IABD :

DISEÑO E IMPLANTACION DE UN *API* PARA EL ACCESO A UN SISTEMA MANEJADOR DE BASES DE DATOS RELACIONAL DESDE UN PROGRAMA C

Claudia Lucía JIMENEZ GUARIN

Rolando BLANCO MALDONADO

Grupo de Investigación SINBAD
Departamento de Ingeniería de Sistemas y Computación
Universidad de los Andes

Apartado Aéreo 4976, Santafé de Bogotá, COLOMBIA

e-mail : cjimenez@andescol.bitnet, rblanco@andescol.bitnet

fax: (57.1).284.18.90

0. Resumen

Este artículo presenta el diseño, implantación y resultados de evaluación de un *API* (*Application Program Interface*) para el acceso a Bases de Datos Relacionales por programadores no expertos en Bases de Datos. Está desarrollado para ser utilizado desde programas C, o desde programas que puedan hacer invocación a funciones en este lenguaje. El objetivo principal es lograr en las aplicaciones portabilidad, independencia del Manejador de Bases de Datos (SMBD), facilidad y rapidez de desarrollo y compatibilidad con la sintáxis de C. Se libera además al programador del manejo complejo de variables, estructuras e instrucciones propias al ambiente de programación del SMBD, permitiendo que programadores no expertos en el manejador específico logren utilizarlo sin necesitar entrenamiento adicional. Se muestran los resultados de una implantación específica, sobre ORACLE. Actualmente se está desarrollando el *API* para otros manejadores, INFORMIX y UNIFY.

Palabras Claves: Bases de Datos Relacionales, C, Programación de Aplicaciones sobre Bases de Datos.

1. Introducción

Gran número de aplicaciones requieren para su desarrollo de las facilidades de un lenguaje de propósito general (e.gr. C, Pascal, Lisp), además de las operaciones de definición y manipulación de datos que provee un Sistema Manejador de Bases de Datos (SMBD). Cada SMBD comercial especifica su propia interface o interfaces de acceso a la Base de Datos (BD) y, aunque en su mayoría son bastante similares, existe un alto riesgo de incompatibilidad entre aplicaciones escritas para distintos SMBD. Por lo general, la utilización de estas interfaces involucra un proceso de aprendizaje adicional, un esfuerzo grande de codificación y control, así como un mayor conocimiento del SMBD sobre el que se está desarrollando la aplicación, lo que lleva a programadores no expertos a problemas adicionales que en gran medida pueden ser evitados. En este documento se presenta la especificación, diseño e implantación de una interface simple, portable y homogénea con el lenguaje C [8] [2] [7], para construir aplicaciones sobre un SMBD-Relacional [4], sin necesitar ser un experto en Bases de Datos.

Este artículo se encuentra organizado de la siguiente manera: en la segunda sección se presentan las formas existentes de acceso a un SMBD-Relacional desde un programa escrito en lenguaje C; en la tercera sección se muestra globalmente la interface propuesta, **IABD**. En la cuarta sección se especifican las funciones que son ofrecidas por IADB, la arquitectura general del API y ejemplos detallados que ilustran la utilización de IABD. Finalmente se presentan algunas consideraciones acerca de su portabilidad y las conclusiones.

2. Formas de Acceso a un SMBD-Relacional desde un Programa C

La comunicación entre un programa C y un SMBD-Relacional puede ser realizada a partir de dos técnicas [9] [11] [12] [8], excluyentes entre sí:

a. Embebiendo instrucciones SQL en el código C del programa.
b. Realizando llamados directos al SMBD en forma de funciones.

La técnica de acceso a la BD a partir del uso de SQL embebido es, por lo general, la interface de comunicación más utilizada por programas C que necesitan acceder SMBDs Relacionales. Es implantada de forma similar por la mayoría de BDs Relacionales, lo que conlleva como principal ventaja la portabilidad de los programas escritos bajo esta técnica. Sin embargo implica una etapa de

precompilación, la cual complica la organización de programas fuente, así como la depuración de la aplicación.

La técnica de acceso directo a la BD aunque no implica labor de precompilación, es por el contrario implantada de manera distinta por cada SMBD Relacional lo que hace prácticamente imposible el transporte a través de distintos SMBD de las aplicaciones que acceden directamente a la BD.

