Trabajo práctico 2: Diseño e implementación
BaseDeDatos

Normativa

Límite de entrega: Domingo 22 de Octubre *hasta las 23:59 hs.* Enviar el zip al mail `algo2.dc+{tn,tm}+TP1@gmail.com`. Ver detalles de entrega abajo.

Normas de entrega: Ver "Información sobre la cursada" en el sitio Web de la materia.
(`http://www.dc.uba.ar/materias/aed2/2017/2017/2c/{tn,tm}/cursada`)

Versión: 1.2 del 9 de Octubre de 2016 (ver CHANGELOG.md)

Enunciado

En este trabajo práctico elaboramos sobre la propuesta del primero. En esta entrega queremos agregarle operaciones a la base de datos y hacer que estas y otras ya implementadas cumplan ciertas restricciones de complejidad. El objetivo del trabajo práctico es producir módulos de diseño nuevos así como modificar los existentes para lograr las funcionalidades requeridas.

El trabajo práctico cuenta con el diseño e implementación de una Base de Datos que cumple los requisitos del trabajo práctico 1. Las clases ya implementadas fueron actualizadas para incorporar los conceptos nuevos de la materia y además fueron dotadas con módulos de diseño completos, incluyendo interfaz, rep y abs. El trabajo práctico también incluye tests para la implementación actual y para las nuevas funcionalidades incluidas. Estos test cubren los mínimos casos de uso y podrían ser extendidos por la cátedra previamente a la fecha de entrega del trabajo.

Nuevas funcionalidades

Actualmente la base de datos permite crear nuevas tablas, consultar si un registro es válido para una tabla de la base, insertar el registro en la tabla, consultar los registros de una tabla, consultar si un criterio de búsqueda es valido para una tabla y aplicar el criterio sobre la misma de forma de obtener un recorte de la tabla sobre la que se aplicó la búsqueda. Además, la base de datos guarda estadísticas del uso de los criterios para saber cuales son las búsquedas más realizadas.

Sobre estas funcionalidades queremos agregar nuevas. Por un lado, se deben poder crear índices para un campo de una tabla. Un índice es una estructura auxiliar que pertenece a la base de datos y permite acceder a registros por valor. Por ejemplo, en la tabla **Alumnos** de campos {LU: String, Nombre: String, Nacimiento: Nat}, un índice para el campo LU tiene como claves todas las libretas y como significado los registros que tienen esa libreta en el campo. Un índice para el campo Nacimiento tendrá una entrada por cada año en el que nació un alumno y asociada el conjunto de registros de alumnos que nacieron en ese año.

Por otra parte se agrega la funcionalidad de hacer **joins** (o uniones) entre dos tablas. La unión entre tablas se hace designando un campo que ambas tengan en común. El resultado de un join es un conjunto de registros con campos igual a la unión de campos de las dos tablas. Las tablas del conjunto de registros resultado es equivalente al producto cartesiano de ambas tablas filtrando solo los resultados donde en los registros provenientes de ambas tablas el valor del campo del join es el mismo (ver ejemplo en fig 1).

La operación de **join** de la base de datos debe devolver un iterador de registros, que recorra el conjunto de registros resultados del join.

Contexto de uso y complejidades requeridas

En la tabla 1 se definen términos útiles para las definiciones de complejidad. Estos términos son los mismos que se utilizan en la interfaz del módulo Base de Datos.

Se presentan los siguientes requerimientos de complejidad:

- `const Dato &Registro::dato(const string&) const`
  $O(1)$

LU	Nombre	Nacimiento
123/23	March	1642
007/25	Bond	1927
321/15	Amanda	1955

(a) Tabla **Alumnos**

LU	Materia
123/23	Algo2
321/15	Algo3
321/15	SisOp

(b) Tabla **Materias**

LU	Nombre	Nacimiento	Materia
123/23	March	1642	Algo2
321/15	Amanda	1955	Algo3
321/15	Amanda	1955	SisOp

(c) Join de **Alumnos** con **Materias** en LU

Figura 1: Ejemplo de una unión entre tablas

L	máxima longitud de un dato string
n, m	cantidad de registros en una tabla
T	cantidad de tablas de la base

C	máxima cantidad de campos en una tabla
c	máxima cantidad de claves en una tabla
cr	máxima cantidad de restricciones en un criterio
cs	cantidad de criterios distintos utilizados

sn	Cantidad de claves definidas en el diccionario para strings
S	máxima longitud de una clave definida en el diccionario para strings

