Envolviendo la API de Firebird/Interbase SQL Server

Cherencio, Guillermo Ruben
Universidad Tecnológica Nacional, Facultad Regional Delta

Abstract
El Sistema de Gestión de Bases de Datos Firebird/Interbase posee un modulo de comunicaciones que exhibe hacia su exterior una API accesible desde lenguaje ANSI C; esta API permite establecer una comunicación con el sistema, debido a su complejidad, muchos desarrolladores ANSI C deberán optar por otros sistemas gestores de bases de datos o bien invertir muchas horas de trabajo para lograr una comunicación eficiente con una base de datos Firebird/Interbase. El presente trabajo implementa una librería que envuelve a esta API , logrando una interfaz mas simple y abstracta sobre la cual desarrollar aplicaciones. Los programas de ejemplo provistos, muestran la resolución de distintos tipos de consultas en forma legible y con pocas lineas de código. Las pruebas realizadas han sido satisfactorias en cuanto a la velocidad de ejecución y su facilidad de uso. La librería se distribuye bajo licencia LGPL. Se abren nuevas líneas de investigación y futuros desarrollos, la posibilidad de portar este trabajo a otros lenguajes y plataformas, facilitando el acceso a una comunidad cada vez mayor de usuarios que deseen utilizar un sistema de gestión de base de datos maduro y de código abierto.

Palabras Clave
ANSI C Firebird Interbase SQL API

Introducción

Los Sistemas de Gestión de Bases de Datos (SGBD)[1] proveen una interfase para programas de aplicación (application programming interface, API) que permiten la vinculación del SGBD con el exterior. A través de esta API podemos establecer una conexión con el SGBD, utilizar, crear y administrar bases de datos. El SGBD Firebird[1] es la versión de código abierto del SGBD Interbase, de la empresa Borland/Inprise®; es un gestor de base de datos que provee al usuario una amplia funcionalidad, a través de distintos objetos: functions, domains, tables, views, indexes, generators, exceptions, triggers, execute stored procedures, select stored procedures, etc.. Ambos SGBD son compatibles en muchos aspectos y en especial en cuanto a su API, por ello se mencionará como Firebird/Interbase (FB).

La API que exhibe al exterior es accesible desde el lenguaje ANSI C, se han desarrollado en el mercado drivers basados en esta API para distintos lenguajes de programación, como por ejemplo, el driver JDBC/JCA Jaybird[2] para Java, la librería IBPP para C++[3], librerías para PHP[4], etc. Sin embargo, no se ha encontrado en el mercado una librería de mayor nivel de abstracción y por ende, de mayor facilidad de uso para programadores ANSI C no experimentados. En la carrera de Ingeniera en Sistemas de Información de la Universidad Tecnológica Nacional, los alumnos utilizan el lenguaje ANSI C como un Taller extracurricular o en asignaturas tales como Algoritmos y Estructuras de Datos, Sistemas Operativos, Gestión de Datos, etc.; cualquier intento de integración curricular, que pretenda la participación de alumnos de distintas asignaturas en un proyecto común en donde se requiera el uso del lenguaje ANSI C y una base de datos FB, se encontrará con el problema de la complejidad de la API de esta base de datos. Esta complejidad llevará al alumno a perder el objetivo global perseguido y deberá invertir horas y esfuerzo para lograr establecer una conexión con la base de

1 Disponible en http://www.firebirdsql.org

2 Disponible en http://www.firebirdsql.org/en/jdbc-driver/

3 Disponible en http://www.ibpp.org/

4 Distribuido con PHP y/o instalable a partir de fuentes PHP, ver http://php.net/manual/en/ibase.installation.php

datos, ejecutar una consulta y manipular todos los tipos de datos FB desde el lenguaje C.

El presente trabajo pretende resolver este problema proponiendo una nueva interfase externa a través del desarrollo de una librería C que envuelva la API C de FB y permita a los alumnos y desarrolladores ANSI C en general, una forma mas abstracta y sencilla para trabajar con este SGBD, sin tener que preocuparse por cuestiones internas de la arquitectura de FB y poder concentrarse en las consultas y procesos que debe realizar en el proyecto o trabajo propuesto. No se trata de ahorrar trabajo al alumno o programador, sino de superar una contradicción: el alumno esta trabajando con abstracciones tales como: "entidades", "relaciones", "atributos", "esquemas", etc. derivadas del modelo de datos[2], luego, para usar estas abstracciones, se requiere de uso de buffers, desplazamientos de punteros, manipuladores, estructuras complejas, con alto nivel de detalle y ligado a la estructura física de la base de datos.