Es claro que ambas opciones involucran serios inconvenientes. El problema es aún mayor para desarrolladores no expertos en BDs. Es por lo tanto deseable una tercera opción que tenga las ventajas de las interfaces existentes y evite en lo posible los problemas que actualmente conllevan.

Estas razones nos condujeron a la especificación, diseño e implantación de una nueva forma de acceso a la BD, la interface IABD. IABD permite accesar la BDR desde un programa C con las siguientes características:

- Sin etapa de precompilación. Esta característica facilita la organización de los archivos así como la depuración de los programas desarrollados.
- Tiene una interface simple y homogénea con el lenguaje C. La especificación de una interface simple garantiza la rápida creación de código fuente para programadores familiarizados con el lenguaje C.
- El código creado debe ser de fácil mantenimiento. Cambios en los requerimientos de una aplicación desarrollada utilizando esta interface no deben generar modificaciones en los módulos encargados del acceso a la BD.
- Alivio al programador de los detalles de acceso a la BD. Característica importante, si se desea facilitar el uso de la interface a personas no familiarizadas con el manejo de BD.
- Acomodarse a las necesidades del programador. Por lo general, distintas aplicaciones necesitan interactuar de manera diferente con el SMBD. Algunas a más alto (o bajo) nivel que otras. La interface debe estar en capacidad de ofrecer alternativas de acceso a la BD que permitan una interacción al nivel deseado, según los requerimientos y experiencia del programador.
- Debe ser portable. Se debe garantizar la máxima portabilidad posible. Esto se logra independizando la interface del SMBD sobre el cual se encuentra implantada. Al expresar los requerimientos que se desean hacer sobre la BD en forma de sentencias SQL se logra algo de esta portabilidad, pues se independiza a la aplicación desarrollada del lenguaje de manipulación de datos (DML) que realmente maneja el SMBD sobre el que corre la aplicación.

Para la utilización de IABD es necesario estar familiarizado con el lenguaje C, pero sin necesidad de ser un experto. Se requieren también algunos conocimientos sobre BDs Relacionales, y en especial sobre SQL, esto con el fin de poder especificar las consultas y demás operaciones que se quieren realizar. El acceso a la BD a través de IABD requiere de los mismos conocimientos que se necesitan para acceder una BD Relacional en forma interactiva. Los usuarios de esta interface serán personas que necesiten manipular o definir datos en una BD, y que tengan requerimientos que hagan necesaria la utilización de un lenguaje de propósito general.

3. Interface Propuesta: IABD

La ***Interface de Acceso a la BD*** o ***IABD*** esencialmente define el protocolo de comunicación con la BD. Este protocolo consiste en un conjunto de funciones y estructuras de datos accesibles a varios niveles de complejidad, según lo requiera el programador.

IABD independiza al programador de los detalles de acceso a la BD al encargarse de la mayoría de aspectos relacionados con el control de la comunicación con la BD: Manejo de *buffers*, control de

códigos de error, recuperación de tuplas y conversión de tipos de datos. IABD especifica por lo tanto una recuperación automática de tuplas, la cual se realiza fundamentalmente asociando una función del programa C usuario al conjunto de tuplas resultado del requerimiento SQL, de tal forma que esta función sea ejecutada para cada tupla de dicho conjunto. Para aquellos programadores que deseen tener un control sobre la recuperación de tuplas y todos los aspectos relacionados con esta acción, IABD especifica una serie de funciones y estructuras que permiten este nivel de interacción.

4. Las Funciones

En IABD se especifican cuatro tipos de funciones:

- **Funciones Básicas :** Son utilizadas para establecer y/o terminar la comunicación con la BD, así como para especificar bajo qué ambiente se desea esta comunicación.
- **Funciones de Alto Nivel** : Son las funciones que realizan una recuperación automática de las tuplas resultado de un requerimiento SQL.
- **Funciones de Bajo Nivel** : Hacen parte de este grupo las funciones que permiten al programa C usuario el control sobre la recuperación de tuplas.
- **Funciones de Manejo de Argumentos** : IABD permite la recuperación de las tuplas resultado de un SELECT, en variables locales y globales del usuario.

Independientemente del tipo de función que se utilice, en ningún caso el manejo de la memoria involucrada en la ejecución de un requerimiento SQL queda a manos del programa usuario. En un mismo programa el usuario puede utilizar indistintamente funciones de uno o varios niveles y realizar simultáneamente tantos accesos como desee.

