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 esto:

$ *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()
```

3.3. Clase TConnect

El constructor de la clase Tconnect y las funciones miembros, son listadas a continuación:

ConnectNew()	Constructor TConnect
TConnect:Command()	Ejecuta una instrucción SQL
TConnect:Commit()	"Commit" (Hace efectiva) una transacción
TConnect:Destroy()	Desconecta desde el servidor y destruye el objeto TConnect
TConnect:CreateRowset()	Constructor TRowset
TConnect:Rollback()	"Rollback" (deshace) una transacción
TConnect:Start()	Comienza una transacción

3.3.1. TConnect:Command()

Command(<cSQL>,[<aParameters>]) --> nAffectedRows

Parámetros

<cSQL>
Una cadena conteniendo la instrucción SQL a ser ejecutada.

<aParameters>

Un arreglo conteniendo parámetros SQL como una serie de subarreglos, uno por cada parámetro. Cada subarreglo debe contener como mínimo dos elementos. El primero es el nombre del parámetro y el segundo es el valor. Un tercer elemento es usado sólo con Oracle (y es obligatorio), contiene un identificador de tipo de dato numérico. Un opcional cuarto elemento es usado para indicar un valor binario; si se pasa un .T. , no

se realizan transformaciones del set de caracteres.

Valor de retorno

El número de filas afectadas.

Descripción

Command() es usado para ejecutar instrucciones “non-SELECT”. Esto significa, cualquier instrucción SQL que no retorna filas. Para ejecutar una instrucción que retorne filas, use Tconnect:CreateRowset() aún si tú no estás interesado en los resultados.

La instrucción SQL puede tener parámetros. Los nombres en **<cSQL>** deben ser precedidos con “:”. Los parámetros de tipo valor, son pasados en un arreglo **<aParameters>** de dos dimensiones.

Si no hay una transacción activa, los cambios hechos por la ejecución de **<cSQL>** son efectuados (“committed”) implícitamente. Si necesitas que se comporte de otra forma, tienes que comenzar la transacción explícitamente invocando Tconnect:Start().

El valor de retorno es el número de filas afectadas, por ejemplo el número de filas borradas, en el caso de la instrucción DELETE.

Ejemplo

```
conn:Command("CREATE TABLE mytable (fname char(20),lname char(20))")
conn:Command("INSERT INTO mytable VALUES (:firstname,:lastname)",;
      {{"firstname","John"},{"lastname","Smith"}})
? conn:Command("DELETE FROM mytable WHERE fname=:fname",;
      {{"fname","John"}}) // 1
```

3.3.2. TConnect:Commit()

Commit() --> NIL

Description

Commit() hace efectivos todos los cambios hechos después de invocar Tconnect:Start() y finaliza la transacción. Si no hay una transacción activa, un error es generado.

Ejemplo

```
conn:Start()
// ...
// algunos cambios
// ...
conn:Commit()
```

3.3.3. TConnect:Destroy()
Destroy() --> NIL

Descripción

Cuando una aplicación completa el acceso a un servidor SQL, se debería desconectar y liberar los recursos del sistema utilizando **Destroy()**.

Ejemplo