El producto de este trabajo también pretende una utilización profesional de esta librería en desarrollos en lenguaje ANSI C en la plataforma Linux, poder incorporar el uso de esta base de datos en proyectos de tipo cliente – servidor basado en lenguaje ANSI C, utilizando interfaz gráfica o bien consola. Permitir el acceso y ejecución de todo tipo de consultas y, de ser requerido, poder tomar el control de la carga de datos en memoria, dependiendo del nivel de experticia que posea el usuario final.

Difundir el uso de FB, como base de datos de código abierto, en el campo académico y profesional, apoyando a todo el movimiento propuesto por la Free Software Foundation y que la misma sea accesible de forma simple por todos los desarrolladores ANSI C, sin necesidad invertir grandes cantidades de tiempo, disminuyendo la curva de aprendizaje que implicaría trabajar directamente con la API de FB; así como también el ahorro de dinero en la adquisición de licencias o software adicional para este tipo de proyectos, ya que, la librería propuesta estará disponible a toda la comunidad bajo licencia LGPL, denominada proyecto libfb en sourceforge y puede ser descargada desde http://sourceforge.net/projects/libfb/?source=directory.

Elementos del Trabajo y metodología

Todo el trabajo se desarrolló en las siguientes etapas, tomando los lineamientos generales de Blanchette[3]:

1. Requerimientos
2. Análisis de la API de FB.
3. Diseño de estructuras y tipos de datos.
4. Diseño de prototipos de funciones.
5. Implementación y documentación de funciones.
6. Implementación de programas de ejemplo y de prueba.
7. Empaquetado, publicación y distribución.

1. Requerimientos

Los requerimientos son genéricos, pues el usuario destinatario de este trabajo es muy amplio, no obstante, considero que el mayor requerimiento a cumplir esta centrado en la facilidad de uso de esta librería y de las abstracciones con las cuales deberá lidiar el usuario, en especial el usuario no experto, el cual requiere de abstracciones simples y fáciles de comprender [4]. La librería deberá soportar los tipos de datos básicos[5] de FB y permitir una representación simple en lenguaje C.
Permitir una forma simple de conexión a una base de datos FB, ejecutar una consulta, recuperar los datos de cada tupla y cerrar la conexión. Permitir activar o desactivar los mensajes de error o advertencias de la

5 Se refiere a los tipos FB: INTEGER, CHAR, VARCHAR, DATE, DECIMAL

librería[6]; permitir recuperar errores SQL (códigos de error, mensajes, etc).

2. Análisis de la API de FB.

Se recomienda la lectura del artículo "Conceptual Architecture for InterBase/Firebird" [5] para comprender la arquitectura FB, antes de comenzar con "InterBase 6 API Guide" (AG) [6]; en el capítulo 4 vemos que para conectar una base de datos debemos trabajar con un buffer de 256 bytes, a través de un puntero, denominado DPB (database parameter buffer) en el cual, en posiciones específicas, debemos cargar los distintos parámetros de la conexión a la base de datos; una vez que logramos conectar una base de datos, obtenemos un manipulador de la misma (algo de tipo isc_db_handle); este manipulador se usará luego para las sucesivas llamadas que involucren a esta base de datos.

El capítulo 5 "Working with Transactions" muestra la forma de utilizar transacciones las cuales son requeridas, incluso, cuando debemos procesar una simple consulta a la base de datos, debemos obtener un manipulador de la transacción (algo de tipo isc_tr_handle) y también se deberá trabajar con un buffer que describa los parámetros de la transacción denominado TPB (transaction parameter buffer).

El capítulo 6 "Working with Dynamic SQL" describe la complejidad de realizar una consulta a partir de un código SQL almacenado en una variable, resumiendo, los pasos a seguir son los siguientes:

1. Determinar si las llamadas a la API pueden procesar el código SQL.
2. Representar el código SQL como una cadena de caracteres dentro de la aplicación.
3. Si es necesario, asignar una o más estructuras de tipo XSQLDA para parámetros de entrada y valores de retorno[7].
4. Usar el método API apropiado para procesar el código SQL.

Los pasos 1 y 2 son bastante triviales[8], la estructura de tipo XSQLDA tiene cierta complejidad, una cabecera y un puntero a un arreglo contiguo de N instancias de tipo XSQLVAR que representan N campos, para describir N parámetros de entrada o bien N parámetros de salida. Es responsabilidad del programador la asignación, uso y liberación de estas estructuras a medida que se procesa el código SQL. De acuerdo al tipo de instrucción SQL, existen 4 métodos distintos, los cuales implican conjuntos mutuamente excluyentes de llamadas API diferentes para procesar el código SQL.

