

May 7-11, 2006

Tampa Convention Center

Tampa, Florida, USA

L03

# Using Perl & DBI With Databases

IDUG® 2006

North America

Darryl Priest

*DLA Piper Rudnick Gray Cary LLP*

Monday, May 8, 2006 • 04:00 p.m. – 05:10 p.m.

Platform: Informix For Application Developers

GoFurther



Session L03  
Using Perl & DBI With Databases

**Darryl Priest**

DLA Piper Rudnick Gray Cary LLP

darryl.priest@dlapiper.com

GoFurther



## Agenda

- What is DBI & DBD::Informix?
- Why Perl?
- Why DBI/DBD::Informix?
- Perl Basics
- Database Connections
- Static SQLs
- Fetching Data
- Other SQLs (Inserts, Deletes, etc.)
- Putting It All Together
- Supported, But Not Covered

3

DBI – Module Database Independent Interface For Perl

DBD::Informix – Informix Database Driver For Perl

DBI and DBD::Informix are object oriented

## Why Perl?

- Easy To Start
- Many Modules Available
- Autovivification and Garbage Collection
- Text Manipulation & Regular Expressions
- Portability
- Easy Access And Interaction With System Commands
- Hashes
- CGI
- Speed
- Code Reusability Using Modules

4

Modules like DBI, DBD::Informix, DBD::ODBC, Text::Wrap, CGI, LWP, SendMail::, XML::, SpreadSheet::WriteExcel ....

Hashes are collections of variables indexes by strings rather than numeric values.

## Why DBI/DBD::Informix?

- Very well tested
- Data Fetch Method Choices
- IBM/Informix Support
- Portability
- Database Connections

Moved application code from AIX/Informix to Windows/SQL Server with very minimal changes. (Only really changed data source to be ODBC.)

## Perl Basics

- `#!/usr/bin/perl -w`
- Variable Types (scalars, arrays or lists, hashes, references)
- use DBI;
- use strict;
- Variable Scope
- TMTOWTDI
- `q#` and `qq#`

6

Use strict should be after other use statements, in case modules don't 'use strict';

Variables `$Var`, `@Var`, `%Var` are all different and ok, probably not clear but ok.

Talk about qualifying variables like `$DBI::errstr` and autovivification

There's More Than One Way To Do It: Perl Motto

`qq#` in my code purely for readability

## DBI Generalizations

- Database connections are referred to as database handles usually named `$dbh`, `$ps_dbh`, etc.
- Data selection SQLs should follow the pattern

```
prepare,  
    execute, fetch, fetch, fetch ...  
execute, fetch, fetch, fetch ...
```
- Non-selection SQLs usually follow the pattern

```
prepare,  
    execute,  
execute,
```

Some of the examples shown are from `perldoc DBI` or `perldoc DBD::Informix`

## Database Connections

```

$dbh = DBI->connect($data_source, $username, $auth, \%attr);
$dbh = DBI->connect("DBI:Informix:$database");

$dbh = DBI->connect("DBI:Informix:$database", '', '',
    { AutoCommit => 0, PrintError => 1 });
my $dbh = DBI->connect("DBI:Informix:MyDatabase")
    or die "MyDatabase Database Open Error: $DBI::errstr\n";
$dbh->{ChopBlanks} = 1;
$dbh->{AutoCommit} = 1;
$dbh->{PrintError} = 1;
$dbh->{RaiseError} = 1;

my $ps_dbh = DBI->connect("DBI:Informix:hrdb@remote_tcp")
    or die "PeopleSoft Database Open Error: $DBI::errstr\n";

$dbh->disconnect();

```

8

Connections are Database Handles.

ChopBlanks can not be specified in the inline attributes.

Commits are done at the handle level, like 4gl, but consider having multiple handles to the same database to commit “part” of a transaction.

@ Must Use \

If populated \$DBI::errstr or \$dbh->errstr contains the SQL Error number and the ISAM error when possible.

Should remember to disconnect from database.