```
conn := ConnectNew("PG",,,,,"template1")
// ...
// Se trabaja un poco
// ...
conn:Destroy()
```

3.3.4. TConnect:CreateRowset()
```
CreateRowset(;
      <cSelectSQL>,;
      [<aParameters>],;
      [<cInsertSQL>],;
      [<cDeleteSQL>],;
      [<cUpdateSQL>],;
      [<cRefreshSQL>],;
      [<cIdName>],;
      [<aOrders>],;
      [<cGenIdSQL>],;
      [<lNoFetch>],;
      [<bEval>],;
      [<nEvery>];
) --> Objeto TRowset
```

Parámetros

<cSelectSQL>
Cadena conteniendo una consulta SQL a ser ejecutada

<aParameters>
Un arreglo conteniendo parámetros SQL como una serie de subarreglos, uno por cada parámetro. Cada subarreglo debe contener como mínimo dos elementos. El primero es el nombre del parámetro y el segundo es el valor. Un tercer elemento es usado sólo con Oracle (y es obligatorio), contiene un identificador de tipo de dato numérico. Un opcional cuarto elemento es usado para indicar un valor binario; si se pasa un .T. , no se realizan transformaciones del set de caracteres. Los parámetros y sus valores pasados en <aParameters> pueden ser usados en cualquiera de las instrucciones SQL descritas más adelante.

<cInsertSQL>

Instrucción SQL a ser ejecutada cuando una nueva fila es añadida al set.

<cDeleteSQL>
Instrucción SQL a ser ejecutada cuando una fila es borrada del set.

<cUpdateSQL>
Instrucción SQL a ser ejecutada cuando una fila es cambiada.

<cRefreshSQL>
Consulta SQL a ser ejecutada para refrescar la fila que está siendo usada en el set.

<cIdName>
Nombre del campo "ID de fila" (sólo para Interbase, ODBC y DBTCP)

<aOrders>
Un arreglo con las definiciones de los órdenes locales (índices) a ser creados como una serie de subarreglos, uno por cada orden. El primer elemento del subarreglo define el nombre del orden, el segundo es el nombre del campo o bloque de código a ser evaluado para obtener el valor de la clave. El tercer elemento es requerido sólo para claves de tipo carácter y, define la longitud de la clave.

<cGenIdSQL>
Instrucción SQL a ser ejecutada para obtener un nuevo y único valor ID de la fila, el cual será usado con <cInsertSQL> (sólo para Interbase).

<lNoFetch>
Valor lógico que define el modo de traida. Si es pasado un .F. , todas las filas son traidas inmediatamente. Si es pasado un .T. , las filas son traidas más tarde o a medida que se necesiten. El valor por defecto es .F.

<bEval>
Un bloque de código que será evaluado durante los procesos de traida. El objeto Trowset es pasado al bloque de código como un parámetro. Si <bEval> retorna .F. el proceso de traida es abortado. Esto es ignorado cuando se trabaja en modo "traida bajo demanda" (cuando <lNoFetch> es .T.)

<nEvery>
Cada cuantas filas (intervalo) debería ser evaluado <bEval>, si es que es pasado. Por defecto es 1, p.e. <bEval> es evaluado por cada una fila traida.

Valor de retorno

Objeto TRowset

Descripción

CreateRowset() ejecuta una consulta SELECT, construye un set con las filas que seleccionadas y retorna un objeto TRowset. De esta forma, puede ser considerado como un constructor de la clase TRowset. "Una consulta SELECT" significa cualquier instrucción SQL válida que retorne filas. El uso de cualquier otro tipo de instrucción SQL en <cSelectSQL> resultará en un error.

<cInsertSQL>, <cDeleteSQL> y <cUpdateSQL> son parámetros opcionales que hacen más fácil la

modificación de la información en las bases de datos. Si no son pasados, los correspondientes cambios en el set, no se reflejarán en las tablas de la base de datos. <cInsertSQL> es ejecutado automáticamente por el método Trowset:Append(). De igual forma <cDeleteSQL> es ejecutado por el método Trowset:Delete() y, <cUpdateSQL> es ejecutado por el método Trowset:Write().

<cRefreshSQL> es automáticamente ejecutado por el método Trowset:RefreshCurrent(). Este debe tener la misma lista de campos de <cSelectSQL>.

Cuando se intenta hacer modificaciones en un set de filas, este debe contener un campo único (ID de fila) el cual es usado para relacionar las filas en el set con las filas en las tablas de la base de datos. Deberías explícitamente incluir ese campo a la lista de los campos requeridos en la consulta SELECT. Existen varios planteamientos para esta situación según sea el RDBMS.

- Algunos RDBMS proporcionan un campo oculto para este propósito (ROWID en Oracle y OID en PostgreSQL). Tal campo por defecto, es creado para cualquier tabla, aún si no lo describes en la instrucción CREATE TABLE. Para PostgreSQL conocemos la siguiente sintaxis:

  SELECT oid,* FROM mitabla

  Al contrario, Oracle no permite tal sintaxis. Deberías enumerar todos los campos por su nombre:

  SELECT rowid,fname,lname FROM mitabla

- Ciertos RDBMS proporcionan campos que se auto incrementan para este propósito. La sintaxis CREATE TABLE de Mysql permite crear una tabla con tal campo, usando la claúsula AUTO_INCREMENT en la definición del campo. CLIP reconoce tal campo y lo usa como ID de fila. Tú lo único que tienes que hacer es indicar un campo como ID de fila, excepto para la definición del campo con la claúsula AUTO_INCREMENT en la instrucción CREATE TABLE.

- Otros RDBMS (Interbase), proporcionan gatilladores (“triggers”) y generadores de números únicos. Una propuesta típica en tal caso es definir un “triggers” BEFORE INSERT, el cual obtiene una ID única desde el generador y lo asigna al campo “ID fila”. Pero no hay forma de determinar el valor del “ID fila” de las filas recien insertadas. Por ejemplo, tú no puedes UPDATE ó DELETE filas recién INSERTadas (una fila es INSERTada después que un objeto Trowset fue creado). Aquí tenemos dos elecciones:

  - Diseñar tu aplicación considerando esta limitación (nunca un UPDATE a una fila recién INSERTada).

  - No usar “trigger” BEFORE INSERT, pero proporcionar el parámetro <cGenIdSQL> al generador, conteniendo la consulta SQL. Esta instrucción debe retornar un únicoID desde el generador, que luego será asignado al campo con nombre <cIdName> durante la ejecución de la instrucción <cInsertSQL>.

    NOTA: El atributo de Interbase RDB$DB_KEY creado para ser un “ID fila”, no lo es realmente. Este más bien es una “dirección de fila” que puede ser cambiada (similar a RECNO() que puede cambiar después de un PACK). Por lo tanto, no es usado como “ID fila”.

- En el caso que el RDBMS no proporcione ninguna de las características descritas anteriormente (ODBC,

DBTCP), deberías pasar el parámetro <cIdName> con el nombre del campo “ID fila” y deberías diseñar tu aplicación para que una fila recién INSERTada, nunca sea cambiada o borrada.

Las instrucciones SQL <cInsertSQL>, <cDeleteSQL>, <cUpdateSQL> y <cRefreshSQL> obtienen valores de los campos de la fila (registro) en uso, a través de los parámetros con el mismo nombre del campo seteado. Por ejemplo:

<cSelectSQL> - SELECT DriverID AS id,fname,lname FROM mitabla
<cInsertSQL> - INSERT INTO mitabla (fname,lname) VALUES (:fname,:lname)
<cUpdateSQL> - UPDATE mitabla SET fname=:fname,lname=:lname WHERE DriverId=:id
<cDeleteSQL> - DELETE FROM mitabla WHERE DriverId=:id
<cRefreshSQL> - SELECT DriverID,fname,lname FROM mitabla WHERE DriverId=:id

Existen tres macros que se pueden utilizar: %FIELDS y %VALUES para usar con <cInsertSQL>; %LIST para usar con <cUpdateSQL>.

%FIELDS es expandido a una lista con los nombres de los campos, %VALUES expandida a los nombres de los parámetros y %LIST como una lista de campos y sus correspondientes parámetros. Por ejemplo:

INSERT INTO mitabla (%FIELDS) VALUES (%VALUES)
UPDATE mitabla SET %LIST

Hay dos formas de traer filas controladas por el parámetro <lNoFetch>. Traida total y traida por demanda (a pedido).

- De la primera forma (<lNoFetch>==.F.), todas las filas seleccionadas son traidas antes que CreateRowset () termine. En este modo, puedes usar inmediatamente Trowset:Lastrec() para determinar el número de filas seleccionadas. También puedes usar el bloque de código <bEval> para observar y administrar el proceso de traida.

- De la segunda forma (<lNoFetch>==.T.), las filas son traidas posteriormente de acuerdo a la demanda (durante la navegación a través del set). Este modo es más rápido, pero Lastrec() retorna el número real de filas seleccionadas sólo después de que todas las filas han sido traidas. En este modo, el parámetro <bEval> es ignorado.

Cuando una aplicación termina de usar el set de filas, debería liberarlo haciendo uso de la llamada a Trowset:Destroy().

Ejemplos

Simple ejemplo que crea un set de filas con la intencion de no modificarlo:

```
rs := conn:CreateRowset("SELECT * FROM mytable WHERE fname = 'John'")
rs:Browse()
```

Crea un set de filas usando parametros SQL en una consulta SELECT:

```
rs := conn:CreateRowset("SELECT * FROM mytable WHERE fname = :par1",;
        {{"par1","John"}})
        rs:Browse()
```

Crea un set de filas con la intencion de modificarlas:

```
rs := conn:CreateRowset("SELECT id,fname,lname FROM mytable",NIL,;
        "INSERT INTO mytable (%FIELDS) VALUES (%VALUES)",;
        "DELETE FROM mytable WHERE id=:id",;
        "UPDATE mytable SET fname=:fname,lname=:lname",;
        "SELECT id,fname,lname FROM mytable WHERE id=:id",;
        "id")
```

Crea un set de filas con ordenes locales (indices) “id” y “fullname”:

```
rs := conn:CreateRowset("SELECT id,fname,lname FROM mytable",,,,,,,;
        {{"id","id"},;
        {"fullname",{|rs| rs:GetValue("fname")+rs:GetValue("lname")},40}})
rs:Browse()             // Muestra las filas en su orden natural
rs:SetOrder("id")
rs:Browse()             // Muestra las filas ordenadas por “id”
rs:SetOrder("fullname")
rs:Browse()             // Muestra las filas ordenadas por “fname” y “lname”
```

Usa <bEval> para observar el proceso de traida (imprime un punto cada 100 filas), el cual puede ser detenido presionando ESC:

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

Trayendo filas de acuerdo a la demanda (de a poco):

```
rs := conn:CreateRowset("SELECT * FROM tablaEnorme",,,,,,,,,.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.
```

3.3.5. TConnect:Rollback()

Rollback() --> NIL

Descripción

Rollback() descarta todos los cambios hechos después de invocar TConnect:Start() y finaliza la transacción. Si no existe una transacción activa, un error es generado.

Ejemplos

```
conn:Start()
// ...
// algunos cambios
// ...
conn:Rollback()
```

3.3.6. TConnect:Start()

Start([<cIsolation>],[<cLockTables>]) --> NIL

Parámetros

<cIsolation>
Una cadena que define el nivel de aislamiento a ser usado durante una transacción. Este parámetro predomina sobre el definido <cIsolation> para la función ConnectNew(). Vea la sección específica de su RDBMS para saber que valores pueden ser pasados con este parámetro.

<cLockTables>
Este parámetro es usado sólo con Interbase y, define que tablas deberían ser bloqueadas y cómo.

Descripción

Start() comienza una nueva transacción. No debe haber una transacción activa, de lo contrario un error es generado. La transacción debería estar en algún momento detenida (ya sea usando la función Tconnect:Commit() ó Tconnect:Rollback()).
Si no hay una transacción activa, todos los cambios a la base de datos son efectuados inmediatamente (“commited”) después de la ejecución de cada instrucción SQL.

Ejemplo

