1

DBD::ODBC

Version

Version 0.20.

Author and Contact Details

The driver authors are Tim Bunce and Jeff Urlwin. The original work was based upon an early version of Thomas Wenrich's `DBD::Solid`. The authors can be contacted via the *dbi-users* mailing list.

Supported Database Versions and Options

The `DBD::ODBC` module supports ODBC Version 2.x and 3.x on Unix and Win32. For all platforms, both an ODBC driver manager *and* an ODBC driver are required in addition to the `DBD::ODBC` module.

For Win32, the driver manager is included with the operating system. For Unix and variants, the iODBC driver manager source is included in the *iodbcsrc* directory. While iODBC acts as the driver manager, you still have to find a driver for your platform and database. Driver providers include:

```
Intersolv  -  http://www.intersolv.com
OpenLink   -  http://www.openlinksw.com
```

There are others - this is not an exhaustive list.

Connect Syntax

The `DBI->connect()` Data Source Name, or *DSN*, has the following forms:

```
dbi:ODBC:odbc_dsnname
dbi:Oracle:driver=Microsoft Access Driver (*.mdb);dbq=\\server\share\access.mdb
```

The first example above requires the user to setup an ODBC Data Source Name. A DSN is simply a name you use to refer to a set of driver specific connection parameters defined elsewhere. Connection parameters typically include the name of the ODBC driver to use, the database name, and any required connection details.

Under Win32, the best method of accomplishing this is by using the ODBC32 control panel applet. Under Unix variants you typically need to edit a text file called *.odbc.ini* in your home directory. Refer to your driver manager documentation for more details.

The second connection example above uses the driver specific connection string. By specifying all the required information, you can bypass the need to use a previously defined DSN.

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

Numeric Data Handling

The numeric data handling for ODBC is dependent upon a variety of factors. One of those critical factors is the end database. For example, Oracle supports different numeric types than Sybase which, in turn, supports different numeric types than a CSV file. You will need to read your database documentation for more information.

Unfortunately, the second critical set of factors are the ODBC driver manufacturer and version of the driver. For example, I have seen great variety in handling of numeric values between versions of Oracle's ODBC drivers. What works with one version, sadly, may not work with even a later version of Oracle's drivers. You will need to read your ODBC driver documentation for more information.

The DBI `type_info()` and `type_info_all()` methods provide information about the data types supported by the database and driver being used.

String Data Handling

As with numeric handling, string data handling is dependent upon the database and driver. Please see above for more information.

Strings can be concatenated using the `CONCAT(s1,s2)` SQL function.

Date Data Handling

As with numeric handling, date data handling is dependent upon the database and driver. Please see above for more information.

You can use ODBC escape sequences to define a date in a database independent way. For example, to insert a date of Jan 21, 1998 into a table, you could use:

```
INSERT INTO table_name (date_field) VALUES ({d '1998-01-21'});
```

You can use placeholders within escape sequences instead of literal values. For example:

```
INSERT INTO table_name (date_field) VALUES ({d ?});
```

Similar escape sequences are defined for other date time types. Here's the full set:

```
{d 'YYYY-MM-DD'}                    -- date
{t 'HH:MM:SS'}                      -- time
{ts 'YYYY-MM-DD HH:MM:SS'}          -- timestamp
{ts 'YYYY-MM-DD HH:MM:SS.FFFFFFF'}  -- timestamp
```

If you specify a DATE value without a time component, the default time is 00:00:00 (midnight). There is also an interval escape clause which is constructed like this:

```
{interval [+|-] 'value' [interval_qualifier]}
```

For example:

```
{interval '200-11' YEAR(3) TO MONTH}
```

Please see an ODBC reference guide for more information.

The current date and time on the server can be found by using an ODBC scalar function escape sequence to call the appropriate function. For example:

```
INSERT INTO table_name (date_field) VALUES ({fn CURDATE});
```

The `{fn ...}` escape sequence isn't required if the entire SQL statement conforms to the level of SQL-92 grammar supported by your ODBC driver.

Other related functions include `CURTIME()`, `NOW()`, `CURRENT_DATE()`, `CURRENT_TIME()`, and `CURRENT_TIMESTAMP()`. The last three require an ODBC v3 driver.

Other date time related functions include: `DAYNAME()`, `DAYOFMONTH()`, `DAYOFWEEK()`, `DAYOFYEAR()`, `EXTRACT()`, `HOUR()`, `MINUTE()`, `MONTH()`, `MONTHNAME()`, `SECOND()`, `WEEK()`, `YEAR()`.

Basic date time arithmetic can be performed using the `TIMESTAMPADD()` and `TIMESTAMPDIFF()` functions.