## Static SQLs

```
$el_dbh->do("set isolation to dirty read");
$el_dbh->do("set lock mode to wait");

$sql = qq#create temp table temp_teamleader
      (tkinit      char(5),
       teamleader  char(5)
       ) with no log in tempdbs#;
$el_dbh->do($sql);

$sql = qq#insert into temp_teamleader(tkinit, teamleader)
      select udjoin, udvalue from udf
      where udf.udtype = "TK" and udf.udfindex = 55#;

my $ins_teamleader_sth = $el_dbh->prepare($sql);
$ins_teamleader_sth->execute();
$el_dbh->do("create index teamldr_idx1 on temp_teamleader(tkinit)");
$el_dbh->do("update statistics high for table temp_teamleader");
```

9

First statements simply executes.

Ins\_teamleader\_sth is prepared and then executed

Do can only be used with non-select SQLs, unless the select is a select ...  
into

## Fetching Data (Static SQL)

```
$sql = qq#select rtype, rtdesc from crltype order by l#;  
my $get_party_type_sth = $el_dbh->prepare($sql);  
$get_party_type_sth->execute();
```

GoFurther

10

“or die” is not required, maybe not even desired if RaiseError/PrintError are on

Execution of select statements return the value of '0E0' which is both true and zero, or undef if it errors.

## Fetching Data with Placeholders

```
$sql = qq#select emplid, contact_name, relationship, phone
        from ps_emergency_cntct
        where emplid = ?
        order by primary_contact desc, contact_name#;

my $get_emerg_contact_sth = $ps_dbh->prepare_cached($sql);

$get_emerg_contact_sth->execute("12345");
```

### Or even better, using a scalar variable

```
my $InEmplid = "12345";
$get_emerg_contact_sth->execute($InEmplid);
```

11

Placeholders are filled in when SQL is executed, does excellent job of quoting, otherwise consider `$dbh->quote()`;

`Prepare_cached` hold statement handles in memory after subroutine has finished.

If getting input from a user, beware of hidden input, such as `12345'` or `emplid = emplid` or `emplid = 'y`

Placeholders are almost always better, quoting is automatic, etc.

## Processing Fetched Data

```
$sql = qq#select rtype, rtdesc from crltype order by l#;

my $get_party_type_sth = $l_dbh->prepare($sql);
$get_party_type_sth->execute();

my (@Row, %PartyTypes);
while ( @Row = $get_party_type_sth->fetchrow_array() ) {
    $PartyTypes{$Row[0]} = $Row[1];
}
```

### Same thing using hash references

```
my ($Row, %PartyTypes);
while ( $Row = $get_party_type_sth->fetchrow_hashref() ) {
    $PartyTypes{ $Row->{rtype} } = $Row->{rtdesc};
}
```

12

Fetchrow\_hashref a bit slower, but easy to see what's happening and table structure changes don't break your code, if you select the entire row.

## Processing Fetched Data, continued

```
opendir (BILLDIR, "/bills") or die "Error Opening $BillDir $!\n";

$sql = qq#select count(*), sum(lamount) from ledger
      where linvoice = ? and lzero != "Y"#;

my $check_sth = $dbh->prepare($sql);

while ( defined ($File = readdir(BILLDIR)) ) {

    @FileNamePieces = split(/\.\/, $File);
    $InvoiceNumber = $FileNamePieces[1];
    $check_sth->execute($InvoiceNumber);

    ($NotPaid, $Amount) = $check_sth->fetchrow_array();

    if ( $NotPaid > 0 ) { print "Not Paid, $NotPaid Ledger Items"; }
    else {
        $New = "$ArchDir/$File";
        move($OldFile, $New) or die "Move $OldFile To $New Failed: $!\n";
        chmod $Mode, $NewFile;
    }
}
```

13

Using `fetchrow_array()` to get two scalars.

Code snippet used to archive paid bills because directory gets too big, 300,000+ files, invoice number is part of bills text file name.

Fetch used in loop to determine which bills should be archived.

Only some fetch methods are shown, other available, consult perldoc DBI or other docs for all fetch methods.