```
conn:Start()
// ...
// algunos cambios
// ...
conn:Commit() // efectúa realmente los cambios
```

3.4. Clase TRowset

Constructor de la clase Trowset, las funciones que la constituyen son listadas a continuación:

TConnect:CreateRowse()	Constructor TRowset
TRowset:Append()	Agrega una nueva fila al set
TRowset:Bof()	Determina si se ha encontrado el inicio del set
TRowset:Browse()	Navega filas dentro de una ventana
TRowset:CreateOrder()	Crea un nuevo orden local (índice)
TRowset:Delete()	Borra una fila desde el set
TRowset:Destroy()	Destruye un objeto TRowset
TRowset:Eof()	Determina si se ha encontrado el fin del set
TRowset:FetchAll()	Trae todas las filas que no se hayan traido
TRowset:Fetched()	Determina el número de filas traidas
TRowset:FieldBinary()	Determina si un campo dado es binario
TRowset:FieldBlock()	Retorna un bloque de código set/get para un campo dado
TRowset:FieldDec()	Determina el número de dígitos decimales en un campo dado
TRowset:FieldLen()	Determina la longitud de un campo dado
TRowset:FieldName()	Retorna el nombre de un campo, dada una posición
TRowset:FieldNo()	Retorna la posición de un campo, dado el nombre
TRowset:FieldNullable()	Determina si un campo es del tipo nulo
TRowset:FieldType()	Retorna el tipo Xbase del campo
TRowset:FieldTypeSQL()	Retorna el tipo RDBMS del campo
TRowset:FieldUnsigned()	Determina si un campo numérico dado no tiene signo
TRowset:GetValue()	Recupera el valor de un campo de la fila (registro) activa
TRowset:GoBottom()	Se mueve al último registro lógico (fila)
TRowset:Goto()	Se mueve a un registro esfecífico
TRowset:GoTop()	Se mueve al primer registro lógico
TRowset:KeyNo()	Determina la posición lógica del registro activo
TRowset:Lastrec()	Determina el número de filas (registros) en el set
TRowset:NFields()	Retorna el número de campos en el set
TRowset:Read()	Lee el registro activo
TRowset:Recno()	Retorna la posición del registro activo
TRowset:RefreshAll()	Refresca el set ejecutando repetidamente la consulta SELECT
TRowset:RefreshCurrent()	Refresca el registro activo ejecutando <cRefreshSQL>
TRowset:Seek()	Se mueve a la fila que tiene el valor clave especificado
TRowset:SetOrder()	Determina el order (índice) que va a controlar el set
TRowset:SetValue()	Setea el valor de un campo en el registro activo
TRowset:Skip()	Se mueve relativamente desde el registro activo
TRowset:Write()	Graba el registro activo

3.4.1. TRowset:Append()

Append(<oRow>) --> NIL

Parámetros

<oRow>
un objeto conteniendo un registro completo (con valores de los campos)

Descripción

Append() agrega un nuevo registro al set de filas y asigna los valores de los atributos de <oRow> a los campos respectivos de la fila recién insertada

Si el set de filas fue creado con el parámetro <cInsertSQL>, éste es ejecutado en el servidor SQL con los valores recién insertados.

Si el set de filas fue creado en modo "traida por demanda", parámetro <lNoFetch> es .T. en la función Tconnect:CreateRowset(), todas las filas no traidas son recuperadas antes del procesamiento.

Append() incrementa la cantidad de filas retornado por Trowset:Lastrec(). La posición activa pasa a ser la nueva fila.

Ejemplo

```
rs := conn:CreateRowset("SELECT id,fname,lname FROM mytable",,;
      "INSERT INTO mytable (fname,lname) VALUES (:fname,:lname)")
obj := map()         // crea un objeto vacío
obj:fname := "John"   // Setea los artibutos con los mismos nombres de los campos
obj:lname := "Smith"  // ...
// Añade la nueva fila y ejecuta
// INSERTA DENTRO de mytable (fname,lname) VALUES ('John','Smith')
rs:Append(obj)
```

3.4.2. TRowset:Bof()

Bof() --> lLímite

Valor de retorno

Verdadero (.T.) después de un intento para saltar más allá del límite lógico del primer registro en el set, de otra forma retorna falso (.F.). Si el set no contiene filas, retorna verdadero (.T.).

Descripción

Bof() es usado para probar una condición de límites cuando estás moviendo el puntero de la fila a través de un set usando la función Trowset:Skip()

Una vez que **Bof()** es verdadero (.T.), retiene este valor hasta que haya otro intento de mover el puntero de la fila.

TRowset:Skip() es la única función de movimiento de fila que puede setear **Bof()** a verdadero (.T.).

Ejemplo

```
rs := conn:CreateRowset("SELECT * FROM mytable")
? rs:Recno() // 1
? rs:Bof()   // .F.
rs:Skip(-1)
? rs:Recno() // 1
? rs:Bof()   // .T.
```

3.4.3. TRowset:Browse()

Browse([<nTop>],[<nLeft>],[<nBottom>],[<nRight>],;
[<asColumns>],[<asHeaders>],[<anWidths>]) --> NIL

Parámetros

<nTop>,<nLeft>,<nBottom>,<nRight>
Define las coordenadas de ventana. Si no se especifican, las coordenadas por defecto de la ventana son 1, 0 y MAXROW(), MAXCOL().

<asColumns>
Un arreglo de cadena que contiene los nombres de campo para usar en las columnas.

<asHeaders>
Un arreglo paralelo de cadena que contiene los encabezados por cada columna.

<anWidths>
Un arreglo paralelo conteniendo los anchos de cada columna.

Descripción

Browse() es una función de interfaz de usuario que proporciona un simple navegador para las filas del set.

Ejemplo

```
rs := conn:CreateRowset("SELECT id,fname,lname FROM mytable",,,,,,,,,.T.)
rs:Browse(,,,,{"fname","lname"},{"First name","Last name"},{20,20})
```

3.4.4. TRowset:CreateOrder()

CreateOrder(<cOrderName>,<cFieldName>|<bExpression>,[<nKeyLength>]) --> NIL

Parámetros

<cOrderName>
Define el nombre de un índice a ser creado.

<cFieldName>
El nombre de un campo cuyo valor será usado para la clave.

<bExpression>
Un bloque de código usado para evaluar claves. Este recibe al objeto Trowset como parámetro.

<nKeyLength>
La longitud de la clave. Requerida para claves tipo carácter.

Descripción

CreateOrder() es usada para crear un orden local. Un "orden local" significa un índice en memoria que controla el orden de las filas en el set. Una vez creado, puede ser controlado por la función Trowset:SetOrder (). Además, el control del orden puede ser usado para localizar filas teniendo su clave, usando la función Trowset:Seek().

Si el set de filas es creado en modo "traida por demanda", parámetro <lNoFetch> es .T. en la función Tconnect:CreateRowset(), todas las filas son traidas implícitamente antes del procesamiento.

Ejemplo

Este ejemplo crea dos órdenes ('birthdate' and 'fullname'). Muestra las filas ordenadas por cumpleaños y nombre completo, y luego busca una persona cuyo nombre comienza con "Joh":

