1

DBD::FreeTDS

Version

Version 0.02.

Author and Contact Details

The driver author is Craig Spannring. He can be contacted via the FreeTDS mailing list. You can subscribe to the mailing list at:

```
http://metalab.unc.edu/freetds/
```

Supported Database Versions and Options

The `DBD::FreeTDS` driver supports Microsoft SQLServer 6.5 and Sybase 11.*x* It has not been tested for Sybase 10 or SQLServer 7.0.

Connect Syntax

The `DBI->connect()` Data Source Name (DSN) is one of the following:

```
dbi:FreeTDS:database=$DATABASE;host=$DBHOST
dbi:FreeTDS:database=$DATABASE;host=$DBHOST;port=$PORT
```

MS SQLServer generally uses port 1433 (the default). Sybase does not have a standard port number so you will have to check with your local DBA.

There are no driver specific attributes for the `DBI->connect()` method.

Numeric Data Handling

The DBD::FreeTDS will support all numerical types supported by Sybase and SQLServer. However, only BIT, FLOAT, INT, REAL, SMALLINT, and TINYINT are currently implemented.

String Data Handling

SQLServer and Sybase support the following string data types:

```
VARCHAR(size)
CHAR(size)
```

CHAR and VARCHAR types and the RAW type have a limit of 255 bytes. DBD::FreeTDS has full support for both of these types.

The CHAR type is fixed length and blank padded. Trailing blanks are not significant for string comparisons.

See the documentation for SQL Server and Sybase for handling of 8 bit characters.

Date Data Handling

SQLServer supports the DATETIME and SMALLDATETIME values. A DATETIME can have a value from Jan 1 1753 to Dec 31 9999 with a 300th of a second resolution. A SMALLDATETIME has a range of Jan 1 1900 to Jun 6 2079 with a 1 minute resolution.

Sybase and SQLServer recognize a wide range of date input formats. Use the CONVERT() SQL function to convert date and time values from other formats. For example:

```
UPDATE a_table
   SET date_field = CONVERT(datetime_field, '1999-02-21', 105)
```

CONVERT() is a generic conversion function that can convert to/from most datatypes. See the CONVERT() function in Chapter 2 of the Sybase Reference Manual.

If you specify a DATE value without a time component, the default time is 00:00:00 (midnight). If you specify a DATE value without a date, the default date is January 1, 1900. If the century is omitted it is assumed to be 1900 if the year < 50 and 2000 if year >= 50.

The current date and time is returned by the GETDATE() function.

The default output format for date types is Day dd yyyy hh:mmAM, e.g. Mar 1 1999 8:02PM. The default date format for a database session can be changed with the Transact-SQL statement SET DATEFORMAT format. The allowed values for format are mdy, dmy, ymd, ydm, myd, and dym.

Sybase and SQLServer can operate on date-time values with the functions DATEADD(), DATEDIFF(), and DATENAME(). You can also use the LIKE operator on date-time values.

Sybase and SQLServer do not understand time zones at all, except that the `getdate()` SQL function returns the date in the time zone that the server is running in (via `localtime`).

The following SQL expression can be used to convert an integer ‘‘seconds since 1-jan-1970’’ value (‘‘unix time’’) to the corresponding database date time:

```
DATEADD(ss, unixtime_field, '1970-01-01')
```

Note however that the server does not understand time zones, and will therefore give the value for local time and not the correct value for the GMT time zone.

If you know that the server runs in the same timezone as the client, you can use:

```
use Time::Local;
$time_to_database = timegm(localtime($unixtime));
```

to convert the unixtime value before sending it to Sybase.

To do the reverse, *i.e.*, convert from a database date time value to ‘‘unix time,’’ you can use:

```
DATEDIFF(ss, '1970-01-01'', datetime_field)
```

The same GMT *vs.* localtime caveat applies in this case. If you know that the server runs in the same timezone as the client, you can convert the returned value to the correct GMT based value with this Perl expression:

```
use Time::Local;
$time = timelocal(gmtime($time_from_database));
```

LONG/BLOB Data Handling

SQLServer and Sybase support an IMAGE and a TEXT type for LONG/BLOB data. Each type can hold up to 2GB of binary data, including nul characters. The main difference between an IMAGE and a TEXT column lies in how the client libraries treat the data on input and output. TEXT data is entered and returned ‘‘as is’’. IMAGE data is returned as a long hex string, and should be entered in the same way.

Having said all that, `DBD::FreeTDS` does not yet support these types.

Other Data Handling issues

The DBD::FreeTDS driver does not support the type_info() method yet.

Transactions, Isolation and Locking

SQLServer and Sybase support transactions, but DBD::FreeTDS does not yet support access to them.

SQLServer uses shared locks for read operations and exclusive page locks for writes. Initially, SQLServer attempts to use page locking, but will quickly escalate the lock to a table level lock. Both readers and writers can be blocked waiting to acquire a lock. SQLServer can detect mutually blocked connections and will rollback the transaction of the one of the connections. Please see the server documentation for more information about how SQLServer chooses the deadlock victim.