Finalmente el requerimiento SQL que el programa C usuario transmite a IABD, a través de funciones de alto o bajo nivel, puede estar parametrizado siguiendo una sintáxis similar a la utilizada en las funciones `printf` y `scanf` [7].

4.1. Funciones Básicas

Hacen parte de este grupo las siguientes funciones:

Funciones Básicas	
conexion	autocommit
`commit`	`impresion`

Fig. 1

La especificación de cada una de ellas es:

```
int conexion (unsigned short cod_op, char *usuario,  char *password)
```

permite establecer o terminar la conexión con la BD, dados los datos del usuario y su clave.

```
int autocommit (unsigned short cod_op)
```

activa o desactiva la opción de validación automática. Por defecto esta opción está *desactivada.*

```
int commit (unsigned short cod_op)
```

permite realizar la validación o anulación de una transacción.

```
int impresion (unsigned short cod_op)
```

activa o desactiva la opción de impresión por pantalla. Por defecto, cuando se ejecuta una operación de selección a alto nivel sobre la BD y el usuario no especifica la acción a realizar para cada tupla resultado de la selección, estas se imprimen por pantalla. Si el usuario no desea que se realice esta acción, debe informarlo a IABD por medio de esta función.

4.2. Funciones de Alto Nivel

Las funciones de Alto Nivel ofrecen la máxima independencia al programador de los detalles relacionados con el acceso a la BD. Estas funciones permiten al usuario ejecutar cualquier operación de definición, inserción, supresión, actualización o búsqueda sobre la BD, en forma de una sentencia en sintáxis SQL. Cuando la sentencia es un SELECT, la recuperación de tuplas es realizada automáticamente. Adicionalmente IABD ofrece a este nivel la posibilidad de manejar o no los códigos de error resultado de la ejecución de la sentencia SQL (Fig. 2), así como la información de control relacionada con dicha ejecución.

Funciones de Alto Nivel	
(Recuperación Automática de Tuplas)	
Sin Inf. de Control	Con Inf. de Control
`sqlo`	`sqloCntr`
`sql`	`sqlCntr`

Fig. 2

Las funciones de Alto Nivel `sql` y `sqlCntr` permiten asociar a la ejecución de una operación de selección sobre la BD una función del programa usuario para ser ejecutada con cada tupla resultado de la selección. Los argumentos de dicha función solo pueden ser conocidos por IABD al momento de ejecución, por lo que requieren de un soporte para su manejo. Este soporte es dado por las funciones de Manejo de Argumentos (ver la sección 4.3).

La interacción más simple del programa usuario con IABD es a partir de las funciones de Alto Nivel que no involucran información de control, por lo tanto las funciones `sqlo` y `sql` serán presentadas primero, para finalizar con la descripción de las funciones `sqloCntr` y `sqlCntr`, las cuales dan al programa usuario información relacionada con la ejecución de la sentencia SQL [8] [2].

```
int sqlo (char *sentencia_sql)
```

El objetivo de esta función es el de permitir, de la forma más sencilla posible, la ejecución de cualquier sentencia SQL. Por ejemplo, la creación de la tabla ESTUDIANTE con atributos código, nombre y promedio a partir de un llamado a esta función, se realiza de la siguiente manera:

```
sqlo("create table estudiante ( código    number  not null, \
                                nombre    char(30) not null, \
                                promedio  number(4,2))");
```

En caso de que la operación ejecutada sea un SELECT y la opción de impresión esté activada, las tuplas seleccionadas se imprimen por pantalla.

```
int sql (char *sentencia_sql [, parámetros] ... )
```

Esta función permite la ejecución de cualquier sentencia SQL *parametrizada*. Además, realiza una recuperación automática de tuplas en caso de que la sentencia SQL sea un SELECT. La sentencia puede contener los caracteres '&', '%', '{', indicativos de los tipos de argumentos pasados como parámetros, los cuales tienen el mismo significado que en las funciones estándar de Entrada/Salida en C (`printf/scanf`) [7].

Los tipos de parámetros permitidos son:

- Variables o constantes.
- Dirección de una función entera, la cual será ejecutada para cada tupla resultado del SELECT.
- Direcciones de variables destinadas a recibir el valor de un atributo obtenido a través de un SELECT. Estas variables pueden ser LOCALES o GLOBALES, dependiendo de si la función entera especificada por el usuario actúa sobre variables globales o locales.