Por último, la desconexión a la base de datos es trivial, pero tipos de datos más complejos, tales como BLOB o ARRAY merecen capítulos completos para su tratamiento y caen fuera del alcance de este trabajo.

3. Diseño de estructuras y tipos de datos.

Acorde con los requerimientos planteados y lo descripto en el punto anterior, surge la necesidad de ocultar este nivel de complejidad al usuario, proponiendo estructuras mas abstractas y fáciles de usar.

3.1 Conexión.

Se debe ocultar la complejidad del buffer DPB y la necesidad de desplazamientos de punteros para completar los datos de conexión, se propone para ello el siguiente tipo de dato (conociendo los límites que tienen los nombres de usuario, contraseñas, nombres de las bases de datos, etc.) fb_db_info:

---

6 Debido a que la librería no debe asumir el contexto de ejecución del usuario, podría utilizarse en una aplicación o script cgi, en donde la salida de la librería no debería afectar a la aplicación.

7 También denominados parámetros de salida.

8 El sentido de la palabra trivial o trivialidad, se utiliza para denotar sencillez, algo no complejo.

```
typedef struct {
/** handle de la conexión de la base de datos */
  isc_db_handle db1;
/** vector de status de conexión */
  ISC_STATUS status_vector[32];
/** nombre de la base de datos */
  char dbname[512];
/** usuario utilizado para conectarme a b.d. */
  char user[32];
/** contraseña para conectarme a b.d. */
  char passw[32];
/** rol del usuario para conectarme a b.d. */
  char role[32];
/** version del servidor FB */
  char fb_version[50];
/** tamaño de la página de la b.d. */
  int fb_page_size;
/** número de buffers de la b.d. */
  int fb_num_buffers;
/** true si la b.d. es read-only  */
  int fb_read_only;
/** dialecto FB utilizado */
  int fb_sql_dialect;
} fb_db_info;
```

Este tipo contiene los datos requeridos para la conexión y también contiene espacio suficiente para almacenar la información de conexión que nos provee FB y que libfb almacenará automáticamente; de esta forma, el prototipo de la función de conexión y desconexión podría ser:

```
int fb_do_connect(fb_db_info *dbinfo);
int fb_do_disconnect(fb_db_info *dbinfo);
```

el valor de retorno representa un valor de verdad, por lo tanto, su uso podría ser:

```
fb_db_info dbinfo;
strcpy(dbinfo.user,"sysdba");
strcpy(dbinfo.passw,"masterkey");
strcpy(dbinfo.dbname,"localhost:/.../x.fdb");
strcpy(dbinfo.role,"sysdb");
if (fb_do_connect(&dbinfo)) {
   // se conecto ok!
….
  fb_do_disconnect(&dbinfo);
}
```

el cuadro indica la forma de conectarse y desconectarse, en donde dbname tiene la forma “server[/port]:base de datos fisica”[9], incluso libfb propone el siguiente prototipo para facilitar aun más la conexión:

```
void fb_pre_connect(fb_db_info *dbinfo, char
*dbname, char *user, char *passwd, char *role);
```

y por lo tanto, puede usarse de la siguiente forma:

```
fb_db_info dbinfo;
fb_pre_connect(&dbinfo, "localhost:/.../x.fdb",
“sysdba”, "masterkey", "sysdb");
if (fb_do_connect(&dbinfo)) {
   // se conecto ok!
….
  fb_do_disconnect(&dbinfo);
}
```

3.2 Consultas.

El diseño de estructuras mas abstractas para el manejo de consultas implica ocultar TPB, XSQLDA, XSQLVAR, etc.. ¿Qué tipo de dato abstracto o estructura de dato[7] mejor se adapta para representar un conjunto de tuplas en memoria? Tuplas que deben recorrerse hacia adelante, hacia atrás y deben recuperarse los valores correspondientes a cada columna o atributo. Se consideró el concepto de lista doblemente enlazada para tal fin. Cada elemento de la lista representa una tupla, cada tupla contendrá un número fijo de campos, tal como lo indica el modelo relacional[8]. Este número fijo de campos será el mismo para toda la consulta. Otra cuestión es ¿Cómo representar los valores de cada campo? La estructura XSQLVAR posee el atributo sqldata de tipo char * el cual representa el valor de un campo; por lo tanto, los valores de los campos serán representados por cadenas de caracteres. Si bien no es el objetivo de libfb los tipos de datos mas complejos de FB, no obstante, se optó por un puntero de tipo void * que