## Processing Fetched Data, continued

```
$sql = qq#select fieldname, fieldvalue, xlatlongname, xlatshortname
        from xlattable x
        where effdt = ((select max(effdt) from xlattable xl
                        where xl.fieldname = x.fieldname and
                        xl.fieldvalue = x.fieldvalue and
                        xl.effdt <= TODAY and xl.language_cd = "ENG")) and
        x.fieldname in ("EMPL_TYPE", "ETHNIC_GROUP", "SEX", "MAR_STATUS",
                        "FULL_PART_TIME", "EMPL_STATUS", "PHONE_TYPE") and
        x.language_cd = "ENG"
        order by 1,2;#;

my $get_xlat_sth = $ps_dbh->prepare($sql);
$get_xlat_sth->execute();

my ($XlatRow);
while ($XlatRow = $get_xlat_sth->fetchrow_hashref()) {
    $Xlat{ $XlatRow->{fieldname} }
        { $XlatRow->{fieldvalue} } = { longname => $XlatRow->{xlatlongname},
                                        shortname => $XlatRow->{xlatshortname} };
}
```

14

%Xlat and \$sql are defined somewhere else

Data structures are great, fast ways to keep data at your fingertips, but remember they are stored in memory, so don't abuse them.

## Processing Fetched Data, continued

Previous example loads %Xlat hash with values such as:

```
$Xlat{MAR_STATUS}->{A}->{longname} = "Head of Household"  
$Xlat{MAR_STATUS}->{A}->{shortname} = "Hd Hsehld"  
$Xlat{MAR_STATUS}->{M}->{longname} = "Married";  
$Xlat{MAR_STATUS}->{M}->{shortname} = "Married";  
$Xlat{SEX}->{F}->{longname} = "Female";  
$Xlat{SEX}->{M}->{shortname} = "Male";
```

Hash values are referenced with:

```
$Xlat{SEX}->{ $Active->{sex} }->{shortname}  
$Xlat{MAR_STATUS}->{ $Active->{mar_status} }->{longname}
```

15

Hash values are shown in alpha order but are not necessarily stored in that order.

`$Active` is a hash reference to an employee data row being processed.

## Binding Columns To Fetch Data

```
$sql = qq#select pcode, pdesc
        from praccodes
        where pdesc is not null
        order by 1#;

my $get_praccodes_sth = $dbh->prepare($sql);

$get_praccodes_sth->execute();

my ($b_pcode, $b_pdesc);
$get_praccodes_sth->bind_columns(undef, \$b_pcode, \$b_pdesc);

while ( $get_praccodes_sth->fetch ) {
    $PracCodes{ $b_pcode } = $b_pdesc;
}
```

16

Fastest way to fetch data by a significant margin.

Return values bound to one memory location(that's why it's a reference to a scalar/array variable, or hash).

Each row goes to same location, so can't access previous row.

This example only shows how, probably not a great example since only a few rows returned.

Bind is after execute, per DBI docs, for portability.

## Binding Columns Continued

```
$sql = qq#select cmatter, to_char(cdisbdt, '%m/%d/%Y') cdisbdt, cbillamt
      from cost
      where cmatter is not null;#;

my $get_cost_sth = $el_dbh->prepare($sql);
my (%CostRow);
$get_cost_sth->bind_columns(undef,          \$CostRow{cmatter},
                          \$CostRow{cdisbdt}, \$CostRow{cbillamt});

while ( $get_cost_sth->fetch() ) {
    ... Do Something With %CostRow Hash Values ...
}

Alternate syntax
$sth->bind_col($col_num, \$col_variable);
$sth->bind_columns(@list_of_refs_to_vars_to_bind);
```

17

Can use `$bind_col` one column at a time and bind to values in an array or hash, so you don't have to keep track of large number of scalars.

## Preparing & Fetching Together

```
my $sql = qq#select emplid, name_first2last name from pm_employees_v#;

my $NamesRef = $dbh->selectall_hashref($sql, "emplid");

while ( $PeopleRow = $get_people_with_subitem_sth->fetchrow_hashref() ){

    if ( defined $NamesRef->{ $PeopleRow->{emplid} } ) {
        print "-- $NamesRef->{ $PeopleRow->{emplid} }{name} ";
    }
    else {
        print "-- Unknown";
    }
}
```

18

selectall\_hashref prepares the passed SQL and executes it immediately.

Can use selectall\_hashref or selectall\_arrayref

This is similar to the use of the fetchall\_\* statements.