Los descriptores para los parámetros son:

a. `%t` Donde t es indicativo del tipo de la variable o constante que se recibe como parámetro, se siguen las mismas convenciones que `printf`, así por ejemplo un descriptor %d indica que uno de los parámetros es de tipo entero.

b. `&nt` Donde t es indicativo del tipo de variable que recibirá el valor de un atributo obtenido a partir de un SELECT. n es un entero, necesario cuando se manejan cadenas de caracteres o de bytes, indica el tamaño del atributo a recuperar. Por ejemplo un descriptor &d indica, al igual que en la función `scanf`, una dirección de una variable de tipo entero. Las variables que se especifican deben ser globales a menos que la invocación a la función `sql` haya sido precedida por una invocación a la función `argumentos` (ver sección 4.3).

c. `{f}` Indicativo de la dirección de una función entera (una sola función es permitida como parámetro) para ejecutar con cada tupla resultado de una selección. IABD soporta la acción de esta función sobre variables tanto locales como globales del usuario. Por defecto la acción de esta función es sobre variables globales.

   El usuario mantiene el control sobre la recuperación de tuplas y puede decidir en cualquier momento interrumpirla. Al terminar la recuperación automática de tuplas se retorna el control al programa del usuario, siguiendo el flujo normal de instrucciones.

Si el usuario considera necesario tener control sobre la ejecución misma de la sentencia SQL, IADB pone a su disposición dos funciones, equivalentes a las anteriormente presentadas, que adicionalmente le ofrecen información sobre número de tuplas involucradas en la ejecución, nombre de las columnas, etc. Estas funciones son sqloCntr y sqlCntr.

```
int sqloCntr(t_estadosql *estado_sql, char *sentencia_sql)
```

Esta función corresponde a la versión de la función `sqlo` que involucra información de control, la cual se encuentra concentrada en la estructura de datos `t_estadosql` especificada por IABD [2].

La función

```
int sqlCntr(t_estadosql *estado_sql, char *sentencia_sql[,parámetros] …)
```

es la correspondiente a `sql`, que involucra información de control. Al igual que para `sql`, la sentencia SQL puede estar parametrizada. El comportamiento de esta función es idéntico al de `sql`, solo que instancia los campos de la estructura de control pasada como parámetro, con la información relacionada a la ejecución de la sentencia SQL.

4.3 Funciones Manejadoras de Argumentos

IABD propone un manejo a los argumentos de las funciones del usuario inspirado en el manejo que da C a las funciones con número variable de parámetros [7]. Sinembargo, como IABD no cuenta con una pila de ejecución en dónde guardar los argumentos que deben pasarse a la función del usuario, este debe informar a IABD sobre la existencia de dicha función y sobre los argumentos que espera recibir.

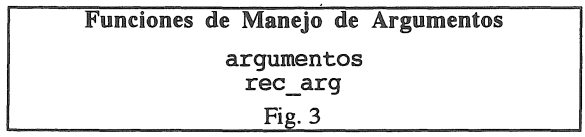

Fig. 3

Cuando el usuario desea realizar un SELECT sobre la BD, y necesita asociar a la recuperación de tuplas una función que actuará sobre variables locales, debe invocar a la función `argumentos` antes que a la función `sql` para especificar los parámetros que espera recibir su función.

La interface de la función `argumentos` es la siguiente:

```
int argumentos (char *tipos_args[, parámetros] …)
```

Donde `tipos_args` es una cadena de caracteres con descriptores del tipo **`&nt`** y/o **`%t`**, los cuales especifican los parámetros que tendrá la función del usuario.

Después de la invocación a la función `argumentos` IABD supondrá que, en el próximo llamado a la función `sql` en el que se especifique una función del usuario, esta función recibirá una pila de argumentos la cual contendrá los argumentos especificados en la invocación a `argumentos`, más *las direcciones de las variables en donde se recuperarán los atributos involucrados en el SELECT que se ejecuta*, variables que pueden ser locales.

Así por ejemplo:

```
...
argumentos("&d %f",&num_veces,prom_total);
sql("select {f} nombre &20s, código & from estudiante
     where promedio >= 4.0", imprima_estudiante,nombre_est,&código_est);
...
```

IABD supondrá que la función del usuario `imprima_estudiante` espera recibir una pila con los siguientes argumentos:

- La dirección de la variable `num_veces`.
- El contenido de la variable `prom_total`.
- La dirección de la variable `nombre_est`.
- La dirección de la variable `codigo_est`.

