A Perl API for the CSQL System

Mark Ito, Clarisse Tur

January 26, 2004

1 Introduction

The CSQL system allows data analyzers to track the progress of running jobs and record the results of completed jobs using a relation database to store a user-specified collection of quantities generated by the analysis routines. The idea was proposed by Arne Freyberger, first implemented by Gagik Gavalian and later refined by Gagik and Harut Avagyan. The system is described in a CLAS note written by Gagik[1]. In the note a C API for defining the data tables and inputting values is described, as well as a web-based browser for looking at results.

This note describes a Perl API, implemented as a Perl module, that allows Perl scripts to easily create tables and input data in the CSQL system. In particular, the Perl module provides an upgrade path for scripts that use the perldb module, allowing data analyzers to access the increased power and flexibility a relational database in a context where the browsing tools have already been developed.

2 Getting Started

We will assume that you have already created a database on an appropriate server as described in Ref. [1]. Now there are two tasks you will want to perform.

1. Create a database table and define the columns.
2. Create a new row in the table and insert data.

In the following sections we will develop an example that does all both.

2.1 Creating a new table

This step needs to done once for each project, *e. g.*, once for each cooking pass, and need not be run for each job in the project. The example creates a table with three columns, the total number of tracks in the job, a resolution for some fictitious quantity and the total number of protons. The table is called test_csql and it is put into the g7_offline database. Note that the specification of the columns is done by creating an array of hashes, `@table_spec` in this example. Each member of the array is a hash that that has two "indices", name and type, whose values are the name of the columns and its type. This hash is passed to the subroutine CreateTable along with the name of the table.

```
#!/usr/bin/env perl

use lib ("$ENV{CLAS_TOOLS}/perl");
use Csql;

$hostname = 'clasdb.jlab.org';
$user = 'offline_g7';
$database = 'g7_offline';
ConnectToServer($hostname, $user, $database);

# fill table specification
@table_spec = (
       {
   name => "tracks",
   type => "int"
   },
       {
   name => "resolution",
   type => "float"
   },
       {
   name => "protons",
   type => "int"
   }
       );

# name the table
$table_name = "test_csql";

# create it
CreateTable($table_name, @table_spec);

DisconnectFromServer();

exit;
```

2.2 Creating a row and adding data

In this example a row is created in the csql_test table. The row is identified by the run and run extension, `$runno` and `$runext` here, and created by the subroutine `InsertRow`. Note that no data is put into the row when it is created.

Next the data is defined, again as an array of hashes. There the hash indices are the column names and the data values, `name` and `value`. The array, `@row` here, is passed to the subroutine `UpdateRow`.

This example is a bit artificial; usually the run, run extension and data values will be

input to the script by some other means, not by being defined explicitly as is done here.

```
#!/usr/bin/env perl

use lib ("$ENV{CLAS_TOOLS}/perl");
use Csql;

$hostname = 'clasdb.jlab.org';
$user = 'offline_g7';
$database = 'g7_offline';
ConnectToServer($hostname, $user, $database);

# create the new row

$table_name = "test_csql";
$runno = 12345;
$runext = 1234503;

InsertRow($table_name, $runno, $runext);

# put data into it

@row = (); # clear the array

%column = {}; # clear the hash
$column{name} = "tracks"; # the column name is tracks
$column{value} = 3000;  # there were 3000 tracks
push(@row, {%column}); # add hash as an element of the array

%column = {}; # clear the hash again, its contents have been pushed
$column{name} = "resolution";
$column{value} = 234.56;
push(@row, {%column});

%column = {};
$column{name} = "protons";
$column{value} = 100;
push(@row, {%column});

UpdateRow(@row);

# finish up

DisconnectFromServer();
```

```
exit;
```

3 The Subroutines

The subroutines are implemented as a Perl module. The CVS home of the module is `tools/perl/Csql.pm`. On the JLab CUE, the module is checked out as `/group/clas/tools/perl/Csql.pm` also known as `$CLAS_TOOLS/perl/Csql.pm`.

3.1 ConnectToServer

Makes the connection to the database server. Must be called at the beginning of all scripts.

Usage:

```
ConnectToServer(<hostname>, <user>, <database>);
```

Arguments:

`host:` name of the database server.

`user:` username on the server.

`database:` name of the database.

3.2 DisconnectFromServer

Disconnects from the server. Should be called at the end of all scripts.

Usage:

```
ConnectToServer();
```

Arguments: no argument for this subroutine

3.3 CreateTable

Creates a table in the chosen database. Defines all columns.

Usage:

```
CreateTable(<table name>, <column specification>);
```

Arguments:

`table name:` name of the table.

`column specification:` names and types of all columns. Specified as an array of hashes. Each hash should have two elements, a `name` element, which specifies the column name and a `type` element which specifies the MySQL data type.

3.4 InsertRow

Creates a new row in the chosen table. Does not enter any data into the row.

Usage:

```
InsertRow(<table name>, <run number>, <run extension>);
```

Arguments:

`table name`: name of the table.

`run number`: the run number being processed.

`run extension`: the file index of the file being processed. This identifies the file within multi-file runs.

Note: The CSQL system assumes that run number and run extension are defined for all tables.

3.5 UpdateRow

Adds data to the previously created row. Assumes that `InsertRow` has been called previously in the script and updates the columns in that row with data. `UpdateRow` may be called multiple times in a given script. For a given call, not all of the columns in the row need to be specified. If a particular column was updated in a previous call to `UpdateRow`, the current call will overwrite that data. Each time `InsertRow` is called, all subsequent calls to `UpdateRow` will operate on the new row.

Usage:

```
InsertRow(<row data>);
```

Argument:

`row data`: data to be put into the row. Specified as an array of hashes. Each hash should have two elements, a `name` element, which specifies the column name and a `value` element which specifies the data value.

References

[1] Gagik Gavalian, CLAS-NOTE 2002-011.

`$Id: csql_perl.tex,v 1.6 2004/01/26 18:17:51 marki Exp $`