TRABAJANDO CON SERVIDORES SQL

Este trabajo fue desarrollado por **Sergio A. Carrasco L.**

Indice

TRABAJANDO CON SERVIDORES SQL

CLIP proporciona simples pero poderosas herramientas para accesar a varios servidores SQL. Los principios ocultos de estas herramientas, son considerados en este capítulo. Algunas características, posibilidades, clases y funciones relacionadas, son descritas a continuación.

1.- CARACTERISTICAS

Algunas de las características y posibilidades son las siguientes:

* CLIP unifica el uso de varios servidores SQL ocultando peculiaridades del API de sus desarrolladores, tanto como le sea posible. Sin embargo, deberías saber el dialecto SQL del RDBMS deseado ("Relational Data Base Manager System" = Sistema de Bases de Datos Relacionales).
* Las transacciones son soportadas. Por defecto cada instrucción SQL trabaja en su propia transacción (cada instrucción es pareada implícitamente con START/COMMIT). Sin embargo es posible lograr una secuencia de las instrucciones en una transacción, llamando explícitamente funciones START, COMMIT o ROLLBACK en puntos apropiados.
* El ordenamiento local permite cambiar el orden de filas localmente, no es necesario cargar el servidor con una claúsula ORDER BY para una consulta similar. Además, los órdenes locales permiten rápidas búsquedas para una fila deseada en enormes set de resultados.
* Automatización de actualizaciones de réplicas en bases de datos. Tú puedes proporcionar las apropiadas instrucciones UPDATE, DELETE e INSERT, que pueden ser ejecutadas automáticamente después de actualizar el set local de las filas seleccionadas.
* Dos modos de traer ("fetch"); (a) traer todo y (b) traer a pedido. En el primer modo, todos los resultados de las filas son traidos después de la ejecución de la instrucción SELECT. El proceso de traida puede ser observado y cancelado por una función definida por el usuario. En el segundo modo, la traida es realizada sólo con el alcance de la fila que ha sido direccionada. Tal tipo de ejecución es útil cuando la cantidad de filas de resultados no pueden ser estimados.

2.- INICIO RÁPIDO

2.1 CONSTRUYENDO UNA APLICACIÓN

Para construir una aplicación con el servidor SQL que desees, debes tener instalado (compilado) el paquete apropiado clip-<rdbms>. Los siguientes paquetes están por ahora disponibles:

Librería: clip-postgres

Sitio oficial: http://www.postgresql.org/index.html
Enlaces interesantes:
http://www.postgresql.org.mx/
http://torresquevedo.hispalinux.es/LuCAS/Tutoriales/NOTAS-CURSO-BBDD/notas-curso-BD.pdf
http://www.postgresql.cl/

Librería: clip-mysql
Sitio oficial: http://www.mysql.com/
Enlaces interesantes:
http://www.mysql-hispano.org/
http://www.cybercursos.net/sql/sql.html

Librería: clip-oracle
Sitio oficial: http://www.oracle.com/index.html
Enlaces interesantes:
http://www.oracle.com/global/es/index.html

Librería: clip-odbc
Administrador del controlador ODBC:
http://support.microsoft.com/default.aspx?scid=kb;es;110093

Librería: clip-interbase
Sitio oficial: http://www.borland.com/us/products/interbase/index.html
Enlaces interesantes:
http://www.firebird.com.mx/modules/news/

Librería: clip-dbtcp
Sitio oficial: Servidor proxy para conexiones ODBC
http://www.fastflow.it/dbftp/

Una vez instalado el paquete deseado, puedes construir tu aplicación de una forma similar a esta:

```
$ clip -es test.prg -lclip-mysql
```

2.2 PASO A PASO

Antes de comenzar a hacer algo, debes crear una conexión al servidor. Para este propósito se debe usar la función ConnectNew(). Esta función es un constructor de la clase Tconnect, o sea, si tiene exito retorna un objeto Tconnect. Una vez obtenido el objeto, puede ser usado para ejecutar instrucciones SQL, para seleccionar un set de filas deseado o para comenzar y finalizar transacciones. Por ejemplo:

```
conn := ConnectNew(...)    // Obtengo una conexión
conn:Start()               // Comienzo una transacción

// Voy a actualizar poniendo 'Total' donde el nombre es 'Rust'.
conn:Command( "UPDATE emp SET name='Total' WHERE name='Rust' " )

// La próxima vez, en la oficina de pago Rust dirá: "Mi nombre es Total" :-)

conn:Rollback()              // Era sólo una broma, cancelo el cambio :)
```

NOTA: Varias conexiones pueden ser hechas simultáneamente. Es más, también es posible conectarse a varios servidores a la vez.
Las consultas e instrucciones SQL pueden tener parámetros. Los nombres de los parámetros deben ser precedidos con un ":" (dos puntos). Los valores de parámetros son pasados a un arreglo bidimensional, una fila por parámetro. La primera columna contiene el nombre y la segunda el valor. Por ejemplo.

```
conn:Command( "UPDATE emp SET fname=:fname,lname=:lname" , ;
                              {{"fname","John"},{"lname","Smith"}} )
```

