ESCUELA POLITÉCNICA SUPERIOR, UAM
Estructuras de Datos 2016 – 2017

Práctica 3. Arquitectura de un Sistema de Gestión de Bases de Datos

Fecha de entrega: última semana de prácticas

CONSEJOS ÚTILES

1. Las memorias, cuando se piden, son importantes: no sólo sirven para que vuestro profesor tenga una idea de lo que habéis hecho y cómo lo habéis organizado; también proporciona indicaciones importantes sobre el razonamiento que habéis seguido para hacer las cosas de una determinada manera. Esto nos ayudará a evaluar mejor vuestro trabajo. Además, escribir una buena memoria es importante para los que, como vosotros, trabajarán en campo técnico: hacer que los colegas, los jefes y los clientes entiendan vuestro trabajo será una de vuestras actividades más importantes.
2. Puede parecer banal, pero... ¡poned el nombre de los componentes del grupo en TODOS los ficheros! No os podéis ni siquiera imaginar cuantas maneras hay de perder el rastro de quien ha hecho algo.
3. Leed la documentación. En el caso concreto de esta práctica, la documentación consiste en la descripción de la misma, los comentarios en las cabeceras de los módulos y el manual de la librería estándar de C (accesible de múltiples formas, especialmente recomendable mediante la utilidad man de Linux, sección 3).

OBJETIVOS

Esta práctica tiene los siguientes objetivos:

1. Proporcionar una perspectiva básica y práctica de la arquitectura de un Sistema de Gestión de Bases de Datos (SGBD), desde el análisis sintáctico o *parsing* de una consulta hasta las operaciones de acceso a disco, pasando por toda la lógica intermedia.
2. Comprender e implementar algunos de los algoritmos y estructuras de datos esenciales detrás de un SGBD: pilas, *demand-pull pipelines* y *nested loop joins*.
3. Aprender las operaciones de acceso a ficheros binarios de la biblioteca estándar de C: fopen, fread, fwrite, fseek, ftell, feof, fclose.

DESCRIPCIÓN

Introducción

Un SGBD moderno es una pieza de software complejo que comprende muchos componentes que se encargan de implementar sus funcionalidades: desde el análisis sintáctico o *parsing* de las consultas en SQL hasta el almacenamiento de datos en un soporte físico, pasando por otras funcionalidades adicionales como control de referencias cruzadas (claves foráneas), procedimientos almacenados, transacciones y seguridad. Por ejemplo, el código fuente de SQLite, un SGBD relativamente

pequeño y sencillo, consta de unas 65.000 líneas de código C. Con respecto a PostgreSQL, el capítulo 44 de su documentación (http://www.postgresql.org/docs/9.1/static/overview.html) provee una breve descripción de los detalles internos del servidor de PostgreSQL.

Por otro lado, el estándar SQL determina un lenguaje de consulta que permite específica *qué* se consulta, pero no el *cómo*, esto es, no proporciona una forma obvia de cómo se accede a las tablas y qué algoritmos y estructuras de datos auxiliares se emplean para obtener los resultados. Es responsabilidad del SGBD elegir un **plan de ejecución u operación** que obtenga los resultados de la forma más rápida y eficiente posible. Para conocer el plan de ejecución de una consulta concreta, el estándar SQL define el comando EXPLAIN (http://www.postgresql.org/docs/9.1/static/sql-explain.html) que da detalles sobre cómo una consulta es procesada.

Con la intención de profundizar más en los detalles concretos de la arquitectura de un SGBD y ser capaces de ir más allá de la mera descripción de los resultados de SQL e ir hacia la definición específica de las operaciones a realizar, presentamos en esta práctica el software ESECUELE. ESECUELE es un SGBD muy básico y simple (en torno a las 2.000 líneas de código C) que sin embargo es fácilmente extensible y permite realizar consultas básicas sobre bases de datos relacionales. Las consultas en ESECUELE no se ejecutan usando un lenguaje declarativo tal como SQL, sino usando una sencilla álgebra relacional, del tipo que las bases de datos usan para expresar el *query plan* intermedio.

