VORTEXperl
Reference Manual

May 30, 2017

Trifox Inc.
3131 S. Bascom Avenue
Campbell, CA 95008

www.trifox.com

Trademarks

TRIMapp, TRImpl, TRIMqmr, TRIMreport, TRIMtools, GENESISsql, DesignVision, DVapp, DVreport, VORTEX, VORTEXcli, VORTEXc, VORTEXcobol, VORTEXperl, VORTEXjdbc, VORTEX++, VORTEXJava Edition, LIST Manager, VORTEXodbc, VORTEXnet, VORTEXclient/server, VORTEXaccelerator, VORTEXreplicator are all trademarks of Trifox, Inc.

All other brand and product names are trademarks or registered trademarks of their respective owners.

Copyright

Printed in the U.S.A.

Contents

Preface

Background

Trifox Inc. has been serving the relational database market since 1984 through consulting and the development of software products. In 1987, Trifox created SQL*QMX for Oracle. This easy-to-use, powerful querying and report writing tool, which is based on IBM's QMF, continues to be used at thousands of sites. In 1989, Trifox created TRIMtools, a family of application and reportwriting tools now known as DesignVision. DesignVision was developed in response to the OLTP requirements of several large application vendors.

Database Access

VORTEX is an integrated family of products that allows nearly any production application to access SQL data:

- On any or all of the major relational databases.
- Across networks.
- Across platforms.
- With a dramatic increase in the number of concurrent users.
- Without any additional hardware.

In a client/server or multi-tier configuration, VORTEX makes it possible for your SQL applications to access data on different platforms over one or more network configurations. Currently it supports only TCP/IP.

Inherent in this approach are services that allow production applications originally written for one relational database (such as Oracle) can access the same data on another database (such as Informix), even if it is spread across different databases.

VORTEX Precompilers for C and COBOL, as well as a variety of program interfaces, allow existing SQL programs to take full advantage of VORTEX services such as performance enhancement, transaction monitoring, and flat-file database access.

With VORTEXaccelerator in your configuration, you dramatically increase the number of concurrent users who can log on to a specific SQL production application. Your users experience faster performance and you won't have to change any programs or add any hardware.

Application and Report Development

DesignVision DVapp lets you design, generate, and maintain forms-based applications. You can easily port the pop-up windows, customizable menus and submenus, and

custom keyboard assignments, in fact the entire application, to Windows .NET, Unix, OpenVMS, or HTML5 with no extra effort.

The reportwriter, TRIMreport, lets you create simple reports quickly, or complex reports with absolute confidence in their power.

When you want to write stand-alone applications (including triggers) without a user interface, the TRIMpl 4GL language gives you the freedom you want. The procedural language has over 100 database-specific functions that help you write powerful applications in very little time.

Reaching Legacy Data

GENESISsql is a SQL processor that accesses low-level data sources such as ISAM, SDMS, ADABAS, RMS, and MicroFocus and makes the data accessible to VORTEX clients. You can add GENESIS data sources to a VORTEX system in a matter of days, simplifying what used to be an enormous task.

Conventions

Screen shots in this manual come from the Windows version of our software.

Trifox documentation uses the following conventions for communicating information:

Example	Describes
`CHOOSE REPORT > [F3] >`	Press [F3] on the CHOOSE REPORT menu and ...
Right-click	Clicking the right mouse button.
Left-click	Clicking the left mouse button.
`connect_string`	Replace italicized text with your own variable.
`vtxnetd`	Text in bold typewriter style represents strings that you type exactly as they appear in the manual.

Support

If you have a question about a TRIFOX product that is not answered in the documentation (paper or online), contact the Customer Support Services group at:

- support@trifox.com
- Trifox Customer Support Services
  2959 Winchester Boulevard
  Campbell, CA 95008
  U.S.A.
- 408-796-1590

Chapter 1 Overview

Introduction

VORTEXperl is a thin layer of Perl code that provides a method of accessing the VORTEXserver. The main VORTEXperl modules are:

- ***dbChannel*** — Encapsulates a database connection. Each instance corresponds to an actual database connection.
- ***dbCursor*** — Encapsulates a database cursor.

Program Flow

The typical flow of a VORTEXperl program has the following steps.

