Trabajo práctico 2: **Álgebra relacional**

Normativa

Límite de entrega: Domingo 20 de octubre, 23:59hs. Enviar el zip al mail: `algo2.dc+TP2@gmail.com`
Normas de entrega: Ver "Información sobre la cursada" en el sitio Web de la materia.
(`http://campus.exactas.uba.ar`)
Versión: 1.0 del 2 de octubre de 2019

Enunciado

El objetivo de este TP es completar el diseño de la base de datos iniciado en el TP1. La interfaz pública propuesta en el TP1 se debe mantener mayormente intacta En cambio deberán seleccionar **estructuras de representación** y **algoritmos** adecuados para cumplir con la interfaz, respetando ciertos **requerimientos de complejidad** detallados más abajo.

Como se detalló en el TP 1 hay ocho tipos de consultas. Al ejecutar una consulta en una base de datos pueden darse dos situaciones, que llamamos *caso típico* y *caso excepcional*:

Tipo de consulta	*Caso típico*	*Caso excepcional*
`FROM`(t)	La tabla t está definida en la base de datos.	La tabla t no está definida en la base de datos.
`SELECT`(q, c, v)	Todos los registros que resultan de la subconsulta q incluyen el campo c.	Algún registro que resulta de la subconsulta q no incluye el campo c.
`MATCH`(q, c_1, c_2)	Todos los registros que resultan de la subconsulta q incluyen los campos c_1 y c_2.	Algún registro que resulta de la subconsulta q no incluye el campo c_1 o no incluye el campo c_2.
`PROJ`(q, C)	Todos los registros que resultan de la subconsulta q incluyen todos los campos del conjunto C.	Algún registro que resulta de la subconsulta q no incluye algún campo del conjunto C.
`RENAME`(q, c_1, c_2)	Todos los registros que resultan de la subconsulta q incluyen el campo c_1 y no incluyen el campo c_2.	Algún registro que resulta de la subconsulta q no incluye el campo c_1 o incluye el campo c_2.
`INTER`(q_1, q_2)	(Siempre).	(Nunca).
`UNION`(q_1, q_2)	(Siempre).	(Nunca).
`PRODUCT`(q_1, q_2)	Si r_1, r_2 son registros tales que r_1 resulta de la subconsulta q_1 y r_2 resulta de la subconsulta q_2, entonces r_1 y r_2 no tienen ningún campo en común.	Hay un registro r_1 que resulta de la subconsulta q_1 y un registro r_2 que resulta de la subconsulta q_2 tales que r_1 y r_2 tienen algún campo en común.

Al momento de cumplir con los requerimientos de complejidad pedidos, únicamente nos concentraremos en los casos típicos, nunca en los excepcionales.

Requerimientos de complejidad

Se imponen los siguientes requerimientos de complejidad en peor caso para responder consultas. Para definir los requerimientos notamos las siguientes complejidades en relación a una Base de Datos:

- Notamos $|x|$ a la longitud máxima encontrada del string x ($|c|$ es el nombre más largo de un campo, $|v|$ es el nombre más largo de un valor y $|t|$ es el nombre mas largo de una tabla).
- La cantidad de campos que puede tener una tabla se considerará *constante*.
- k nota la cantidad de registros en el resultado de la consulta.

A continuación listamos las restricciones. De la 1. a la -3. son obligatorias. Además se deben incorporar al menos dos de las optimizaciones 4.–8.. Como se mencionó arriba, todos los requerimientos de complejidad pedidos aplican para los casos típicos, mientras que en los casos excepcionales no se imponen requerimientos de complejidad.

1. **Select con clave** — responder una consulta $\texttt{SELECT}(\texttt{FROM}(t), c, v)$ cuando c es la clave de t, debe tener costo $O(|t| + |c| + |v|)$.
2. **Select sin clave** — responder una consulta $\texttt{SELECT}(\texttt{FROM}(t), c, v)$ cuando c *no* es la clave de t, debe tener costo $O(|t| + |c| + n \cdot |v| + k \cdot (|v| + |c|))$ donde n es la cantidad total de registros de la tabla t.
3. **Join con claves** — responder una consulta $\texttt{MATCH}(\texttt{PRODUCT}(\texttt{FROM}(t_1), \texttt{FROM}(t_2)), c_1, c_2)$, también conocida como "join", suponiendo que t_1 y t_2 son tablas distintas, y tales que c_1 es la clave de t_1 y c_2 es la clave de t_2, debe tener costo $O(|t| + |c| + \text{mín}(n_1, n_2) \cdot |v| + k \cdot (|c| + |v|))$ donde n_1 es la cantidad total de registros de la tabla t_1 y n_2 es la cantidad total de registros de la tabla t_2.

