NAME

A (Modest) Proposal for Replacing Gramene's Quick Search

AUTHOR

Ken Youens-Clark <kclark@cshl.edu>

SYNOPSIS

I offer an idea for a comprehensive search when most of your data is locked away in databases.

```
    http://dev.gramene.org/db/searches/unisearch
```

DESCRIPTION

Gramene's current ``quick'' search isn't all that quick:

```
    http://www.gramene.org/db/searches/browser?search_type=All&query=c86&RGN=on
```

This script takes the ``query'' and uses Perl and LWP to query each module's search scripts with the added argument of ``table_only=1'' which instructs the scripts to leave out the header and footer wrappers, returning only the ``table'' of results. It's not entirely quick, but it does the job and maintains a level of consistency at least in that any search result for a module here will be the same as if the user had gone to the specific module search interface.

What I think could be improved:

- **Allow more tables and fields to be searched**
- **Use a search engine for more faster searches**
- **Use a search engine for more interesting matches (stems, soundex)**
- **Present the results in a uniform, Googlish way**

While developing Gramene's diversity module around the GDPDM schema:

```
    http://www.maizegenetics.net/gdpdm/
```

I realized that a field-by-field, table-by-table search in a schema of 30+ tables would be inefficient (to say the least), so

I had an idea to unfield the data and cram everything that needed to be searched into a single MySQL table that would use the FULLTEXT index:

```
    http://dev.mysql.com/doc/refman/5.0/en/fulltext-search.html
```

I decided to port the idea over for Gramene and added the facility to also index all the static HTML on our site. I introduced the idea to the ``gramene-dev'' list many months ago; however, the implementaion left some to be desired, and the idea was shelved. Recently I decided to dust it off, having learned some things that I wanted to apply.

SEARCH SCHEMA

First, let me show you the schema:

```
  CREATE TABLE 'doc_search' (
    'path' varchar(255) NOT NULL default '',
    'title' varchar(255) default NULL,
    'contents' text,
    UNIQUE KEY ('path'),
    FULLTEXT KEY ('path','title','contents')
  ) ENGINE=MyISAM;

  CREATE TABLE 'module_search' (
    'module_name' varchar(40) NOT NULL default '',
    'table_name' varchar(40) NOT NULL default '',
    'record_id' int(10) unsigned NOT NULL default '0',
    'record_text' text,
    UNIQUE KEY ('module_name','table_name','record_id'),
    KEY ('module_name','table_name'),
    FULLTEXT KEY ('record_text')
  ) ENGINE=MyISAM;
```

LOADING

The ``doc_search'' is pretty simple. I crawl the Gramene docroot and find all the documents I want to index, including HTML, Word Documents, and PDFs. (PDFs are actually pretty horrendous to index as the text comes out pretty mangled, e.g., the word ``the'' might come out like ``t he'' if the text was, say, justified. Luckily, it seems Gramene's PDFs are alternate formats of other texts like HTML, which are just fine.) For each document, I strip out any ugly stuff like HTML tags and put the contents into the ``contents'' field (duh). There is a FULLTEXT index on the title, path, and contents of the document.

The module search is just a little more interesting, and I'll have to give you some background real quick-like: Gramene has a number of ``modules'' -- discrete units of databases and code that represent some type of data we store like ``markers'' or ``QTLs''. A ``module'' is defined in the ``<modules>'' section of ``gramene.conf'' and has a corresponding ``<section>'' where it defines how to connect to its database. This is important for the ``Gramene::DB'' module to work which is used by the Gramene::CDBI modules. A while back, the Gramene team decided to start using a standardized object-relational model (ORM). We settled on the CPAN module ``Class::DBI'':

```
    http://search.cpan.org/dist/Class-DBI/
```

Our namespace is called ``Gramene::CDBI'', and to use, say the CDBI interface to the markers db, you just say ``use Gramene::CDBI::Markers.''

OK, so back to loading the modules search. I run through each table in the schema for each module and extract all the ``text'' of each record. (The text is basically anything in a field that isn't the primary key or a foreign key, and I know which are which because of the details that are in the Gramene::CDBI classes, which were generated directly and automatically from the MySQL schemas [see ``bin/mk-gcdbi.pl'' and ``templates/class-dbi/gramene-cdbi.tmpl'' under

``/usr/local/gramene/''].)

