1

DBD::Illustra

Version

Version 0.03.

Author and Contact Details

The driver author is Peter Haworth. He can be contacted via the *dbi-users* mailing list, although direct mail to *pmh@edison.ioppublishing.com* is likely to be read more quickly.

Supported Database Versions and Options

The `DBD::Illustra` module supports Illustra version 3.3.1. Other versions may be supported.

Connect Syntax

The `DBI->connect()` Data Source Name, or *DSN* must be in the following format:

```
dbi:Illustra:dbname
```

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

Numeric Data Handling

Illustra supports the following numeric data types:

```
INT1            - 1 byte signed integer
SMALLINT        - 2 byte signed integer
INTEGER         - 4 byte signed integer
NUMERIC         - fixed point number, precision=15, scale=0
NUMERIC(p)      - fixed point number, precision=p, scale=0
NUMERIC(p,s)    - fixed point number, precision=p, scale=s
DECIMAL         - NUMERIC
```

```
DECIMAL(p)       - NUMERIC(p)
DECIMAL(p,s)     - NUMERIC(p,s)
REAL             - single precision floating point number
DOUBLE PRECISION - double precision floating point number
```

There appears to be no limit on either the precision or scale of NUMERIC and DECIMAL types, except that neither may be negative and the scale may not be less than the precision.

`DBD::Illustra` always returns all numbers as strings, so it supports numbers outside the valid range for Perl numbers.

String Data Handling

Illustra supports the following string data types:

```
CHAR1           - single character
CHAR(size)      - fixed length string
CHARACTER(size) - fixed length string
VARCHAR(size)   - variable length string
TEXT            - variable length string, up to 8KB
LARGE_TEXT      - text, may be greater than 8KB
```

The 8KB mentioned above is the size of a page in Illustra. The combined size of all data in a single row may not exceed the page size. The LARGE_TEXT type uses a large object held outside the page to store the data.

CHAR and VARCHAR types don't appear to have a size limit other than the page size.

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

Illustra doesn't seem to notice or care when the 8th bit is set. The 8th bit is preserved as entered. Unicode UTF-8 strings can be stored but strings with embedded NUL characters can't.

Strings can be concatenated using the `||` operator.

Date Data Handling

Illustra supports the SQL2 datetime data types and intervals:

```
DATE            - single day resolution, from 1 AD to 9999 AD
TIME            - one second resolution, from 00:00:00 to 23:59:61
TIME(p)         - p digits of fractional seconds (max 8 digits)
TIMESTAMP       - microsecond resolution, from 1 AD to 9999 AD
TIMESTAMP(p)    - p digits of fractional seconds (max 8 digits)
```

The TIME, `TIME(p)`, `TIMESTAMP(p)` and `TIMESTAMP(p)` types all accept a ''WITH TIME ZONE'' modifier.

The default output formats for datetime types are:

```
DATE                      YYYY-MM-DD
TIME                      HH:MM:SS[.NNNNNNNN]
TIME WITH TIME ZONE       HH:MM:SS[.NNNNNNNN]+HH:MM
TIMESTAMP                 YYYY-MM-DD HH:MM:SS[.NNNNNNNN]
TIMESTAMP WITH TIME ZONE  YYYY-MM-DD HH:MM:SS[.NNNNNNNN]+HH:MM
```

The default output format cannot be changed. Note that the timezone offsets may be negative (`-HH:MM`) as well as positive.

Individual components of dates and times may be extracted using the EXTRACT function: `EXTRACT(field FROM date_value)`, where *field* may be one of YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, TIMEZONE_HOUR, or TIMEZONE_MINUTE.

The default input formats for all the above types are the same as the output format. Only those formats are recognized. Literal values must be quoted. All parts of dates and times must be specified. So two digit years, for example, are not permitted.

To get the correct date and time, the literal string `'now'` may be type cast to any of the above types using a ‘‘::type’’ notation:

```
'now'::date
'now'::time
'now'::timestamp
```

Functions are also available:

```
current_date
current_time
current_time(precision)
current_timestamp
current_timestamp(precision)
```

For DATE, the two methods are equivalent. However, for TIME and TIMESTAMP, `'now'` does not include the timezone, whereas the `current_*()` functions do. If an integer argument is passed to the `current_*()` functions, it indicates the number of digits of fractional seconds to display. The default for `current_time()` is 0, with a maximum of 8. The default for `current_timestamp()` is 6, with a maximum of 8.