1. Initialize a dbChannel.

   ```
   $db = new dbChannel(128,32,128,8192);
   ```

2. Open a VORTEXserver database connection.

   ```
   $db->dbConnect("tri11",1958,"VTX0","scott/tiger",
   "ORACLE_HOME=/usr4/oracle/product/8.0.4,ORACLE_SID=A");
   ```

   Refer to the *VORTEX Installation and Operations Manual* for more information concerning VORTEXserver connection parameters.

3. Create a cursor.

   ```
   $c1 = new dbCursor($db);
   ```

4. Associate a SQL statement to the cursor.

   ```
   $db->sql($c1,"select * from staff where id >= :1",1,1);
   ```

5. Bind any parameters.

   ```
   $c1->setIntParam(0,0,15);
   ```

6. Execute for DML SQL or fetch for SELECT statements

   ```
   $db->fetch($c1);
   ```

7. Fetch the data.

   ```
   $nco = $c1->numOutputCols();
   while (!$c1->endOfScan()) {
      for ($i=0;$i<$nco;$i++) {
         $buf = $db->getString($c1);
         print "$buf ";
      }
   print "\n";
   }
   ```

8. Close the connection.

   ```
   $db->release();
   ```

Chapter 2 dbChannel

Module dbChannel

new

Allocates and initializes a dbChannel, which is the connection used to communicate with the database. You can open several dbChannels, each pointing to a different database or even the same database. If you open more than one dbChannel to the same database, however, you may find that you are deadlocked against yourself. This method corresponds to the VTXINIT call in VORTEXcli.

Parameters

```
dbChannel ( const int maxLogicalCursors,
            const int maxDbCursors,
            const int maxColumns,
            const int fetchBufferSize);
```

maxLogicalCursors Maximum number of logical cursors to allocate for the connection. A logical cursor is mapped to a real database cursor and lets an application reuse cursors using parameters. This approach minimizes the amount of work done to reuse a statement because it is more efficient than creating unique statements with parameter values hard-coded in the SQL statement. For best performance you should allocate one cursor for each SQL statement.

maxDbCursors Number of actual database cursors to allocate. Logical cursors are automatically mapped to database cursors. The dbChannel keeps track of which cursors are in use and which are soft-closed (meaning that all results have been retrieved). If a logical cursor needs an database cursor and none are available, dbChannel looks for the first soft-closed cursor, marks it as hard-closed and gives its database cursor to the requesting logical cursor. Because database cursors are fairly expensive to allocate (database side), you should allocate fewer than the number of logical cursors.

maxColumns Maximum number of output columns that any given SELECT statement can have.

fetchBufferSize Size of the fetch buffer to allocate. When fetching data from the database the number of records to pre-fetch will depend on the record size. The actual fetch buffer is per dbCursor and is not allocated until the `fetch()` is invoked. A good starting fetch buffer is 8192. If your application is mostly reading lots of data sequentially, then you should experiment with a larger buffer. For random reads or reads where your application only uses the first few records, a smaller buffer size will actually be faster.

Example

Allocates a connection with an 8KB fetch buffer:

```
$db = new dbChannel(256,64,128,8192);
```

Methods

These functions are listed in `dbChannel.hxx`.

cancel

Cancels an outstanding request. This method is typically called after a user interrupt. Returns false on success, true on failure.

command

Sends a database-specific command. A description of the supported commands is located in the *VORTEXcli Reference Manual* in the VTXCMD section.

Parameters

cursor	dbCursor object.
cmd	Command identifier.
value	Command value. The value is always passed as a string.

Example

Sets the resource timeout value to 5 seconds.

```
$db->command($cursor,TDB_CMD_TIMEOUT(),``5'');
```

commit

Commit a transaction.

Parameters

startUpdateTrans	If true, a begin transaction is performed immediately following the commit. All activity takes place on the server side which minimizes network traffic. Returns false on success, true on failure.

cursor_close

Closes a dbCursor. Returns false on success, true on failure.

Parameters

cursor dbCursor object.

hard If true, performs a ``hard'' close, which sends a message to the database driver to close the cursor and release all associated resources. If not true, the cursor is ``soft'' closed. Hard close is not usually necessary and causes extra overhead. Use the soft close option whenever possible.

dbConnect

Connects to a remote database. Returns false on success, true on failure. The complete connection syntax is described in the VORTEX Installation and Operations Manual available at www.trifox.com.