## Inserting Rows

### Declare The Insert Statement Handle

```
$sql = qq#insert into winoutstat(wouser, wouser1, woreport, wotitle, wofile,  
                                wodate0, wotime0, wostat1, wopid)  
                                values(?, ?, ?, ?, ?,  
                                ?, ?, ?, ?);#;  
  
my $ins_win_sth = $sel_dbh->prepare_cached($sql);
```

### Insert The Row

```
$ins_win_sth->execute($Logon, $Logon, "Reminders", $Title, $FileName,  
                    $OutDate, $OutTime, "RUNNING", $$);  
  
my @Errd = @{$ins_win_sth->{ix_sqlerrd}};  
$Hold{woindex} = $Errd[1];
```

#### Alternate syntax

```
$Hold{woindex} = $ins_win_sth->{ix_sqlerrd}[1];
```

19

Also shows example of getting the new row inserted value of the serial value woindex.

## Deleting Data

### Declare The Delete Statement Handle

```
$sql = qq#delete from pm_reminders
      where matter_num = ? and
            location = ? and
            run_date = TODAY and
            run_by = ?;#;

my $del_remind_sth = $el_dbh->prepare($sql);
```

### Delete Row(s) Based On Passed Parameters

```
$del_remind_sth->execute($MatRow->{mmatter},
                        $Hold{location},
                        $ThisLogon);
```

## Using DBI With CGI, continued

Elite Report Files For User Darryl Priest				
	Report	Title	Report Date	Report Description
<a href="#">Archive</a>	Time Batch Report		10/01/2002 09:09	1 batches processed. 1 reported.
<a href="#">Archive</a>	Time Batch Finalization		10/01/2002 09:09	1 batches processed. 1 finalized.

## Using DBI With CGI

```
sub show_elite_files {  
  
    print header(),  
        start_html(-title=>"User File Manager",  
                  -style=>{'src'=>'/styles/inSite_Style.css'});  
  
    $sql = qq#select woindex, woreport, wotitle, wodate0, wotime0, wodatel, wotimel, wodescl  
              from winoutstat  
              where (wostat1 = "COMPLETE" or wostat2 = "COMPLETE") and wouser = ?  
              order by wodate0 desc, wotime0#;  
  
    my $get_files_sth = $el_dbh->prepare($sql);  
    $get_files_sth->execute($ThisLogon);  
  
    my ($FileRow, $ViewLink, $ShowDate, $Count);  
    $Count = 0;  
    while ( $FileRow = $get_files_sth->fetchrow_hashref() ) {  
        $ViewLink = a({-href=>"getfiles.cgi?Session=${InSession}&FileNum=$FileRow->{woindex}"}, "Archive");  
  
        $ShowDate = "$FileRow->{wodate0} $FileRow->{wotime0}";  
        if ( $FileRow->{wodate0} ne $FileRow->{wodatel} ) {  
            $ShowDate .= " - " . $FileRow->{wodatel} . " " . $FileRow->{wotimel};  
        }  
        elsif ( $FileRow->{wotime0} ne $FileRow->{wotimel} ) {  
            $ShowDate .= " - " . $FileRow->{wotimel};  
        }  
    }  
}
```

22

Entire subroutine which shows a users Elite File Manager files to them in HTML.

## Using DBI With CGI, continued

```
### If This Is The First File Printed, Print The Headers First
if ( $Count == 0 ) {
    my $ThisName = get_user_name($ThisLogon);
    print start_table({-width=>'100%',
                      -border=>1,
                      -cellpadding=>'5'}),
          $NewLine,
          Tr ( th ({-colspan=>'5'}, h4("Elite Report Files For User $ThisName") ) ),
          Tr ( th ( "&nbsp;" ),
                th ( h4("Report") ),
                th ( h4("Title") ),
                th ( h4("Report Date") ),
                th ( h4("Report Description") )
              );
}
### Print Information For This File
print Tr ( td ({-align=>'center'}, "$ViewLink"),
          td ({-align=>'left'}, "$FileRow->{woreport}"),
          td ({-align=>'left'}, "$FileRow->{wotitle}"),
          td ({-align=>'center'}, "$ShowDate"),
          td ({-align=>'left'}, "$FileRow->{wodescl}"));

$Count++;
}
```

## Using DBI With CGI, continued

```
### If No File Rows Found Show Error & Back Button, Otherwise
### Print The End Of The Table
if ( $Count == 0 ) {
    print br, br,
        textfield(-name=>'ProcessMessage',
            -size=>'80',
            -style=>$ErrorStyle,
            -maxlength=>'80',
            -value=>'No Files Were Found In Your Elite File Manager!'),
        br, br;
    print_back();
    return;
} else { print end_table(); }

print end_html();
} ### End Of SubRoutine show_elite_files
```