9 Corresponde a la forma habitual de identificar en FB a una base de datos determinada.

facilitará soportarlos a futuro. Como los valores pueden ser de longitud variable, sabiendo que contamos con un número fijo de campos, podemos utilizar un arreglo contiguo de punteros a void * para representar cada valor, de cada campo, en cada tupla. Este concepto se representa en el tipo de dato rquery:

```
typedef struct rquery {
/** arreglo de apuntadores que apunta a los valores
de los atributos */
  void *col;
  /** puntero a siguiente tupla */
  struct rquery *next;
  /** puntero a tupla anterior */
  struct rquery *prev;
} rquery;
```

esta estructura permitirá cargar en memoria las tuplas recuperadas de la base de datos, no obstante, existen una serie de datos que aplican al conjunto de tuplas, por ejemplo: la cantidad de campos, cantidad de tuplas, nombres de los campos, códigos de error, mensajes de error, cantidad de tuplas insertadas, actualizadas, seleccionadas o borradas en ultima consulta, dónde comienza la primer tupla y dónde termina la última tupla, etc. para ello se utilizó el tipo de dato query:

```
typedef struct query {
  /** cantidad de tuplas */
  int rows;
  /** cantidad de columnas */
  int cols;
  /** arreglo de apuntadores char * de longitud
    query.cols con los nombres de las columnas */
  void *colname;
  /** flag de error true->error, false->no error */
  int fb_error;
  /** código sql de error */
  long SQLCODE;
  /** ultimo status del ultimo fetch realizado */
  ISC_STATUS FETCHCODE;
  /** mensajes de error */
  char *errmsg;
  /** cuántas tuplas fueron actualizadas */
  int rows_updated;
  /** cuántas tuplas fueron insertadas */
  int rows_inserted;
  /** cuántas tuplas fueron borradas */
  int rows_deleted;
  /** cuántas tuplas fueron recuperadas */
  int rows_selected;
  /** cuántas tuplas obtenidas (fetched) en cada
ejecución del query */
  int rows_fetched;
  /** puntero a la primer tupla en memoria */
  struct rquery *top;
  /** puntero a la ultima tupla en memoria */
  struct rquery *bottom;
} query;
```

Esta estructura representa el punto de partida para acceder a los resultados de la consulta desde un punto común, así como también obtener información acerca de la ejecución de consulta. Es el punto inicial y argumento utilizado por muchas funciones de la librería, que permiten realizar operaciones sobre los resultados de la consulta en forma global, por ejemplo: exportación de datos, liberación de memoria, obtener los nombres de los campos, etc..

La librería también permite un uso más profesional de la misma o bien con un mayor nivel de control sobre la ejecución de todo tipo de consultas, a través de las funciones fb_do_exec_query() y fb_do_query(), cuyos prototipos son:

```
int fb_do_exec_query(fb_db_info *dbinfo,char
*sql_stmt,int pinput,int poutput,int (*onDoQuery)
(int eventType,fb_query_info *qi,void
*buffer2),void *buffer);
int fb_do_query(fb_db_info *dbinfo,int queryId,char
*sql_stmt,int (*)(int eventType,fb_query_info
*qi,void *buffer),void *buffer);
```

estas funciones permiten que el usuario utilice una misma llamada a la función para ejecutar N veces una misma consulta parametrizada; cuando comienza la ejecución de la consulta, sucederán una serie de eventos (eventType), por cada evento, se llamará a una función de callback[10] pasada como parámetro de estas

10 Se utiliza el concepto en el setnido de retrollamada: una función A que se usa como argumento de otra

funciones, la cual tiene el siguiente prototipo[11]:

```
int onDoQuery(int eventType,fb_query_info
*qi,void *buffer2);
```

esta función de callback será llamada ante la ocurrencia de los siguientes eventos:

Eventos en consultas XXXX

Evento:FB_EXECUTE_QUERY_OK Descripción: consulta ejecutada Ok Valor EventType: 1
Evento:FB_EXECUTE_QUERY_ERROR Descripción: consulta ejecutada con error Valor EventType: 2
Evento:FB_SET_QUERY_INPUT Descripción: se deben indicar los valores de los parametros de entrada de la consulta Valor EventType: 3
Evento:FB_SET_QUERY_OUTPUT Descripción: se debe indicar los valores de los parametros de salida de la consulta Valor EventType: 4
Evento:FB_START_QUERY_ERROR Descripción: la consulta en cuestión dio error al inicio de su ejecución y no puede avanzar en las etapas subsiguientes del proceso Valor EventType: 5
Evento:FB_MEMORY_QUERY_ERROR Descripción: la consulta en cuestión dio error al intentar asignar memoria en forma dinámica dentro de fb_do_query() y no puede avanzar en las etapas subsiguientes del proceso Valor EventType: 6
Evento:FB_FETCH_RECORDS Descripción: la consulta en cuestión ha avanzado en su ejecución y esta lista para recuperar tuplas Valor EventType: 7

