1

DBD::Oracle

Version

This driver summary is for `DBD::Oracle` version 0.60.

Author and Contact Details

The driver author is Tim Bunce. He can be contacted via the *dbi-users* mailing list.

Supported Database Versions and Options

The `DBD::Oracle` module supports both Oracle 7 and Oracle 8.

Building for Oracle 8 defaults to use the new Oracle 8 OCI interface, which enables use of some Oracle 8 features including LOBs and ‘‘INSERT ... RETURNING ...’’.

An emulation module for the old Perl4 oraperl software is supplied with `DBD::Oracle`, making it very easy to upgrade `oraperl` scripts to Perl5.

Connect Syntax

The `DBI->connect()` Data Source Name, or *DSN*, can be one of the following:

```
dbi:Oracle:tnsname
dbi:Oracle:sidname
dbi:Oracle:T:hostname:sidname
```

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

Numeric Data Handling

Oracle only has one flexible underlying numeric type, NUMBER. But Oracle does support several ANSI standard and IBM data type names as aliases, including:

```
INTEGER      = NUMBER(38)
INT          = NUMBER(38)
SMALLINT     = NUMBER(38)
DECIMAL(p,s) = NUMBER(p,s)
NUMERIC(p,s) = NUMBER(p,s)
FLOAT        = NUMBER
FLOAT(b)     = NUMBER(p)   where b is the binary precision, 1 to 126
REAL         = NUMBER(18)
```

The NUMBER datatype stores positive and negative fixed and floating-point numbers with magnitudes between 1.0 x 10-130 and 9.9...9 x 10125 (38 nines followed by 88 zeroes), with 38 digits of precision.

You can specify a fixed-point number using the following form: `NUMBER(p,s)` where *s* is the scale, or the number of digits to the right of the decimal point. The scale can range from -84 to 127.

You can specify an integer using `NUMBER(p)`. This is a fixed-point number with precision *p* and scale 0. This is equivalent to `NUMBER(p,0)`.

You can specify a floating-point number using NUMBER. This is a floating-point number with decimal precision 38. A scale value is not applicable for floating-point numbers.

`DBD::Oracle` always returns all numbers as strings. Thus the driver puts no restriction on size of PRECISION or SCALE.

String Data Handling

Oracle supports the following string data types:

```
VARCHAR2(size)
NVARCHAR2(size)
CHAR
CHAR(size)
NCHAR
NCHAR(size)
RAW(size)
```

The RAW type is presented as hexadecimal characters. The contents are treated as non-character binary data and thus are never ‘‘translated’’ by character set conversions.

CHAR types and the RAW type have a limit of 2000 bytes. For VARCHAR types the limit is 2000 bytes in Oracle 7 and 4000 in Oracle 8.

The NVARCHAR2 and NCHAR variants hold string values of a defined national character set (Oracle 8 only). For those types the maximum number of characters stored may be lower when using multibyte character sets.

The CHAR and NCHAR types are fixed length and blank padded.

Oracle automatically converts character data between the character set of the database defined when the database was created and the character set of the client, defined by the NLS_LANG parameter for the CHAR and VARCHAR2 types or the NLS_NCHAR parameter for the NCHAR and NVARCHAR2 types.

CONVERT(string, dest_char_set, source_char_set) can be used to convert strings between character sets. Oracle 8 supports 180 storage character sets. UTF-8 is supported. See the National Language Support section of the Oracle Reference manual for more details on character set issues.

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

Date Data Handling

Oracle supports one flexible date/time data type: DATE. A DATE can have any value from January 1, 4712 BC to December 31, 4712 AD with a one second resolution.

Oracle supports a very wide range of date formats and can use one of several calendars (Arabic Hijrah, English Hijrah, Gregorian, Japanese Imperial, Persian, ROC Official (Republic of China) and Thai Buddha). We'll only consider the Gregorian calendar here.

The default output format for the DATE type is defined by the NLS_DATE_FORMAT configuration parameter, but it's typically `DD-MON-YY`, *e.g.*, 20-FEB-99 in most western installations. The default input format for the DATE type is the same as the output format. Only that one format is recognised.

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 the first day of the current month. If a date format that has a two digit year, such as the `YY` in `DD-MON-YY` (a common default) then the date returned is always in the current century. The `RR` format can be used instead to provide a 50 year pivot.

The default date format is specified either explicitly with the initialization parameter NLS_DATE_FORMAT or implicitly with the initialization parameter NLS_TERRITORY. For information on these parameters, see Oracle8 Referencee.

You can change the default date format for your session with the ''ALTER SESSION'' command. For example:

```
ALTER SESSION SET NLS_DATE_FORMAT='MM/DD/YYYY'
```

The `TO_DATE()` function can be used to parse a character string containg a date in a known format. For example:

```
UPDATE table SET date_field=TO_DATE('1999-02-21', 'YYYY-MM-DD')
```

The TO_CHAR() function can be used to format a date. For example:

```
SELECT TO_CHAR(SYSDATE, 'YYYY-MM-DD') FROM DUAL
```

The current datetime is returned by the SYSDATE() function.