Curators can limit which tables in a schema are indexed in the ``gramene.conf'' section for ``<gramene_search>'' using the ``<limit_index>''. E.g., here's the line for the qtl db:

```
<limit_index>
  qtl  index_only=qtl+qtl_trait,qtl_trait_synonym
</limit_index>
```

This says to index only the ``qtl'' table and to aggregate (the ``+'' sign) the tables ``qtl_trait'' and ``qtl_trait_synonym'' for each ``qtl'' record. (Use a semi-colon to separate the tables+aggregates.) This ``aggregation'' of data is one of the ideas I got after working more with the GDPDM/diversity schema -- sometimes your data is very normalized and the thing being sought is actually pretty far away from the concept that the user cares to see. So in the above example, if I found a search string in the ``qtl_trait_synonym'' table, I would return the find in the context of which QTLs matched, not which trait synonyms. FWIW, it seems to me that it's best to index only on table per module and to aggregate as much data as possible around that.

So now, this should make the the script that loads the schema understandable:

```
#!/usr/local/bin/perl

# vim: tw=78: sw=4: ts=4: et:

# $Id: load-gramene-search.pl,v 1.2 2007/02/01 19:46:43 kclark Exp $

use strict;
use warnings;
use DBI;
use Data::Dumper;
use English qw( -no_match_vars );
use File::Basename;
use File::Extract::PDF;
use File::Extract::Result;
use CAM::PDF;
use File::Find::Rule;
use File::Spec::Functions;
use Getopt::Long;
use HTML::HeadParser;
use HTML::Strip;
use IO::Prompt;
use PDF::API2;
use Perl6::Slurp;
use Pod::Usage;
use Readonly;

$| = 1;

use Gramene::Config;
use Gramene::DB;
use Gramene::Utils qw( commify get_logger table_name_to_gramene_cdbi_class );

delete @ENV{qw(IFS CDPATH ENV BASH_ENV)};
$ENV{'PATH'} = '/bin:/usr/bin:/usr/local/bin';

Readonly my $ANTIWORD  => '/usr/local/bin/antiword';
Readonly my $COMMA     => q{, };
Readonly my $DBL_COLON => q{::};
Readonly my $EMPTY_STR => q{};
Readonly my $NL        => qq{\n};
Readonly my $SLASH     => q{/};
Readonly my $SPACE     => q{ };
Readonly my $VERSION   => sprintf '%d.%02d',
                          qq$Revision: 1.2 $ =~ /(\d+)\.(\d+)/;

my $modules      = $EMPTY_STR;
my $list_modules = 0;
my ( $help, $man_page, $show_version );
GetOptions(
    'l|list'      => \$list_modules,
    'm|modules:s' => \$modules,
    'help'        => \$help,
    'man'         => \$man_page,
    'version'     => \$show_version,
) or pod2usage(2);

if ( $help || $man_page ) {
    pod2usage({
        -exitval => 0,
        -verbose => $man_page ? 2 : 1
    });
};

if ( $show_version ) {
    my $prog = basename( $PROGRAM_NAME );
    print "$prog v$VERSION\n";
    exit 0;
}

my %process      = map { s/^\s+|\s+//g; lc $_, 1 } split /,/, $modules;
my $config       = Gramene::Config->new;
my $modules_conf = $config->get('modules');
my $search_conf  = $config->get('gramene_search');
( my $db_name    = $search_conf->{'db_dsn'} ) =~ s/.*://;
my @modules      = ref $modules_conf->{'module'} eq 'ARRAY'
                   ? @{ $modules_conf->{'module'} }
                   : ( $modules_conf->{'module'} );
my %valid_module = map { $_, 1 } ( @modules, 'documents' );

if ( $list_modules ) {
    print join $NL,
        'Valid modules:',
        ( map { "  $_" } sort keys %valid_module ),
        $EMPTY_STR;
    exit 0;
}

if ( %process ) {
    my @bad = grep { !exists $valid_module{ $_ } } keys %process;

    if ( @bad ) {
        my $bad   = join $COMMA, sort @bad;
        my $valid = join $COMMA, sort @modules;
        die "Bad modules ($bad).\nPlease choose from:\n$valid\n";
    }
```

```
        my $ok = prompt -yn,
            sprintf( "OK to recache %s in '$db_name'? ",
                %process
                    ? join($COMMA, sort keys %process)
                    : join($COMMA, sort @modules)
            )
        ;

        if ( !$ok ) {
            print "Exiting.\n";
            exit 0;
        }
    }
    else {
        %process = map { $_, 1 }
            prompt -menu => [ (sort keys %valid_module), 'ALL' ], 'Which module?';