## Defining Reusable Code

```
#!/usr/bin/perl
package MyLib;
use strict;
require Exporter;
use vars qw($VERSION @ISA @EXPORT);
$VERSION = 0.01;
@ISA = qw(Exporter);
@EXPORT = qw(get_names);

sub get_names {
    my ($UseDbh, $Emplid) = @_;
    my (@RetVal);
    my $sql = qq#select first_name, last_name from pm_employees_v where emplid_s = ?#;

    my $get_names_sth = $UseDbh->prepare_cached($sql);
    $get_names_sth->execute($Emplid);

    @RetVal = $get_names_sth->fetchrow_array();
    return @RetVal;
}
1;
```

## Using Your Module

```
#!/usr/bin/perl -w
use DBI;
use strict;

use lib q{/perl/modules/};
use MyLib;

.....

if ( defined $Emplid ) {
    my (@RetNames) = MyLib::get_names($dbh, $Emplid);
    if ( defined $RetNames[0] ) { $Name = $RetNames[0]; }
    else { $Name = "Name Unknown"; }
}
```

26

Don't need to use MyLib::, but that fully qualifies it, just in case there are more than one routines of that name.

Only really want the name in first to last order.

No "runner" needed!!

## Database Connection Module

```

sub default_db_connect {
my ($DB, $Server) = @_;
my ($dbh);
if ( defined $Server and length($Server) > 1 ) {
    $dbh = DBI->connect("DBI:Informix:${DB}\@$Server");
}
else {
    $dbh = DBI->connect("DBI:Informix:${DB}", undef, undef, { PrintError => 0, RaiseError => 0 });
    if ( ! defined $dbh ) {
        $Server = default_informix_tcp(); ### Change INFORMIXSERVER To _tcp
        $dbh = DBI->connect("DBI:Informix:${DB}\@$Server");
    }
}
if ( defined $dbh ) {
    $dbh->{AutoCommit} = 1;
    $dbh->{ChopBlanks} = 1;
    $dbh->{PrintError} = 1;
    $dbh->{RaiseError} = 1;
    if ( $dbh->{ix_LoggedDatabase} ) { $dbh->do("set lock mode to wait"); }
    if ( $dbh->{ix_ModeAnsiDatabase} ) { $dbh->do("set isolation to dirtyread"); }
    return $dbh;
}
else {
    die "SDB Database Open Error. Error: $DBI::errstr";
}
} ### End Of SubRoutine default_db_connect

```

27

default\_informix\_tcp is another 'default' module, which simply replaces \_shm in the Informix server name with \_tcp

```

sub default_informix_tcp {
my $InformixTCP = $ENV{INFORMIXSERVER};
$InformixTCP =~ s/_shm/_tcp/;
return $InformixTCP;
} ### End Of SubRoutine default_informix_tcp

```

## Get Employee Data Example

```
#!/usr/local/bin/perl -w
### Script Name: empl_info.pl
$| =1;
use DBI;
use Term::Size;
use Getopt::Std;
use strict;

use lib q{/custom/perl/modules/bin};
use Defaults;

use vars qw($opt_b $opt_c $opt_d $opt_e $opt_l $opt_n $opt_t $opt_v);

my $Usage = qq#
Usage: empl_info.pl [ -b Show Blanks -c Columns -d Database -e Emplid -l Logon -n Name -v Verbose ]
                    -b Show Data Columns That Are Unpopulated, Default Is To Skip
                    -c Column Name Match To Be Reported
                    -d Database Server To Select Data From
                    -e Employee ID To Report
                    -l Employee Logon ID To Report
                    -n Employee Name To Report
                    -t Include Terminated Employees & Contractor / Temps
                    -v Verbose Column Output
#;

getopts('bc:d:e:l:n:tv');

### Just In Case See If There's A Single Argument
my @Args = @ARGV;
```