Illustra supports the SQL2 datetime data types and intervals:

```
INTERVAL start[(p1[,p2])] [TO end[(p3)]]
```

The following interval qualifications are possible:

```
YEAR, YEAR TO MONTH,
MONTH,
DAY, DAY TO HOUR, DAY TO MINUTE, DAY TO SECOND,
HOUR, HOUR TO MINUTE, HOUR TO SECOND,
MINUTE, MINUTE TO SECOND,
```

```
SECOND
```

Where *p1* specifies the number of digits specified in the value, with a maximum of 10 and a default of 2. *p2* and *p3* specify the number of digits specified in fractional seconds, with a maximum of 8 and a default of 0.

Literal interval values may be specified using the following syntax:

```
INTERVAL value start[(p1,[p2])] [TO end[(p3)]]
```

e.g.:

```
INTERVAL '2' DAY
INTERVAL '02:03' HOUR TO MINUTE
INTERVAL '12345:67.891' MINUTE(5) TO SECOND(3)
```

A full range of operations can be performed on dates and intervals, *e.g.*, datetime-datetime=interval, datetime+interval=datetime, interval/number=interval.

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

```
'epoch'::ABSTIME::TIMESTAMP + INTERVAL seconds_since_epoch SECOND(10)
```

The following SQL expression can be used to go the other way, to convert a database date time value into a ''seconds since 1-jan-1970 GMT'' value:

```
(date_time_field - 'epoch'::ABSTIME::TIMESTAMP) INTERVAL SECOND(10)
```

No time zone adjustments are perfomed on values of TIME WITH TIME ZONE or TIMESTAMP WITH TIME ZONE data types. Their time zones are output as they were entered.

Normal TIME and TIMESTAMP values are treated as being in the current time zone of the current client. They are converted to GMT when stored in the database and converted back to the current clients time zone when fetched. For example:

```
CREATE TABLE tz_demo (loctime TIME, tztime TIME WITH TIME ZONE);

SET TIME ZONE INTERVAL '3:00' HOUR TO MINUTE;
INSERT INTO tz_demo VALUES('12:40:00','12:40:00+03:00');

SELECT * FROM tz_demo;
     loctime    tztime
     12:40:00   12:40:00+03:00



SET TIME ZONE INTERVAL '0:00' HOUR TO MINUTE;
SELECT * FROM tz_demo;
     loctime    tztime
     09:40:00   12:40:00+03:00
```

LONG/BLOB Data Handling

Illustra supports the following large object types:

```
LARGE_OBJECT    - Binary large object
LARGE_TEXT      - Large object masquerading as normal text
EXTERNAL_FILE   - Locator for external large binary file
```

The maximum size of large objects is not documented but is probably 4GB. None of the types are passed to and from the database as pairs of hex digits.

The *LongReadLen* and *LongTruncOk* attributes are untested as yet.

The `bind_param()` method is currently unsupported by `DBD::Illustra`, so these types must be input as literal string values. Fortunately, Illustra accepts very long SQL statements (over 100KB).

Direct support for large objects is currently under development. LARGE_TEXT fields may be treated just like any other fields, but LARGE_OBJECT fields and EXTERNAL_FILE fields currently need to be accessed with Illustra's `FileToLO()` and `LOToFile()` SQL functions.

Other Data Handling issues

The `DBD::Illustra` driver does not currently support the `type_info()` method. This is under development, but Illustra doesn't generally provide enough information to make this particularly useful.

Illustra automatically converts dates to strings, strings to dates, and strings to numbers, but numbers must be explicitly converted to strings:

```
INSERT INTO foo (num_field, str_field) VALUES ('42', 42::text)
```

Transactions, Isolation and Locking

Illustra and `DBD::Illustra` support transactions. The default transaction isolation level is Serializable.

Illustra supports all four standard isolation levels: Serializable, Repeatable Read, Read Commited, and Read Uncommited. 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.

The default locking behavior is for readers to block writers.

Rows returned by a SELECT statement can be locked to prevent them from being changed by another transaction, by including ''LOCK=EXCLUSIVE'' or ''LOCK=UPDATE'' in the optimizer hints for a given table:

```
SELECT * FROM xyz USING(LOCK=UPDATE) WHERE xid = 'abc'
```

Exclusive locks provide less concurrency, but update locks must be upgraded to exclusive when updates are required. This can cause deadlock if another transaction has acquired a read lock in the meantime.