Cuadro 1: Términos para definición de complejidades

- `void BaseDeDatos::agregarRegistro(string&, Registro&)`
  $O(copy(Registro))$ si la tabla en la que se inserta no tiene índices, $O([L + log(m)] * C + copy(Registro))$ sino.
- `bool BaseDeDatos::registroValido(const Registro&, const string&) const`
  $O(C + (c * n * L))$
- `bool BaseDeDatos::criterioValido(const Criterio &r, const string &nombre) const`
  $O(cr)$
- `void BaseDeDatos::crearIndice(const string &nombre, const string &campo)`
  $O(m * [L + log(m)])$ donde m hace referencia a la cantidad de registros de la tabla en la que se crea el índice.
- `join_iterator BaseDeDatos::join(const string &tabla1, const string &tabla2, const string &campo) const`
  $O(n * [L + log(m)])$ donde n y m hacen referencia a la cantidad de registros que tienen cada una de las tablas.
- `join_iterator BaseDeDatos::join_end() const`
  $O(1)$
- `BaseDeDatos::join_iterator BaseDeDatos::join_iterator::operator++()`
  $O(n * [L + log(m)])$ donde n y m hacen referencia a la cantidad de registros que tienen cada una de las tablas.
- `Registro BaseDeDatos::join_iterator::operator*()`
  $O(copy(Registro))$

Para las complejidades se debe tener en cuenta que se consideran que los nombres de tablas y los nombres de campos son de largo acotado.

Sobre la escritura de módulos

Los módulos de diseño provistos por la cátedra se encuentran escritos como parte del código .h en forma de comentarios. Estos comentarios son interpretables por doxygen[1] que genera una página web en un lindo formato. Doxygen utiliza un archivo de configuración de proyecto. Este archivo se encuentra en docs/Doxyfile.in. Para producir la documentación formateada en linux se debe ejecutar:

```
$> doxygen docs/Doxyfile.in
```

El resultado se encontrará en docs/html/index.html. Para referencia sobre cómo escribir la documentación pueden utilizar los módulos ya documentados. La documentación de la cátedra utiliza comandos personalizados que fueron configurados en el archivo Doxyfile.in.

[1] http://www.stack.nl/~dimitri/doxygen/

Diccionario para Strings

Para resolver las complejidades es necesario implementar un diccionario especializado para claves de tipo `string`[2]. Como parte de los requisitos del trabajo se deberá implementar el mismo acorde a un subconjunto de la interfaz de `map` de la `std`. El enunciado base provee un `.h` con la interfaz requerida así como una serie de tests de cobertura mínima.

Para la implementación deberán completar la sección privada de la clase `string_map`, agregar todas las implementaciones y completar el módulo, incluyendo pre y postcondición de las funciones, así como rep y abs. La estructura debe respetar las complejidades exigidas (ver archivo `.h`). Asimismo, el diccionario debe contar con un iterador a pares clave-valor en sus versiones regular y **`const`**. Avanzar el iterador sn veces debe tener complejidad $\mathcal{O}(sn * S)$.

Consideraciones para la entrega

- Todos los módulos que diseñen deben contar con una interfaz completa donde cada función pública debe tener precondición, postcondición, complejidad y aspectos de aliasing. Esto no es necesario para las funciones privadas pero es sumamente recomendado. Como parte del módulo también es necesario aclarar con que TAD se explica el módulo y explicitar el invariante de representación (rep) y la función de abstracción (abs).
- El rep debe estar escrito en lenguaje formal exceptuando las secciones del mismo que refieran a punteros o iteradores. Estas últimas pueden estar escritas en lenguaje natural.
- De ser útil para la escritura de pre/post condición, rep o abs, pueden declarar nuevas funciones de TADs en un documento auxiliar.
- En el TP se permite modificar las implementaciones provistas por la cátedra. Los cambios realizados deben ser informados y justificados en un documento auxiliar. Los cambios deben estar reflejados en la documentación doxygen de los módulos modificados.
- En el TP se permite utilizar los módulos provistos por la `STL`.
- El trabajo práctico se debe entregar como un único ZIP con el código fuente, el código de test y la documentación auxiliar. Se espera que el ZIP contenga instrucciones para compilar el código y ejecutar los tests (son admisibles instrucciones que impliquen el uso de CLion). El zip no debería contener código compilado (se recomienda hacer `$> make clean` o correr clean en CLion antes de producir el ZIP).
- La entrega se realizará la dirección `algo2.dc+{tn,tm}+tp2@gmail.com` utilizando `tm` o `tn` según corresponda el turno del grupo. El subject del mail deberá conformarse por las libretas universitarias de los integrantes separadas por ;.

Criterios de corrección

- La base de datos debe seguir cumpliendo toda la funcionalidad que cumplía originalmente.
- La base de datos debe poder realizar las nuevas funcionalidades.
- Se deben cumplir las complejidades temporales requeridas.
- Se deben respetar los criterios de interfaz establecidos en el enunciado.
- Todos los nuevos módulos deben contar con interfaz completa, rep y abs.
- Las funciones agregadas a módulos existentes deben contar con interfaz completa.
- Las modificaciones realizadas en módulos existentes deben estar reflejados en el rep y el abs del módulo modificado.
- Se evaluará la prolijidad del código.

[2]Para más información ver: Sedgewick, Robert, and Kevin Wayne. Algorithms. Addison-Wesley Professional, 2011.