## Get Employee Data Example, cont'd

```
### Get User Input, Make Sure To Get An Emplid, Name Or Logon
my (%In, $IncludeTerms, $ShowBlanks, $Verbose);
if ( defined $opt_b ) { $ShowBlanks = 1; }
else { $ShowBlanks = 0; }

### If Specific Columns Are Requested, Make Sure Verbose Is On So All Columns Are
### Available, Also Make Sure If Phone Number Is Selected To Include Dial Prefix
if ( defined $opt_c ) {
    $In{columns} = $opt_c;
    if ( $In{columns} =~ /phon/ ) { $In{columns} .= "|dial_prefix"; }
    $opt_v = 1;
}

if ( defined $opt_d ) { $In{db} = "mydb@$opt_d"; }
else { $In{db} = "mydb"; }

if ( defined $opt_e ) { $In{emplid} = $opt_e; }

if ( defined $opt_l ) { $In{logon} = lc($opt_l); }

if ( defined $opt_n ) { $In{name} = lc($opt_n); }

if ( defined $opt_t ) { $IncludeTerms = 1; }
else { $IncludeTerms = 0; }

if ( defined $opt_v ) { $Verbose = 1; }
else { $Verbose = 0; }
```

## Get Employee Data Example, cont'd

```
### If No Options Were Passed, Check For Valid Argument,  
### Or Die With Usage Displayed  
if ( ! exists $In{emplid} and ! exists $In{logon} and ! exists $In{name} ) {  
  
    ## Check The Possible Argument For Possible Usage  
    if ( defined $Args[0] and length($Args[0]) > 1 ) {  
        if ( $Args[0] =~ /^[0-9]{5}$/ ) { $In{emplid} = $Args[0]; }  
        elsif ( $Args[0] =~ /^[A-Z,a-z,\-\_\`]+$/ ) { $In{name} = lc($Args[0]); }  
        elsif ( $Args[0] =~ /^[A-Z,a-z,0-9]{1,8}$/ ) { $In{logon} = lc($Args[0]); }  
    }  
    else {  
        die "\n$Usage\n\n";  
    }  
}  
  
### If Looking For A Name Make Sure It Has Wild Cards  
if ( defined $In{name} ) {  
    if ( $In{name} !~ /\*[\?]/ ) { $In{name} = "*" . $In{name} . "*"; }  
}  
  
## Get Terminal Width  
my ($Columns, $Rows) = Term::Size::chars *STDOUT{IO};  
my $PrintWidth = $Columns - 2;
```

## Get Employee Data Example, cont'd

```
### Set Default Columns String, Which Will Be Reported Unless Overridden With -c Or -v
my %DefaultColumns = (
    assignments => '',
    empl_status_desc => '',
    full_prt_time_desc => '',
    hire_date => '',
    job_family => '',
    logon_id => '',
    published_phones => '',
    secretaries => '',
    term_date => '',
    work_phone => ''
);

### Connect To Database
my ($pm_dbh);
if ( defined $opt_d ) { $pm_dbh = default_db_connect("mydb", $opt_d); }
else { $pm_dbh = default_db_connect("mydb"); }

### Select Emplid & Name For Passed Emplid/Logon/Name Match
my ($sql, $Where, $TermSql, $TempSql);
$sql = qq#select emplid, name_first2last, 'E' from pm_empl_search#;

$TermSql = qq#select emplid, name_first2last, 'T' from pmhr_terminations#;

$TempSql = qq#select emplid, name_first2last, 'C' from pmhr_temps#;
```

## Get Employee Data Example, cont'd

```
my ($get_emplid_sth, $where);
SWITCH: {
  if ( exists $In{emplid} ) {
    $Where = qq# emplid = ?#;

    if ( $IncludeTerms ) {
      $sql .= qq# where $Where union $TermSql where $Where union $TempSql where $Where#;
      $get_emplid_sth = $pm_dbh->prepare($sql);
      $get_emplid_sth->execute($In{emplid}, $In{emplid}, $In{emplid});
    }
    else {
      $sql .= qq# where $Where#;
      $get_emplid_sth = $pm_dbh->prepare($sql);
      $get_emplid_sth->execute($In{emplid});
    }
  }
  last SWITCH;
}
```