Tabla 1

La función de callback deberá realizar su trabajo en función del evento que haya ocurrido y una vez realizado su trabajo, podrá informar a fb_do_query() o fb_do_exec_query() -a través de su valor de retorno- para que éstas reaccionen en consecuencia:

Valores de retorno de función de callback

Valor de Retorno:FB_CONTINUE Descripción: indica a fb_do_query() que continúe con la normal ejecución de la consulta en cuestión Valor: 1
Valor de Retorno:FB_ABORT Descripción: indica a fb_do_query() que cancele, aborte la normal ejecución de esta consulta en cuestión, pero continuar con el resto de las consultas pendientes que pudiera haber. Valor: 2
Valor de Retorno:FB_ABORT_ALL Descripción: le indica a fb_do_query() que cancele, aborte la normal ejecución de todas las consultas en cuestión. Valor: 3

Tabla 2

La función de callback, para llevar adelante la programación de cada evento, puede requerir acceder a información menos abstracta que la exhibida por el tipo de dato query, para ello, se utiliza el tipo de dato fb_query_info al cual tendrá acceso esta función:

```
typedef struct {
  /** conexión con la base de datos */
  fb_db_info *dbinfo;
  /** manipulador (handle) de la transacción
asociada con esta consulta */
  isc_tr_handle *tr1;
  /** manipulador (handle) del SQL statement
asociada con esta  consulta */
  isc_stmt_handle *stmt;
  XSQLDA *in_sqlda;
  XSQLDA *out_sqlda;
  /** SQL consulta a ejecutar */
  char *sql;
  /** nombre del cursor  */
  char *cursor;
  /** Id de la consulta desde el punto de vista del
usuario, permite identificarlo en caso de usar
    la misma función de callback para varias
consultas  */
  int queryId;
  /** método FB de resolución de la consulta */
  int method;
  /** flag de error true->error, false->no error */
  int fb_error;
```

función B. Cuando se ejecuta B, ésta ejecuta A. Ver http://es.wikipedia.org/wiki/Callback_(inform%C3%A1tica)

11 Ejemplo de prototipo de función de callback onDoQuery() provista por la librería.

```
  /** código sql de error */
  long SQLCODE;
  /** status del ultimo fetch realizado */
  ISC_STATUS FETCHCODE;
  /** mensajes de error */
  char *errmsg;
  /** cuántas veces se lanzo el evento
FB_SET_QUERY_INPUT, comenzando por 0 */
  int set_time;
  /** cuántas veces debe ser ejecutado esta consulta
parametrizada? */
  int set_maxtimes;
  /** cuántas tuplas fueron actualizadas */
  int rows_updated;
  /** cuántas tuplas fueron insertadas */
  int rows_inserted;
  /** cuántas tuplas fueron borradas */
  int rows_deleted;
  /** cuántas tuplas fueron recuperadas,
seleccionadas */
  int rows_selected;
  /** cuántas tuplas fueron obtenidas (fetched) en
cada ejecución de la consulta */
  int rows_fetched;
} fb_query_info;
```

4. Diseño de prototipos de funciones.

Todos los prototipos de funciones comienzan con "fb_"[12], las funciones que devuelven valores enteros indican el éxito (1) o fracaso (0) de la operación. Los nombres de las funciones están en inglés, lo cual no significa un problema para el programador avanzado y en cuanto a los alumnos o programadores novatos, libfb apoya al aprendizaje de estos términos que debe realizarse, pues la mayoría de las librerías importantes con las que deberá lidiar en su vida profesional están en inglés. El diseño se realizó en función de los objetivos perseguidos: algunas ocultan la complejidad de la API C de FB, otras son más triviales y sólo aportan a la facilidad de uso de la librería, otras son más complejas y permiten un uso más profesional o mayor control sobre la ejecución de consultas.