Además, el diseño debe incorporar **al menos dos** de las siguientes optimizaciones:

4. **Selección con clave de selección sin clave** — si c_1 es la clave de la tabla t y c_2 no es la clave de la tabla t, responder una consulta de la forma:

$$\texttt{SELECT}(\texttt{SELECT}(\texttt{FROM}(t), c_2, v_2), c_1, v_1)$$

   debe tener costo $O(|t| + |c| + |v|)$.
5. **Selección con clave de un producto** — si t_1 y t_2 son tablas distintas y c_1 es la clave de la tabla t_1, responder una consulta de la forma:

$$\texttt{SELECT}(\texttt{PRODUCT}(\texttt{FROM}(t_1), \texttt{FROM}(t_2)), c_1, v_1)$$

   debe tener costo $O(|t| + |c| + |v| + n_2 \cdot (|c| + |v|))$ donde n_2 es la cantidad total de registros de la tabla t_2.
6. **Intersección de dos selecciones de la misma tabla** — si c_1 y c_2 no son la clave de la tabla t, responder una consulta de la forma:

$$\texttt{INTER}(\texttt{SELECT}(\texttt{FROM}(t), c_1, v_1), \texttt{SELECT}(\texttt{FROM}(t), c_2, v_2))$$

   debe tener costo $O(|t| + |c| + n \cdot |v| + k \cdot (|v| + |c|))$ donde n es la cantidad total de registros de la tabla t.
7. **Intersección de dos tablas** — si t_1 y t_2 son tablas distintas que tienen el mismo conjunto de campos y la misma clave, entonces responder una consulta de la forma:

$$\texttt{INTER}(\texttt{FROM}(t_1), \texttt{FROM}(t_2))$$

   debe tener costo $O(|t| + \text{mín}(n_1, n_2) \cdot |v| + k \cdot (|c| + |v|)$ donde n_1 es la cantidad total de registros de la tabla t_1 y n_2 es la cantidad total de registros de la tabla t_2.
8. **Caché de consultas** — si q es alguna de las últimas 10 consultas que se hicieron en el sistema y únicamente involucra tablas que no fueron modificadas desde la última vez que se realizó dicha consulta, responder la consulta q debe tener costo $O(\#q)$, donde $\#q$ representa el *tamaño* de la consulta q. (El tamaño de una consulta se puede medir, por ejemplo, como la cantidad total de símbolos que se necesitan para escribir la consulta q).

No hay ningún requisito de complejidad sobre las demás operaciones de la base de datos (agregar y eliminar tablas, agregar y eliminar registros) ni sobre consultas de otros tipos no contemplados arriba, pero se espera que los algoritmos para esas operaciones no hagan usos excesivos de tiempo o memoria.

Entrega

La entrega consistirá de un único documento digital con el diseño completo de todos los módulos. Se debe diseñar el módulo principal (BaseDeDatos) y todos los módulos auxiliares. De definirse módulos que no pueden ser apropiadamente explicados por un TAD existente, deberá desarrollarse un TAD para describir su comportamiento. La única excepción son los módulos disponibles en el Apunte de Módulos Básicos, que se pueden utilizar sin diseñarlos: lista enlazada, pila, cola, vector, diccionario lineal, conjunto lineal y conjunto acotado de naturales. Además, en caso de utilizarse un diccionario implementado sobre la estructura Trie, el módulo puede estar dado solamente presentando su interfaz, completa con complejidades y aliasing. Las complejidades descriptas en la interfaz deben ser posibles de cumplir.

En este TP se debe diseñar íntegramente los módulos, que constan de las siguientes partes.

