1

DBD::Pg

Version

Version 0.91.

Author and Contact Details

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

Supported Database Versions and Options

The DBD-Pg-0.91 module supports Postgresql 6.4.

Connect Syntax

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

```
dbi:Pg:dbname=$dbname
dbi:Pg:dbname=$dbname;host=$host;port=$port;options=$options;tty=$tty
```

All parameters, including the userid and password parameter of the connect command, have a hard-coded default which can be overridden by setting appropriate environment variables:

Parameter	Environment Variable	Default
dbname	PGDATABASE	current userid
host	PGHOST	localhost
port	PGPORT	5432
options	PGOPTIONS	""
tty	PGTTY	""
username	PGUSER	current userid
password	PGPASSWORD	""

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

Numeric Data Handling

Postgresql supports the following numeric types:

Postgresql	Range
int2	-32768 to +32767
int4	-2147483648 to +2147483647
float4	6 decimal places
float8	15 decimal places

Some platforms also support the int8 type. `DBD::Pg` always returns all numbers as strings.

String Data Handling

Postgresql supports the following string data types:

CHAR	single character
CHAR(size)	fixed length blank-padded
VARCHAR(size)	variable length with limit
TEXT	variable length

All string data types have a limit of 4096 bytes. The CHAR type is fixed length and blank padded.

There is no special handling for data with the 8th bit set. They are stored unchanged in the database. None of the character types can store embedded nulls and Unicode is not formally supported.

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

Date Data Handling

Postgresql supports the following date time data types:

Type	Storage	Recommendation	Description
abstime	4 bytes	original date and time	limited range
date	4 bytes	SQL92 type	wide range
datetime	8 bytes	best general date and time	wide range, high precision
interval	12 bytes	SQL92 type	equivalent to timespan
reltime	4 bytes	original time interval	limited range, low precision
time	4 bytes	SQL92 type	wide range
timespan	12 bytes	best general time interval	wide range, high precision
timestamp	4 bytes	SQL92 type	limited range

Data Type	Range		Resolution
abstime	1901-12-14	2038-01-19	1 sec
timestamp	1901-12-14	2038-01-19	1 sec

```
reltime      -68 years         +68 years         1 sec
tinterval    -178000000 years  +178000000 years  1 microsec
timespan     -178000000 years  178000000 years   1 microsec
date         4713 BC             32767 AD        1 day
datetime     4713 BC           1465001 AD        1 microsec
time         00:00:00:00       23:59:59:99       1 microsec
```

Postgresql supports a range of date formats:

```
Name           Example
-----------    ----------------------
ISO            1997-12-17 0:37:16-08
SQL            12/17/1997 07:37:16.00 PST
Postgres       Wed Dec 17 07:37:16 1997 PST
European       17/12/1997 15:37:16.00 MET
NonEuropean    12/17/1997 15:37:16.00 MET
US             12/17/1997 07:37:16.00 MET
```

The default output format does not depend on the client/server locale. It depends on, in increasing priority: the PGDATESTYLE environment variable at the server, the PGDATESTYLE environment variable at the client, and the `SET DATESTYLE` SQL command.

All of the formats described above can be used for input. A great many others can also be used. There is no specific default input format. If the format of a date input is ambiguous then the current DATESTYLE is used to help disambiguate.

If you specify a date/time value without a time component, the default time is 00:00:00 (midnight). To specify a date/time value without a date is not allowed. If a date with a two digit year is input then if the year was less than 70, add 2000; otherwise, add 1900.

The currect date/time is returned by the keyword `'now'` or `'current'`, which has to be casted to a valid data type. For example:

```
SELECT 'now'::datetime
```

Postgresql supports a range of date time functions for converting between types, extracting parts of a date time value, truncating to a given unit, etc. The usual arithmetic can be performed on date and interval values, e.g., date-date=interval, etc.

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(unixtime_field)
```

and to do the reverse:

```
DATE_PART('epoch', datetime_field)
```

The server stores all dates internally in GMT. Times are converted to local time on the database server before being sent to the client frontend, hence by default are in the server time zone.

The TZ environment variable is used by the server as default time zone. The PGTZ environment variable on the client side is used to send the time zone information to the backend upon connection. The SQL `SET TIME ZONE` command can set the time zone for the current session.

LONG/BLOB Data Handling

Postgresql handles BLOBS using a so called ''large objects'' type. The handling of this type differs from all other data types. The data are broken into chunks, which are stored in tuples in the database. Access to large objects is given by an interface which is modelled closely after the UNIX file system. The maximum size is limited by the file size of the operating system.

If you just select the field, you get a ''large object identifier'' and not the data itself. The *LongReadLen* and *LongTruncOk* attributes are not implemented because they don't make sense in this case. The only method implemented by the driver is the undocumented DBI method `blob_read()`.

Other Data Handling issues

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

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

Transactions, Isolation and Locking

Postgresql supports transactions. The current default isolation transaction level is ''Serializable'' and is currently implemented using table level locks. Both may change. No other isolation levels for transactions are supported.

With AutoCommit on, a query never places a lock on a table. Readers never block writers and writers never block readers. This behavior changes whenever a transaction is started (AutoCommit off). Then a query induces a shared lock on a table and blocks anyone else until the transaction has been finished.

The `LOCK TABLE table_name` statement can be used to apply an explicit lock on a table. This only works inside a transaction (AutoCommit off).

To ensure that a table being selected does not change before you make an update later in the transaction, you must explicitly lock it with a `LOCK TABLE` statement before executing the select.

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, just omit the ‘‘from’’ clause. Here’s an example that selects the current time as a datetime:

```
SELECT ’now’::datetime;
```

Table Join Syntax

Outer joins are not supported. Inner joins use the traditional syntax.

Table and Column Names

The max size of table and column names cannot exceed 31 charaters in length. Only alphanumeric characters can be used; the first character must be a letter.

If an identifier is enclosed by double quotation marks ("), it can contain any combination of characters except double quotation marks.

Postgresql converts all identifiers to lower-case unless enclosed in double quotation marks. National character set characters can be used, if enclosed in quotation marks.

Case Sensitivity of LIKE Operator

Postgresql has the following string matching operators:

```
Glyph Description                              Example
----- ---------------------------------------- ----------------------------
~~    Same as SQL "LIKE" operator              ’scrappy,marc’ ~~ ’%scrappy%’
!~~   Same as SQL "NOT LIKE" operator          ’bruce’ !~~ ’%al%’
~     Match (regex), case sensitive            ’thomas’ ~ ’.*thomas.*’
~*    Match (regex), case insensitive          ’thomas’ ~* ’.*Thomas.*’
!~    Does not match (regex), case sensitive   ’thomas’ !~ ’.*Thomas.*’
!~*   Does not match (regex), case insensitive ’thomas’ !~ ’.*vadim.*’
```

Row ID

The Postgresql ‘‘row id’’ pseudocolumn is called *oid*, object identifier. It can be treated as a string and used to rapidly (re)select rows.

Automatic Key or Sequence Generation

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

However, Postgresql does support ‘‘sequence generators’’. Any number of named sequence generators can be created in a database. Sequences are used via functions called `NEXTVAL` and `CURRVAL`. Typical usage:

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

To get the value just inserted, you can use the corresponding `currval()` SQL function in the same session, or

```
SELECT last_value FROM seq_name
```

Automatic Row Numbering and Row Count Limiting

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

Parameter Binding

Parameter binding is emulated by the driver. Both the `?` and `:1` style of placeholders are supported.

The TYPE attribute of the `bind_param()` method may be used to influence how parameters are treated. These SQL types are bound as VARCHAR: SQL_NUMERIC, SQL_DECIMAL, SQL_INTEGER, SQL_SMALLINT, SQL_FLOAT, SQL_REAL, SQL_DOUBLE, SQL_VARCHAR.

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

Unsupported values of the TYPE attribute generate a warning.

Stored Procedures

`DBD::Pg` does not support stored procedures.

Table Metadata

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

The *pg_attribute* table contains detailed information about all columns of all the tables in the database, one row per table.

The *pg_index* table contains detailed information about all indexes in the database, one row per index.

Primary keys are implemented as unique indexes. See *pg_index* above.

Driver-specific Attributes and Methods

There are no significant `DBD::Pg` driver-specific database handle attributes.

`DBD::Pg` has the following driver-specific statement handle attributes:

pg_size

Returns a reference to an array of integer values for each column. The integer shows the storage (not display) size of the column in bytes. Variable length columns are indicated by -1.

pg_type

Returns a reference to an array of strings for each column. The string shows the name of the data type.

pg_oid_status

Returns the OID of the last INSERT command.

pg_cmd_status

Returns the name of the last command type. Possible types are: INSERT, DELETE, UPDATE, SELECT.

`DBD::Pg` has no private methods.

Positioned updates and deletes

Postgresql does not support positioned updates or deletes.

Differences from the DBI Specification

`DBD::Pg` has no significant differences in behavior from the current DBI specification.

Note that `DBD::Pg` does not fully parse the statement until it's executed. Thus attributes like *$sth->{NUM_OF_FIELDS}* are not available until after `$sth->execute` has been called. This is valid behaviour but is important to note when porting applications originally written for other drivers.

URLs to More Database/Driver Specific Information

```
http://www.postgresql.org
```

Concurrent use of Multiple Handles

`DBD::Pg` 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 statement handle, provided it is associated with the same database handle.

Other Significant Database or Driver Features

Postgres offers substantial additional power by incorporating the following four additional basic concepts in such a way that users can easily extend the system: classes, inheritance, types, and functions.

Other features provide additional power and flexibility: constraints, triggers, rules, transaction integrity, procedural languages, and large objects.

It’s also free Open Source Software with an active community of developers.