2 marcos marcos@gmail.com
sql>
```

Si el statement es inválido se imprimirá al stdout "Operación inválida"
Ej:

```
sql>SELECT *
Operación inválida
sql>
```

3- Meta-comando .table-metadata

El metacomando .table-metadata dará información sobre el estado interno de la tabla:

- número de páginas alojadas al momento
- número de registros insertados hasta el momento.

El formato del output será el siguiente:

```
sql>.table-metadata
Paginas: 2
Registros: 30
sql>
```