4.1 Funciones para ejecución de consultas.

Ya vimos los prototipos de las funciones fb_do_exec_query() y fb_do_query(); la primera se utiliza para consultas que no devuelven tuplas y/o consultas de actualización de la base de datos, veamos un ejemplo de uso simple utilizando la función de callback provista onDoGenericQuery()[13] :

```
query myquery;
fb_init(&myquery);
char *squery = "SELECT * FROM ...";
if ( fb_do_query(&dbinfo, 1, squery,
onDoGenericQuery, &myquery) ) { // query ok!
  if ( myquery.rows ) { // hay tuplas!
     printf("Recupero %d filas\n",myquery.rows);
     rquery *q = myquery.top;
     for(q=myquery.top; q; q=q->next) {
       int ncol;
       for(ncol=0;ncol < myquery.cols;ncol++)
         printf("[%s]", fb_get_col(&myquery, q,
ncol));
       printf("\n");
     }
  }
  fb_free(&myquery);
}
```

el tipo de dato query requiere una inicialización fb_init() y al final, una liberación de los recursos fb_free(); si bien el prototipo de fb_do_query() parece complicado, su uso no lo es, más cuando se trata de una consulta no parametrizada, el código recupera todas las tuplas y todos los campos, básicamente se trata de recorrer la lista y para facilitar el acceso a los datos de las columnas, se utiliza fb_get_col() que facilita el acceso a través de punteros; también puede usarse fb_get_col_byname() si hay necesidad de recuperar los valores de los campos a través de su nombre.

4.2 Funciones para ejecución de consultas de alto nivel.

12 A excepción de las funciones de callback o de tratamiento de consultas. Se provee una función de callback, a modo de ejemplo, denominada onDoGenericQuery.

13 Asumimos que ya estamos conectados a la base de datos, tal como se ejemplificó anteriormente.

Muchas veces, el programador solo necesita ejecutar una consulta y recuperar sus tuplas, similar al caso anterior, pero utilizando una función con un prototipo mas simple, como es el caso de fb_do_single_query():

```
query *myquery;
printf("Ejecuto Query!\n");
char *squery = "Mitabla";
if ( (myquery=fb_do_single_query(&dbinfo,
squery)) != NULL ) {
  printf("Recupero %d filas\n",myquery->rows);
  rquery *q = myquery->top;
…..
  fb_free(myquery);
}
```

el resto del código es igual al ejemplo anterior, cabe destacar que "Mitabla" puede ser una tabla, una vista, o un select stored procedure[14] , el programador no necesita especificar código SQL, sólo la fuente del dato. Esta aplicación es ideal para proyectos que involucren a alumnos o programadores novatos que no poseen conocimientos de SQL.

También es posible que todo el objetivo del programa sea ejecutar una consulta, con lo cual se puede utilizar fb_do_connect_squery(), la cual realiza la conexión a la base de datos, ejecuta la consulta y devuelve las tuplas para su proceso:

```
query *myquery;
char *squery = "Mitabla";
if ((myquery =
fb_do_connect_squery( "localhost:/var/lib/firebird/2.
5/data/isft.fdb", "sysdba", "masterkey", "sysdb",
squery)) != NULL) {
  printf("Recupero %d filas\n",myquery->rows);
…..
  fb_free(myquery);
}
```

14 En términos de FB se refiere a un procedimiento almacenado que permite la devolución de tuplas.

el primer argumento es la URL a la base de datos FB, luego se indica el usuario, la contraseña, el rol del usuario y la consulta a realizar, la función devuelve un puntero de tipo query, que luego puede usarse de la forma ya explicada.

4.2 Otras Funciones.

Esta librería posee 86 funciones[15] , muchas de las cuales no han sido mencionadas, existen funciones para el manejo de errores SQL, para uso de FB events, para uso de arreglos, blobs[16], exportación de datos a archivos, etc..

5. Implementación y documentación de funciones.

Los puntos 5 y 6 se fueron realizando en forma simultánea, se diseñaba el prototipo de la función pensando en los objetivos que se perseguían, luego se escribía un programa de ejemplo para ver su uso, se implementaba la función y se documentaba utilizando Doxygen[17], al mismo tiempo que se escribía el código, luego se comprobaba su correcto funcionamiento con los programas de ejemplo. Se realizaban pruebas e implementaciones parciales de las funciones, en forma incremental y reiterativa hasta lograr el objetivo perseguido.

6. Implementación de programas de ejemplo y de prueba.

15 Al momento de escribir este documento, pues libfb es un proceso dinámico, en donde alumnos y usuarios van recomendando ampliaciones y cambios.

16 FB permite lanzar eventos desde el código de un trigger o stored procedure, esos eventos pueden ser reconocidos por una aplicación que esté en ejecución y utilice esta librería, de esta forma, se puede programar la respuesta automática a dichos eventos. FB también soporta arreglos y blobs.

17 Herramienta que permite la documentación de programas fuentes, a medida que escribimos el código, véase http://www.doxygen.org

La librería se distribuye con 6 programas de ejemplo que documentan 6 formas distintas de uso, que se encuentran dentro de la carpeta /examples ; también se incluye una base de datos FB isft.fdb que debe utilizarse para ejecutar los programas de ejemplo. Todas las funciones se han probado utilizando estos programas de ejemplos y otros que fueron realizados ad-doc para las pruebas.

