NAME

DBI::ProfileDumper - profile DBI usage and output data to a file

SYNOPSIS

To profile an existing program using DBI::ProfileDumper, set the DBI_PROFILE environment variable and run your program as usual. For example, using bash:

```
DBI_PROFILE=2/DBI::ProfileDumper program.pl
```

Then analyze the generated file (*dbi.prof*) with dbiprof:

```
dbiprof
```

You can also activate DBI::ProfileDumper from within your code:

```
use DBI;

# profile with default path (2) and output file (dbi.prof)
$dbh->{Profile} = "!Statement/DBI::ProfileDumper";

# same thing, spelled out
$dbh->{Profile} = "!Statement/DBI::ProfileDumper/File:dbi.prof";

# another way to say it
use DBI::ProfileDumper;
$dbh->{Profile} = DBI::ProfileDumper->new(
Path => [ '!Statement' ],
File => 'dbi.prof' );

# using a custom path
$dbh->{Profile} = DBI::ProfileDumper->new(
Path => [ "foo", "bar" ],
File => 'dbi.prof',
);
```

DESCRIPTION

DBI::ProfileDumper is a subclass of DBI::Profile which dumps profile data to disk instead of printing a summary to your screen. You can then use dbiprof to analyze the data in a number of interesting ways, or you can roll your own analysis using DBI::ProfileData.

NOTE: For Apache/mod_perl applications, use DBI::ProfileDumper::Apache.

USAGE

One way to use this module is just to enable it in your `$dbh`:

```
$dbh->{Profile} = "1/DBI::ProfileDumper";
```

This will write out profile data by statement into a file called *dbi.prof*. If you want to modify either of these properties, you can construct the DBI::ProfileDumper object yourself:

```
use DBI::ProfileDumper;
$dbh->{Profile} = DBI::ProfileDumper->new(
Path => [ '!Statement' ],
File => 'dbi.prof'
);
```

The `Path` option takes the same values as in DBI::Profile. The `File` option gives the name of the file where results will be collected. If it already exists it will be overwritten.

You can also activate this module by setting the DBI_PROFILE environment variable:

```
$ENV{DBI_PROFILE} = "!Statement/DBI::ProfileDumper";
```

This will cause all DBI handles to share the same profiling object.

METHODS

The following methods are available to be called using the profile object. You can get access to the profile object from the Profile key in any DBI handle:

```
my $profile = $dbh->{Profile};
```

flush_to_disk

```
$profile->flush_to_disk()
```

Flushes all collected profile data to disk and empties the Data hash. Returns the filename written to. If no profile data has been collected then the file is not written and *flush_to_disk()* returns undef.

The file is locked while it's being written. A process 'consuming' the files while they're being written to, should rename the file first, then lock it, then read it, then close and delete it. The `DeleteFiles` option to DBI::ProfileData does the right thing.

This method may be called multiple times during a program run.

empty

```
$profile->empty()
```

Clears the Data hash without writing to disk.

filename

```
$filename = $profile->filename();
```

Get or set the filename.

The filename can be specified as a CODE reference, in which case the referenced code should return the filename to be used. The code will be called with the profile object as its first argument.

DATA FORMAT

The data format written by DBI::ProfileDumper starts with a header containing the version number of the module used to generate it. Then a block of variable declarations describes the profile. After two newlines, the profile data forms the body of the file. For example:

```
DBI::ProfileDumper 2.003762
Path = [ '!Statement', '!MethodName' ]
Program = t/42profile_data.t

+ 1 SELECT name FROM users WHERE id = ?
+ 2 prepare
= 1 0.0312958955764771 0.000490069389343262 0.000176072120666504 0.00140702724456787 1023115
+ 2 execute
1 0.0312958955764771 0.000490069389343262 0.000176072120666504 0.00140702724456787 102311581
+ 2 fetchrow_hashref
= 1 0.0312958955764771 0.000490069389343262 0.000176072120666504 0.00140702724456787 1023115
+ 1 UPDATE users SET name = ? WHERE id = ?
+ 2 prepare
= 1 0.0312958955764771 0.000490069389343262 0.000176072120666504 0.00140702724456787 1023115
+ 2 execute
= 1 0.0312958955764771 0.000490069389343262 0.000176072120666504 0.00140702724456787 1023115
```

The lines beginning with `+` signs signify keys. The number after the `+` sign shows the nesting level of the key. Lines beginning with `=` are the actual profile data, in the same order as in DBI::Profile.

Note that the same path may be present multiple times in the data file since `format()` may be called more than once. When read by DBI::ProfileData the data points will be merged to produce a single data set for each distinct path.

The key strings are transformed in three ways. First, all backslashes are doubled. Then all

newlines and carriage-returns are transformed into `\n` and `\r` respectively. Finally, any NULL bytes (`\0`) are entirely removed. When DBI::ProfileData reads the file the first two transformations will be reversed, but NULL bytes will not be restored.

AUTHOR

Sam Tregar <sam@tregar.com>

COPYRIGHT AND LICENSE