Parameters

hostName Host to connect to. Can be in either the symbolic name format or the IP address dot notation. If the VORTEXserver (vtxnetd) requires authentication then you must append a userid and password to hostName: hostName(uid/pwd)

port Port number on which VORTEXserver (vtxnetd) is listening.

hostProgram Executable to service the connection. On server platforms that support DLLs this is the name of the DLL. See the *VORTEX Installation and Operations Manual* for more information.

dbConnectString Connect string passed to the actual database.

envVariables Optional environment variables to be set on the server; for example, database environment variables.

Examples

Connects to an Oracle database on a UNIX machine.

```
   $db->dbConnect(``orahost'',1958,``VTX0'', ``scott/tiger'',
``ORACLE_HOME=/usr/local/oracle,ORACLE_SID=A'');
```

Connects to a SQL Server on a NT machine.

```
   $db->dbConnect(``sqlnt'',1958,``vtx12'',``sa/sa/master2/SQLNT'',```'');
```

Connects to a DB2 database on MVS with authentication.

```
   $db->dbConnect(``sys1(SYS1/SYSPWD)'',1958,``7'',``/'',```'');
```

execute

Executes a non-SELECT dbCursor. Returns false on success, true on failure. You must bind all the parameters specified in the `sql()` method using either `setCharParam()`, `setIntParam()`, or `setBlobParam()`. The SQL statement is only sent to the database the first time. On subsequent calls only the parameters (if any) are sent.

Parameter

cursor dbCursor object.

executeIO

Executes a stored procedure dbCursor. Returns false on success, true on failure. You must bind all the parameters specified in the sql method using `setCharParam()` or `setIntParam()`. Note that output parameters must set the flag parameter with `setCharParam()` and `setIntParam`. If the database's stored procedure facility supports a function return value, this value is located in the message buffer. Use `sqlError()` to retrieve the value. The SQL statement is only sent to the database the first time. On subsequent calls only the parameters (if any) are sent.

Parameter

cursor dbCursor object.

fetch

Opens and fetches from a SELECT dbCursor. Returns false on success, true on failure. You must bind all the parameters specified in the `sql()` method using `setIntParam()` and `setCharParam()`. The SQL statement is only sent to the database the first time. On subsequent calls only the parameters (if any) are sent.

Data is automatically retrieved into a fetch buffer (using the fetch buffer size specified when the dbChannel was created). The number of rows per fetch varies depending upon the total length of a row.

The column data must be retrieved sequentially using the `getString()` method.

Parameter

cursor dbCursor object.

getRow

Gets the next row of data and return an array of data items. Returns ($errorcode,$row). Errorcode is false on success, true on failure.

Parameters

cursor dbCursor object.

getRowsAffected

Get the number of rows affected by the last `execute()` call. This value is valid only for INSERT, UPDATE, and DELETE statements. If you are doing a bulk operation that fails due to a uniqueness violation for example, getRowsAffected returns the number of rows that were successfully executed before the error.

getSkip

Skips (passes over) columns. A large `numColsToSkip` with `stopOnColZero` set to true advances to the next row. Returns false on success, true on failure.

Parameters

cursor dbCursor object.

numColsToSkip Number of columns to skip over.

stopOnColZero Never skip past column zero of the next row.

getString

Gets the next column as a string. Numeric and datetime data are automatically converted to a string representation. Returns ($errorcode,$string). Errorcode is false on success, true on failure.

Parameters

cursor dbCursor object.

getStringIO

Gets the next stored procedure column as a string. Numeric data is automatically converted to a string representation. Returns ($errorcode,$string). Errorcode is false on success, true on failure.

Parameters

cursor dbCursor object.

putBlob

Sends BLOB data. The VORTEXperl blob/clob interface is rather involved. To bind a blob/clob parameter, you must first call `setBlobParam()` with the length of the blob/clob. Then you use `putBlob` to send the blob/clob parameter. To see an example, refer to `sample3.pl` in the VORTEXperl download.

Parameters

cursor	dbCursor object.
col	Blob column number.
blob	Blob data.
len	Blob length.

release

Releases a connected database. Returns false on success, true on failure. A `rollback()` takes place before the actual database release request is sent.

release2

Sends a temporary release, used with VORTEXweb. Returns false on success, true on failure.

rollback

Rollback a transaction.