        if ( $process{'ALL'} ) {
            %process = %valid_module;
        }
    }

    my $logger = get_logger();
    $logger->info("Reloading search db '$db_name'");

    my $db = Gramene::DB->new('gramene_search');

    my $insert_sql = q[
        insert
        into   module_search (module_name, table_name, record_id, record_text)
        values (?, ?, ?, ?)
    ];

    my $num_modules = 0;
    my $num_records = 0;

    for my $module ( @modules ) {
        if ( %process ) {
            next if !defined $process{ lc $module };
        }

        my ( %index_only, %skip );
        if ( my $search_limit = $search_conf->{'limit_index'}{ $module } ) {
            my ( $directive, $tables ) = split /=/, $search_limit;
            if ( $directive !~ /^(index_only|skip)$/ ) {
                die "Bad directive ($directive) in limit clause ($search_limit).\n";
            }

            for my $table ( split /;/, $tables ) {
                if ( $directive eq 'skip' ) {
                    $skip{ $table } = 1;
                }
                else {
                    if ( $table =~ /(\w+)\+(.+)/ ) {
                        $index_only{ $1 } = [ split /,/, $2 ];
                    }
                    else {
                        $index_only{ $table } = 1;
                    }
                }
            }
        }

        my $CDBI   = join $DBL_COLON, 'Gramene', 'CDBI', ucfirst(lc($module));
        my $CDBIPM = join $SLASH, 'Gramene', 'CDBI', ucfirst(lc($module)) . '.pm';

        eval { require $CDBIPM };

        if ( my $err = $@ ) {
            $logger->debug("Error requiring $CDBIPM: $err");
            die $err;
        }

        $num_modules++;

        print "Removing previous data for module '$module'.\n";
        $db->do(
            'delete from module_search where module_name=?', {},
            ( $module )
        );

        TABLE:
        for my $table ( $CDBI->represented_tables ) {
            if (
                   $skip{ $table }
                || ( %index_only && !exists $index_only{ $table } )
            ) {
                next TABLE;
            }

            my $class    = table_name_to_gramene_cdbi_class( $module, $table );
            my $id_field = $class->columns('Primary');

            if ( !$id_field ) {
                print STDERR "No PK in $class\n";
                next TABLE;
            };

            my @has_a    = keys %{ $class->meta_info('has_a') || {} };
            my $skip     = join '|', $id_field, @has_a;
            my @columns  = grep { !/($skip)/ } $class->columns('All');

            if ( scalar @columns == 0 ) {
                print STDERR "No usable columns in $class.\n";
                next TABLE;
            };

            my $count
                = $class->db_Main->selectrow_array("select count(*) from $table");

            printf "\rProcessing %s record%s in '%s.%s'\n",
                commify($count),
                $count == 1 ? '' : 's',
                $module,
                $table;
```

```
my $sth = $class->db_Main->prepare("select $id_field from $table");
$sth->execute;

my @index_also
    = $index_only{ $table } && ref $index_only{ $table } eq 'ARRAY'
    ? @{ $index_only{ $table } }
    : ();

my ( $c, $i ) = ( 1, 1 );
my $max = 50;
while ( my $id = $sth->fetchrow_array ) {
    my $rec = $class->retrieve( $id );
    if ( $i == $max ) {
        $i = 0;
    }
    else {
        print "$c ", ( '.' x int(100*($c/$count)/2) ), "\r";
        $i++;
    }
    $c++;

    $num_records++;
    my $text
        = join $SPACE,
        map { defined $_ ? $_ : ()  }
        $rec->get(@columns);

    OTHER_TABLE:
    for my $other_table ( @index_also ) {
        my $other_class
            = table_name_to_gramene_cdbi_class($module, $other_table);
        my $other_id_field = $other_class->columns('Primary');

        if ( !$other_id_field ) {
            print STDERR "No PK in $class\n";
            next OTHER_TABLE;
        };

        my @other_has_a
