



## Profiling DBI Queries

brian d foy  
brian.d.foy@gmail.com

You new Perl programmers don't know how good you have it! When I was a kid, we didn't have any of this fancy database stuff that worked with just about every database server or format out there, and even when we got it, we were just happy to have it. Performance? Knowing what it's doing? Bah! The Perl Database Interface (DBI) keeps getting better, though, and it keeps responding to developer's needs.

My latest favorite DBI feature is its profiling capabilities. I can turn on accounting features with an environment variable to profile everything, or set a value in a database or statement handle so it applies to just that part of my program.

### The Old Way

When I first started using DBI, I quickly discovered that just because it was really easy to write database code didn't mean that it would necessarily perform well. Of course, that's my fault, but I also have to fix it.

Instead of using the DBI directly in my code, I wrote a wrapper around it so I could count the queries, track their parameters, and time the database response. Roughly, it looked something like this, but more complicated:

```
package My::DBI;
use Time::HiRes
    qw(gettimeofday tv_interval);

my %Queries;

sub query {
    my( $sql, @params ) = @_;
    $Queries{ $sql }{ _count }++;
    my $start = gettimeofday;
    $dbi->prepare( $sql );
    $dbi->execute( @params )
    my $end = gettimeofday;
    $Queries{ $sql }{ _time } =
        tv_interval( $end, $start );
}
```

At the end of a session, I'd dump the data in %Queries to see what happened. Usually I'd find some queries that I could optimize or some results that I could cache. It was a bit crude, but it worked.

### The New Way

The Perl DBI has actually had profiling capabilities for some time, but recently Tim Bunce had made them much easier to use. The DBI already does what I was doing before: it uses a single point of dispatch through which all queries run, so the DBI already has the set-up to count and time queries. I just have to tell it to turn on those features, which it can apply globally or only to certain handles.

To turn on profiling for my entire program, I use the `DBI_PROFILE` environment variable. This way, I don't have to do anything to the code to get my report. The particular format depends on my shell, and I'll talk about the actual values for the environment variables in a moment. The actual command line syntax may be different in your particular shell (I use `bash` for my examples).

```
$ env DBI_PROFILE=... program.pl
```

That way gets me a report (as in Figures 1 and 2 on page 11) when the program finishes.

If I want to limit its effect, I turn it on for a particular handle, whether a database handle (to profile only that connection) or a statement handle. In the next section, I show the particular values that I can use.

```
my $dbh = DBI->connect( ...,
    { Profile => ... }
);

$dbh->{Profile} = ...;

my $sth = $dbh->prepare( ... );
$sth->{Profile} = ...;
```