7. Empaquetado, publicación y distribución.

La librería libfb se distribuye bajo licencia LGPL[18], se incluyen los programas fuentes, ejemplos, documentación, etc. en una estructura de directorios comprimidos en formato zip que los usuarios pueden descargar, descomprimir y compilar utilizando make sobre la carpeta principal. La distribución y publicación de este trabajo se realiza a través del portal sourceforge[19] bajo el nombre libfb.

Resultados

Este trabajo se ha aplicado en trabajos prácticos integrados, tal como es el caso del proyecto SGANS[20], también se han recibido experiencias y recomendaciones por correo electrónico de usuarios que la han utilizado con fines profesionales desde que fuera registrado el proyecto en sourceforge el dia 16/11/2010, destacando su facilidad de uso y velocidad. Se ha realizado especial seguimiento de alumnos de 1er año que no saben SQL y que han logrado implementar programas client – server y también CGI[21] utilizando servidores como Mongoose.[22] También ha sido utilizada por alumnos de nivel superior[23] en sus trabajos prácticos finales.

Discusión

El presente trabajo ha sido realizado sobre una plataforma Linux, no obstante, podría ser portado a otras plataformas, más allá incluso, de las plataformas en donde hoy puede ejecutarse FB.

Todo el desarrollo esta realizado en lenguaje ANSI C y el resultado de la compilación de esta librería permite generar dos tipos de librerías: una estática y otra dinámica, para que luego el usuario pueda vincular la compilación de su proyecto con la librería estática o bien en tiempo de ejecución, poder cargarla en forma dinámica; esto permite la vinculación de esta librería con distintos lenguajes de programación y entornos de desarrollo.

Se han obtenido comentarios favorables en cuanto a la velocidad de recuperación de tuplas, pero no se ha realizado ningún trabajo comparativo en tal sentido, en relación con otros drivers o productos similares. De obtenerse resultados favorables, se podrían desarrollar nuevos drivers basados en esta librería.

Existe en FB una forma de pre-compilar[24] código utilizando la API C de FB, no obstante, podría modificarse al mismo (ya que es de código abierto) o bien partir de un proyecto totalmente nuevo que permita pre-compilación utilizando esta librería.

---

18 http://www.gnu.org/licenses/lgpl.html

19 http://www.sourceforge.net

20 El proyecto SGANS involucra a alumnos de nivel superior de 1ro, 2do y 3er año, se puede ver su enunciado en http://www.grch.com.ar/docs/pp/2012/sgans/tpfinal.pdf

21 Se refiere a Common Gateway Interface, que permite ejecutar programas que utilizan esta librería y que realizan operaciones sobre bases de datos FB en un entorno web.

22 https://code.google.com/p/mongoose/

23 Se refiere a alumnos del ISFT Nº 189 http://www.isft189.edu.ar , en donde el autor dicta clases, de primero a tercer año, para la realización de sus trabajos prácticos finales en las asignaturas: Programación I, Práctica Profesional, Bases de Datos.

24 Se refiere a la posibilidad de realizar programas C que hagan uso intensivo de la pre-compilación para generar código de esta librería. Una herramienta similar a Pro*C, conocido en el mundo Oracle®.

El trabajo permite el desarrollo de nuevos productos basados en esta librería que amplíen el alcance de FB a otras plataformas y servicios, por ejemplo, podría desarrollarse un servidor o middleware que admita peticiones de clientes y permita -a través de él- la ejecución de consultas FB desde clientes livianos, sin necesidad de instalación de ningún driver ni librería. El servidor podría implementar una técnica de "pool de conexiones", cache de consultas de uso intensivo, seguridad, etc. Desde el punto de vista didáctico, bien podrían hacerse proyectos desde asignaturas tales como Sistemas Operativos, Bases de Datos, Gestión de Datos, etc.

La librería facilita el uso de FB por parte de programadores ANSI C en proyectos client – server, en los cuales, muchas aplicaciones podrían ser migradas a esta base de datos o permitir que la misma aplicación pueda utilizarse con distintas bases de datos o realizar herramientas de migración y transferencias de datos entre distintas bases de datos.

La implementación actual esta orientada a resolver consultas que recuperen un número de tuplas almacenable en memoria principal, es de suponer, que si se pretende recuperar un volumen de datos que supere la memoria principal disponible (a pesar de los mecanismos de virtualización de memoria principal con los cuales pudiera contar el sistema operativo), el procesamiento de este tipo de consultas tendría un impacto negativo en el rendimiento general del sistema. La arquitectura actual no impide que se desarrollen nuevas implementaciones que contemplen esta situación[25] y permitan el uso de cursores y almacenamiento secundario para disminuir los requerimientos de memoria principal.

