NAME

DBI::ProfileData - manipulate DBI::ProfileDumper data dumps

SYNOPSIS

The easiest way to use this module is through the dbiprof frontend (see dbiprof for details):

```
dbiprof --number 15 --sort count
```

This module can also be used to roll your own profile analysis:

```
# load data from dbi.prof
$prof = DBI::ProfileData->new(File => "dbi.prof");

# get a count of the records (unique paths) in the data set
$count = $prof->count();

# sort by longest overall time
$prof->sort(field => "longest");

# sort by longest overall time, least to greatest
$prof->sort(field => "longest", reverse => 1);

# exclude records with key2 eq 'disconnect'
$prof->exclude(key2 => 'disconnect');

# exclude records with key1 matching /UPDATE/i
$prof->exclude(key1 => qr/UPDATE/i);

# remove all records except those where key1 matches /SELECT/i
$prof->match(key1 => qr/SELECT/i);

# produce a formatted report with the given number of items
$report = $prof->report(number => 10);

# clone the profile data set
$clone = $prof->clone();

# get access to hash of header values
$header = $prof->header();

# get access to sorted array of nodes
$nodes = $prof->nodes();

# format a single node in the same style as report()
$text = $prof->format($nodes->[0]);

# get access to Data hash in DBI::Profile format
$Data = $prof->Data();
```

DESCRIPTION

This module offers the ability to read, manipulate and format DBI::ProfileDumper profile data.

Conceptually, a profile consists of a series of records, or nodes, each of each has a set of statistics and set of keys. Each record must have a unique set of keys, but there is no requirement that every record have the same number of keys.

METHODS

The following methods are supported by DBI::ProfileData objects.

$prof = DBI::ProfileData->new(File => "dbi.prof")
$prof = DBI::ProfileData->new(File => "dbi.prof", Filter => sub { ... })
$prof = DBI::ProfileData->new(Files => ["dbi.prof.1", "dbi.prof.2"])

Creates a new DBI::ProfileData object. Takes either a single file through the File option or a list of Files in an array ref. If multiple files are specified then the header data from the first file is used.

Files

Reference to an array of file names to read.

File

Name of file to read. Takes precedence over `Files`.

DeleteFiles

If true, the files are deleted after being read.

Actually the files are renamed with a C.deleteme> suffix before being read, and then, after reading all the files, they're all deleted together.

The files are locked while being read which, combined with the rename, makes it safe to 'consume' files that are still being generated by DBI::ProfileDumper.

Filter

The `Filter` parameter can be used to supply a code reference that can manipulate the profile data as it is being read. This is most useful for editing SQL statements so that slightly different statements in the raw data will be merged and aggregated in the loaded data. For example:

```
Filter => sub {
my ($path_ref, $data_ref) = @_;
s/foo = '.*?'/foo = '...'/ for @$path_ref;
}
```

Here's an example that performs some normalization on the SQL. It converts all numbers to `N` and all quoted strings to `S`. It can also convert digits to N within names. Finally, it summarizes long "IN (...)" clauses.

It's aggressive and simplistic, but it's often sufficient, and serves as an example that you can tailor to suit your own needs:

```
Filter => sub {
my ($path_ref, $data_ref) = @_;
local $_ = $path_ref->[0]; # whichever element contains the SQL Statement
s/\b\d+\b/N/g; # 42 -> N
s/\b0x[0-9A-Fa-f]+\b/N/g; # 0xFE -> N
s/'.*?'/'S'/g; # single quoted strings (doesn't handle escapes)
s/".*?"/"S"/g; # double quoted strings (doesn't handle escapes)
# convert names like log_20001231 into log_NNNNNNNN, controlled by $opt{n}
s/([a-z_]+)(\d{$opt{n},})/$1.('N' x length($2))/ieg if $opt{n};
# abbreviate massive "in (...)" statements and similar
s!(([NS],){100,})!sprintf("$2,{repeated %d times}",length($1)/2)!eg;
}
```

It's often better to perform this kinds of normalization in the DBI while the data is being collected, to avoid too much memory being used by storing profile data for many different SQL statement. See DBI::Profile.

$copy = $prof->*clone()*;

Clone a profile data set creating a new object.

`$header = $prof->`*header()*;
Returns a reference to a hash of header values. These are the key value pairs included in the header section of the DBI::ProfileDumper data format. For example:

```
$header = {
Path => [ '!Statement', '!MethodName' ],
Program => 't/42profile_data.t',
};
```

Note that modifying this hash will modify the header data stored inside the profile object.

`$nodes = $prof->`*nodes()*
Returns a reference the sorted nodes array. Each element in the array is a single record in the data set. The first seven elements are the same as the elements provided by DBI::Profile. After that each key is in a separate element. For example:

```
$nodes = [
[
2, # 0, count
0.0312958955764771, # 1, total duration
0.000490069389343262, # 2, first duration
0.000176072120666504, # 3, shortest duration
0.00140702724456787, # 4, longest duration
1023115819.83019, # 5, time of first event
1023115819.86576, # 6, time of last event
'SELECT foo FROM bar' # 7, key1
'execute' # 8, key2
# 6+N, keyN
],
# ...
];
```

Note that modifying this array will modify the node data stored inside the profile object.

`$count = $prof->`*count()*
Returns the number of items in the profile data set.

`$prof->sort(field => "field")`
`$prof->sort(field => "field", reverse => 1)`
Sorts data by the given field. Available fields are:

```
longest
total
count
shortest
```

The default sort is greatest to smallest, which is the opposite of the normal Perl meaning. This, however, matches the expected behavior of the dbiprof frontend.

`$count = $prof->exclude(key2 => "disconnect")`
`$count = $prof->exclude(key2 => "disconnect", case_sensitive => 1)`
`$count = $prof->exclude(key1 => qr/^SELECT/i)`
Removes records from the data set that match the given string or regular expression. This method modifies the data in a permanent fashion - use *clone()* first to maintain the original data after *exclude()*. Returns the number of nodes left in the profile data set.

`$count = $prof->match(key2 => "disconnect")`
`$count = $prof->match(key2 => "disconnect", case_sensitive => 1)`
`$count = $prof->match(key1 => qr/^SELECT/i)`
Removes records from the data set that do not match the given string or regular expression. This method modifies the data in a permanent fashion - use *clone()* first to maintain the original data

after *match()*. Returns the number of nodes left in the profile data set.

`$Data = $prof->`*Data()*
Returns the same Data hash structure as seen in DBI::Profile. This structure is not sorted. The *nodes()* structure probably makes more sense for most analysis.

`$text = $prof->`**format($nodes->[0])**
Formats a single node into a human-readable block of text.

`$text = $prof->`**report(number => 10)**
Produces a report with the given number of items.

AUTHOR

Sam Tregar <sam@tregar.com>

COPYRIGHT AND LICENSE