1

DBD::Ingres

Version

Version 0.16 and, where noted, the forthcoming 0.20 release.

Author and Contact Details

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

Supported Database Versions and Options

The `DBD::Ingres` module supports both Ingres 6.4 and OpenIngres (1.x & II).

Connect Syntax

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

```
dbi:Ingres:dbname
dbi:Ingres:vnode::dbname
dbi:Ingres:dbname;options
```

Where *options* are the SQL Option Flags as defined in the CA-OpenIngres System Reference Guide.

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

Numeric Data Handling

The database and driver supports 1 byte, 2 byte and 4 byte INTEGERS, 4 byte and 8 byte FLOATS, and a currency type. The database and the driver (from version 0.20) supports the DECIMAL-number type.

```
Type       Description          Range
----       -----------          -----
INTEGER1   1-byte integer       -128 to +127
SMALLINT   2-byte integer       -32,678 to +32,767
INTEGER    4-byte integer       -2,147,483,648 to +2,147,483,647
FLOAT4     4-byte floating      -1.0e+38 to 1.0e+38 (7 digits)
FLOAT      8-byte floating      -1.0e+38 to 1.0e+38 (16 digits)
MONEY      8-byte money         $-999,999,999,999.99 to $999,999,999,999.99
DECIMAL    fixed point numeric  Depends on precision (max 31) and scale.
```

DBD::Ingres always returns all numbers as Perl numbers—integers where possible, floating point otherwise. It is therefore possible that some precision may be lost when fetching DECIMAL types with a precision greater than Perl numbers (usually 16).

String Data Handling

Ingres and DBD::Ingres supports the following string data types:

```
VARCHAR(size)
CHAR(size)
TEXT(size)
C(size)
```

All string types have a limit of 2000 bytes. The CHAR, TEXT, and C types are fixed length and blank padded.

All string types can handle national character sets. The C type will only accept printing characters. CHAR and VARCHAR accept all character values including embedded nul characters ("\0"). Unicode is not formally supported yet.

Strings can be concatenated using the SQL + operator.

Date Data Handling

Ingres has just one date datatype: DATE. However, it can contain either an absolute date and time or a time interval. Dates and times are in second resolution between approx 1-JAN-1581 and 31-DEC-2382. Intervals are stored to a one second resolution.

Ingres supports a variety of date formats, depending on the setting of the environment variable II_DATE_FORMAT. The default output format is US: DD-MMM-YYYY HH:MM:SS.

Many input formats are allowed. For the default format the following are accepted: MM/DD/YYYY, DD-MMM-YYYY, MM-DD-YYYY, YYYY.MM.DD, YYYY_MM_DD, MMDDYY, MM-DD, and MM/DD.

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 following date-related functions are supported:

```
DATE(string)              - converts a string to a date
DATE_TRUNC(unit, date)    - date value truncated to the specified unit
DATE_PART(unit, date)     - integer containing the specified part
DATE_GMT(date)            - converts date to string "YYYY_MM_DD HH:MM:SS GMT".
INTERVAL(unit, interval)  - express interval as numeric count of units
```

The currect date and time is returned by the `DATE('now')` function. The current date is returned by `DATE('today')`.

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

```
DATE('01.01.1970 00:00 GMT')+DATE(CHAR(seconds_since_epoch)+' seconds')
```

And to do the reverse:

```
INT4(INTERVAL('seconds', DATE('now')-DATE('01.01.1970 00:00 GMT')))
```

A three letter time zone name (from a limited set) can be appended to a date. If no time zone name is given, then the current client time zone is assumed. All datetimes are stored in the database as GMT and are converted back to the local time of the client fetching the data. All date comparisions in the server and done in GMT.

LONG/BLOB Data Handling

Ingres supports these LONG types:

```
LONG VARCHAR  - Character data of variable length upto 2GB
LONG BYTE     - Raw binary data of variable length upto 2GB
```

However, the `DBD::Ingres` driver does not yet support these types.

Other Data Handling issues

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

Ingres supports automatic conversions between data types wherever it’s reasonable.

Transactions, Isolation and Locking

`DBD::Ingres` support transactions. The default transaction isolation level is ‘‘Serializable’’. OpenIngres II supports ‘‘Repeatable Read’’, ‘‘Read Commited’’, and ‘‘Serializable’’.