SQLServer supports several extensions to the SELECT statement that affect locking. See the SQLServer documentation for more details.

No-Table Expression Select Syntax

To select a constant expression (one that doesn't involve data from a database table or view), you can select it without naming a table:

```
SELECT getdate()
```

Table Join Syntax

Outer joins are supported in Sybase using the =* (right outer join) and *= (left outer join) operators:

```
SELECT customer_name, order_date FROM customers, orders WHERE
customers.cust_id =* orders.cust_id
```

For all rows in the customers table that have no matching rows in the orders table, Sybase returns NULL for any select list expressions containing columns from the orders table.

SQLServer uses ANSI standard syntax for outer joins.

```
SELECT publishers.pub_id, titles.title, titles.title_id
FROM titles RIGHT OUTER JOIN publishers
ON publishers.pub_id = titles.pub_id
```

Table and Column Names

The names of SQLServer identifiers, such as tables and columns, cannot exceed 30 characters in length.

The first character must be a letter, or one of the symbols _, @, or #. The remaining characters con be any combination of letters, digits, or the symbols _, @, or #. Identifiers that start with the pound sign (#) symbol are temporary objects; objects starting with a double pound sign (##) are global temporary objects. Identifiers for temporary objects should be less than 14 characters including the pound sign.

Identifiers can be enclosed in double quote marks ("). Quoted identifiers may contain any combination of characters in the current code page except for double quote marks.

SQLServer stores identifiers as entered but does not otherwise distinguish between case. Two identifiers are considered equivalent if they differ only in case.

National characters can be used in identifier names without quoting.

Case Sensitivity of LIKE Operator

The Sybase and SQLServer LIKE operator is case sensitive.

The `UPPER` function can be used to force a case insensitive match, *e.g.*, `UPPER(name) LIKE 'TOM%'`. However, this prevents the server from making use of any index on the name column to speed up the query.

Row ID

SQLServer does not have an implicit table row ID column.

A table can be created with an explicit identity column using the IDENTITY keyword. Please refer to the server documentation for more details. The column with the IDENTITY property can be accessed either through the name given in the `CREATE` statement or with the alias IDENTITYCOL.

To fetch the value generated and used by the last insert, you can

```
SELECT @@IDENTITY
```

Automatic Key or Sequence Generation

SQLServer has two types of automatically generated columns: IDENTITY and TIMESTAMP.

Columns with the IDENTITY property are automatically and uniquely set when a row is inserted. The value of the IDENTITY column can not be changed for a row after it's been inserted. To fetch the value generated and used by the last insert, you can:

```
SELECT @@IDENTITY
```

TIMESTAMP columns are set to a unique value when a row is first inserted and automatically updated with a new and unique value each time the row is modified.

SQLServer does not support sequence generators other than the IDENTITY and TIMESTAMP columns.

Automatic Row Numbering and Row Count Limiting

SQLServer does not provide any pseudo-columns for sequentially numbering the rows in a result set.

Parameter Binding

Parameter binding is not yet supported by the `DBD::FreeTDS` driver. Both the `?` and `:1` style of placeholders will be supported in the future.

Stored Procedures

`DBD::FreeTDS` can execute stored procedures with the EXEC Transact-SQL keyword. For example, if you want to call a stored procedure named ‘‘foo’’ that takes two parameters, you would write:

```
$sth = $dbh->prepare("exec foo 1, 2");
$sth->execute;
```

SQLServer and Sybase stored procedures are implemented in the Transact-SQL language (a procedural extension to SQL that supports variables, control flow, packages, exceptions, etc).

Stored procedures often return multiple result sets. This can be handled using the *syb_more_results* statement handle attribute. For example:

```
do {
    while (@rs = $sth->fetchrow_array) {
        ...process the data...;
    }
} while ($sth->{syb_more_results});
```

Table Metadata

`DBD::FreeTDS` does not support the `table_info()` method yet.

The *syscolumns* table has one row per column per table. See the definitions of the Sybase system tables for details. However, the easiest method is to use the `sp_help` stored procedure.

The easiest way to get detailed information about the indexes of a table is to use the `sp_helpindex` (or `sp_helpkey`) stored procedure.

Driver-specific Attributes and Methods

`DBD::FreeTDS` has no significant driver-specific handle attributes or private methods.

Positioned updates and deletes

SQLServer supports updatable cursors but `DBD::FreeTDS` does not implement those yet.

Differences from the DBI Specification

`DBD::FreeTDS` is a work in progress. Significant sections of the DBI specification have not been implemented yet.

URLs to More Database/Driver Specific Information

```
http://www.microsoft.com/sql
http://techinfo.sybase.com
http://sybooks.sybase.com
```

Concurrent use of Multiple Handles

`DBD::FreeTDS` supports an unlimited number of concurrent database connections to one or more databases.

`DBD::FreeTDS` allows multiple outstanding statements for each database handle. The driver maintains one connection for every active statement.

Note: SQLServer will count each active statement as one user connection. You should be careful to close statement handles so that you don’t run out of licensed connections.