Instrucciones de Compilación y Ejecución

ESECUELE fue desarrollado usando el IDE Netbeans, aunque se puede compilar directamente en una terminal mediante el comando make. Para trabajar sobre el código fuente se puede elegir cualquier alternativa (Netbeans, otro IDE, terminal más editor de texto, etc.) siempre que el resultado sea compilable mediante make.

La compilación genera un único ejecutable esecuele al cual hay que pasarle siempre dos argumentos: el modo de operación y la base de datos sobre la que trabajar. Se distinguen cuatro modos de operación:

1. createdb, que sirve para crear una nueva base de datos vacía.
2. define, que muestra un prompt interactivo (d>) para la definición de tablas.
3. insert, que muestra un prompt interactivo (i>) para la inserción de datos en las tablas.
4. query, que muestra un prompt interactivo (q>) para hacer consultas.

El script adjunto bank.bash proporciona un ejemplo de ciclo de creación y consulta sobre una pequeña base de datos de un banco. Se recomienda hacer pruebas con este ejemplo antes de probar con los datos de la biblioteca de la última práctica.

Limitaciones de ESECUELE

Buscando un equilibrio entre simplicidad y funcionalidad, hemos tenido que sacrificar algunas funcionalidades y características generalmente necesarias en un SGBD:

- La comprobación de errores es mínima y se centra en tratar de evitar errores producidos por un mal uso del programa. En general, la comprobación de fopen's, calloc's y malloc's se ha omitido para facilitar en lo posible la legibilidad del código. En todo caso, todo recurso es convenientemente liberado cuando deja de usarse, y eso se tiene que mantener con las modificaciones que hagáis (recuerda que valgrind es una herramienta muy útil para estos menesteres).
- Las columnas de una tabla no tienen nombres, teniendo que referenciarse mediante su posición (contando desde 0).
- En algunas operaciones es necesario especificar el tipo de dato de algún operando. Esto es necesario para que el analizador sintáctico sea lo más sencillo posible.
- No se pueden realizar operaciones de actualización ni borrado de registros.

Arquitectura de ESECUELE

La parte de organización y almacenamiento de bases de datos se compone de los siguientes componentes:

- Bases de datos: una base de datos se compone de tablas. El módulo database.c/h contiene el código que gestiona una base de datos mediante el tipo de dato database_t. Por cada base de datos se crea un directorio homónimo con un fichero binario bd con la información manejada por la base de datos.
- Tablas: el módulo table.c/h contiene la definición de table_t, el tipo de dato que sirve para operar con tablas. Cada tabla nueva creará un fichero con el mismo nombre de la tabla y extensión .table, donde se guardarán la descripción de la tabla y los registros de la misma.
- Registros: es la representación de una fila de una tabla. El módulo record.c/h contiene las primitivas del tipo de dato record_t para acceder a los registros de la tabla.

En lo respectivo a la parte de consultas, estas son

- Operaciones: una operación es un mecanismo de acceso iterativo a una fuente de datos, que puede ser una tabla a su vez otra operación. Una consulta será expresada mediante una operación construida por el analizador sintáctico. El módulo operation.c/h contiene la descripción del tipo de dato operation_t.
- Condiciones: las condiciones con tipo de dato condition_t evalúan a VERDADERO/FALSO para un resultado de una operación fuente y son usadas para implementar la operación SELECT.
- Proyecciones: distintas proyecciones projection_t de una sola columna con las que se implementa el operador PROJECT.
- Analizador sintáctico o parser: es el componente que procesa una consulta y la convierte en una operación sobre una base de datos. Debido a que el estudio de gramáticas, gramáticas de atributos y analizadores sintácticos pertenece a asignaturas posteriores a EDAT en el Plan de Estudios, hemos optado por un lenguaje de consulta basado en la notación posfija o notación polaca inversa que, como se vio en Programación 2, es fácilmente procesable mediante una pila.

Cada vez que el analizador sintáctico crea una operación, esta se recorre de forma iterativa siguiendo una *demand-pull pipeline*. Para ello, una operación tiene las siguientes funciones asociadas:

- next: avanza al siguiente resultado de la operación (siguiente fila de la consulta), devolviendo 0
  si ya no hay más resultados o != 0 en caso contrario.
- get(i): obtiene el i-ésimo campo (columna) del resultado.
- close: cierra la operación liberando los recursos.

