NAME

DBI::DBD::SqlEngine - Base class for DBI drivers without their own SQL engine

SYNOPSIS

```
package DBD::myDriver;

use base qw(DBI::DBD::SqlEngine);

sub driver
{
...
my $drh = $proto->SUPER::driver($attr);
...
return $drh->{class};
}

package DBD::myDriver::dr;

@ISA = qw(DBI::DBD::SqlEngine::dr);

sub data_sources { ... }
...

package DBD::myDriver::db;

@ISA = qw(DBI::DBD::SqlEngine::db);

sub init_valid_attributes { ... }
sub init_default_attributes { ... }
sub set_versions { ... }
sub validate_STORE_attr { my ($dbh, $attrib, $value) = @_; ... }
sub validate_FETCH_attr { my ($dbh, $attrib) = @_; ... }
sub get_myd_versions { ... }
sub get_avail_tables { ... }

package DBD::myDriver::st;

@ISA = qw(DBI::DBD::SqlEngine::st);

sub FETCH { ... }
sub STORE { ... }

package DBD::myDriver::Statement;

@ISA = qw(DBI::DBD::SqlEngine::Statement);

sub open_table { ... }

package DBD::myDriver::Table;

@ISA = qw(DBI::DBD::SqlEngine::Table);

sub new { ... }
```

DESCRIPTION

DBI::DBD::SqlEngine abstracts the usage of SQL engines from the DBD. DBD authors can concentrate on the data retrieval they want to provide.

It is strongly recommended that you read DBD::File::Developers and DBD::File::Roadmap, because many of the DBD::File API is provided by DBI::DBD::SqlEngine.

Currently the API of DBI::DBD::SqlEngine is experimental and will likely change in the near future to provide the table meta data basics like DBD::File.

DBI::DBD::SqlEngine expects that any driver in inheritance chain has a DBI prefix.

Metadata

The following attributes are handled by DBI itself and not by DBI::DBD::SqlEngine, thus they all work as expected:

```
Active
ActiveKids
CachedKids
CompatMode (Not used)
InactiveDestroy
AutoInactiveDestroy
Kids
PrintError
RaiseError
Warn (Not used)
```

The following DBI attributes are handled by DBI::DBD::SqlEngine:

AutoCommit

Always on.

ChopBlanks

Works.

NUM_OF_FIELDS

Valid after `$sth->execute`.

NUM_OF_PARAMS

Valid after `$sth->prepare`.

NAME

Valid after `$sth->execute`; probably undef for Non-Select statements.

NULLABLE

Not really working, always returns an array ref of ones, as DBD::CSV does not verify input data. Valid after `$sth->execute`; undef for non-select statements.

The following DBI attributes and methods are not supported:

bind_param_inout
CursorName
LongReadLen
LongTruncOk

DBI::DBD::SqlEngine specific attributes

In addition to the DBI attributes, you can use the following dbh attributes:

sql_engine_version

Contains the module version of this driver (**readonly**)

sql_nano_version

Contains the module version of DBI::SQL::Nano (**readonly**)

sql_statement_version

Contains the module version of SQL::Statement, if available (**readonly**)

sql_handler

Contains the SQL Statement engine, either DBI::SQL::Nano or SQL::Statement (**readonly**).

sql_parser_object

Contains an instantiated instance of SQL::Parser (**readonly**). This is filled when used first time (only when used with SQL::Statement).

sql_sponge_driver

Contains an internally used DBD::Sponge handle (**readonly**).

sql_valid_attrs

Contains the list of valid attributes for each DBI::DBD::SqlEngine based driver (**readonly**).

sql_readonly_attrs

Contains the list of those attributes which are readonly (**readonly**).

sql_identifier_case

Contains how DBI::DBD::SqlEngine deals with non-quoted SQL identifiers:

```
* SQL_IC_UPPER (1) means all identifiers are internally converted
into upper-cased pendants
* SQL_IC_LOWER (2) means all identifiers are internally converted
into lower-cased pendants
* SQL_IC_MIXED (4) means all identifiers are taken as they are
```

These conversions happen if (and only if) no existing identifier matches. Once existing identifier is used as known.

The SQL statement execution classes doesn't have to care, so don't expect `sql_identifier_case` affects column names in statements like

```
SELECT * FROM foo
```

sql_quoted_identifier_case

Contains how DBI::DBD::SqlEngine deals with quoted SQL identifiers (**readonly**). It's fixated to SQL_IC_SENSITIVE fIs0 (3), which is interpreted as SQL_IC_MIXED.

sql_flags

Contains additional flags to instantiate an SQL::Parser. Because an SQL::Parser is instantiated only once, it's recommended to set this flag before any statement is executed.

sql_dialect

Controls the dialect understood by SQL::Parser. Possible values (delivery state of SQL::Statement):

```
* ANSI
* CSV
* AnyData
```

Defaults to ''CSV''. Because an SQL::Parser is instantiated only once and SQL::Parser doesn't allow to modify the dialect once instantiated, it's strongly recommended to set this flag before any statement is executed (best place is connect attribute hash).

sql_engine_in_gofer

This value has a true value in case of this driver is operated via DBD::Gofer. The impact of being operated via Gofer is a read-only driver (not read-only databases!), so you cannot modify any attributes later - neither any table settings. **But** you won't get an error in cases you modify table attributes, so please carefully watch `sql_engine_in_gofer`.

sql_meta

Private data area which contains information about the tables this module handles. Table meta data might not be available until the table has been accessed for the first time e.g., by issuing a select on it however it is possible to pre-initialize attributes for each table you use.