La función CreateRowset(), perteneciente a Tconnect, se usa para obtener un set de filas - resultado de la instrucción SELECT. Esta retorna un objeto de la clase Trowset. Por ejemplo:

```
rs := conn:CreateRowset( "SELECT * FROM emp WHERE
fname=:fname",{{"fname","John"}} )
rs:Browse()      //Un simple BROWSE para Trowset.
```

Las funciones miembros de Trowset, te permiten navegar a través de un set de filas de resultados. Ellas son: Bof(), Eof(), Skip(), Goto(), GoTop(), GoBottom(), Lastrec() y Recno().

Se han implementado dos funciones para leer/escribir en la fila en uso: Read () y Write(). La función Read() retorna un objeto cuya estructura es la misma que la estructura de la fila. Por ejemplo:

```
rs := conn:CreateRowset( "SELECT fname,lname FROM emp" )
? rs:Recno(), rs:Read()         //Imprime: 1   {FNAME: John, LNAME: Smith}
```

La función Write() recibe un objeto y setea los valores de los campos cuyos nombres concuerdan con los nombres de los atributos de ese objeto. Por ejemplo:

```
? rs:Read()                     // {FNAME: John, LNAME: Smith}
obj := map()                    // Creo una objeto vacío
obj:fname := "Robert"           //Añado atributos y valores
obj:salary := 10000
rs:Write(obj)                   //Adopto valores de nombres del objeto
                                   concordantes
? rs:Read()                     // {FNAME: Robert, LNAME: Smith}
```

Tu puedes añadir y borrar filas de un set usando Append() y Delete(). Append() puede recibir parámetros de un obj. Por ejemplo:

```
rs := conn:CreateRowset( "SELECT fname,lname FROM emp")
? rs:Lastrec()        // 100
obj := map()          // Creo un objeto vacío
obj:fname := "Homer"
obj:lname := "Simpson"
rs:Append( obj )      // Agrego un objeto
? rs:Lastrec()        // 101
? rs:Read()           // {FNAME: Homer, LNAME: Simpson}
rs:Delete()           //Borro la fila activa (generada del objeto)
? rs:Lastrec()        // 100
```

NOTA: Todos los cambios ejecutados por Write(), Append() y Delete() son sólo aplicados a ese set. Sin embargo, tres parámetros adicionales (<cInsertSQL>, <cDeleteSQL>, <cUpdateSQL>) pueden ser pasados a CreateRowset(). Si es pasado <cInsertSQL>, será ejecutado implícitamente cuando se ejecute el método Append(). De forma similar, <cDeleteSQL> y <cUpdateSQL> serán usados cuando sean invocados Delete() y Write(). Una única identificación (ID) de la fila debería ser SELECTcionada en el caso de Write() y Delete(). Por ejemplo:

```
rs := conn:CreateRowset( "SELECT rowid,fname,lname FROM emp", , ;
         "INSERT INTO emp values (:fname,:lname) ", ;
         "DELETE FROM emp WHERE rowid=:rowid", ;
```

```
        "UPDATE emp SET fname=:fname,lname=:lname WHERE rowid=:rowid" )
```
Si la cantidad de filas coincidentes no puede ser estimada, dos parámetros <bEval> y <nEvery> serán de gran ayuda. El código de bloque <bEval>, será ejecutado durante el proceso de traida después de cierto número de filas traidas, definido por <nEvery>. Si este retorna .F. (falso) el proceso se cancela. Así tú podrías hacer una barra de progreso para enormes set de resultados y detener la traida en cualquier momento. El siguiente ejemplo imprime un "." (punto) por cada 100 filas y, puede ser cancelado presionando ESC.

```
rs := conn:CreateRowset( "SELECT * FROM tabla_enorme", , , , , , , , , , ;
                         { | | qqout( "." ), inkey() != K_ESC},100 )
```

Por la misma razón (cuando la cantidad de filas coincidentes no puede ser estimada) tú puedes dirigir a Trowset para que no traiga todas las filas inmediatamente, pero que las traiga de acuerdo las vaya necesitando. Existe otro parámetro para este propósito, <lNoFetch>. Si se pasa como .T., CreateRowset() termina de una vez. Pero el número de filas coincidentes no puede ser obtenido mientras haya filas remanentes por traer. Para traer al resto, podría ser usada la función Trowset:FetchAll(). Para obtener el número de filas traidas en el momento, se debería usar Trowset:Fetch(). Por ejemplo:

```
rs := conn:CreateRowset( "SELECT * FROM
tabla_enorme", , , , , , , , ,.T. )
rs:Gotop()
? rs:Fetched()             // 1
? rs:Lastrec()             // 0
for i:= 1 to 100
      rs:Skip()
      ? rs:Fetched()     // 2, 3, ..., 101
next
rs:FetchAll()
? rs:Lastrec() == rs:Fetched()     // .T.
```

