1

DBD::DB2

Version

Version 0.71.

Author and Contact Details

Support for the `DBD::DB2` driver is provided by IBM through its service agreements for DB2 UDB. Any comments, suggestions, or enhancement requests can be sent via e-mail to *db2perl@ca.ibm.com*. Please see the web site at

```
http://www.software.ibm.com/data/db2/perl
```

for more information.

Supported Database Versions and Options

The `DBD::DB2` driver supports DB2 UDB V5.2 and later.

Connect Syntax

The `DBI->connect()` Data Source Name, or *DSN*, is specified as follows:

```
dbi:DB2:database_name
```

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

Numeric Data Handling

DB2 UDB supports the following numeric data types:

```
SMALLINT
INTEGER
BIGINT
REAL
DOUBLE
FLOAT
DECIMAL or NUMERIC
```

A SMALLINT is a two byte integer than can range from -32768 to +32767. The maximum precision is 5. Scale is not applicable.

An INTEGER is a four byte integer that can range from -2147483648 to +2147483647. The maximum precision is 10. Scale is not applicable.

A BIGINT is an eight byte integer that can range from -9223372036854775808 to +9223372036854775807. The maximum precision is 19. Scale is not applicable.

A REAL is a 32 bit approximation of a real number. The number can be 0 or can range from -3.402E+38 to -1.175E-37, or from +1.175E-37 to +3.402E+38. The maximum precision is 7. Scale is not applicable.

A DOUBLE or FLOAT is a 64 bit approximation of a real number. The number can be 0 or can range from -1.79769E+308 to -2.225E-307, or from 2.225E-307 to 1.79769E+308. The maximum precision is 15. Scale is not applicable.

A DECIMAL or NUMERIC value is a packed decimal number with an implicit decimal point that can range from -10**31+1 to +10**31-1. The maximum precision is 31 digits. The scale cannot be negative or greater than the precision.

Notice that DB2 supports numbers outside the typical valid range for Perl numbers. This isn't a major problem because `DBD::DB2` always returns all numbers as strings.

String Data Handling

DB2 UDB supports the following string data types:

```
CHAR
CHAR FOR BIT DATA
VARCHAR
VARCHAR FOR BIT DATA
GRAPHIC
VARGRAPHIC
```

CHAR is a fixed-length character string that can be up to 254 bytes long. VARCHAR is a varying-length character string that can be up to 32672 bytes. The FOR BIT DATA variants are used for data not associated with a particular coded character set.

GRAPHIC is a fixed-length string of double-byte characters that can be up to 127 characters long.

VARGRAPHIC is a varying-length string of double-byte characters that can be up to 16336 characters long.

The CHAR and GRAPHIC types are fixed-length strings, padded with blanks.

For DB2 UDB, CHAR fields can be in mixed codesets (national character sets). The non-ASCII characters are handled according to the mixed code page definition. For example, Shift-JIS characters in the range 0x81 to 0x9F and 0xE0 to 0xFC are DBCS introducer bytes, and characters in the range 0xA0 to 0xDF are single-byte Katakana characters. Blank padding for CHAR fields is always with ASCII blank (single-byte blank). For UTF-8, characters with the sign bit set are interpreted according to the UTF-8 definition.

GRAPHIC data types are stored as pure double-byte in the default code page of the database, or in UCS-2 in the case of a Unicode database. Blank padding for GRAPHIC fields is always with the DBCS blank of the corresponding code page, or with the UCS-2 blank (U+0020) in the case of a Unicode database.

Code page conversions between the client code page and the database code page are automatically performed by DB2 UDB.

Unicode support is provided with DB2 UDB Version 5 + FixPak 7 (DB2 UDB V5.2 is actually DB2 UDB V5 + FixPak 6). In a Unicode database, CHAR data types are stored in UTF-8 format and GRAPHIC data types are stored in UCS-2 format.

With DB2 UDB Version 6.1, the `VARCHAR()` function has been extended to convert graphic string data types to a VARCHAR, with the exception of LONG VARGRAPHIC and DBCLOB. This function is valid for UCS-2 databases only. For non-Unicode databases, this is not allowed.

All character types can store strings with embedded nul (`"\0"`) bytes.

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

Date Data Handling

DB2 UDB supports the following date, time, and date/time data types:

```
DATE
TIME
TIMESTAMP
```

DATE is a three-part value consisting of year, month, and day. The range of the year part is 0001 to 9999. Two digit years cannot be used with DB2 UDB. Years must be specified with all four digits.

TIME is a three-part value consisting of hour, minute, and second designates a time of day under a 24-hour clock.

TIMESTAMP is a seven-part value, consisting of year, month, day, hour, minute, second, and microsecond, that designates a date and time as defined above, except that the time includes a fractional specification of microseconds. If you specify a TIMESTAMP value without a time component, the default time is 00:00:00 (midnight).

The current date, time, and date/time can be retrieved using the CURRENT DATE, CURRENT TIME, and CURRENT TIMESTAMP special registers.

