Perl's DBI Module
Scott D. Anderson

Perl's built-in capabilities can be extended by various "add-on" modules, written in Perl. They can be installed in the Perl include path (a series of directories where Perl looks for modules whenever the input has a `use` statement). In this course, we are using two important ones: CGI and DBI. Thus, our scripts will have, near the top:

```
use CGI;
use DBI;
```

The CGI module I have already discussed, and it is also well explained in the O'Reilly book *CGI Programming with Perl*, by Scott Guelich, Shishir Gundavaram, and Gunther Birznieks, which I've put copies of on reserve in the library. It's also documented online using `perldoc`. This handout briefly describes the DBI module. I can't possibly do justice to this module in a page, so you're encouraged to read the online documentation (using `perldoc`) and the O'Reilly book `ProgrammingthePerlDBI`, by Alligator Descartes and Tim Bunce, which I've also put on reserve.

Basic Concepts

You begin by *connecting* to a database. This returns an object called a database *handle*. You use the handle to invoke various methods, particularly *disconnect*. You can have multiple connection objects, all of which are separate, and as many connections as your database allows. In the documentation, a database handle is the value of the `dbh` variable.

Given a database handle, you can have the database process SQL statements. For a SELECT statement, the sequence is

1. prepare: this sends the SQL code to the database, which parses it, figures out an execution plan, and does whatever else it needs to do. This step also returns a statement handle (abbreviated `sth` in the documentation), which is an object that is used to invoke the subsequent steps. You can have multiple statements active simultaneously.

2. execute: requests that the database execute the statement and get ready to send data back, usually by setting up a *cursor* that will move through the data. (Cursors live in the database, not in the Perl script, but it's an important concept.)

3. fetch: gets one or more rows from the result, advancing the cursor. When there are no more rows, the result is false, which is usually the right value for exiting a loop.

4. finish: you may not have to invoke this method explicitly. It's mostly used if you're iterating through the results and discover that you don't need any more. By finishing the statement, you inform the database that no more data will ever be requested, thereby freeing up database resources.

For statements like INSERT, UPDATE or DELETE, which are executed for side-effect rather than value, the fetch stage is entirely unnecessary. Thus, the prepare and execute stages are combined into one, called "do."

Often, our scripts will use SQL statements that are constructed "on the fly," by the script. One easy way to do this is to simply construct the desired string, using the Perl concatenate operator, which is a dot, and string interpolation. Sometimes, you even need to use `sprintf`, which you

can look up in the documentation. This is straightforward string manipulation. One helpful thing to know about is that the DBI provides a database-specific quoting mechanism so that you don't have to re-write your script if you change from Oracle to some other database that uses different quoting conventions. This is done by the `quote` method on the database handle. Another way of constructing SQL statements on the fly is to prepare a statement with placeholders in it, denoted by the question mark, and bind those placeholders to actual values during execution. This has the advantage of efficiency if a series of related queries are made, since the parsing and preparation is only done once.

By default, the DBI checks for errors and prints out an error message if one occurs, so you don't have to check any return values for errors. This is good, but you should keep an eye on those error messages during debugging. CGI scripts run unattended, so (ideally), we should write our scripts to check for errors and return an appropriate error-reporting web page to the user if something goes wrong.

Code

The following are extracted from the online `perldoc` documentation. You should also check the example scripts.

Connecting and disconnecting:

```
$dbh = DBI->connect($data_source, $username, $auth);
$rc  = $dbh->disconnect;
```

For SELECT statements:

```
$sth = $dbh->prepare($statement);
$rv = $sth->execute;
@row_ary  = $sth->fetchrow_array;
$rv = $sth->finish;        if necessary
```

For non-SELECT statements:

```
$rv  = $dbh->do($statement);
```

When constructing statements by constructing strings:

```
$statement = "SELECT * FROM emps WHERE name = " . $dbh->quote($look_for);
$sth = $dbh->prepare($statement);
$rv = $sth->execute;
```

When constructing statements using `bind_param`:

```
$sth = $dbh->prepare("SELECT * FROM emps WHERE name = ?");
$rv = $sth->bind_param(1, $look_for);
$rv = $sth->execute;
```