```
rs := conn:CreateRowset("SELECT bdate,fname,lname FROM employee")
rs:CreateOrder("birthdate","bdate")
rs:CreateOrder("fullname",{|rs| rs:GetValue("fname")+rs:GetValue("lname")},40)
rs:SetOrder("birthdate")
rs:Browse()
rs:SetOrder("fullname")
rs:Browse()
? rs:Seek("Joh") // .T.
row := rs:Read()
? row:fname       // John
```

3.4.5. TRowset:Delete()
Delete() --> NIL

Descripción

Delete() borra la fila activa desde el set.

Si el set de filas fue creado con el parámetro <cDeleteSQL>, éste es ejecutado en el servidor SQL con el valor del campo "row id" (identificador de fila) del registro activo.

Si el set de filas fue creado en modo "traida por demanda", parámetro <lNoFetch> es .T. en la función Tconnect:CreateRowset(), todas las filas no traidas son recuperadas antes del procesamiento.

Delete() decrementa la cantidad de filas retornada por TRowset:Lastrec().

Si no hay un índice controlando, la posición de la fila activa permanece igual, o se mueve a la última fila (cuando se borra una fila donde Recno() == Lastrec()).

Si hay un índice controlando (usando Trowset:SetOrder()), la posición de la fila activa se mueve a la que tenga la próxima clave, o a la última fila lógica (cuando la fila que tiene la clave mayor es borrada).

Si la última y única fila es borrada, ambos estados Bof() y Eof() son llevados a verdadero (.T.) y Trowset:Recno() retorna cero (0).

Ejemplo

```
rs := conn:CreateRowset("SELECT id,fname,lname FROM mytable",,,;
      "DELETE FROM mytable WHERE id=:id")
? rs:Lastrec() // 10
// borra una fila y ejecuta
// BORRA DESDE mytable DONDE id=...
rs:Delete()
? rs:Lastrec() // 9
```

3.4.6. TRowset:Destroy()

Destroy() --> NIL

Descripción

Cuando una aplicación termina de usar el set de filas, debería liberarlo llamando al Trowset:Destroy().

Ejemplo

```
rs := conn:CreateRowset("SELECT * FROM mytable")
// ...
// Hacer algún trabajo
// ...
rs:Destroy()
```

3.4.7. TRowset:Eof()

Eof() --> lLímite

Valor de retorno

Verdadero (.T.) después de un intento por saltar más allá del último registro en el set, de otra forma retorna falso (.F.). Si el set no contiene filas, retorna verdadero (.T.).

Descripción

Eof() es usado para probar una condición de límite cuando estás moviendo el puntero de fila hacia adelante a través de un set usando la función Trowset:Skip().
Una vez que **Eof()** es verdadero (.T.), retiene su valor hasta que haya otro intento por mover el puntero de la fila.

TRowset:Skip() es la única función de movimiento de fila que puede poner Eof() a verdadero (.T.).

NOTA: A diferencia del comando estándar Xbase SKIP, Trowset:Skip() nunca se mueve más allá del último registro lógico. Cuando Trowset:Eof() retorna .T. ciertamente la posición es el último registro lógico, no Lastrec() + 1.

Ejemplo

```
rs := conn:CreateRowset("SELECT * FROM mytable")
rs:GoBottom()
? rs:Recno() // 100
? rs:Eof()   // .F.
rs:Skip()
? rs:Recno() // 100
? rs:Eof()   // .T.
rs:Skip(-1)
? rs:Recno() // 99
? rs:Eof()   // .F.
```

3.4.8. TRowset:FetchAll()

FetchAll() --> nLastrec

Valor de retorno

El número de registros en el set.

Descripción

FetchAll() trae todo el resto de filas del set. Es útil cuando el set de filas ha sido creado en modo "traida por demanda" (el parámetro <lNoFetch> de Tconnect:CreateRowset() es .T.). Si no hay filas por traer, FetchAll () no tiene efecto.

Ejemplo

```
rs := conn:CreateRowset("SELECT * FROM mytable",,,,,,,,,.T.)
? rs:Lastrec()  // 0
? rs:Fetched()  // 1
? rs:FetchAll() // Número de filas seleccionadas.
? rs:Lastrec()  //        - " -
? rs:Fetched()  //        - " -
```

3.4.9. TRowset:Fetched()
Fetched() --> nNúmero de filas traidas

Valor de retorno

Número de filas ya traidas.

Descripción

Fetched() es usado para determinar cuantas filas ya han sido traidas. Es útil en <bEval>, bloque de código parámetro de Tconnect:CreateRowset(), cuando el set de filas ha sido creado en modo "traida por demanda".

Ejemplos

- Imprime el número de filas ya traidas, durante el proceso de traer:
  ```
  rs := conn:CreateRowset("SELECT * FROM hugetable",,,,,,,,,,;
        {|rs| qout(rs:Fetched())},100)
  ```

- Usa **Fetched()** con set de filas creado en modo "traida por demanda":
  ```
  rs := conn:CreateRowset("SELECT * FROM hugetable",,,,,,,,,.T.)
  ? rs:Fetched() // 1
  rs:Skip()
  ? rs:Fetched() // 2
  rs:Skip(100)
  ? rs:Fetched() // 102
  ```

3.4.10. TRowset:FieldBinary()
FieldBinary(<nFieldNo> | <cFieldName>) --> lBinario

Parámetros

<nFieldNo>
La posición del campo en la lista de campos.

<cFieldName>
El nombre del campo.

Valor de retorno

Verdadero (.T.) si el campo dado es binario, de otra forma es falso (.F.).

Descripción

FieldBinary() es usado para determinar si un campo carácter es binario. Las transformaciones "charset" no toman lugar con los valores de un campo binario.

Ejemplo

? rs:FieldBinary("fname") // .F.

3.4.11. TRowset:FieldBlock()

FieldBlock(<nFieldNo> | <cFieldName>) --> bBlock

Parámetros

<nFieldNo>
La posición del campo en la lista de campos

<cFieldName>
El nombre del campo

Valor de retorno

Un bloque de código que, cuando es evaluado, setea (asigna) u obtiene (recupera) el valor de un campo dado.

Descripción

FieldBlock() construye un bloque de código que setea u obtiene un valor para un campo dado. Cuando es ejecutado con un argumento, el bloque de código creado por esta función, asigna el valor del argumento al campo dado. Cuando es ejecutado sin argumento, el bloque de código recupera el valor de un campo dado.

NOTA: Cambiar el set usando un bloque de código, no causa ejecución inmediata de <cUpdateSQL> (parámetro pasado a Tconnect:CreateRowset()). Sólo Write() y todas las funciones de movimiento de fila (Gotop(), Gobottom(), Goto() y Skip()) son las funciones que inician el reflejo de los cambios sobre el servidor SQL.

Ejemplo