DBI::DBD::SqlEngine recognizes the (public) attributes `col_names`, `table_name`, `readonly`, `sql_data_source` and `sql_identifier_case`. Be very careful when modifying attributes you do not know, the consequence might be a destroyed or corrupted table.

While `sql_meta` is a private and readonly attribute (which means, you cannot modify it's values), derived drivers might provide restricted write access through another attribute. Well known accessors are `csv_tables` for DBD::CSV `ad_tables` for DBD::AnyData and `dbm_tables` for DBD::DBM

sql_table_source

Controls the class which will be used for fetching available tables.

See "DBI::DBD::SqlEngine::TableSource" for details.

sql_data_source

Contains the class name to be used for opening tables.

See "DBI::DBD::SqlEngine::DataSource" for details.

Driver private methods

Default DBI methods

data_sources

The `data_sources` method returns a list of subdirectories of the current directory in the form "dbi:CSV:f_dir=$dirname".

If you want to read the subdirectories of another directory, use

```
my ($drh) = DBI->install_driver ("CSV");
my (@list) = $drh->data_sources (f_dir => "/usr/local/csv_data");
```

list_tables

This method returns a list of file names inside `$dbh->{f_dir}`. Example:

```
my ($dbh) = DBI->connect ("dbi:CSV:f_dir=/usr/local/csv_data");
my (@list) = $dbh->func ("list_tables");
```

Note that the list includes all files contained in the directory, even those that have non-valid table names, from the view of SQL.

Additional methods

The following methods are only available via their documented name when DBI::DBD::SQlEngine is used directly. Because this is only reasonable for testing purposes, the real names must be used instead. Those names can be computed by replacing the `sql_` in the method name with the driver prefix.

sql_versions

Signature:

```
sub sql_versions (;$) {
my ($table_name) = @_;
$table_name ||= ".";
...
}
```

Returns the versions of the driver, including the DBI version, the Perl version, DBI::PurePerl version (if DBI::PurePerl is active) and the version of the SQL engine in use.

```
my $dbh = DBI->connect ("dbi:File:");
my $sql_versions = $dbh->func( "sql_versions" );
print "$sql_versions\n";
__END__
# DBI::DBD::SqlEngine 0.05 using SQL::Statement 1.402
# DBI 1.623
# OS netbsd (6.99.12)
# Perl 5.016002 (x86_64-netbsd-thread-multi)
```

Called in list context, sql_versions will return an array containing each line as single entry.

Some drivers might use the optional (table name) argument and modify version information related to the table (e.g. DBD::DBM provides storage backend information for the requested table, when it has a table name).

sql_get_meta

Signature:

```
sub sql_get_meta ($$)
{
my ($table_name, $attrib) = @_;
...
}
```

Returns the value of a meta attribute set for a specific table, if any. See sql_meta for the possible attributes.

A table name of "." (single dot) is interpreted as the default table. This will retrieve the appropriate attribute globally from the dbh. This has the same restrictions as `$dbh->{$attrib}`.

sql_set_meta

Signature:

```
sub sql_set_meta ($$$)
{
my ($table_name, $attrib, $value) = @_;
...
}
```

Sets the value of a meta attribute set for a specific table. See sql_meta for the possible attributes.

A table name of "." (single dot) is interpreted as the default table which will set the specified attribute globally for the dbh. This has the same restrictions as `$dbh->{$attrib} = $value`.

sql_clear_meta

Signature:

```
sub sql_clear_meta ($)
{
my ($table_name) = @_;
...
}
```

Clears the table specific meta information in the private storage of the dbh.