DB2 UDB supports the following date, time, and date/time formats:

```
ISO   (International Standards Organization)
USA   (IBM USA standard)
EUR   (IBM European standard)
JIS   (Japanese Industrial Standard Christian era)
LOC   (site-defined, depends on database country code)
```

You can input date and date/time values in any supported format. For example:

```
create table datetest(dt date);
insert into datetest('1991-10-27');
insert into datetest('10/27/1991');
```

The default output format for DATE, TIME, and TIMESTAMP is that format which is associated with the country code of the database (LOC format above). You can use the `CHAR` function and specify an alternate format.

Datetime values can be incremented, decremented, and subtracted. DB2 UDB provides a wide range of date functions including `DAY`, `DAYOFWEEK`, `DAYOFYEAR`, `MONTHNAME`, and `TIMESTAMPDIFF`. See the DB2 UDB documentation for additional functions.

The following SQL expression can be used to convert an integer ‘‘seconds since 1-jan-1970’’ value to the corresponding database date time (local time not GMT):

```
TIMESTAMP('1970-01-01','00:00') + seconds_since_epoch
```

There is no simple expression that will do the reverse. Subtracting `timestamp('1970-01-01','00:00')` from another timestamp gives a timestamp duration which is a `DECIMAL(20,6)` value with format yyyymmddhhmmss.zzzzzz.

DB2 does no automatic time zone adjustments.

LONG/BLOB Data Handling

DB2 UDB supports the following LONG/BLOB data types:

```
BLOB
CLOB
DBCLOB
LONG VARCHAR
LONG VARCHAR FOR BIT DATA
```

```
LONG VARGRAPHIC
```

BLOB (Binary Large Object) is a varying-length string measured in bytes that can be up to 2 gigabytes long. A BLOB is primarily intended to hold non-traditional data such as pictures, voice, and mixed media. BLOBs are not associated with a coded character set (similar to FOR BIT DATA character strings — see below).

CLOB (Character Large Object) is a varying-length string measured in bytes that can be up to 2 gigabytes long. A CLOB is used to store large character-based data.

DBCLOB (Double-Byte Character Large Object) is a varying-length string of double-byte characters that can be up to 1,073,741,823 characters long. A DBCLOB is used to store large DBCS character based data.

LONG VARCHAR is a varying-length character string that can be up to 32,700 bytes long. LONG VARCHAR FOR BIT DATA is used for data not associated with a coded character set.

LONG VARGRAPHIC is a varying-length string of double-byte characters that can be up to 16,350 characters long.

None of these types need to be passed to and from the database as pairs of hex digits.

Sadly the `DBD::DB2` driver does not yet support the *LongReadLen* and *LongTruncOk* attributes. Values of any length can be inserted and fetched up to the maximum size of the corresponding data type although system resources may be a constraint.

The `DBD::DB2` driver is unusual in that it requires heavy use of bind parameter attributes both for ordinary types and for LONG/BLOB types. (See Parameter Binding section for discussion on attribute hashes.) For example, here's an attribute hash for a CLOB, which will have a maximum length of 100K in this particular application:

```
$attrib_clob = {
  ParamT => SQL_PARAM_INPUT,
  Ctype  => SQL_C_CHAR,
  Stype  => SQL_CLOB,
  Prec   => 100000
  Scale  => 0,
};
```

Other Data Handling issues

The `DBD::DB2` driver does not yet support the `type_info` method.

DB2 does not automatically convert strings to numbers or numbers to strings.

Transactions, Isolation and Locking

DB2 UDB supports transactions and four transaction isolation levels: Repeatable Read, Read Stability, Cursor Stability, Uncommited Read. The default transaction isolation level is Cursor Stability.

For the `DBD::DB2` driver, the isolation level can be changed by setting the TXNISOLATION keyword in the *db2cli.ini* file to the desired value. This keyword is set in a database-specific section, meaning that it will affect all applications that connect to that particular database. There is no way to change the isolation level from SQL.

The default behaviour for reading and writing is based on the isolation level. Rows returned by a SELECT statement can be explicitly locked by appending ''FOR UPDATE'' and a list of field names to the SELECT statement. For example:

```
SELECT colname1, colname2
FROM tablename
WHERE colname1 = 'testvalue'
FOR UPDATE OF colname1, colname2
```

The ''LOCK TABLE table_name IN lock_mode'' statement can be used to apply an explicit lock on an entire table.

No-Table Expression Select Syntax

There is no current method for selecting a constant expression using the `DBD::DB2` driver. That is, an expression that doesn't involve data from a database table or view.

Table Join Syntax

You can perform an equi-join, or inner join, using the standard `WHERE a.field = b.field` syntax. You can also use the following syntax:

```
SELECT tablea.col1, tableb.col1
FROM tablea INNER JOIN tableb
ON tableb.name = tablea.name
```

DB2 UDB supports left outer joins, right outer joins, and full outer joins. For example, to perform a left outer join, you can use the following statement:

```
SELECT tablea.col1, tablea.col2, tableb.col1, tableb.col2
FROM tablea LEFT OUTER JOIN tableb
ON tableb.name = tablea.name
```

Changing ‘‘LEFT’’ to ‘‘RIGHT’’ or ‘‘FULL’’ gives you the other forms of outer join.