## Get Employee Data Example, cont'd

```
if ( exists $In{logon} ) {
    $Where = qq# lower(logon_id) matches ?#;

    if ( $IncludeTerms ) {
        $sql .= qq# where $Where union $TermSql where $Where union $TempSql where $Where order by 2#;
        $get_emplid_sth = $pm_dbh->prepare($sql);
        $get_emplid_sth->execute($In{logon}, $In{logon}, $In{logon});
    }
    else {
        $sql .= qq# where $Where order by 2#;
        $get_emplid_sth = $pm_dbh->prepare($sql);
        $get_emplid_sth->execute($In{logon});
    }
    last SWITCH;
}

if ( exists $In{name} ) {
    $Where = qq# lower(name_first2last) matches ?#;

    if ( $IncludeTerms ) {
        $sql .= qq# where $Where union $TermSql where $Where union $TempSql where $Where order by 2#;
        $get_emplid_sth = $pm_dbh->prepare($sql);
        $get_emplid_sth->execute($In{name}, $In{name}, $In{name});
    }
    else {
        $sql .= qq# where $Where order by 2#;
        $get_emplid_sth = $pm_dbh->prepare($sql);
        $get_emplid_sth->execute($In{name});
    }
    last SWITCH;
}
}
```

## Get Employee Data Example, cont'd

```
### Fetch All Employees Found For Passed Match
my $EmplidRef = $get_emplid_sth->fetchall_arrayref();

### If Only Employee Matches, Call Show Subroutine, Else Show List Of Matches
### And Allow User To Select In A Loop From The List And Report
my $ListsShown = 0;
if ( @{$EmplidRef} > 0 ) {
    if ( @{$EmplidRef} == 1 ) {
        list_empl_info($EmplidRef->[0][0], $EmplidRef->[0][2]);
    }
    else {
        show_list($EmplidRef);
        my ($Choice);
        while (<STDIN>) {
            chomp;
            if ( $_ =~ /[Xx]/ ) { last; }
            $Choice = $_ - 1;
            if ( $Choice < @{$EmplidRef} ) {
                list_empl_info($EmplidRef->{ $Choice } [0], $EmplidRef->{ $Choice } [2]);
            }
            show_list($EmplidRef);
        }
    }
}
else {
    print "\n\nNo Matches Found For Passed Criteria\n\n";
}
$pm_dbh->disconnect(); ### End Of Main Program ###
```

## Get Employee Data Example, cont'd

```
### SubRoutine: show_list
### This subroutine list the passed list reference of employee ids and names.
sub show_list {
my ($ListRef) = @_;

### If This Isn't The First Time This Was Called
if ( $ListsShown > 0 ) {
    print "Press Enter To Continue";
    while (<STDIN>) { last; }
}
$ListsShown++;

my ($x, $y);
print "\n\n    Selected Employees\n";
print "    -----\n";
for ($x = 0; $x < @{$ListRef}; $x++) {
    $y = $x + 1;
    if ( $ListRef->[$x][2] eq "E" ) {
        printf("%3d.) %s (%s)\n", $y, $ListRef->[$x][1], $ListRef->[$x][0]);
    }
    elsif ( $ListRef->[$x][2] eq "C" ) {
        printf("%3d.) %s (%s) - Contractor / Temp\n", $y, $ListRef->[$x][1], $ListRef->[$x][0]);
    }
    else {
        printf("%3d.) %s (%s) - Terminated\n", $y, $ListRef->[$x][1], $ListRef->[$x][0]);
    }
}
print "\nEnter Choice(or x to exit): ";
} ### End Of SubRoutine show_list
```

## Get Employee Data Example, cont'd

```
### SubRoutine: list_empl_info
### This subroutine will list the employee information
### from pm_employees_v or pmhr_terminations based on
### employee status for the passed emplid.
sub list_empl_info {

my ($ThisEmplid, $EmplStatus) = @_;

### Select All Potential Data Columns For Passed Emplid
if ( $EmplStatus eq "E" ) {
    $sql = qq#select * from pm_employees_v where emplid = ?#;
}
elseif ( $EmplStatus eq "C" ) {
    $sql = qq#select * from pmhr_temps where emplid = ?#;
}
else {
    $sql = qq#select * from pmhr_terminations where emplid = ?#;
}

my $get_pdata_sth = $pm_dbh->prepare_cached($sql);

$get_pdata_sth->execute($ThisEmplid);
```