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

```
TIMESTAMPADD(SQL_TSI_SECOND, seconds_since_epoch, {d '1970-01-01'})
```

to do the reverse you can use:

```
TIMESTAMPDIFF(SQL_TSI_SECOND, {d '1970-01-01'}, date_field)
```

ODBC itself does not have any support for timezones, though the database to which you are connected may.

LONG/BLOB Data Handling

Support for LONG/BLOB data types and their maximum lengths are very dependent on the database to which you are connected.

The *LongReadLen* and *LongTruncOk* attributes work as defined. However, the driver implementations do affect this. Some drivers do not properly indicate that they have truncated the data, or they have more data available than was actually returned. The `DBD::ODBC` tests attempt to determine correct support for this.

No special handling is required for LONG/BLOB data types. They can be treated just like any other field when fetching or inserting etc.

Other Data Handling issues

The `DBD::ODBC` driver supports the `type_info()` method.

Transactions, Isolation and Locking

`DBD::ODBC` supports transactions. However, some databases do not.

Supported isolation levels, the default isolation level, and locking behavior are all dependent on the database to which you are connected.

No-Table Expression Select Syntax

This is dependent upon the database to which you are connected. The ODBC standard SQL does require a table name in SELECT statements. To prevent multiple values being returned you should use the DISTINCT keyword in the query or, better yet, write the query to only match one row in a table you know exists, such as a system catalog.

Table Join Syntax

This is dependent on the database to which you are connected. The ODBC standard SQL defines the standard syntax for inner joins and an escape sequence to use for outer joins:

```
{oj outer_join}
```

where *outer_join* is defined as:

```
table_name [LEFT | RIGHT | FULL]
  OUTER JOIN [ table_name | outer_join] ON condition
```

An outer join request must appear after the FROM clause of a SELECT but before a WHERE clause, if one exists.

Table and Column Names

The maximum length of table and column names, the case-sensitivity of names, and the ability to quote them are all dependent on the database to which you are connected.

Case Sensitivity of LIKE Operator

The ODBC standard SQL defines the LIKE operator as case sensitive. However, a few databases may have case insensitive LIKE operators.

Row ID

The ODBC standard SQL does not define a ‘‘table row id’’ pseudocolumn. However, some databases do provide one, for example, ROWID in Oracle.

Automatic Key or Sequence Generation

This is dependent on the database to which you are connected.

Automatic Row Numbering and Row Count Limiting

This is dependent on the database to which you are connected.

Parameter Binding

Parameter binding is supported by `DBD::ODBC` if the underlying ODBC driver driver supports it. Only the standard `?` style of placeholders is supported.

The *TYPE* attribute to the `bind_param()` method is supported.

Stored Procedures

Stored procedures can be called using the following ODBC escape sequence:

```
{call procedure1_name}
{call procedure2_name(?, ?)}
{?= call procedure3_name(?, ?)}
```

The last form would be used to return values from the procedure, but DBD::ODBC currently does not support output parameters.

Table Metadata

DBD::ODBC supports the table_info() method.

DBD::ODBC also supports many of the ODBC *metadata* functions that can be used to discover information about the tables within a database. These can be accessed as driver-specific private methods:

```
SQLGetTypeInfo   --  $dbh->func(xxx, GetTypeInfo)
SQLDescribeCol   --  $sth->func(colno, DescribeCol)
SQLColAttributes --  $sth->func(xxx, colno, ColAttributes)
SQLGetFunctions  --  $dbh->func(xxx, GetFunctions)
SQLColumns       --  $dbh->func(catalog, schema, table, column, 'columns')
```

The DBI will provide standard methods for all these soon, possibly by the time you read this.

Driver-specific Attributes and Methods

DBD::ODBC has no driver-specific handle attributes.

In addition to the private methods described in above, the GetInfo() private method can be used to discover many many details about the driver and database you are using.

Positioned updates and deletes

This is dependent upon the database to which you are connected. Positioned updates and deletes are supported in ODBC SQL using the ''WHERE CURRENT OF'' syntax. For example:

```
$dbh->do("UPDATE ... WHERE CURRENT OF $sth->{CursorName}");
```

Differences from the DBI Specification

DBD::ODBC does not currently support ''out'' parameter binding. That should be fixed in a later release.

URLs to More Database/Driver Specific Information

```
http://www.openlinksw.com
http://www.intersolv.com
http://www.microsoft.com
```

Links related to the FreeODBC project:

```
http://www.openlinksw.com/iodbc
http://www.genix.net/unixODBC
```

To subscribe to the *freeodbc* development mailing list, send a message to *freeodbc-request@as220.org* with just the word `subscribe` in the body of the message.

Concurrent use of Multiple Handles

Most ODBC drivers and databases let you make multiple concurrent database connections to the same database. A few do not.

Some ODBC drivers and databases, most notably Sybase and SQL Server, do not let you prepare and execute a new statement handle while still fetching data from another statement handle associated with the same database handle.