Extensibility

DBI::DBD::SqlEngine::TableSource

Provides data sources and table information on database driver and database handle level.

```
package DBI::DBD::SqlEngine::TableSource;

sub data_sources ($;$)
{
my ( $class, $drh, $attrs ) = @_;
...
}

sub avail_tables
{
my ( $class, $drh ) = @_;
...
}
```

The `data_sources` method is called when the user invokes any of the following:

```
@ary = DBI->data_sources($driver);
@ary = DBI->data_sources($driver, \%attr);

@ary = $dbh->data_sources();
@ary = $dbh->data_sources(\%attr);
```

The `avail_tables` method is called when the user invokes any of the following:

```
@names = $dbh->tables( $catalog, $schema, $table, $type );

$sth = $dbh->table_info( $catalog, $schema, $table, $type );
$sth = $dbh->table_info( $catalog, $schema, $table, $type, \%attr );

$dbh->func( "list_tables" );
```

Every time where an `\%attr` argument can be specified, this `\%attr` object's `sql_table_source` attribute is preferred over the `$dbh` attribute or the driver default, eg.

```
@ary = DBI->data_sources("dbi:CSV:", {
f_dir => "/your/csv/tables",
# note: this class doesn't comes with DBI
sql_table_source => "DBD::File::Archive::Tar::TableSource",
# scan tarballs instead of directories
});
```

When you're going to implement such a DBD::File::Archive::Tar::TableSource class, remember to add correct attributes (including `sql_table_source` and `sql_data_source`) to the returned DSN's.

DBI::DBD::SqlEngine::DataSource

Provides base functionality for dealing with tables. It is primarily designed for allowing transparent access to files on disk or already opened (file-)streams (eg. for DBD::CSV

Derived classes shall be restricted to similar functionality, too (eg. opening streams from an archive, transparently compress/uncompress log files before parsing them,

```
package DBI::DBD::SqlEngine::DataSource;

sub complete_table_name ($$;$)
```

```
{
my ( $self, $meta, $table, $respect_case ) = @_;
...
}
```

The method `complete_table_name` is called when first setting up the *meta information* for a table:

```
"SELECT user.id, user.name, user.shell FROM user WHERE ..."
```

results in opening the table `user`. First step of the table open process is completing the name. Let's imagine you're having a DBD::CSV handle with following settings:

```
$dbh->{sql_identifier_case} = SQL_IC_LOWER;
$dbh->{f_ext} = '.lst';
$dbh->{f_dir} = '/data/web/adrmgr';
```

Those settings will result in looking for files matching `[Uu][Ss][Ee][Rr](\.lst)?$` in `/data/web/adrmgr/`. The scanning of the directory `/data/web/adrmgr/` and the pattern match check will be done in `DBD::File::DataSource::File` by the `complete_table_name` method.

If you intend to provide other sources of data streams than files, in addition to provide an appropriate `complete_table_name` method, a method to open the resource is required:

```
package DBI::DBD::SqlEngine::DataSource;

sub open_data ($)
{
my ( $self, $meta, $attrs, $flags ) = @_;
...
}
```

After the method `open_data` has been run successfully, the table's meta information are in a state which allowes the table's data accessor methods will be able to fetch/store row information. Implementation details heavily depends on the table implementation, whereby the most famous is surely DBD::File::Table.

SQL ENGINES

DBI::DBD::SqlEngine currently supports two SQL engines: SQL::Statement and DBI::SQL::Nano::Statement_. DBI::SQL::Nano supports a *very* limited subset of SQL statements, but it might be faster for some very simple tasks. SQL::Statement in contrast supports a much larger subset of ANSI SQL.

To use SQL::Statement, you need at least version 1.401 of SQL::Statement and the environment variable `DBI_SQL_NANO` must not be set to a true value.

SUPPORT

You can find documentation for this module with the perldoc(1) command.

```
perldoc DBI::DBD::SqlEngine
```

You can also look for information at:

- RT: CPAN's request tracker

  <http://rt.cpan.org/NoAuth/Bugs.html?Dist=DBI>
  <http://rt.cpan.org/NoAuth/Bugs.html?Dist=SQL-Statement>

- AnnoCPAN: Annotated CPAN documentation

  <http://annocpan.org/dist/DBI> <http://annocpan.org/dist/SQL-Statement>

- CPAN Ratings

  <http://cpanratings.perl.org/d/DBI>

- Search CPAN

  <http://search.cpan.org/dist/DBI/>

Where can I go for more help?

For questions about installation or usage, please ask on the dbi-dev@perl.org mailing list.

If you have a bug report, patch or suggestion, please open a new report ticket on CPAN, if there is not already one for the issue you want to report. Of course, you can mail any of the module maintainers, but it is less likely to be missed if it is reported on RT.

Report tickets should contain a detailed description of the bug or enhancement request you want to report and at least an easy way to verify/reproduce the issue and any supplied fix. Patches are always welcome, too.

ACKNOWLEDGEMENTS

Thanks to Tim Bunce, Martin Evans and H.Merijn Brand for their continued support while developing DBD::File, DBD::DBM and DBD::AnyData. Their support, hints and feedback helped to design and implement this module.

AUTHOR

This module is currently maintained by

H.Merijn Brand < h.m.brand at xs4all.nl > and Jens Rehsack < rehsack at googlemail.com >

The original authors are Jochen Wiedmann and Jeff Zucker.

COPYRIGHT AND LICENSE

SEE ALSO

DBI, DBD::File, DBD::AnyData and DBD::Sys.