## Get Employee Data Example, cont'd

```
### For Each Returned Column
for $Var ( sort keys %{$Row} ) {
  if ( $Var =~ /_s$/ ) { next; }

  ### If User Selected Specific Columns To Report, Only
  ### Report The Selected Columns
  if ( exists $In{columns} ) {
    if ( $Var !~ /$In{columns}/ ) { next; }
  }

  ### If Not Verbose And This Column Isn't A Default, Skip It
  if ( ! $Verbose ) {
    if ( ! exists $DefaultColumns{ $Var } ) { next; }
  }

  ### If Column Contains Data, Report It, Unless Blanks
  ### Are To Be Shown, Then Set It To "" & Print It Anyway
  if ( $ShowBlanks ) {
    if ( ! defined $Row->{$Var} ) { $Row->{$Var} = ""; }
  }

  if ( $ShowBlanks or ( defined $Row->{$Var} and length($Row->{$Var}) > 0 ) ) {
    write:
  }
}
print "<" x $PrintWidth, "\n:"
}
} ### End Of SubRoutine list_empl_info
```





## Supported, But Not Covered In Detail

### Accessing The Informix SQLCA Values

```
$sqlcode = $sth->{ix_sqlcode};  
$sqlerrm = $sth->{ix_sqlerrm};  
$sqlerrp = $sth->{ix_sqlerrp};  
@sqlerrd = @{$sth->{ix_sqlerrd}};  
@sqlwarn = @{$sth->{ix_sqlwarn}};
```

### Transactions using `$dbh->commit();` and `$dbh->rollback();`

#### Do With Parameters

```
$dbh->do($stmt, undef, @parameters);  
$dbh->do($stmt, undef, $param1, $param2);
```

#### Quoting with `$dbh->quote($string);`

```
$sth->finish; and undef $sth;
```

#### Blob fields

Blob fields can be selected, inserted and deleted but not updated.  
Blobs should be stored in memory.

## Supported, But Not Covered, continued

- \$sth attributes, NUM\_OF\_FIELDS, NAME, etc.

- DBI->trace(\$level, \$tracefile);

- Fetch methods `selectrow_array()` & `selectall_array()`

- `$dbh->func()`

- Statement Handles For Update

```
$st1 = $dbh->prepare("SELECT * FROM SomeTable FOR UPDATE");
$wc = "WHERE CURRENT OF $st1->{CursorName}";
$st2 = $dbh->prepare("UPDATE SomeTable SET SomeColumn = ? $wc");
$st1->execute;
$row = $st1->fetch;
$st2->execute("New Value");
```

- `$sth->rows();`

42

`@{$sth->{NAME}}` returns column names being selected, `@{sth->{NUM_OF_FIELDS}}` the number of fields being selected

`select[row|all][_array|arrayref|hashref]` methods combine prepare, execute and fetch methods

`$dbh->func()` can be used to get tables in database, columns, views and other Informix database specific information.

Statement handles / cursor for update example from DBD::Informix, works for updates and deletes

Rows affected accessible by `$rv = $sth->do()` or `$sth->rows`, however not effective with select statements. I don't use this, I count rows when I want to know what happened. Returns -1 if unknown.

## Additional Information

- [dbi.perl.org](http://dbi.perl.org) - DBI Home Page
- [www.perl.com](http://www.perl.com)
- [www.perl.org](http://www.perl.org)
- [www.cpan.org](http://www.cpan.org) - Comprehensive Perl Archive Network
- [www.activestate.com/perl](http://www.activestate.com/perl) - Windows Based Perl Solutions
  
- [perldoc DBI](#) – DBI Man Pages
- [perldoc DBD::Informix](#) – DBD::Informix Man Pages
  
- Learning Perl by Randal Schwartz
- Programming Perl by Larry Wall, Tom Christiansen & Jon Orwant
- Programming the Perl DBI, by Alligator Descartes and Tim Bunce
- Perl Cookbook by Tom Chistiansen & Nathan Torkington
- Perl Objects, References & Modules by Randal Schwartz & Tom Phoenix

See DBI docs for many more helpful links and resources.

## Thanks!

- To the authors who brought us:
  - Perl
    - Larry Wall
  - DBI
    - Tim Bunce
    - Alligator Descartes
  - DBD::Informix
    - Jonathan Leffler

And thanks for their continued support and maintenance and their patience with newbies.

May 7-11, 2006

Tampa Convention Center

Tampa, Florida, USA

L03

## Using Perl & DBI With Databases

IDUG® 2006

North America

Darryl Priest

*DLA Piper Rudnick Gray Cary LLP*

Monday, May 8, 2006 • 04:00 p.m. – 05:10 p.m.

Platform: Informix For Application Developers

GoFurther