The reading of a record sets a read-lock preventing writers from changing that record and, depending on lock granularity, possibly other records. Other readers are not hindered in their reading. Writing a record sets a lock that prevent other writers from writing, and readers from reading.

The `SET LOCKMODE` statement allows you to change the locking granularity. It can be set to:

```
ROW    - lock only the affected rows (OpenIngres II only)
PAGE   - lock the page that contains the affected row
TABLE  - lock the entire table
```

With the statement `SET LOCKMODE SESSION WHERE READLOCK=NOLOCK` it is possible, but definitely *not* recommended, to set the isolation level to ‘‘Read Uncommited’’.

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 just select the expression. For example:

```
SELECT DATE('now')
```

Table Join Syntax

OpenIngres support outer joins in ANSI SQL-92 syntax. Ingres 6.4 does not support outer joins.

Table and Column Names

The names of indentifiers cannot exceed 32 characters. The first character must be a letter or an underscore (_), but the rest can be any combination of letters, numerals, dollar signs ($), pound signs (#), and at signs (@).

However, if an identifier is enclosed by double quotation marks ("), it can contain any combination of legal characters, including spaces but excluding quotation marks. This is not supported in Ingres 6.4.

Case significance is determined by the settings for the Ingres installation as set by the administrator when Ingres is installed.

National character sets can be used in identifiers, if enclosed in double quotation marks.

Case Sensitivity of LIKE Operator

The LIKE operator is case sensitive.

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

Row ID

The Ingres ‘‘row ID’’ pseudocolumn is called tid. It is an integer. It can be used without special handling. For example:

```
SELECT * FROM table WHERE tid=1029;
```

Automatic Key or Sequence Generation

OpenIngres II supports ‘‘logical key’’ columns. They are defined by using a special data type: `TABLE_KEY WITH SYSTEM MAINTAINED`. Ingres 6.4 required an extra-cost option to support that.

A column can be defined as either TABLE_KEY or OBJECT_KEY. Table_keys are unique in the table, whereas object_keys are unique in the entire database.

`DBD::Ingres` can’t currently find the value of the last automatic key inserted, though it may do so in the future if enough people ask nicely, or someone contributes the code.

Automatic Row Numbering and Row Count Limiting

There is no simple way to select a pseudocolumn that sequentially numbers the rows fetched by a select statement.

Parameter Binding

Parameter binding is directly suported by Ingres. Only the default `?` placeholder style is supported.

When using the `bind_param()` method, the common integer, float, and char types can be defined using the TYPE attribute. Unsupported values of the TYPE attribute generate a warning.

Stored Procedures

Calling a stored procedure is done by the ‘‘execute procedure’’ statement. For example:

```
$dbh->do("execute procedure my_proc(param1='value')");
```

It is not yet possible to get results.

Table Metadata

`DBD::Ingres` version 0.20 supports the `table_info()` method.

The IICOLUMNS catalog contains information about all columns of a table.

The IIINDEXES catalog contains detailed information about all indexes in the database, one row per index. The IIINDEX_COLUMNS catalog contains information about the columns that make up each index.

Primary keys are indicated in the key_sequence field of the IICOLUMNS catalog.

Driver-specific Attributes and Methods

`DBD::Ingres` has no driver-specific database handle attributes. However, it does support a number of statement handle attributes. Each returns a reference to an array of values, one for each column of the select results.

ing_type

`'i'` for integer columns, `'f'` for float and `'s'` for strings

ing_ingtype

The numeric Ingres type of the columns

ing_length

The Ingres length of the columns (as used in the database)

`DBD::Ingres` supports just one private method:

get_dbevent()

This private method calls `GET DBEVENT` and `INQUIRE_INGRES` to fetch a pending database event. If called without an argument, a blocking `GET DBEVENT WITH WAIT` is called. A numeric argument results in a call to `GET DBEVENT WITH WAIT= :seconds`.

Positioned updates and deletes

Positioned updates and deletes are supported in `DBD::Ingres` version 0.20 using the `WHERE CURRENT OF` syntax. For example:

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

The CursorName is automatically defined by `DBD::Ingres` for each prepared statement.

Differences from the DBI Specification

Prepared statements do not work across transactions because commit/rollback close/invalidate are all prepared statements. Work is underway to fix this.

URLs to More Database/Driver Specific Information

```
http://www.cai.com/products/ingres.htm
```

Concurrent use of Multiple Handles

`DBD::Ingres` 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 associated with the same database handle.