1

DBD::Informix

Version

Version 0.60.

Author and Contact Details

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

Supported Database Versions and Options

The `DBD::Informix` module supports Informix OnLine and SE from version 5.00 onwards. There are some restrictions in the support for IUS (aka IDS/UDO). It uses Informix-ESQL/C, aka Informix ClientSDK. You must have a development licence for Informix-ESQL/C (or the C-code version of Informix-4GL) to be able to compile the `DBD::Informix` code.

Connect Syntax

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

```
dbi:Informix:I<xyz>
```

where *xyz* is any valid string that can be passed to the CONNECT statement (or to the DATABASE statement for version 5.x systems). The acceptable notations include:

```
dbase
dbase@server
@server
/path/to/dbase
//machine/path/to/dbase
```

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

Numeric Data Handling

Informix supports these numeric data types:

```
INTEGER           - signed 32-bit integer, excluding -2**31
SERIAL            - synonym for INTEGER as far as scale is concerned
SMALLINT          - signed 16-bit integer, excluding -2**15
FLOAT             - Native C 'double'
SMALLFLOAT        - Native C 'float'
REAL              - Synonym for SMALLFLOAT
DOUBLE PRECISION  - Synonym for FLOAT
DECIMAL(s)        - s-digit floating point number (non-ANSI databases)
DECIMAL(s)        - s-digit integer (MODE ANSI databases)
DECIMAL(s,p)      - s-digit fixed-point number with p decimal places
MONEY(s)          - s-digit fixed-point number with 2 decimal places
MONEY(s,p)        - s-digit fixed-point number with p decimal places
NUMERIC(s)        - synonym for DECIMAL(s)
NUMERIC(s,p)      - synonym for DECIMAL(s,p)
INT8              - signed 64-bit integer, excluding -2**63 (IDS/UDO)
SERIAL8           - synonym for INT8 as far as scale is concerned
```

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

String Data Handling

Informix supports the following string data types:

```
VARCHAR(size)
NVARCHAR(size)
CHAR
CHAR(size)
NCHAR
NCHAR(size)
CHARACTER VARYING(size)
NATIONAL CHARACTER VARYING(size)
NATIONAL CHARACTER(size)
CHARACTER(size)
VARCHAR(size,min)   -- and synonyms for this type
NVARCHAR(size,min)  -- and synonyms for this type
LVARCHAR            -- IDS/UDO only
```

Arguably, TEXT and BYTE blobs should also be listed here, as they are automatically converted from/to strings.

CHAR types have a limit of 32767 bytes in OnLine and IDS and a slightly smaller value (325xx) for SE. For VARCHAR types the limit is 255. LVARCHAR columns are limited to 2 KB; when used to transfer other data types, up to 32 KB. `DBD::Informix` 0.61 doesn't have fully operational LVARCHAR support.

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

Handling of national character sets depends on the database version (and is different for versions 5, for versions 6 and 7.1*x*, and for versions 7.2*x* and later). Details for version 8.*x* vary depending on *x*. It depends on the locale, determined by a wide range of standard (*e.g.*, LANG, LC_COLLATE) and non-standard (*e.g.*, DBNLS, CLIENT_LOCALE) environment variables. For details, read the relevant manual. Unicode is not currently directly supported by Informix (as of 1999-02-28).

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

Date Data Handling

There are two basic date/time handling types: DATE and DATETIME. DATE supports dates in the range 01/01/0001 through 31/12/9999. It is fairly flexible in its input and output formats. Internally, it is represented by the number of days since December 31 1899, so January 1 1900 was day 1. It does not understand the calendric gyrations of 1752, 1582-4, or the early parts of the first millenium, and imposes the calendar as of 1970-01-01 on these earlier times.