There doesn't seem to be any way to explicitly lock a table other than by issuing a dummy SELECT statement with a LOCK=EXCLUSIVE hint as above.

No-Table Expression Select Syntax

To select a constant expression (one that doesn't involve data from a database table or view), you must use ''RETURN'' rather than ''SELECT''.

```
$dbh->prepare("RETURN 'now'::date");
```

Table Join Syntax

Illustra does not appear to support outer joins, but normal, inner joins are supported with the standard syntax.

Table and Column Names

The maximum size of table and column names appears to be 212 characters. The first character must be a letter, but the rest can be any combination of letters, numerals and underscores (_).

However, if an Illustra identifier is enclosed by double quotation marks ("), it can contain any combination of characters, including spaces. Double quotes in identifier names must be escaped with another double quote, e.g, `"double""quote"`.

Identifiers are stored as entered. All identifiers are case sensitive. National character set characters can be used if enclosed in double quotation marks.

Case Sensitivity of LIKE Operator

The Illustra 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 Illustra from making use of any index on the name column to speed up the query).

Row ID

The Illustra ‘‘row ID’’ pseudocolumn is called *oid*. Illustra *oid*’s look like ‘‘2d52.2001’’. *Oid*’s can be treated as a string and used to rapidly (re)select rows.

Automatic Key or Sequence Generation

Illustra does not support automatic key generation such as ‘‘auto increment’’ or ‘‘system generated’’ keys.

It also doesn’t offer sequence generators.

Automatic Row Numbering and Row Count Limiting

Illustra does not support any way of automatically numbering returned rows.

Parameter Binding

Parameter binding is not supported by Illustra. Emulation by the driver is under development.

Stored Procedures

The closest match to stored procedures that Illustra supports is user defined functions. These may be used just like system defined functions in SELECT or RETURN statements:

```
$sth = $dbh->prepare("RETURN foo('bar')");
$sth->execute;
@result = $sth->fetchrow_array;
```

Table Metadata

`DBD::Illustra` supports the `table_info()` method. However, since the information comes from Illustra’s ‘‘tables’’ table, `table_info()` will only return useful information if ‘‘tables’’ is readable. By default, only the DBA has access to ‘‘tables’’.

The ‘‘columns’’ table contains detailed information about all columns of all the table in the database, one row per table. However, the same access restrictions described above for the ‘‘tables’’ table will probably apply.

The ‘‘tables’’ and ‘‘columns’’ tables contain information about indexes as well as normal tables. Use `tables.table_kind='i'` for index tables (and `'t'` for normal tables).

The `tables.table_unique` field holds an array of column numbers. Each unique constraint, including the primary key, is held as a -1 terminated list of column numbers. The primary key is always the first list, even if it is not present.

```
(no constraints)                  ->  []
primary key(c1,c2)                ->  [1,2,-1]
primary key(c1,c2),unique(c1,c3)  ->  [1,2,-1,1,3,-1]
unique(c1,c3)                     ->  [-1,1,3,-1]
```

These arrays are returned as strings although you can retrieve individual elements instead using SQL functions.

Driver-specific Attributes and Methods

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

Positioned updates and deletes

Illustra supports positioned updates or deletes in cursors that have been explicitly created and opened FOR UPDATE. For example:

```
$dbh->do("DECLARE cur1 CURSOR FOR SELECT * FROM tab1 FOR UPDATE");
$dbh->do("OPEN cur1");
$sth = $dbh->prepare("FETCH NEXT FROM cur1");
while ($sth->execute && $row = $sth->fetchrow_arrayref) {
    $dbh->do("UPDATE tab1 SET col2='zyx' WHERE CURRENT OF cur1");
}
```

The statements must all occur in the same transaction:

Differences from the DBI Specification

DBD::Illustra does not currently support parameter binding, but does not have any other significant differences in behavior from the current DBI specification.

URLs to More Database/Driver Specific Information

The Illustra database is being absorbed into the Informix database after Informix bought the company. Since Illustra is no longer being developed or supported, it's a bit hard to find online information.

Concurrent use of Multiple Handles

DBD::Illustra supports 32 concurrent database connections to one or more databases.

It also supports the preparation of a new statement handle while still fetching data from another statement handle associated with the same database handle. However, only one statement handle per database handle may be executing concurrently. So you can prepare a new one but not execute it. This applies to all statements, whether DML or DDL.