1. **Interfaz.**

a) *Tipo abstracto ("se explica con ...").* Género (TAD) que sirve para explicar las instancias del módulo, escrito en el lenguaje de especificación formal de la materia.
b) *Signatura.* Listado de todas las funciones públicas que provee el módulo, escrito con la notación de módulos de la materia, por ejemplo:

apilar(**in**/**out** pila : PILA, **in** x : ELEMENTO)

c) *Contrato.* Precondición y postcondición de todas las funciones públicas. Las pre y postcondiciones de las funciones de la interfaz deben estar expresadas **formalmente** en lógica de primer orden.[1]
d) *Complejidades.* Complejidades de todas las funciones públicas, cuando corresponda.
e) *Aspectos de* aliasing. De ser necesario, aclarar cuáles son los parámetros y resultados de los métodos que se pasan por copia y cuáles por referencia, y si hay *aliasing* entre algunas de las estructuras.

Importante: la interfaz pública propuesta en el TP1 no debería sufrir mayores cambios. Todas las modificaciones que se hagan a la interfaz propuesta en el TP1 deben estar debidamente mencionadas y justificadas.

2. **Implementación.**
   a) *Representación ("se representa con ...").* Módulo con el que se representan las instancias del módulo actual.
   b) *Invariante de representación.* Puede estar expresado en lenguaje natural o formal.
   c) *Función de abstracción.* Puede estar expresada en lenguaje natural o formal.
   d) *Algoritmos.* Pueden estar expresados en pseudocódigo, usando si es necesario la notación del lenguaje de módulos de la materia o notación tipo C++. Las pre y postcondiciones de las funciones auxiliares pueden estar expresadas en lenguaje natural (no es necesario que sean formales). Indicar de qué manera los algoritmos cumplen con el contrato declarado en la interfaz y con las complejidades pedidas. No se espera una demostración formal, pero sí una justificación adecuada.
3. **Servicios usados.** Módulos que se utilizan, detallando las complejidades, *aliasing* y otros aspectos que dichos módulos deben proveer para que el módulo actual pueda cumplir con su interfaz.

Sobre el uso de lenguaje natural y formal.

Las precondiciones y poscondiciones de las funciones auxiliares, el invariante y la función de abstracción pueden estar expresados en lenguaje natural. No es necesario que sean formales. Asimismo, los algoritmos pueden estar expresados en pseudocódigo. Por otro lado, está permitido que utilicen fórmulas en lógica de primer orden en algunos lugares puntuales, si consideran que mejora la presentación o subsana alguna ambigüedad. El objetivo del diseño es convencer al lector, y a ustedes mismos, de que la interfaz pública se puede implementar usando la representación propuesta y respetando las complejidades pedidas. Se recomienda priorizar la **claridad** y **legibilidad** antes que el rigor lógico por sí mismo. Por ejemplo:

Más claro	**Menos claro**
"Cada clave del diccionario D debe ser una lista sin elementos repetidos." ✓ "sinRepetidos?(claves(D))" ✓	"No puede haber repetidos." (¿En qué estructura?).
"Ordenar la lista A usando mergesort." ✓ "A.mergesort()" ✓	"Ordenar los elementos." (¿Qué elementos? ¿Cómo se ordenan?).
"Para cada tupla (x, y) en el conjunto C { x.apilar(y) n++ }" ✓	"Miro las tuplas del conjunto, apilo la segunda componente en la primera y voy incrementando un contador." (Ambiguo y difícil de entender).

La entrega se realizará por mail a la dirección `algo2.dc+tp2@gmail.com`. El documento desarrollado se entregará como un archivo en formato pdf hasta el día 20 de octubre a las 23:59hs. El mail deberá tener como **Asunto** los números de libreta separados por punto y coma (;). Por ejemplo:

Acerca del recuperatorio: para el **recuperatorio** del TP2, se deberá reentregar el informe, debidamente corregido o extendido, en la fecha correspondiente a la entrega del TP3. Notar que el TP3 consistirá en implementar el TP2 en C++.

[1] Si la implementación requiere usar funciones auxiliares, sus pre y postcondiciones pueden estar escritas en lenguaje natural, pero esto no forma parte de la interfaz.

To: algo2.dc+tp2@gmail.com
From: alumno-algo2@dc.uba.ar
Subject: 123/45; 67/8; 910/11; 12/13
Adjunto: tp2.pdf