Los dos primeros argumentos fueron los especificados en la función `argumentos`. Los dos últimos son los que se asumen al realizar la ejecución de la función `sql`. La función `imprima_estudiante` será invocada para cada tupla resultado de la selección.

Si la invocación a la función `sql` no hubiera estado precedida por la función `argumentos`, en la ejecución de `sql` se habría supuesto que la función `imprima_estudiante` no tiene parámetros. Entonces, las variables en donde se recuperan los atributos de la selección (`nombre_est` y `código_est`) deben ser variables globales.

Para el manejo de la pila de parámetros que se pasa a la función del usuario IABD define el siguiente tipo.

```
type args_var
```

Para recibir la pila de argumentos, la función del usuario debe tener un único parámetro de tipo `args_var` (argumentos variables). Así la función `imprima_estudiante` del ejemplo, debe tener la siguiente forma:

```
int imprima_estudiante (args_var pila_args)
```

El tipo args_var puede ser definido como un apuntador a un bloque de memoria de tipo indeterminado:

```
typedef void *args_var
```

La función del usuario recupera los argumentos de la pila de argumentos a partir de la función `rec_arg`. Esta función retorna el primer argumento de una pila de argumentos de tipo `args_var`. En caso de que la pila esté vacia no se garantiza una terminación normal del programa en ejecución.

La interface de la función es la siguiente:

```
type  rec_arg(args_var pila_argumentos, type)
```

4.4. Funciones de Bajo Nivel

Las funciones de Bajo Nivel permiten al usuario administrar la recuperación de tuplas según sus necesidades. A este nivel de interacción solo es posible ejecutar selecciones (SELECTs) sobre la BD y el control de la ejecución es responsabilidad del usuario.

La ejecución y recuperación de tuplas resultado de una selección, a partir de las funciones de Bajo Nivel se realiza en dos fases:

Funciones de Bajo Nivel	
sqlb	sqls
sql_fetch	sql_abort

Fig. 4

a. Ejecución de la operación de selección, a partir de las funciones `sqlb` o `sqls`.
b. Recuperación de las tuplas resultado del SELECT (una a una), a partir de la función `sql_fetch`.

Eventualmente existe una tercera fase, en caso de que la recuperación de tuplas sea abortada intencionalmene por el programador al invocar la función `sql_abort`.

La función

```
int sqlb (t_estadosql *estadosql, char *sentencia_sql,
          tab_encabezados *encabezados)
```

ejecuta una selección sobre la BD. A diferencia de las funciones de Alto Nivel, no realiza la recuperación de las tuplas seleccionadas. La función `sqlb` es la encargada de obtener la información correspondiente a las columnas involucradas en la selección. Para cada columna se maneja el tipo y nombre del atributo [2].

La función

```
int sqls (t_estadosql *estadosql, char *sentencia_sql,
          tab_encabezados *encabezados [,parámetros] ...)
```

es muy similar a la función anterior, permite una selección SQL parametrizada con directivas de instanciación (%t). No se permiten directivas del tipo '{f}' y '&nt', ya que estas directivas se relacionan con la recuperación automática de tuplas, labor que a este nivel es realizada por el usuario. Esta función, a diferencia de la función `sqlb`, realiza una acción de instanciación. Después de esa acción su comportamiento es idéntico al de `sqlb`.

La función

```
int sql_fetch (t_estadosql estado_sql, int identificador,
                tab_buffers *buffers)
```

es la encargada de la recuperación de cada tupla; para tal efecto debe manejar una estructura que contenga los buffers en donde se almacenan los valores de los atributos de la tupla recuperada [2].

La función recupera la siguiente tupla resultado de la selección. En caso de que no queden tuplas por recuperar, o se presente algún error, retorna un descriptor inválido (negativo), y actualiza el campo `error` en `estado_sql`.

La función

```
void sql_abort (int identificador)
```

permite abortar la recuperación de tuplas y se encarga de liberar la memoria reservada para la recuperación de tuplas. El parámetro `identificador` corresponde al identificador de la sentencia SQL que se desea abortar.

