Tarea 1: Consultas en memoria secundaria

Prof: Pablo Barceló, Gonzalo Navarro
Auxiliares: Matilde Rivas, Bernardo Subercaseaux
Ayudantes: Daniel Diomedi, Thibault Swysen

1 Entrega de la Tarea

Se espera que se *implementen* los algoritmos, *realicen* los experimentos correspondientes y se entregue un *informe* que indique claramente los siguientes puntos:

1. Las *hipótesis* escogidas antes de realizar los experimentos.
2. El *diseño experimental*, incluyendo la idea general de la implementación de los algoritmos, la generación de las instancias y las medidas de rendimiento utilizadas.
3. La *presentación de los resultados* en forma de una descripción textual, tablas y/o gráficos.
4. El *análisis e interpretación* de los resultados.

1.1 Instrucciones

- La tarea puede realizarse en grupos de a lo más 3 personas.
- Para la implementación solo puede utilizar C, C++ o Java. Para el informe se recomienda utilizar LaTeX.
- Siga buenas prácticas (*good coding practices*) en sus implementaciones.
- Escriba un informe claro y conciso (no más de 10 páginas). Las ponderaciones del informe y la implementación en su nota final son las mismas.
- Tenga en cuenta las sugerencias realizadas en las primeras clases sobre la forma de realizar y presentar experimentos.
- La entrega será a través de U-Cursos y deberá incluir el informe junto con el código fuente de la implementación, y todas las indicaciones necesarias para su ejecución en un archivo `README.md`.
- Se permiten atrasos con un descuento de 0.5 punto por día.

2 Enunciado

En esta tarea implementaramos el *core* de `MiSQL`, un gestor de bases de datos inventado para propósitos de este curso.

Nos centraremos en el estudio de una base de datos particular y predefinida, en el contexto de modernizar un negocio del barrio República implementando un programa de fidelización basado en puntos.

El programa de fidelización funciona de la siguiente manera: el negocio tiene un registro de sus clientes, y cuándo compran algún producto ganan una cierta cantidad de puntos. Los puntos que acumulan se almacenan en su registro, y luego le permiten a los clientes obtener productos de manera gratuita.

Inicialmente, consideraremos el siguiente esquema relacional, que modela las tablas:

`Producto(INT id, INT precio, INT puntosNec, INT puntosRec)`

`Consumidor(INT id, STRING rut, INT puntosAcumulados)`

Consumidor		
id	rut	puntosAcumulados
1	19136938-6	1000
2	6499130-2	31415926
3	1-9	666

Producto			
id	precio	puntosNec	puntosRec
1	2717281	1000	2
2	160833	666	77

Los *puntos necesarios* (variable `puntosNec`) de un producto se refieren a la cantidad que se requiere canjear para obtener el producto sin pagar. Los *puntos de recompensa* (variable `puntosRec`) a aquellos que se obtienen al comprarlo normalmente.

Parte 1: Base de Datos y Ordenamiento

En esta primera parte, implementará la inserción de datos usando diferentes estructuras. Para esto considere que una entrada (fila) puede representarse como un nodo (un nodo en el sentido más general, simplemente una unidad de información) que tiene atributos según las diferentes columnas de la tabla.

a) (1.0 pto) Construya una clase `nodo` que contenga un diccionario, donde las llaves serán los nombres de las columnas, y los valores serán los valores correspondientes al nodo (fila) en cuestión.

b) (1.0 pto) Construya una clase `Database` que permita agregar nodos. En su primera implmentación, la estructura subyacente será simplemente un archivo de texto que contendrá un nodo por línea (necesitará que la clase `nodo` pueda ser serializada (representada) en una línea). Abstraiga eso de

la implementación, ya que en la sección siguiente usará otra estructura subyacente. Implemente un método `ordenar`, que recibirá un nombre de atributo, y ordenará el archivo de texto de acuerdo a ese atributo. Deberá ocupar el algoritmo de ordenamiento MergeSort para memoria externa.

c) (1.0 pto) Estudie el tiempo de inserción y ordenamiento (por separado) con diferentes números de datos. En particular, pruebe con $N = \{10, 10^2, 10^3, 10^4, 10^5, 10^6, 10^7\}$.

Parte 2: B-Trees

En esta sección implementará B-Trees como estructura subyacente.

a) (1.0 pto) Implemente una clase `Btree` con los métodos para inserción y búsqueda.

b) (1.0 pto) Estudie el tiempo de inserción con diferentes números de datos. En particular, pruebe con $N = \{10, 10^2, 10^3, 10^4, 10^5, 10^6, 10^7\}$.

Parte 3: Cruzando tablas

Una consulta natural para el esquema dado sería:

`SELECT * FROM Producto, Consumidor WHERE puntosNec <= puntosAcumulados`.

Para responder una consulta de este estilo hay varias estrategias posibles, estudiaremos 3. El resultado de la consulta deberá ser un archivo, donde en cada línea se encuentra impreso un nodo perteneciente al conjunto definido por la query.

1. Para cada cliente, consultar por todos los productos que no requieren más puntos de los que él posee, y listar al cliente con cada producto.
2. Para cada producto, consultar por todos los clientes que tienen al menos tantos puntos como cuesta el producto, y listar al producto con cada uno de esos clientes.
3. Ordenar ambas tablas, los consumidores según los puntos acumulados (de menor a mayor) y los productos según los puntos necesarios (de menor a mayor). Luego iterar por los consumidores, manteniendo siempre un cursor (puntero) en la tabla de productos. En cada momento se lista el consumidor actual junto a todos los productos hasta el cursor (sin incluir) y luego, mientras el cliente tenga más puntos acumulados que el costo del producto apuntado por el cursor, se lista también el cliente junto a ese producto y el cursor se avanza en una posición.

a) (1.0 pto) Implemente las dos primeras estrategias indexando con Btrees. (Cada nodo del Btree correspondería a un valor particular del atributo según el cuál indexa, y tendría un puntero a un archivo donde se encuentran los nodos que poseen ese valor en el atributo correspondiente).

b) (1.0 pto) Fundamente la tercera estrategia e impleméntela en base al ordenamiento de la primera parte.

c) (1.0 pto) Experimente cómo se comparan las estrategias con todas las combinaciones de números de datos para consumidores y productos, respectivamente: $C = \{10, 10^2, 10^3\}$ y $P = \{10, 10^2, 10^3, 10^4\}$

Bonus: Manejando Consultas

En esta parte nos meteremos de lleno a la implementación necesaria para responder consultas a la base de datos. Dejaremos de lado la parte de *parsear* la consulta, y nos concentraremos además en consultas no-recursivas sobre una sola tabla.

a) (0.5 pto) Implemente una clase `Filtro`, donde un filtro se define por una lista de subfiltros. Un subfiltro es un triplete `(nombreAtributo, tipoComparación, valorComparación)`, donde `nombreAtributo` es el nombre de una columna, por ejemplo: RUT.

`tipoComparación` puede tomar cualquiera de los siguientes valores `{ <, >, !=, ==, <=, >= }` e indica la relación que los valores deben cumplir con respecto a `valorComparación`.

Así, el triplete `(RUT, >, '5')` filtra (guarda) aquellas columnas cuyo RUT es mayor que '5'. Note que $>$ y $<$ se definen de acuerdo al tipo de datos, y considere los usuales (lexicográfico para palabras y el orden clásico de los enteros).

b) (1.0 pto) Implemente una función de su clase Tabla que reciba un filtro y retorne las filas que cumplen con el filtro.