```
cb := rs:FieldBlock("fname")
? eval(cb)   // 'John'
eval(cb,'Richard')
? eval(cb)   // 'Richard'
rs:Write(rs:Read()) // causa UPDATE (Actualización) en el servidor
```

3.4.12. TRowset:FieldDec()

FieldDec(<nFieldNo> | <cFieldName>) --> nDec

Parámetros

<nFieldNo>
La posición del campo en la lista de campos.

<cFieldName>
El nombre del campo

Valor de Retorno

Número de dígitos decimales usados en un campo numérico dado.

Descripción

FieldDec() es usado para determinar el número de dígitos decimales en un campo dado.

Ejemplo

```
? rs:FieldDec("salary")
```

3.4.13. TRowset:FieldLen()

FieldLen(<nFieldNo> | <cFieldName>) --> nDec

Parámetros

<nFieldNo>
La posición del campo en la lista de campos.

<cFieldName>
El nombre del campo.

Valor de Retorno

Largo del campo (en términos del RDBMS usado).

Descripción

FieldLen() es usado para determinar la longitud de un campo dado, en términos del RDBMS usado. Por ejemplo, la longitud de un campo de tipo “FLOAT” es 4.

Ejemplo

```
? rs:FieldLen("salary")
```

3.4.14. TRowset:FieldName()

FieldName(<nFieldNo>) --> cFieldName

Parámetros

<nFieldNo>
La posición del campo en la lista de campos.

Valor de retorno

El nombre del campo.

Descripción

FieldName() es usado para determinar el nombre de un campo, dada una posición.

Ejemplo

? rs:FieldName(1) // fname

3.4.15. TRowset:FieldNo()

FieldName(<cFieldName>) --> nFieldNo

Parámetros

<cFieldName>
El nombre del campo.

Valor de retorno

La posición del campo.

Descripción

FieldNo() es usado para determinar la posición de un campo, dado un nombre.

Ejemplo

? rs:FieldName('fname') // 1

3.4.16. TRowset:FieldNullable()

FieldNullable(<nFieldNo> | <cFieldName>) --> lNullable

Parámetros

<nFieldNo>
La posición del campo en la lista de campos.

<cFieldName>
El nombre del campo.

Valor de retorno

Verdadedo (.T.) si el campo dado puede ser nulo, de otra forma es falso(.F., NOT NULL).

Descripción

FieldNullable() es usado para determinar si un campo puede ser nulo o no.

Ejemplo

```
? rs:FieldNullable("fname") // .T.
```

3.4.17. TRowset:FieldType()

FieldType(<nFieldNo> | <cFieldName>) --> cType

Parámetros

<nFieldNo>
La posición del campo en la lista de campos.

<cFieldName>
El nombre del campo.

Valor de retorno

Tipo de campo (en términos de XBase).

Descripción

FieldType() es usado para determinar el tipo de un campo dado. Retorna un carácter único que designa el tipo de dato del campo. **FieldType()** retorna los siguientes caracteres según sean los tipos de campos de datos:

- *C* – carácter, cadena
- *N* - numérico
- *D* – fecha (date)
- *T* – hora (datetime)
- *L* - logico

Ejemplo

```
? rs:FieldType("salary") // 'N'
```

3.4.18. TRowset:FieldTypeSQL()

FieldTypeSQL(<nFieldNo> | <cFieldName>) --> nType

Parámetros

<nFieldNo>
La posición del campo en la lista de campos.

<cFieldName>
El nombre del campo.

Valor de retorno

Tipo numérico de un campo (en términos del RDBMS usado).

Descripción

FieldTypeSQL() es usado para determinar el tipo de un campo dado, en términos del RDBMS usado. Los tipos específicos de cada RDBMS son definidos en sus correspondientes archivos *.ch.

Ejemplo

```
? rs:FieldTypeSQL("salary")
```

3.4.19. TRowset:FieldUnsigned()

FieldUnsigned(<nFieldNo> | <cFieldName>) --> lUnsigned

Parámetros

<nFieldNo>
La posición del campo en la lista de campos.

<cFieldName>
El nombre del campo.

Valor de retorno

Verdadero (.T.) si el campo dado no tiene signo, de otra forma es falso (.F.).

Descripción

FieldUnsigned() es usado para determinar si un campo numérico no tiene signo.

Ejemplo

```
? rs:FieldUnsigned("salary") // .T.
```

3.4.20. TRowset:GetValue()

GetValue(<nFieldNo> | <cFieldName>) --> xValue

Parámetros

<nFieldNo>
La posición del campo en la lista de campos

<cFieldName>
El nombre del campo.

Valor de retorno

El valor del campo de la fila activa.

Descripción

GetValue() es usada para obtener el valor de la fila activa.

Ejemplo