Parameter

startUpdateTrans	If true then a begin transaction is performed immediately following the rollback. All activity takes place on the server side which minimizes the network traffic. Returns false on success, true on failure.

sql

Associates a SQL statement to a dbCursor.

This method simply caches the SQL statement and other information. If the dbCursor was previously used, its database cursor is hard closed and all resources are freed. The new SQL statement is not sent to the database until either a `fetch()` or `execute()` is performed. If you use parameters in your SQL, you do not have to call `sql()` again for this SQL statement. You merely use the `setIntParam()`, `setCharParam()`, and `setBlobParam()` methods to change the parameter values. This approach is much more efficient than creating unique SQL strings.

Parameters

cursor dbCursor object.

sqlStatement The SQL statement.

numDimension If the SQL statement contains parameter markers (? or :n) then this specifies how many "sets" of parameters there are. For a SELECT statement with parameters this must be 1. For INSERT/UPDATE/DELETE statements this parameter indicates how many records to process. If the SQL statement does not contain any parameters the value must be set to 0.

numParameters If numDimensions is not zero (0), then this indicates the number of parameters in each "set" of parameters.

Examples

Prepares a SELECT statement with no parameters.

```
$db->sql($cursor,``select * from staff'',0,0);
```

Prepares a bulk INSERT statement with 7 parameters in batches of 50.

```
$db->sql($cursor,``insert into staff values
        (:1,:2,:3,:4,:5,:6,:7)'',50,7);
```

synchronize

Resynchronizes a cursor after a release2/connect. Used only with VORTEXweb. Returns false on success, true on failure.

Parameters

cursor dbCursor object.

cursor number The cursor number value returned from the $dbCursor->save function.

cursor status The cursor status was returned from the $dbCursor->save function.

sqlError

Return the last SQL error code and message

Chapter 3 dbCursor

Module dbCursor

new

Returns `$errorcode,$dbCursor`.

A dbCursor is required for each SQL statement that is to be processed. VORTEX automatically maps a logical cursor to an actual database cursor which allows an application to allocate many more cursors than the underlying database actually can support.

A dbCursor has two close states: soft and hard.

Normally an application closes a cursor in soft mode, which means that the cursor is simply marked as closed on the client. When the dbCursor is used again VORTEX determines if the cursor can be re-used without having to open (re-parse) the database cursor. Reuse is a big performance boost. In a client/server environment the cursor cache becomes even more important as it cuts down the number of network roundtrips needed.

A LRU algorithm is used when soft closed cursors need to be re-mapped.

Methods

endOfScan

Returns true if no more (fetch) data is available. This call is only valid after a `fetch()`.

getColDesc

Returns `$errorcode,$dbDescriptor` for an output column. This call is only valid after a `fetch()`.

Parameters

columnIndex	Column index.	
dbDescriptor	type	Datatype (1 - Character, 12 - Datetime, 2 - Numeric)
	length	Maximum length of data item (applicable only to Character fields).
	name	Name of column.
	nullsAllowed	Nulls allowed.
	precision	Numeric precision.
	scale	Numeric scale.

numOutputCols

Returns the number of output columns. This call is only valid after a `fetch()`.

setBlobParam

Set (bind) a Blob parameter. Use ``'' to pass a NULL value to the database.

Parameters

col Column of parameter (zero-based).

datatype Datatype (TDT_BLOB or TDT_CLOB).

data Data item.

setCharParam

Sets (bind) a String parameter. Uses ““to pass a NULL value to the database.

Parameters

row Row of parameter (bulk operations).

col Column of parameter (zero-based).

data Data item.

flag Optional flag for stored procedures. For output parameters, set this to `TDB_FLG_OUT()`.

name Name of the parameter. This parameter is used by Oracle, Sybase, and SQL Server stored procedures.

setIntParam

Set (bind) an Integer parameter. Use ``'' to pass a NULL value to the database.

Parameters

row Row of parameter (bulk operations).

col Column of parameter (zero-based).

data Data item.

flag Optional flag for stored procedures. For output parameters, set this to `TDB_FLG_OUT()`

name Name of the parameter. This parameter is used by Oracle, Sybase, and SQL Server stored procedures.

save

Returns cursor number and state used for later synchronize operation.