You can add numbers to DATE values. The number is interpreted as numbers of days; for example, SYSDATE + 1 is this time tomorrow, and SYSDATE - (3/1140) is three minutes ago. You can subtract two dates to find the difference, in days, between them.

Oracle provides a wide range of date functions including ROUND(), TRUNC(), NEXT_DAY(), ADD_MONTHS(), LAST_DAY() (of the month), and MONTHS_BETWEEN().

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

```
to_date(trunc(:unixtime/86400, 0) + 2440588, 'J') -- date part
+(mod(:unixtime,86400)/86400)                     -- time part
```

To do the reverse you can use:

```
(date_time_field - TO_DATE('01-01-1970','DD-MM-YYYY')) * 86400
```

Oracle does no automatic time zone adjustments. However it does provide a NEW_TIME() function that calculates time zone adjustments for a range of time zones. NEW_TIME(d, z1, z2) returns the date and time in time zone *z2* when the date and time in time zone *z1* are represented by *d*.

LONG/BLOB Data Handling

Oracle supports these LONG/BLOB data types:

```
LONG      - Character data of variable length
LONG RAW  - Raw binary data of variable length
CLOB      - A large object containing single-byte characters
NCLOB     - A large object containing national character set data
BLOB      - Binary large object
BFILE     - Locator for external large binary file
```

The LONG types can hold upto 2 gigabytes. The other types (LOB and FILE) can hold upto 4 gigabytes. The LOB and FILE types are only available when using Oracle 8 OCI.

The LONG RAW, and RAW, types are passed to and from the database as strings consisting of pairs of hex digits.

The *LongReadLen* and *LongTruncOk* attributes work as defined. However, the LongReadLen attribute seems to be limited to 65535 bytes on most platforms when using Oracle 7. Building DBD::Oracle with Oracle 8 OCI raises that limit to 4 gigabytes.

The maximum length of `bind_param()` parameter value that can be used to insert LONG data seems to be limited to 65535 bytes on most platforms when using Oracle 7. Building `DBD::Oracle` with Oracle 8 OCI raises that limit to 4 gigabytes.

The TYPE attribute value SQL_LONGVARCHAR indicates an Oracle LONG type. The value SQL_LONGVARBINARY indicates an Oracle LONG RAW type. These values are not always required but their use is strongly recommended.

No other 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::Oracle` driver supports the `type_info()` method.

Oracle supports automatic conversions between data types wherever it's reasonable.

Transactions, Isolation and Locking

`DBD::Oracle` supports transactions. The default transaction isolation level is 'Read Commited'.

Oracle supports READ COMMITED and SERIALIZABLE isolation levels. The level be changed per-transaction by executing a `SET TRANSACTION ISOLATION LEVEL x` statement (where *x* is the name of the isolation level required).

Oracle also supports transaction-level read consistency. This can be enabled by issuing a SET TRANSACTION statement with the READ ONLY option.

In Oracle, the default behavior is that a lock never prevents other users from querying the table. A query never places a lock on a table. Readers never block writers and writers never block readers.

Rows returned by a SELECT statement can be locked to prevent them from being changed by another transaction by appending `FOR UPDATE` to the select statement. Optionally, you can specify a column list in parentheses after the FOR UPDATE clause.

The `LOCK TABLE table_name IN lock_mode` statement can be used to apply an explicit lock on an entire table. A range of row and table locks are supported.

No-Table Expression Select Syntax

To select a constant expression, that is, an expression that doesn't involve data from a database table or view, you can select from the DUAL table. For example, `SELECT SYSDATE FROM DUAL`.

Table Join Syntax

Oracle supports inner joins with the usual syntax:

```
SELECT * FROM a, b WHERE a.field = b.field
```

To write a query that performs an outer join of tables A and B and returns all rows from A, the Oracle outer join operator (+) must be applied to all column names of B that appear in the join condition. For example:

```
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, Oracle returns NULL for any select list expressions containing columns from the orders table.

Table and Column Names

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

The first character must be a letter, but the rest can be any combination of letters, numerals, dollar signs ($), pound signs (#), and underscores (_).

However, if an Oracle identifier is enclosed by double quotation marks ("), it can contain any combination of legal characters including spaces but excluding quotation marks.

Oracle converts all identifiers to upper-case unless enclosed in double quotation marks. National characters can also be used when identifiers are quoted.

Case Sensitivity of LIKE Operator

The Oracle LIKE operator is case sensitive.

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

Row ID

The Oracle ‘‘row id’’ pseudocolumn is called ROWID. Oracle ROWIDs look like `AAAAf-SAABAAAClaAAA`. It can be treated as a string and used to rapidly (re)select rows.

Automatic Key or Sequence Generation

Oracle supports ‘‘sequence generators’’. Any number of named sequence generators can be created in a database using the `CREATE SEQUENCE seq_name` SQL command. Each has pseudocolumns called NEXTVAL and CURRVAL. Typical usage:

```
INSERT INTO table (k, v) VALUES (seq_name.nextval, ?)
```

To get the value just inserted you can use

```
SELECT seq_name.currval FROM DUAL
```

Oracle does not support automatic key generation such as ‘‘auto increment’’ or ‘‘system generated’’ keys. However they can be emulated using triggers and sequence generators. For example:

```
CREATE TRIGGER trigger_name
  BEFORE INSERT ON table_name FOR EACH ROW
  DECLARE newid integer;
BEGIN
  IF (:NEW.key_field_name IS NULL)
  THEN
    SELECT sequence_name.NextVal INTO newid FROM DUAL;
    :NEW.key_field_name := newid;
  END IF;
END;
```

Automatic Row Numbering and Row Count Limiting

The ROWNUM pseudocolumn can be used to sequentially number selected rows (starting at 1). Sadly, however, Oracle’s ROWNUM has some frustrating limitations. Refer to the Oracle SQL documentation.

Parameter Binding

Parameter binding is directly suported by Oracle. Both the `?` and `:1` style of placeholders are supported.

The `bind_param()` method TYPE attribute can be used to indicate the type a parameter should be bound as. These SQL types are bound as VARCHAR2: SQL_NUMERIC, SQL_DECIMAL, SQL_INTEGER, SQL_SMALLINT, SQL_FLOAT, SQL_REAL, SQL_DOUBLE, and SQL_VARCHAR. Oracle will automatically convert from VARCHAR2 to the required type.

The SQL_CHAR type is bound as a CHAR thus enabling fixed-width blank padded comparison semantics.

The SQL_BINARY and SQL_VARBINARY types are bound as RAW. SQL_LONGVARBINARY is bound as LONG RAW and SQL_LONGVARCHAR as LONG.

Unsupported values of the TYPE attribute generate a warning.

Stored Procedures

Oracle stored procedures are implemented in the Oracle PL/SQL language*.

> * A procedural extension to SQL that supports variables, control flow, packages, exceptions, etc.

The `DBD::Oracle` module can be used to execute a block of PL/SQL code by starting it with a `BEGIN` and ending it with an `END;`. PL/SQL blocks are used to call stored procedures. Here's a simple example that calls a stored procedure called ''foo'' and passes it two parameters:

```
$sth = $dbh->prepare("BEGIN foo(:1, :2) END;");
$sth->execute("Baz", 24);
```

Here's a more complex example:

```
$sth = $dbh->prepare("BEGIN bar(:bang, :bong) END;");
$sth->bind_param(":bang", "Baz");
my $pong = 42;
$sth->bind_param_inout(":bong", \$bong, 100);
$sth->execute;
print "Pong is now: $pong\n";
```

Table Metadata

`DBD::Oracle` supports the `table_info()` method.

The ALL_TAB_COLUMNS view contains detailed information about all columns of all the tables in the database, one row per table. (Note that fields containing statistics derived from the actual data in the corresponding table are updated only when the `ANALYSE` command is executed for that table.)

The ALL_INDEXES view contains detailed information about all indexes in the database, one row per index. (Note that fields containing statistics derived from the actual data in the corresponding table are updated only when the `ANALYSE` command is executed for that table.) The ALL_IND_COLUMNS view contains information about the columns that make up each index.

Primary keys are implemented as unique indexes. See ALL_INDEXES above.

Driver-specific Attributes and Methods

DBD::Oracle has no significant driver-specific database or statement handle attributes.

The following private methods are supported:

plsql_errstr

```
$plsql_errstr = $dbh->func('plsql_errstr');
```

Returns error text from the USER_ERRORS table.

dbms_output_enable

```
$dbh->func('dbms_output_enable');
```

Enables the DBMS_OUTPUT package. The DBMS_OUTPUT package is typically used to receive trace and informational messages from stored procedures.

dbms_output_get

```
$msg  = $dbh->func('dbms_output_get');
@msgs = $dbh->func('dbms_output_get');
```

Gets a single line or all available lines using DBMS_OUTPUT.GET_LINE.

dbms_output_put

```
$msg  = $dbh->func('dbms_output_put', @msgs);
```

Puts messages using DBMS_OUTPUT.PUT_LINE.

Positioned updates and deletes

Oracle does not support positioned updates or deletes.

Differences from the DBI Specification

DBD::Oracle has no known significant differences in behavior from the current DBI specification.

URLs to More Database/Driver Specific Information

```
http://www.oracle.com
http://technet.oracle.com
```

Concurrent use of Multiple Handles

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

It also supports the preparation and execution of a new statement handle while still fetching data from another statment handle, provided it is associated with the same database handle.