            = keys %{ $other_class->meta_info('has_a') || {} };
        my $other_skip
            = join '|', $other_id_field, @other_has_a;
        my @other_columns
            = grep { !/($other_skip)/ } $other_class->columns('All');

        for my $other_obj ( $rec->get_related($other_table) ) {
            $text
                .= join $SPACE,
                $EMPTY_STR,
                map { defined $_ ? $_ : () }
                $other_obj->get(@other_columns);
        }
    }

    next if !$text;

    $text =~ s/^\s+|\s+$//g; # trim
    $text =~ s/\s+/ /g;      # collapse spaces
```

```
                $db->do( $insert_sql, {}, ( $module, $table, $rec->id, $text ) );
            }
            print "\n";
        }
    }

    my $num_docs = 0;
    if ( $process{'documents'} ) {
        my $ok = prompt -yn, "OK to recache documents in '$db_name'? ";

        if ( !$ok ) {
            print "Exiting.\n";
            exit 0;
        }

        my $gr_conf  = $config->get('gramene') or die "No 'gramene' config.\n";
        my $base_dir = $gr_conf->{'base_dir'} or die "No 'base_dir' setting.\n";
        my $doc_root = catdir( $base_dir, 'html' );
        my @files    = File::Find::Rule->file()
                                       ->name('*.html', '*.pdf', '*.doc')
                                       ->in($doc_root);

        print "Removing previous data for 'documents'.\n";
        $db->do('truncate table doc_search');

        my $hs = HTML::Strip->new;

        FILE:
        for my $file ( @files ) {
            ( my $path = $file ) =~ s/^$doc_root//;
            print "Indexing $path\n";

            my $contents = $EMPTY_STR;
            my $title    = $EMPTY_STR;

            if ( $file =~ /\.html$/ ) {
                my $text  = slurp $file;
                $contents = $hs->parse( $text );
                $hs->eof;

                my $p = HTML::HeadParser->new;
                $p->parse( $text );
                $title = $p->header('Title');
            }
            elsif ( $file =~ /\.doc$/ ) {
                $contents = `$ANTIWORD $file`;
                my $xml   = `$ANTIWORD -x db $file`;
                if ( $xml =~ /<title>(.*?)<\/title>/ ) {
                    $title = $1;
                }
            }
            elsif ( $file =~ /\.pdf$/ ) {
                my $pdf = CAM::PDF->new( $file ) or do {
                    warn "No PDF: $CAM::PDF::errstr\n";
                    next FILE;
                };

                for my $p ( 1..$pdf->numPages() ) {
```

```
                if ( my $page = $pdf->getPageText( $p ) ) {
                    $contents .= $page;
                }
            }

            my $pdf2  = PDF::API2->new( $file ) ;
            my %info  = $pdf2->info;
            $title    = $info{'Title'};
        }
        else {
            warn "Don't know what to do with '$file'\n";
        }

        $contents =~ s/^\s+|\s+$//g;

        if ( !$title ) {
            my @lines = grep { /\w+/ } split /\n/, $contents;
            my $first_line_len = @lines ? length $lines[0] : 0;
            if ( $first_line_len > 1 && $first_line_len < 100 ) {
                $title = $lines[0];
            }
            else {
                $title = basename( $file );
                $title =~ s/\..*$//;
                $title =~ s/_/ /g;
            }
        }

        if ( $contents ) {
            $contents =~ s/[^[:ascii:]]//g;
            $contents =~ s/\s+/ /g;

            $db->do(
                q[
                    insert
                    into   doc_search (path, title, contents)
                    values (?, ?, ?)
                ],
                {},
                ( $path, $title, $contents )
            );
            $num_docs++;
        }
    }
}

printf
    "Done, processed %s record%s in %s module%s, %s document%s.\n",
    commify($num_records),
    $num_records == 1 ? '' : 's',
    commify($num_modules),
    $num_modules == 1 ? '' : 's',
    commify($num_docs),
    $num_docs == 1 ? '' : 's',
;
```