Un ejemplo de cómo imprimir todos los resultados de una consulta sería este (extraído de la función query de esecuele.c):

```
types = operation_types(operation);
        ncols = operation_ncols(operation);
        nrows = 0;
        /* for every result of the operation*/
        while (operation_next(operation)) {
                /* the values are printed */
        nrows++;
                for (i = 0; i < ncols; i++) {
                        print_value(stdout, types[i], operation_get(i, operation)); printf("\t");
                        }
                printf("\n");
        }
        operation_close(operation);
```

Lenguaje de Definición, Inserción y Consulta

En el modo de definición (con el prompt d>) se pueden crear tablas. Los comandos son los siguientes:

TABLE table_name no_of_columns col0_type col1_type ... donde colX_type=INT|STR

Los siguientes ejemplos muestran la equivalencia entre el lenguaje de consulta de ESECUELE y sus casi-equivalentes en SQL:

ESECUELE	**SQL**	
TABLE table1 2 INT STR	CREATE TABLE	table1 (
	x INTEGER,	
	y TEXT	
	);	

En el modo de inserción (con el prompt i>) el único operador disponible es COPY, que toma como

parámetros el nombre de una tabla y la ruta a un fichero cuyos campos, separados por tabuladores, se copiarán a la tabla. Las líneas que empiecen por "#" son ignoradas y no se copian a las tablas.

En el modo de consulta (con el prompt q>) las consultas con considerablemente más complejas y se asimilan más a la notación del álgebra relacional que a SQL. Para simplificar el análisis semántico el lenguaje de consulta sigue una notación posfija o notación polaca inversa. En el apéndice al final del documento se adjunta una descripción de las operaciones, condiciones y proyecciones definidas en el lenguaje de consulta.

Formato de los registros

NOTA IMPORTANTE: Conviene familiarizarse previamente con las funciones de acceso a ficheros binarios de la librería estándar de C (entre otras, fopen, fclose, fseek, ftell, fread, fwrite, feof, ferror) antes de intentar implementar el formato de los registros que se detalla a continuación.

Cada tabla se compone de una cabecera y los registros. La cabecera contendrá el número de columnas seguido de un array con el tipo de cada una. Inmediatamente después irán los registros, que irán consecutivos. El formato de un registro consistirá de una secuencia de pares tamaño-contenido para cada columna.

Un ejemplo de una tabla con tres registros (entero, cadena, entero) sería el siguiente:

3	INT	STR	INT			
sizeof(int)	1	8*sizeof(char)	Johnson	sizeof(int)		234
sizeof(int)	2	6*sizeof(char)	Kenny	sizeof(int)		455
sizeof(int)	3	7*sizeof(char)	Connor	sizeof(int)		102

TAREAS A REALIZAR

0. (Incalculable) Leer detenidamente la descripción de la práctica. Comprender el código adjunto y familiarizarse con él. Ayudará crear la base de datos del script bank.bash, y ejecutar ESECUELE con un depurador para ir viendo el flujo de ejecución de creación, definición, inserción y consulta.
1. Añadir los tipos de dato double (DBL) y el tipo long (LNG).
2. Implementar los módulos table.c/h y record.c/h siguiendo el formato de registros descrito anteriormente. Proporciona una función main de ejemplo en el que se pruebe el correcto funcionamiento de estos módulos.
3. Implementar las operaciones COUNT, UNION, LIMIT, OFFSET. Para cada una de estas operaciones, prueba su funcionamiento mediante una consulta sobre la base de datos de bancos.
4. Crea una base de datos llamada "libros_db", con los datos de edicion, ventas y usuarios, y expresa y evalúa tanto en SQL como en el lenguaje de consulta de ESECUELE las siguientes preguntas:
    1. Lista de libros comprados por "jack".
    2. Números de libros comprados por ´jack´.
5. (opcional) implementar la operación join sobre una sola columna, especificando en

que columna se quiere hacer el join.

