NAME

DBD::File::Roadmap - Planned Enhancements for DBD::File and pure Perl DBD's

Jens Rehsack - May 2010

SYNOPSIS

This document gives a high level overview of the future of the DBD::File DBI driver and groundwork for pure Perl DBI drivers.

The planned enhancements cover features, testing, performance, reliability, extensibility and more.

CHANGES AND ENHANCEMENTS

Features

There are some features missing we would like to add, but there is no time plan:

LOCK TABLE
: The newly implemented internal common table meta storage area would allow us to implement LOCK TABLE support based on file system `flock ()` support.

Transaction support
: While DBD::AnyData recommends explicitly committing by importing and exporting tables, DBD::File might be enhanced in a future version to allow transparent transactions using the temporary tables of SQL::Statement as shadow (dirty) tables.

  Transaction support will heavily rely on lock table support.

Data Dictionary Persistence
: SQL::Statement provides dictionary information when a ''CREATE TABLE ...'' statement is executed. This dictionary is preserved for some statement handle attribute fetches (as `NULLABLE` or `PRECISION`).

  It is planned to extend DBD::File to support data dictionaries to work on the tables in it. It is not planned to support one table in different dictionaries, but you can have several dictionaries in one directory.

SQL Engine selecting on connect
: Currently the SQL engine selected is chosen during the loading of the module DBI::SQL::Nano. Ideally end users should be able to select the engine used in `DBI->connect ()` with a special DBD::File attribute.

Other points of view to the planned features (and more features for the SQL::Statement engine) are shown in SQL::Statement::Roadmap.

Testing

DBD::File and the dependent DBD::DBM requires a lot more automated tests covering API stability and compatibility with optional modules like SQL::Statement.

Performance

Several arguments for support of features like indexes on columns and cursors are made for DBD::CSV (which is a DBD::File based driver, too). Similar arguments could be made for DBD::DBM, DBD::AnyData, DBD::RAM or DBD::PO etc.

To improve the performance of the underlying SQL engines, a clean re-implementation seems to be required. Currently both engines are prematurely optimized and therefore it is not trivial to provide further optimization without the risk of breaking existing features.

Join the DBI developers IRC channel at <irc://irc.perl.org/dbi> to participate or post to the DBI Developers Mailing List.

Reliability

DBD::File currently lacks the following points:

duplicate table names
: It is currently possible to access a table quoted with a relative path (a) and additionally using an absolute path (b). If (a) and (b) are the same file that is not recognized (except for flock protection handled by the Operating System) and two independent tables are handled.

invalid table names
: The current implementation does not prevent someone choosing a directory name as a physical file name for the table to open.

Extensibility

I (Jens Rehsack) have some (partially for example only) DBD's in mind:

DBD::Sys
: Derive DBD::Sys from a common code base shared with DBD::File which handles all the emulation DBI needs (as getinfo, SQL engine handling, ...)

DBD::Dir
: Provide a DBD::File derived to work with fixed table definitions through the file system to demonstrate how DBI / Pure Perl DBDs could handle databases with hierarchical structures.

DBD::Join
: Provide a DBI driver which is able to manage multiple connections to other Databases (as DBD::Multiplex), but allow them to point to different data sources and allow joins between the tables of them:

```
# Example
# Let table 'lsof' being a table in DBD::Sys giving a list of open files using lsof util
# Let table 'dir' being a atable from DBD::Dir
$sth = $dbh->prepare( "select * from dir,lsof where path='/documents' and dir.entry = ls
$sth->execute(); # gives all open files in '/documents'
...

# Let table 'filesys' a DBD::Sys table of known file systems on current host
# Let table 'applications' a table of your Configuration Management Database
# where current applications (relocatable, with mountpoints for filesystems)
# are stored
$sth = dbh->prepare( "select * from applications,filesys where " .
"application.mountpoint = filesys.mountpoint and ".
"filesys.mounted is true" );
$sth->execute(); # gives all currently mounted applications on this host
```

PRIORITIES

Our priorities are focused on current issues. Initially many new test cases for DBD::File and DBD::DBM should be added to the DBI test suite. After that some additional documentation on how to use the DBD::File API will be provided.

Any additional priorities will come later and can be modified by (paying) users.

RESOURCES AND CONTRIBUTIONS

See <http://dbi.perl.org/contributing> for *how you can help*.

If your company has benefited from DBI, please consider if it could make a donation to The Perl Foundation "DBI Development" fund at <http://dbi.perl.org/donate> to secure future development.

Alternatively, if your company would benefit from a specific new DBI feature, please consider sponsoring it's development through the options listed in the section "Commercial Support from the Author" on <http://dbi.perl.org/support/>.

Using such targeted financing allows you to contribute to DBI development and rapidly get something specific and directly valuable to you in return.

My company also offers annual support contracts for the DBI, which provide another way to support the DBI and get something specific in return. Contact me for details.

Thank you.