Trabajo práctico 2: Diseño

Base De Datos

Normativa

Límite de entrega: miércoles 25 de mayo *hasta las 22:00 hs*. Enviar PDF a `algo2.dc+TP2@gmail.com` con asunto "Grupo N" siendo N el número de grupo.

Normas de entrega: Ver "Información sobre la cursada" en el sitio Web de la materia.
(`http://www.dc.uba.ar/materias/aed2/2016/1c/cursada`)

Versión: 1.2 del 17 de mayo de 2016 (ver TP2_Changelog.txt)

Enunciado

El objetivo de este TP es diseñar el módulo correspondiente al TAD Base de Datos, junto con todos los módulos que sean necesarios para contar con un diseño **completo**.

Este enunciado contiene la especificación del TAD y demás tipos necesarios tal como deberá ser diseñado. Considerando las siguientes diferencias en el comportamiento con respecto a lo pedido en el TP1:

- Se agrega la posibilidad de crear (pero no borrar) *índices* (ver apéndice).
- Se definen las operaciones auxiliares sobre el TAD Tabla: *mínimo, máximo* para campos con índice.
- Se agrega la operación *buscar* del RTP1.

Estos cambios están especificados como son deseados. Sugerimos que confíen en la axiomatización más que en el castellano.

Contexto de uso y complejidades requeridas

Se requiere que todas las operaciones exportadas de los TADs tengan una contraparte en los módulos diseñados. Recomendamos modularizar adecuadamente: no necesariamente se aconseja hacer un único módulo por cada TAD.

Se **asume** lo siguiente a lo largo de todo el trabajo práctico:

- Los nombres de las tablas y de los campos son STRING acotados.
- Se puede asumir que las tablas de la base de datos tienen una cantidad acotada de campos para los cálculos de complejidad. No así la cantidad de tablas.
- Lo valores NAT se insertan y borran con distribución uniforme.
- Se puede definir a lo sumo un índice de cada tipo por tabla, es decir, por tabla puede haber a lo sumo dos índices, uno para algún campo del tipo NAT y otro para algún campo el tipo STRING.

Además, las operaciones indicadas a continuación deberán cumplir las complejidades temporales detalladas para el módulo Base De Datos. Todas las complejidades corresponden al peor caso salvo indicación contraria.

- tablas: $O(1)$ debe devolver un iterador con siguiente en $O(1)$.
- insertarEntrada: $O(T * L + in)$
  - in es $O(\log(n))$ en promedio, si hay índice sobre un campo de tipo NAT, $O(1)$ sino.
- borrar(r, t): $O(T * L + in)$
  - in es $O(\log(n))$ en promedio, si el criterio de borrado r es un campo clave con índice en la tabla t.
- dameTabla, tablaMáxima, campoJoin, HayJoin?, borrarJoin: $O(1)$
- mínimo y máximo del módulo Tabla: $O(1)$ si hay índice en el campo pedido.
- buscar(r, t): $O(L + \log(n))$ en promedio si alguno de los criterios de búsqueda r es un campo clave con índice en la tabla t.

- generarVistaJoin(t_1, t_2, ca):
  - $O((n+m)*(L+\log(n+m)))$ en promedio si existe un índice en ca para las tablas t_1 y t_2.
  - Debe devolver un iterador al resultado con siguiente en O(1).
- vistaJoin(t_1, t_2):
  - $O(R*(L+\log(n*m)))$ donde R son los registros nuevos y borrados luego de la generación o última visualización del join. Si $R = 0$ la complejidad debería ser O(1).
  - Además, debe devolver un iterador al resultado con siguiente en O(1)

donde:

- n y m son la cantidad de registros de la tablas pasadas por parámetro.
- T es la cantidad de tablas en la base de datos.
- L es la máxima longitud de un valor STRING de un registro en la tabla pasada por parámetro

Requisitos y consideraciones

- Todos los algoritmos *deben* tener su desarrollo que justifique los órdenes de complejidad. Si algún paso es no trivial pueden hacer notas a continuación del mismo.
  - Para las operaciones que involucren índices se espera un uso eficiente de los mismo y una justificación detalladas en los órdenes de complejidad.
- Se pretende que la creación de índices no replique la información contenida en los registros.
- Todas las operaciones auxiliares deben ser especificadas formalmente según las herramientas vistas en clase. Agregar comentarios necesarios para entender la forma en la cual deben ser utilizadas para su correcto funcionamiento.
- Cuando se formalicen los invariantes y funciones de abstracción, *deben* identificar cada parte de la fórmula del Rep y comentar en castellano lo que describe.
- El invariante de representación de las estructuras recursivas (aquellas que involucren punteros) puede escribirse en castellano.
- Tener en cuenta que las complejidades son en peor caso. Soluciones más eficientes serán bien recibidas.
- También, tener en cuenta que hay estructuras que pueden servir para más de una finalidad, sobre todo los contenedores.
- Pueden crear módulos adicionales si así lo necesitan.
- Para los módulos que se expliquen con TADs básicos o TADs que están en la especificación adjunta, solo es obligatorio incluir en el diseño los generadores y observadores básicos. Las otras operaciones pueden incluirlas o no según las precisen.
- Cuentan con los siguiente TADs y módulos ya diseñados:
  - CHAR que representan los posibles caracteres. Siendo un tipo enumerado de 256 valores. con funciones ord y ord^{-1} para la correspondencia de cada $Char$ a Nat.
  - STRING como sinónimo de $Vector(Char)$.
  - Todos los definidos en el apunte de TADs básicos.
  - Todos los definidos en el apunte de módulos básicos.

Apéndice: índices sobre campos de las tablas

En la vida real, las tablas de una base de datos pueden ser accedidas a través de *índices* que aprovechan la eficiencia con la que determinados tipos de datos pueden ser organizados. La creación de un índice para una tabla se hace a través de la elección de un campo de la tabla, que puede ser uno de sus campos clave o no, y a partir de considerarlo como clave de un diccionario, se puede acceder a todos los elementos que compartan un valor dado para dicho campo.

Entre las enormes ventajas que provee la utilización de índices, están:

1. la posibilidad de realizar el borrado de los registros accediendo solo a aquellos que satisfacen la condición,
2. la provisión de una implementación eficiente de la operaciones que involucren búsquedas y join tomando en cuenta la existencia de índices en ambas tablas, y
3. la posibilidad de optimizar la inserción de elementos en una tabla minimizando el impacto en el costo temporal de mantener actualizados los joins existentes.