DATETIME has to be qualified by two components from the set:

```
YEAR MONTH DAY
HOUR MINUTE SECOND FRACTION FRACTION(n) for n = 1..5
```

These store a date using ISO 8601 format for the constants. For example, `DATE(‘‘29/02/2000’’)` is equivalent to:

```
DATETIME("2000-02-29") YEAR TO DAY,
```

and The Epoch for POSIX systems can be expressed as:

```
DATETIME(1970-01-01 00:00:00) YEAR TO SECOND
```

There is no direct support for timezones.

The default date time format depends on the environment locale settings and the version and the data type. The DATETIME types are rigidly ISO 8601 except for converting 1-digit or 2-digit years to a 4-digit equivalent, subject to version and environment.

Handling of two digit years depends on the version, the bugs fixed, and the environment. In general terms (for current software), if the environment variable DBCENTURY is unset or is set to `'R'`, then the current century is used. If DBCENTURY is `'F'`, the date will be in the future; if DBCENTURY is `'P'`, it will be in the past; if DBCENTURY is `'C'`, it will be the closest date (50 year window, based on current day, month and year, with the time of day untested).

The current datetime is returned by the `CURRENT` function, usually qualified as CURRENT YEAR TO SECOND.

Informix provides no simple way to input or output dates and times in other formats. Whole chapters can be written on this subject.

Informix supports a draft version of the SQL2 INTERVAL data type:

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

(Where `[]` indicates optional parts.)

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, FRACTION
```

p1 specifies the number of digits specified in the most significant unit of the value, with a maximum of 9 and a default of 2 (except YEAR that defaults to 4). *p2* specifies the number of digits in fractional seconds, with a maximum of 5 and a default of 3.

Literal interval values may be specified using the following syntax:

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

For example:

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

The expression ‘‘2 UNITS DAY’’ is equivalent to the first of these, and similar expressions can be used for any of the basic types.

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:

```
DATETIME(1970-01-01 00:00:00) YEAR TO SECOND + seconds_since_epoch UNITS SECOND
```

There is no simple expression for inline use that will do the reverse. Use a stored procedure; see the *comp.databases.informix* archives at DejaNews, or the Informix International Users Group (IIUG) web site at *http://www.iiug.org*.

Informix does not handle multiple time zones in a simple manner.

LONG/BLOB Data Handling

Informix supports the following large object types:

```
BYTE  - binary data     max 2GB
TEXT  - text data       max 2GB
BLOB  - binary data     max 2GB (maybe bigger); IDS/UDO only
CLOB  - character data  max 2GB (maybe bigger); IDS/UDO only
```

`DBD::Informix` does not currently have support for BLOB and CLOB data types, but does support the BYTE and TEXT types. None of the types are passed to and from the database as pairs of hex digits.

The DBI *LongReadLen* and *LongTruncOk* attributes are not implemented. If the data selected is a BYTE or TEXT type, then the data is stored in the relevant Perl variable, unconstrained by anything except memory up to a limit of 2GB.

The maximum length of `bind_param()` parameter value that can be used to insert BYTE or TEXT data is 2 GB. No specialized treatment is necessary for fetch or insert. UPDATE simply doesn't work.

The `bind_param()` method doesn't pay attention to the TYPE attribute. Instead, the string presented will be converted automatically to the required type. If it isn't a string type, it needs to be convertible by whichever bit of the system ends up doing the conversion. UPDATE cannot be used with these types in `DBD::Informix`; only version 7.30 IDS provides the data necessary to be able to handle blobs.

Other Data Handling issues

The `type_info()` method is not supported.

Non-BLOB types can be automatically converted to and from strings most of the time. Informix also supports automatic conversions between pure numeric data types whereever it is reasonable. Converting from DATETIME or INTERVAL to numeric data types is not automatic.

Transactions, Isolation and Locking

Informix databases can be created with or without transaction support.

Informix supports several transaction isolation levels: REPEATABLE READ, CURSOR STABILITY, COMMITTED READ, DIRTY READ. Refer to the Informix documentation for their exact meaning. Isolation levels only apply to OnLine and IDS and relatives; SE supports only a level somewhere in between COMMITTED READ and DIRTY READ.

The default isolation level depends on the type of database to which you're connected. You can use `SET ISOLATION TO x` to change the isolation level. If the database is unlogged, that is, it has no transaction support, you can't set the isolation level. In some more recent versions, you can also set a transaction to READ ONLY.

The default locking behaviour for reading and writing depends on the isolation level, the way the table was defined, and on whether the database has a transaction log or not.

Rows returned by a SELECT statement can be locked to prevent them 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. This has no effect on the query or locking strategy.

The `LOCK TABLE table_name IN lock_mode` statement can be used to apply an explicit lock on a table. The lock mode can be SHARED or EXCLUSIVE. There are constraints on when tables can be unlocked, and when locks can be applied. Row/Page locking occurs with cursors FOR UPDATE. In some types of database, some cursors are implicitly created FOR UPDATE.

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 write:

```
CREATE VIEW dual /* to mimic predefined DUAL table in Oracle */
AS (SELECT Tabid FROM "informix".SysTables WHERE TabID = 1);

SELECT 42 FROM dual;
```

Or:

```
SELECT 42 FROM "informix".SysTables WHERE TabID = 1;
```

Or, in most databases:

```
SELECT 42 FROM SysTables WHERE TabID = 1;
```

There are other equivalent formulas based on the system catalog, which are the only tables guaranteed to exist in the database.

Table Join Syntax

All Informix versions support the basic `WHERE a.field = b.field` style join notation. Support for SQL-92 join notation depends on DBMS version; most do not.

Outer joins are supported. The basic version is:

```
SELECT * FROM A, OUTER B WHERE a.col1 = b.col2
```

All rows from A will be selected. Where there is one or more rows in B matching the row in A according to the join condition, the corresponding rows will be returned. Where there is no matching row in B, NULL will be returned in the B-columns in the SELECT list. There are all sorts of other contortions, such as complications with criteria in the WHERE clause, or nested outer joins.

Table and Column Names

For most versions, the maximum size of a table name or column name is 18 characters, as required by SQL-86. For the latest versions (Centaur, provisionally 9.2 or 7.4), the answer will be 128, as required by SQL-92. Owner (schema) names can be 8 characters in the older versions and 32 in the versions with long table/column names.

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

If the DELIMIDENT environment variable is set, then table and column and owner names can be quoted inside double quotes, and any characters become valid. To embed a double-quote in the name, use two adjacent double-quotes, such as `"I said, “"Don’t!"""`. (Normally, Informix is very relaxed about treating double quotes and single quotes as equivalent, so often you could write `’I said, “Don’’t"’` as the equivalent of the previous example. With DELIMIDENT set, you have to be more careful.) Owner names are delimited identifiers and should be embedded in double quotes for maximum safety.

The case preserving and case sensitive behavior of table and column names depends on the environment and the quoting mechanisms used.

Support for using national character sets in names depends on the version and the environment (locale).

Case Sensitivity of LIKE Operator

The ANSI standard LIKE operator is case sensitive, so the Informix version is.

With version 7.30 and later versions, the `UPPER` or `LOWER` function can be used to force a case insensitive match, *e.g.*, `UPPER(name) LIKE ’TOM%’` although that does prevent the DBMS from making use of any index on the name column to speed up the query.

In earlier versions, it is possible to write excruciatingly slow stored procedures to achieve the same effect. In the pre-Centaur (9.2) releases of IUS and IDS/UDO, you can write a UDR (user defined routine) to provide the same functionality and performance as the 7.3 `LOWER` and `UPPER` functions.

Row ID

Most tables have a virtual ROWID column which can be selected. Fragmented tables do not have one unless it is specified in the `WITH ROWIDS` clause when the table is created or altered. In that case, it is a physical ROWID column which otherwise appears as a virtual column (meaning `SELECT *` does not select it).

As with any type except the blob types, a ROWID can be converted to a string and used as such. Note that ROWIDs need not be contiguous, nor start at either zero or one.

Automatic Key or Sequence Generation

The SERIAL and SERIAL8 datatypes are ‘‘auto incrementing’’ keys. If you insert a zero into these columns, the next previously unused key number is *unrollbackably* allocated to that row. Note that NULL can’t be used; you have to insert a zero. If you insert a non-zero value into the column, the specified value is used instead. Usually, there is a unique constraint on the column to prevent duplicate entries.

To get the value just inserted, you can use:

```
$sth->{ix_sqlerrd}[1]
```

Informix doesn’t support sequence generators directly, but you can create your own with stored procedures.

Automatic Row Numbering and Row Count Limiting

Informix does not support a way to automatically number returned rows.

Parameter Binding

Parameter binding is directly suported by Informix. Only the `?` style of place holder is supported. (The `:abc` and `:1` notations can occur in other contexts than placeholders, so supporting the notation requires a full SQL parser.)

The TYPE attribute to `bind_param()` is not currently supported, but some support is expected in a future release.

Stored Procedures

Some stored procedures can be used as functions in ordinary SQL:

```
SELECT proc1(Col1) FROM SomeTable WHERE Col2 = proc2(Col3);
```

All stored procedures can be executed via the EXECUTE PROCEDURE statement. If the procedure returns no values it can just be executed. If the procedure does return values, even single values via a `RETURN` statement, then it can be treated like a SELECT statement. So after calling `execute()` you can fetch results from the statement handle as if a select statement had been executed. For example:

```
$sth = $dbh->prepare("EXECUTE PROCEDURE CursoryProcedure(?,?)");
$sth->execute(1, 12);
$ref = $sth->fetchall_arrayref();
```

Table Metadata

The DBI `table_info()` method is not currently supported. The private `_tables()` method can be used to get a list of all tables or a subset.

Details of the columns of a table can be fetched using the private `_columns()` method.

The keys/indexes of a table be fetched by querying on the system catalog.

Further information about these and other issues can be found via the *comp.databases.informix* news group, and via the International Informix User Group (IIUG) at *http://www.iiug.org*.

Driver-specific Attributes and Methods

Refer to the `DBD::Informix` documentation for details of driver-specific database and statement handle attribues.

Private `_tables()` and `_columns()` methods give easy access to table and column details.

Positioned updates and deletes

Positioned updates and deletes are supported using the `WHERE CURRENT OF` syntax. For example:

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

Differences from the DBI Specification

Anywhere ‘‘perldoc `DBD::Informix`’’ says what goes on is different from the current DBI specification and the actual behavior does not match the DBI documentation—or anywhere where the `DBD::Informix` documentation says it does what the DBI documentation says it should do but it doesn’t.

If you change `AutoCommit` after preparing a statement, you will probably run into problems which you don’t expect. So don’t do that. See the `DBD::Informix` documentation for more details.

URLs to More Database/Driver Specific Information

```
http://www.informix.com
http://www.iiug.org
```

Concurrent use of Multiple Handles

If you’re using version 6.00 or later of ESQL/C, then the number of database handles is only limited by your imagination and the computer’s physical constraints. If you’re using 5.*x*, you’re stuck with one connection at a time.

`DBD::Informix` supports the preparation and execution of a new statement handle while still fetching data from another statment handle associated with the same database handle.

Other Significant Database or Driver Features

Temporary tables can be created during a database session that are automatically dropped at the end of that session if they have not already been dropped explicitly. Very handy.

The latest versions of Informix (IDS/UDO, IUS) support user defined routines and user defined types which can be implemented in the server in C or (shortly) Java.

The SQL-92 ‘‘CASE WHEN’’ syntax is supported by some versions of the Informix servers. That greatly simplifies some kinds of queries.