```
? rs:GetValue('fname') // John
```

3.4.21. TRowset:GoBottom()

GoBottom() --> NIL

Descripción

GoBottom() mueve la posición de la fila a la última fila lógica. Si la fila fue creada en modo "traida por demanda" (parámetro <lNoFetch> de Tconnect:CreateRowset() es .T.), todas las filas por recuperar, son traidas antes del procesamiento.

Ejemplo

```
rs:CreateRowset("SELECT * FROM mytable",,,,,,,,,.T.)
? rs:Recno()   // 1
? rs:Lastrec() // 0 (hay filas remanentes no traidas)
rs:GoBottom()
? rs:Recno()   // número de filas seleccionadas.
? rs:Lastrec() // número de filas seleccionadas (sin traer las filas remanentes).
```

3.4.22. TRowset:Goto()

Goto(<nRowPosition>) --> nNewPosition

Parámetros

<nRowPosition>
La posición donde moverse.

Valor de retorno

Nueva posición de la fila activa.

Descripción

Goto() es usada para moverse a una posición física de fila específica. Si el orden de control no es físico (seteado por Trowset:SetOrder()), la posición física puede ser diferente de la posición lógica. Si <nRowPosition> es menor que 1, **Goto()** mueve a la primera fila física, y el estado Bof() es seteado a verdadero (.T.). Si <nRowPosition> es mayor que Lastrec(), **Goto()** mueve a la última fila física, y el estado Eof() es seteado a verdadero (.T.).

Ejemplo

```
for i:=1 to rs:Lastrec()
      rs:goto(i)
      ? rs:Read()
next
```

3.4.23. TRowset:GoTop()

```
GoTop() --> NIL
```

Descripción

GoTop() mueve la posición de la fila a la primera fila lógica.

Ejemplo

```
rs:CreateRowset("SELECT fname,lname FROM mytable")
? rs:Recno()   // 1
rs:CreateOrder("fname","fname",20)
rs:SetOrder("fname")
rs:GoTop()
? rs:Recno()   // la posición de la primera fila lógica.
```

3.4.24. TRowset:KeyNo()

```
KeyNo() --> nPosition
```

Valor de retorno

La posición lógica de la fila activa.

Descripción

KeyNo() es usado para determinar el número lógico de la fila activa. Si no hay un orden (índice) controlando (creado por TRowset:CreateOrder() y seteado por Trowset:SetOrder()), **KeyNo()** retorna la posición física de la fila, p.e. trabaja similar a Trowset:Recno().

Ejemplo

```
rs := conn:CreateRowset("SELECT fname,lname FROM mytable")
rs:CreateOrder("fname","fname",20)
? rs:KeyNo(), rs:Recno() // 1, 1 (sin índice activo)
```

```
rs:SetOrder("fname")      // índexado por "fname"
? rs:KeyNo(), rs:Recno() // N, 1
rs:GoTop()
? rs:KeyNo(), rs:Recno() // 1, M
```

3.4.25. TRowset:Lastrec()

```
Lastrec() --> nLastrec
```

Valor de retorno

El número de filas en el set.

Descripción

Lastrec() determina el número de filas en el set. Si el set fue creado en modo "traída por demanda" (parámetro <lNoFetch> de Tconnect:CreateRowset() es .T.), **Lastrec()** puede retornar 0 (si existen filas remanentes por traer). El número de filas traídas puede ser determinado usando la función Trowset:Fetched(). Trowset:FetchAll() puede ser usado para traer las filas remanentes que esperan a ser traídas. Después de esto, **Lastrec()** retornará el número real de filas.

Ejemplo

```
rs := conn:CreateRowset("SELECT * FROM mytable",,,,,,,,,.T.)
? rs:Lastrec() // 0
rs:FetchAll()
? rs:Lastrec() // Número de filas seleccionadas.
```

3.4.26. TRowset:NFields()

```
NFields() --> nFields
```

Valor de retorno

El número de campos en el set de filas

Descripción

NFields() determina el número de campos en el set de filas.

Ejemplo

```
rs := conn:CreateRowset("SELECT fname,lname FROM mytable")
? rs:NFields() // 2
```

3.4.27. TRowset:Read()

```
Read() --> oRow
```

Valor de retorno

Un objeto conteniendo todos los campos de la fila activa.

Descripción

Read() recupera los valores de la fila activa dentro de un objeto.

Ejemplo

```
rs := conn:CreateRowset("SELECT fname,lname FROM mytable")
? rs:Read():fname // John
? rs:Read():lname // Smith
```

3.4.28. TRowset:Recno()

```
Recno() --> nPosition
```

Valor de retorno

La posición física de la fila activa.

Descripción

Recno() usada para determinar la posición física de la fila activa. Si no hay filas en el set, retorna 0.

Ejemplo

```
rs := conn:CreateRowset("SELECT * FROM mytable")
? rs:Recno() // 1
rs:Skip()
? rs:Recno() // 2
rs:Goto(100)
? rs:Recno() // 100
```

3.4.29. TRowset:RefreshAll()

```
RefreshAll() --> nLastrec
```

Valor de retorno

El número de filas en el set.

Descripción

RefreshAll() refresca el set de filas ejecutando repetídamente <cSelectSQL> que es pasado como parámetro SQL en <aParámetros> del Trowset constructor Tconnect:CreateRowset(). La posición física de la fila se mantiene o se mueve a la última fila física.

Ejemplo

```
rs := conn:CreateRowset("SELECT * FROM mytable")
rs:Browse()
rs:RefreshAll()
rs:Browse()
```

3.4.30. TRowset:RefreshCurrent()