5. Implantación

IADB fue implantado sobre ambiente Unix Versión 5 de AT&T [1], para el SMBD Relacional ORACLE Versión 6.0, accesado a partir de la librería OCI versión 1.1 [9]. El compilador utilizado fué gcc versión 1.40 [6]. Compilaciones sobre DOS se realizaron con turboC [3]. Las medidas de rendimiento nos indican que la carga adicional a la aplicación, con la utilización de IABD, no supera el 8% en el peor de los casos, siendo un 1% la sobrecarga normal.

IABD fue implantado siguiendo una arquitectura por capas, de tal manera que las funciones de bajo nivel implantan las funciones de alto nivel. La comunicación entre los módulos es netamente funcional, de tal manera que es explícita la forma de interacción entre los distintos componentes. Desde el punto de vista del desarrollo mismo del API, y en aras a lograr un máximo de portabilidad, los puntos de dependencia de un SMBD específico son minimizados.

Los grados de dependencia los podemos clasificar de la siguiente manera:

a - Con respecto a la Máquina y el Sistema Operacional.

Para lograr independencia de la máquina y del sistema se realizó la implantación en Ansi-C. Así, una aplicación desarrollada sobre ORACLE en ambiente UNIX, será compilable y ejecutable sobre el mismo SMBD en ambiente DOS o VMS. No se recomienda implantar IABD en lenguajes distintos a Ansi-C, ya que se pierde la portabilidad que garantiza este ambiente de desarrollo.

b- Con respecto al SMBD.

Cada implantación de la Interface se supone totalmente dependiente del SMBD sobre la que se implantó. Así por ejemplo una implantación sobre UNIFY [5] no podrá utilizarse cuando se desea acceder a una BD SQL/DS [12]. Sin embargo, a nivel de aplicaciones, implantaciones sobre distintos SMBD serán totalmente compatibles. Esto implica que una aplicación desarrollada sobre una implantación de la interface en un SMBD específico, puede ser utilizada

sobre cualquier otro SMBD sobre el que exista una implantación de IABD, siendo necesario solamente recompilar y reencadenar la aplicación.

6. Conclusiones

La interface IABD es la implantación de una forma de acceso directo a un SMBD Relacional. Esta interface no requiere de etapa de precompilación, es homogénea con el lenguaje C, es portable y permite una interacción a varios niveles con la BD. Existe una implantación sobre ORACLE en ambiente UNIX.

Una de las mayores bondades de IABD es el encapsular completamente los detalles de acceso al SMBD, de tal manera que se garantiza uniformidad en el desarrollo de software e independencia con respecto a un Manejador en particular, con un costo prácticamente despreciable en el rendimiento de la aplicación. Además, se logra un fácil mantenimiento y desarrollo al evitar al programador el aprendizaje de los detalles de conexión e interacción con el SMBD.

IABD está actualmente en etapa de implantación sobre UNIFY e INFORMIX y es utilizado en el desarrollo de varias aplicaciones usuarias de SMBD, entre ellas aplicaciones transaccionales distribuídas [10].

Referencias

[1] M. J. Bach. "The design of the Unix Operating System". Ed. Prentice Hall, 1986.

[2] R. Blanco. "Interface de Acceso a un Sistema Manejador de Bases de Datos Relacional desde un Programa C". Proyecto de Grado ISC-91-II-03 Ingeniero de Sistemas y Computación Universidad de los Andes, febrero de 1992.

[3] "Turbo C Reference Guide, Version 2.0". Borland International, 1988.

[4] E.F. Codd. "A Relational model for Large Shared Data Banks". ACM Vol. 13, Num. 6 (jun.1970).

[5] UNIFY. "Direct HLI Reference".

[6] GNU, Compilador gcc, obtenido vía FTP-anónimo.

[7] B. Kernighan, D. M. Ritchie. "The C Programming Language" (Second Edition). Prentice-Hall.

[8] M. C. Pajon. "OPIDUM Une Interface Procedurale de Haut Niveau pour ORACLE dans un Environnement Distribué". Centre de Recherche Groupe, Grenoble Francia, 1988.

[9] Oracle Corporation. "ORACLE References". 1988.

[10] C. Franky, J. Abásolo, R. Cucalón, G. Acosta, S. Maya. "PERSEO: Ambiente de Programación de Aplicaciones Transaccionales Distribuídas sobre Bases de Datos Relacionales". XVIII Conferencia Latinoamericana de Informática, Las Palmas de Gran Canaria, 1992.

[11] Digital Corp. "Relational Data Manipulation Language".

[12] IBM Corp. "SQL/Data System Application Programming for VM/System Product".