SEARCHING

So that's how all data the from six Gramene modules and a couple thousand static documents data gets put into two search tables. It's really the more intersting part of the serach, IMO, because it manipulates the schemas and data so abstractly. The CGI search isn't really all that interesting because it just takes the query string and use MySQL ``MATCH AGAINST'' for the FULLTEXT search to retrieve the record from each module.

I divide the search results by module and show the user a summary at the top of the number of hits for each with links to them. What I think you might find interesting is:

- You can control the fields displayed for each module via the ``<list_columns>'':

  ```
  <list_columns>
    literature.reference=title,source.source_name,year
  </list_columns>
  ```

  Here I'm indicating that, when showing the results for the ``literature'' module's ``reference'' table, show just the ``title'' and ``year'' from that table plus the ``source_name'' from the record's ``source'' method (a CDBI method!). This is made possible by taking the ``module_name'' and ``table_name'' from the ``module_search'' results turning those into a Gramene::CDBI class name and then retrieving the object via the ORM, like so:

  ```
    my $class = table_name_to_gramene_cdbi_class( $module, $object )
                or die "Invalid object ($module.$object)";
    my $item  = $class->retrieve( $id ) or die "Bad id '$id'";
  ```

- You can indicate how to get to the module-specific search interfaces using the ``<view_link>'' template:

  ```
  <view_link>
      genes.gene_gene="/db/genes/search_gene?id="
  </view_link>
  ```

  This says that the ``View'' link for a ``gene_gene'' record in the ``genes'' module should put the string in quotes before the ``record_id'' (the primary key value of the record. This, of course, means that the script needs to be able to work with PK values, and many of Gramene's modules primarily use accessions, so I had to hack a couple of scripts to recognize this additional parameter.

- Each record indicates the context of the search hits a la Google.

This last point took me a few hours to work out. (Those Google guys are really pretty smart, you know.) Anyway, here's the sub I wrote to return the portions matching the search strings:

```
sub match_context {
    my ( $find, $s ) = @_;

    for my $f ( @$find ) {
        $s =~ s/($f)/{{$1}}/gi;
    }

    my $pos         = -1;
    my $match_num   = 0;
    my $max_matches = 7;
    my $last_match  = 0;
    while ( ( $pos = index( $s, '}}', $pos )) > -1 ) {
        $match_num++;
        $last_match = $pos;
```

```
            if ( $match_num == 7 ) {
                $s = substr( $s, 0, $pos + 25 );
            }
            else {
                $pos++;
            }
        }

        if ( $match_num < $max_matches && length($s) - $last_match > 50 ) {
            $s = substr( $s, 0, $last_match + 25 );
        }

        $s =~ s/^.*?((?:\S+\s+){2}\S*[{]{2})/...$1/xms;
        $s =~ s/
            ([}]{2}\S*)
            ((?:\s+\S+){3})
            .{$MAX_CHARS_MATCH_CONTEXT,}?
            ((?:\S+\s+){3})
            (\S*[{]{2})
            /$1$2...$3$4/gxms;

        $s =~ s/[{]{2}/<b>/g;
        $s =~ s/[}]{2}/<\/b>/g;

        return $s;
    }
```

First I put double braces around each search hit, then I cut off the text after a maximum of seven matches. Then I remove any long bits of text between matches and try to add a couple words around them. Then I turn my double braces into HTML ``bold'' tags.

This is all pretty neat, except that I'm not matching in quite the same way that MySQL's FULLTEXT engine does -- it can do nifty things like word stemming and soundex and similarity and misspellings, and I'm doing exact matches. However, it's better than nothing.

PROBLEMS

One drawback I'm dealing with is the enormity of Gramene's markers db, which currently has 13.5M markers with 38.4M marker synonyms. To load the markers db, I would have to aggregate millions of records (and a trial run did 150K in 12 hours) and add 13.5M records to the search schema. Given that most people search the marker names and that the markers schema was optimized around just such searches, it makes no sense to denormalize all this data and bloat the search db, thereby slowing it down. So I think I will probably just tap the markers search API directly when I get to including markers in the quick search.

Another possible problem is that this search might work differently from the module-specific searches, possibly confusing the user. This is especially true of the similarity-based matches that the FULLTEXT engine will find, e.g., a search for ``Texmont'' will also return ``Rosemont,'' whereas a stricter field-based search common in the other interfaces would never find such a match.

SUMMARY

So I've tried to show you how I do a lot of fancy manipulation of data using very high-level interfaces that are generated by underlying code. Here's a little graph showing how all those early foundation pieces fit together:

```
    mk-gcdbi.pl
         |
         v
    SQL::Translator          gramene.conf
```

```
       |                           ^
       v                           |
MySQL db (markers)        Gramene::Config
       |                           ^
       v                           |
Gramene::CDBI::Markers  <-- Gramene::DB
```

Once those CDBI classes are in place, it makes it easy to abstract the logic for digesting the data into the search schema and then turning around and making the abstractions back into objects using the ORM for purposes of displaying them.

I hope you've enjoyed yourselves. Be sure to tip your waiter and bartender. I'll be here all week.