Conclusión

El SGBD FB posee un modulo de comunicaciones, el cual es accesible a través de una API escrita en Lenguaje ANSI C. Los programadores en Lenguaje ANSI C sólo cuentan con esta API si desean desarrollar aplicaciones client – server utilizando FB. Esta API C es compleja, requiere uso de punteros, comprender la arquitectura de FB, implementar, aplicar, todo lo documentado en el manual de referencia de esta API denominado FB API Guide, etc.. Esto implica que no es usable por un programador ANSI C novato o un alumno de una Ingeniería en Sistemas (o de cualquier otra carrera de sistemas), incluso un programador profesional, pues se requieren muchas horas de entrenamiento y el desarrollo de un driver o librería que permita su uso de una forma más abstracta, desvinculando al programador de cuestiones técnicas internas de FB y permitiendo que éste se dedique a la resolución del problema, haciendo posible, de esta forma, que FB sea un facilitador de su tarea y no un obstáculo. Este trabajo permite resolver este problema, implementando una librería en ANSI C que dota a la API C de FB de un mayor nivel de abstracción, facilitando su uso, ya sea en experiencias pedagógicas de laboratorio como así también a nivel profesional. La implementación propuesta esta disponible para toda la comunidad[26], las pruebas realizadas han sido satisfactorias, los programas de ejemplo propuestos permiten un uso muy sencillo y al mismo tiempo, de requerirse, un uso también más avanzado.

Este trabajo abre nuevas lineas de investigación en cuanto a mejoras que puedan introducirse como así también la posibilidad de extender esta librería a otros lenguajes y plataformas disponibles, permitir nuevos desarrollos que aporten

25 Por ejemplo, proveyendo otras funciones de callback que implementen nuevas tecnologías para el tratamiento de las consultas.

26 Puede descargar la librería desde http://sourceforge.net/projects/libfb

otras funcionalidades que requieran como sustento acceder a un SGBD como FB desde el lenguaje ANSI C, de una forma fácil, rápida y eficiente, sin necesidad de invertir grandes cantidades de tiempo y esfuerzo en comprender los detalles de implementación de la API C de FB.

Agradecimientos
Al Ing. Luis H. Perna de la Universidad Tecnológica Nacional, Facultad Regional Delta, por motivarme a la presentación del presente trabajo.

Referencias

[1] Castaño Miguel, Adoración de; Velthuis Piattini, Mario; Martinez, Esperanza Marcos, Diseño de Bases de Datos Relacionales, Editorial Ra-Ma, Madrid, 2000, ISBN 84-7897-385-0, Pag 5.
[2] Silberschatz, Abraham; Korth, Henry F.; Sudarshan S., Fundamentos de Bases de Datos, 4ta.Ed., Ed. Mc Graw Hill, Madrid, 2002, ISBN: 0-07-228363-7, Pag. 17.
[3] Blanchette, Jasmin, The Little Manual of API Design, Trolltech, a Nokia company, June 19, 2008, disponible en http://www4.in.tum.de/~blanchet/api-design.pdf
[4] Jackson, Daniel, Software Abstractions, MIT Press, 2006.
[5] Chan, Hubert; Yashkir, Dmytro, Conceptual Architecture for InterBase/Firebird, Faculty of Mathematical, Statistical and Computer Sciences, University of Waterloo, Canada, January 29, 2002, disponible en http://www.ibphoenix.com/resources/documents/design/doc_25 . Tambien disponible en formato pdf en http://www.grch.com.ar/docs/bd/tutorial/firebird/Conceptual.Architecture.Firebird.pdf
[6] InterBase 6 API Guide, Borland/INPRISE, disponible en http://www.firebirdsql.org/en/reference-manuals/
[7] Aho, Alfred; Hopcroft, John E.; Ullman, Jeffrey E., Estructuras de Datos y Algoritmos, Ed Revisada, Addison-Wesley, 1998, Pag. 51-53.
[8] Elmasri, Ramez; Navathe, Shamkant B., Sistemas de Bases de Datos, Conceptos Fundamentales, 2da Ed, Addison-Wesley Iberoamericana, ISBN 0-201-65370-2, Pag. 139-145.

Datos de Contacto:

Cherencio, Guillermo Ruben. Universidad Tecnológica Nacional, Facultad Regional Delta. San Martin 1171, CP 2804, Campana, Provincia de Buenos Aires, República Argentina. E-mail: grchere@yahoo.com.