Table and Column Names

In DB2 UDB Version 5.2, the maximum length of table names and column names is 18. In DB2 UDB Version 6.1, the maximum length of table names will be increased to 128 and the maximum length of column names will be increased to 30.

The first character must be a letter, but the rest can be any combination of uppercase letters, digits, and underscores.

Table and field names can be delimited by double quotation marks (") and can contain the same characters as described above plus lowercase letters.

Table and column names are stored as upper case in the catalogs unless delimited. Delimited identifiers preserve the case. Two consecutive quotation marks are used to represent one quotation mark within the delimited identifier. This is handy to know if you’re passing an SQL statement in quotation marks and you have a delimited identifier.

National characters can be used in table and column names.

Case Sensitivity of LIKE Operator

The DB2 UDB LIKE operator is case sensitive.

The `UCASE` function can be used to force a case insensitive match, *e.g.*, `UCASE(name) LIKE 'TOM%'` although that does prevent DB2 from making use of any index on the name column to speed up the query.

Row ID

DB2 UDB does not support a ‘‘table row ID’’ pseudocolumn.

Automatic Key or Sequence Generation

The `GENERATE_UNIQUE` function can be used to provide unique values (keys) in a table. For example:

```
CREATE TABLE EMP_UPDATE (
  UNIQUE_ID CHAR(13) FOR BIT DATA,  -- note the "FOR BIT DATA"
  EMPNO CHAR(6),
  TEXT VARCHAR(1000)
)

INSERT INTO EMP_UPDATE VALUES
  (GENERATE_UNIQUE(), '000020', 'Update entry...'),
  (GENERATE_UNIQUE(), '000050', 'Update entry...')
```

Sadly, DB2 does not provide any way to discover the most recent value generated by `GENERATE_UNIQUE`.

DB2 UDB does not support named sequence generators.

Automatic Row Numbering and Row Count Limiting

There is no pseudocolumn that can be used to sequentially number the rows fetched by a select statement. However, you can number the rows of a result set using the OLAP function `ROWNUMBER`. For example:

```
SELECT ROWNUMBER() OVER (order by lastname) AS number, lastname, salary
FROM employee ORDER BY number;
```

This returns the rows of the employee table with numbers assigned according to the ascending order of last names, ordered by the row numbers.

A cursor can be declared with the `FETCH FIRST n ROWS ONLY` clause to limit the number of rows returned.

Parameter Binding

Parameter binding is directly suported by DB2 UDB. Only the standard `?` style of placeholders is supported.

The `DBD::DB2` driver does not support the TYPE attribute exactly as described in the DBI documentation. Attribute hashes are used to pass type information to the `bind_param()` method. An attribute hash is simply a collection of information about a particular type of data. Each attribute can be determined at compile time, created at run time, or modified at run time. (See *DB2.pm* for a list of predefined attribute hashes).

The following is an example of how a complete new attribute hash can be created:

```
$attrib_char = {
  ParamT => SQL_PARAM_INPUT,
  Ctype  => SQL_C_CHAR,
  Stype  => SQL_CHAR,
  Prec   => 254,
  Scale  => 0,
};
```

Stored Procedures

Stored procedures are invoked by using the following SQL syntax:

```
CALL procedure-name(argument, ...)
```

Table Metadata

DBD::DB2 does not yet support the table_info() method.

The SYSCAT.COLUMNS view contains one row for each column that is defined for all tables and views in the database.

The SYSCAT.INDEXES view contains one row for each index that is defined for all tables in a database. Primary keys are implemented as unique indexes.

Driver-specific Attributes and Methods

DBD::DB2 has no driver-specific attributes or methods.

Positioned updates and deletes

DB2 UDB supports positioned updates and deletes. Since specific testing of this functionality has not been done with the DBD::DB2 driver, it's not officially supported; however, no problems are anticipated.

The syntax for a positioned update is as follows. DELETE has a similar syntax.

```
UPDATE ... WHERE CURRENT OF $cursor_name
```

Differences from the DBI Specification

The only significant difference in behaviour from the current DBI specification is the way in which data types are specified in the bind_param() method. Please see the information earlier in the document about using the bind_param() method with the DBD::DB2 driver.

URLs to More Database/Driver Specific Information

```
http://www.software.ibm.com/data/db2/perl
http://www.software.ibm.com/data/db2
http://www.software.ibm.com/data/db2/library
http://www.software.ibm.com/data/db2/udb/ad
```

Concurrent use of Multiple Handles

DBD::DB2 supports concurrent database connections to one or more databases.

DBD::DB2 supports the preparation and execution of a new statement handle while still fetching data from another statment handle associated with the same database handle.