Trowset soporta algo llamado "órdenes locales". Una "Orden Local" es un índice creado en el lado del cliente y que permite cambiar el orden de las filas (registros) en un set. Esto es mayoritariamente lo mismo que los índices estándar RDD, pero su duración está limitada por el tiempo de vida del set de filas., p.e. éste se localiza en la memoria y no ocupa archivos. Trowset:CreateOrder() crea un orden con un nombre determinado, Trowset:SetOrder() activa un orden . Por ejemplo:

```
rs := conn:CreateRowset("SELECT fname,lname FROM emp")

// crea un orden 'Firstname' sobre el campo 'fname'
// El largo de la clave es 20 caracteres.
rs:CreateOrder("Firstname","fname",20)

// crea un orden 'Lastname' sobre el campo 'lname'
// El largo de la clave es 20 caracteres.

rs:CreateOrder("Lastname","lname",20)

// crea un orden 'Fullname' sobre ambos campos 'fname' y 'lname'.
// El largo de la clave es 40 caracteres.

rs:CreateOrder("Fullname",{|rs|  rs:GetValue("fname")+rs:GetValue("lname")},
20)

rs:SetOrder("Firstname")
rs:Browse() // Muestra las filas ordenadas por el nombre
```

```
rs:SetOrder("Lastname")
rs:Browse() // Muestra las filas ordenadas por apellido

rs:SetOrder("Fullname")
rs:Browse() // Muestra las filas ordenadas por nombre y apellido
```

A continuación procederemos a la descripción de las clases y funciones relacionadas con SQL.

3.- REFERENCIA AL API DE SQL

3.1. SQLList()

```
SQLList() --> aControladoresDisponibles
```

Valor de retorno

Un arreglo de controladores como una serie de subarreglos, uno por cada controlador disponible. El primer elemento del subarreglo contiene la ID corta del controlador, el segundo el nombre del RDBMS accesado por el controlador y el tercero, la descripción del controlador.

Descripción
SQLList() se usa para obtener la lista de los controladores disponibles. El controlador está disponible cuando su librería está enlazada con una aplicación. Si no hay controladores SQL enlazados un arreglo vacío es retornado.

El primer elemento del subarreglo que representa a un controlador, contiene una ID del RDBMS (valor CHARACTER corto asociado con el controlador), el cual es usado como el parámetro <cRDBMS> del constructor de Tconnect llamado ConnectNew().

Ejemplo

```
$cat test.prg

// test.prg
procedure Main()
? SQLList()[1]
? SQLList()[2]
return NIL

$clip -e test.prg -lclip-mysql -lclip-postgres
$./test
{MS, MySQL, Generic MySQL for CLIP driver, v.1.0},
{PG, PostgreSQL, Generic PostgreSQL for CLIP driver v.1.0}
```

3.2. ConnectNew()

```
ConnectNew(<cRDBMS>,[<RDBMS specific>,...],[<cCharset>],[<cIsolation>])
        --> TConnect object
```

Parámetros
<cRDBMS>
Identificador RDBMS
<RDBMS especific>
Un número de parámetros específicos del RDBMS

<cCharset>
(noveno parámetro) "backend" del set de caracteres

<cIsolation>

(Décimo parámetro) Por defecto el nivel de aislamiento de la transacción

Valor de retorno
Objeto TConnect

Descripción
ConnectNew() conecta a un servidor SQL, construye y retorna un objeto TConnect. Este objeto puede ser usado para comenzar o detener transacciones, ejecutar instrucciones SQL y, para obtener un set de filas mediante el uso de la instrucción SELECT.

El parámetro opcional <cCharset> se usa para indicar al servidor que use un diferente "charset" desde el cliente. Todas las transformaciones de cadenas son luego hechas automáticamente. Si no es pasado, entonces se usa SET ("SQL_CHARSET"). Observe que no tiene efecto cambiar el SET("SQL_CHARSET"), después de haberse conectado al servidor.

Si el parámetro opcional <cIsolation> no es pasado, entonces una variable apropiada para ese SET es usada. Por ejemplo si usamos SET ("OR_ISOLATION_LEVEL"), para Oracle. Si el SET para esa variable no existe, luego, el SET("SQL_ISOLATION_LEVEL") es usado.

NOTA: El nivel de aislamiento por defecto puede ser sustituido por los parámetros de Tconnect:Start(). Cambiando el valor de una variable con el apropiado SET después de conectarse, no sirve.
Cuando una aplicación completa el acceso a un servidor SQL, debería desconectarse del servidor y liberar los recursos del sistema llamando a la función Tconnect:Destroy().
Ejemplo

En este ejemplo se ejecuta una conexión al servidor local PostgreSQL.

```
conn := ConnectNew("PG",,,,,"template1")
...
conn:Destroy()
```