A ENTREGAR

La entrega de esta práctica consistirá en un fichero p3_XXX_YY.zip que contendrá, al menos:

1. El código fuente y un Makefile para su compilación de vuestra implementación de ESECUELE.
2. El directorio de la base de datos "libros_db" y otras que hayáis podido necesitar durante la implementación de ESECUELE.
3. Una memoria detallando el trabajo hecho para resolver las tareas solicitadas, **así como ejemplos que demuestren el funcionamiento de lo implementado en cada apartado**. **IMPORTANTE: la memoria es el lugar donde justificar el diseño de alto nivel de vuestro código y las pruebas sobre el mismo. Para discusiones sobre bajo nivel (codificación) el lugar adecuado son los comentarios en el propio código.**

APÉNDICE: Descripción del lenguaje de consulta de ESECUELE

Operaciones

- SEQUENTIAL: Escaneo secuencial, sus resultados son todos los registros de una tabla.
  Argumentos: nombre de la tabla a recorrer.
  Formato: table_name SEQUENTIAL
- SELECT: Selección, dada una condición, muestra los resultados de una operación de entrada que cumplen con la condición.
  Argumentos: operación, condición.
  Formato: operation condition SELECT
- PROJECT: Proyección, dado un número de proyecciones, las aplica sobre los resultados de una operación de entrada y muestra como resultados.
  Argumentos: operación, proyecciones, número de proyecciones.
  Formato: operation proj1 proj2 … projN N PROJECT
- PRODUCT: Producto, dadas dos operaciones de entrada, hace el producto cartesiano de los resultados de las dos operaciones.
  Argumentos: dos operaciones.
  Formato: operation1 operation2 PRODUCT
- COUNT: Contar, devuelve un único resultado con el número de resultados de la operación de entrada.
  Argumentos: operación.
  Formato: operation COUNT
- UNION: Unión, concatena los resultados de dos operaciones con las mismas columnas.
  Argumentos: dos operaciones.
  Formato: operation1 operation2 UNION
- LIMIT: Acotar, muestra los primeros N resultados de una operación de entrada. Argumentos: operación, límite.
  Formato: operation N LIMIT
- OFFSET: Desplazar, muestra todos salvo los primeros N resultados de una operación de entrada.
  Argumentos: operación, offset.
  Formato: operation offset OFFSET

Condiciones

- C_TRUE: evalúa siempre a verdadero.
  Formato: C_TRUE
- C_NOT: NOT booleano de una condición previa.
  Argumentos: condición.
  Formato: condition C_NOT
- C_AND: AND booleano de dos condiciones de entrada.
  Argumentos: dos condiciones.
  Formato: condition C_AND
- C_OR: OR booleano de dos condiciones de entrada.

Argumentos: dos condiciones.
Formato: condition C_OR

- C_COLEQCTE: evalúa si una columna del resultado de una operación es igual a una constante.
  Argumentos: columna, tipo de dato, constante.
  Formato: column type value C_COLEQCTE
- C_COLEQCOL: evalúa si una dos columnas del resultado de una operación son iguales:
  Argumentos: dos columnas.
  Formato: column1 column2 C_COLEQCOL

Proyecciones

- P_COL: proyecta una columna del resultado de una operación de entrada:
  Argumentos: tipo de la columna, columna.
  Formato: type column P_COL
- P_SUM: proyecta la suma de dos columnas de tipo entero del resultado de una operación de entrada:
  Argumentos: dos columnas.
  Formato: column1 column2 P_SUM

Algunas de las operaciones más comunes sobre tablas y sus equivalentes en SQL se muestran en la siguiente tabla:

ESECUELE	**SQL**
table1 SEQUENTIAL	SELECT * FROM table1;
operation COND SELECT	SELECT * FROM operation WHERE COND;
operation 1 0 2 PROJECT	SELECT y, x FROM operation;
operation1 operation2 PRODUCT	SELECT * FROM operation1, operation2;
operation COUNT	SELECT COUNT(*) FROM operation;
operation1 operation2 UNION	operation1 UNION ALL operation2:
operation 1000 LIMIT	operation LIMIT 1000;
operation 40 OFFSET	operation OFFSET 40;