```
RefreshCurrent() --> NIL
```

Descripción

RefreshCurrent() refresca la fila activa del set ejecutando <cRefreshSQL> pasada al Trowset constructor Tconnect:CreateRowset(). Si <cRefreshSQL> no es pasado, **RefreshCurrent()** no tiene efecto.

Ejemplo

```
rs := conn:CreateRowset("SELECT id,fname,lname FROM mytable",,,,,;
        "SELECT id,fname,lname FROM mytable WHERE id=:id")
? rs:Read()
rs:RefreshCurrent()
? rs:Read()
```

3.4.31. TRowset:Seek()

```
Seek(<xKeyValue>,[<lSoft>]) --> lFound
```

Parámetros

<xKeyValue>
Un valor asociado a la clave de la fila deseada.

<lSoft>
Valor lógico que especifica si se va a ejecutar una búsqueda suave. Esto determina como se posiciona en el área de trabajo si el valor de la clave especificada no es encontrada (ver abajo).

Valor de retorno

Verdadero (.T.) si el valor de la clave es encontrado, de otra forma es falso (.F.)

Descripción

Seek() mueve a la primera fila lógica cuyo valor de la clave sea igual a <xKeyValue>. Si tal fila es encontrada, la convierte en la fila activa y **Seek()** retorna verdadero (.T.), de otra forma retorna falso (.F.). Para una búsqueda normal (no suave), el set de filas es posicionado en la última fila lógica y Trowset:Eof() retorna verdadero (.T.). Para una búsqueda suave, el set de filas es posicionado en la primera fila cuya clave es mayor que la clave especificada. Si no existe tal fila, El set de filas es posicionado en la última fila lógica y **TRowset:Eof()** retorna verdadero (.T.).

Para un set de filas sin control por orden (índices), **Seek()** no tiene efecto.

Ejemplo

```
rs := conn:CreateRowset("SELECT fname,lname FROM mytable")
rs:CreateOrder("fname","fname",20)
rs:SetOrder("fname")
? rs:Seek("John")
```

3.4.32. TRowset:SetOrder()

```
SetOrder([<cOrderName>]) --> cOrderName
```

Parámetros

<cOrderName>
El nombre del orden (índice) a ser seteado para tener el control.

Valor de retorno

El nombre del orden que previamente tenía el control o NIL (orden físico).

Descripción

SetOrder() activa un orden a ser creado por Trowset:CreateOrder() (lo hace con el control). Este retorna el nombre del orden de control previo. Si es pasada una cadena vacía, **SetOrder()** causa que el set de filas sea accesada en un orden físico.

Ejemplo

```
rs := conn:CreateRowset("SELECT fname,lname FROM mytable")
rs:CreateOrder("fname","fname",20)
rs:Browse() // Navega filas en orden físico.
rs:SetOrder("fname")
rs:Browse() // Navega filas ordenadas por "first name".
rs:SetOrder("")
rs:Browse() // Navega en orden físico nuevamente.
```

3.4.33. TRowset:SetValue()

```
SetValue(<nFieldNo> | <cFieldName>, <xValue>) --> NIL
```

Parámetros

<nFieldNo>
La posición del campo en la lista de campos.

<cFieldName>
El nombre del campo.

<xValue>
El valor a ser asignado al campo.

Descripción

SetValue() asigna <xValue> al campo en la fila activa.

NOTA: A diferencia de la función Trowset:Write(), SetValue() no causa un reflejo inmediato del cambio en el servidor SQL. Debe usarse una de las funciones de movimiento de filas para causar un reflejo (ejecución de la instrucción <cUpdateSQL> pasado al constructor de Trowset, Tvonnect:CreateRowset()).

Ejemplo

```
rs := conn:CreateRowset("SELECT fname,lname FROM mytable")
? rs:GetValue("fname") // John
rs:SetValue("fname","Sean")
? rs:GetValue("fname") // Sean
```

3.4.34. TRowset:Skip()

```
Skip([<nRows>]) --> nRows
```

Parámetros

<nRows>
El número de filas lógicas a mover, en relación a la fila en curso. Un valor positivo significa saltos hacia adelante, y un valor negativo significa saltar hacia atrás. Si se omite, el valor 1 es asumido.

Valor de retorno

Número real de filas saltadas.

Descripción

Skip() mueve ya sea hacia adelante o hacia atrás en relación a la fila actual. Un intento para saltar más allá de la posición de la última fila lógica, quedará en la última fila y Trowset:Eof() retornará verdadero (.T.). Un intento para saltar más allá de la posición de la primera fila lógica, quedará en la primera fila y Trowset:Bof() retornará verdadero (.T.).

Ejemplo

```
rs := conn:CreateRowset("SELECT * FROM mytable")
while !rs:Eof()
        ? rs:Read()
        rs:Skip()
enddo
```

3.4.35. TRowset:Write()

```
Write(<oRow>) --> NIL
```

Parámetros

<oRow>
Un objeto conteniendo nuevos valores para los campos.

Descripción

Write() asigna nuevos valores a los campos de la fila activa. Luego trata de reflejar los cambios en el servidor SQL por ejecución de la instrucción <cUpdateSQL> parada al constructor Trowset Tconnect:CreateRowset().

Ejemplo

```
rs := conn:CreateRowset("SELECT id,fname,lname FROM mytable",,,,;
        "UPDATE mytable SET fname=:fname,lname=:lname WHERE id=:id")
oRow := map()
oRow:fname := "John"
oRow:lname := "Smith"
rs:Write(oRow)
? rs:GetValue("fname") // John
? rs:GetValue("lname") // Smith
```