C interface to mySQL for monitoring and storing CLAS data analyses results

Gagik Gavalian, UNH (2002)

1 Introduction

The CSQI is a package that allows to store characteristic results from the “cooking” of CLAS data into a mySQL database. The produced tables can be used for monitoring the cooking process as well as for post-cooking analysis of run period. CSQI has been recently statically linked to RECSIS (standard distribution) and can be activated with a TCL switche (explained later). There are two basic functionalities of CSQI: first - real time generated tables to monitor progress of the “cooking”, second - static final tables which store run related statistics in the database table.

A web interface exists for quick data displaying and table viewing.

The CSQI package can also be used as a standalone library. This document also provides CSQI API documentation to help in developing of any stand alone codes.

2 RECSIS switches and ENV variables

To activate CSQI package in RECSIS one needs to set a switche in the TCL input file. The following switche is available:

\[
\text{set lmysql=-1; // activates csqI package (default is = 0;)}
\]

Before starting the “cooking” process one needs to set up several environment variables needed by the CSQI package. Following variables are defined:
<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSQ_LDBHOST</td>
<td>database host name</td>
</tr>
<tr>
<td>CSQ_DB</td>
<td>database name</td>
</tr>
<tr>
<td>CSQ_USER</td>
<td>the user name of database account</td>
</tr>
<tr>
<td>CSQ_TABLE</td>
<td>final result table</td>
</tr>
<tr>
<td>CSQ_DDL</td>
<td>ddl file (for the group) to define columns for reconstruction related variables e.g. Ne,Np, Nd</td>
</tr>
<tr>
<td>CSQ_CALIB</td>
<td>ddl file to define column names for calibration related parameters, e.g. RF mean, RF sigma ..</td>
</tr>
<tr>
<td>CSQ_COM</td>
<td>specify comment</td>
</tr>
</tbody>
</table>

**NOTE:** There is no password required. Cooking account must be without password.

CSQL package also stores information about calibration database used for the cooking. Environmental variables that are defined for RECSIS are filled into final database for future reference. This variables are:

<table>
<thead>
<tr>
<th>CLAS_CALDB_HOST</th>
<th>calibration database host</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLAS_CALDB_RUNINDEX</td>
<td>RunIndex of constants</td>
</tr>
<tr>
<td>CLAS_CALDB_TIME</td>
<td>time stamp</td>
</tr>
</tbody>
</table>

After all those variables are set, RECSIS can be run with modified TCL file. Currently there are 3 groups defined in the default table. This groups are “SYST”, “CSQL” and “CALB”.

- “SYST” group contains run-time system information (i.e. user name, process id, node name). A typical printout from recsis can be found in Appendix G.

- “CSQL” group contains run statistic information such as number of events processed, number of electrons reconstructed etc. (see Appendix F)

- “CALB” group contains calibration quality information for different detectors (i.e. RF mean, RF sigma, EC time resolution etc.) (see Appendix F)

Information in “SYST” goes from defined enviromental variables. “CSQL” group is filled in “ana” package. Values of parameters in “CALB” are filles in the end of the run by “user_ana”.

### 3 Viewing Database

Once you have configured and run the produced monitoring database can be viewed using either a standard WEB Browser (at http://clasweb.jlab.org/csquilb/) or ROOT
based tools which can be downloaded from the same web page.

There are some predefined graphs you can access through menu buttons of RootC-SQLTools such as number of electrons reconstructed per run, RF mean and RF sigma for each run, number of files “cooked” hourly and daily etc. On Figure 1 hourly and daily “cooked” file statistics can be seen for 1d run period.

![Cooked files per hour](image1)

![Cooked files per day](image2)

Figure 1: Time graph produced by RootCSQLTools

The web based interface has already defined database settings for each run group (to create a new one, please, contact clasdb administrator). On the Figure 2 the starting page can be seen where run group can be selected. On the next page (after hitting submit) the menu will let the user to select table within the selected database (not on the figure). After selecting a run group and the table the full set of table variables will be shown. One can select any combination of these variables to be viewed, sorted by any of the variables. The number of rows per page and column name for sorting can be pulled from given buttons (Figure 3).

On the Figure 4 the resulting table of some selection is displayed.

On the Figure 3 screen just below the table with column names and options there is a link (not in the Figure) to the page where dynamic plots can be viewed by selecting column on X axis and Y axis and adjusting some options.

On Figure 5 (in the Appendix E) an example plot can be seen. Simply choose column name in the “X axis” and “Y axis” menu and press submit. One can also
normalize “Y axis” value to other column in the table. In the case of Figure 5 number of reconstructed electrons were normalized to number of processed triggers (NPROC). The scale of the graph can be changed if desired. Just enter minimum and maximum values of “Y axis” in the input boxes then press “Submit”.

4 CSQL Package description

CSQL package can be obtained from any CUE account at JLAB by checking out from CLAS CVS tree (use: cvs co c_sql). The TOP_DIR environment variable should be set to point to the directory where the compiled libraries will appear. The package is written in C and has wrappers for FORTRAN code.
Figure 3: This page allows to choose columns from the selected tables and display them in the Browser

4.1 Initializing the library

Once the library has been compiled the CSQL package must be initialized the library before any call to its functions is made. It is done by calling `init_csdl()` functions.

*NOTE:* If one is using this library as a stand alone or on network other than JLAB one should set database parameters (i.e. host, database, user) by calling `set_database(char *hostname, char *username, char *password, char *DB_name)` since `init_csdl()` sets default values (for JLAB users) and one may get execution error trying to connect to databases without having permission.

When using as part of CLAS software `initclas()` function can be used to initialize library from environmental variables (see section RECSIS switches and ENV variables).

4.2 Creating Groups

Groups are collection of related variables. Each of them can contain different number and types of variable. For instance, BEAM group can contain information
about the beam energy, the beam current e.t.c., ECCH parameters related to EC (Electromagnetic Calorimeter). The names of groups are limited to 4 characters (following BOS convention).

After the package has been initialized one can create groups. It is done by add_group( const char *name) where ‘name’ is a 4-character string. During one session one can create as many as 20 groups (hard coded limitation, that can be changed in the source code). After creating groups one can print them out at execution time with print_all_groups(); which will display all runtime defined groups.

Groups can also be created from a file in DDL format (DDL is a format used at CLAS for formatting BOS banks). For more information about creating groups from DDL file see the next section.

Example 1:

/* this example demonstrates how to create groups and view them on the screen */
#include ‘<stdio.h>’
#include ‘<csql.h>’
int main()
{
    init_sql();
    add_group("BEAM");
    add_group("SYST");
    print_all_groups();
}

4.3 Creating Columns

Columns are variables within a group. Each group can contain as many as 52 columns (elements). Each column is associated with one group and specified by name, type (integer, float or character) and mode. Mode defines weather the variable will be created and written in the table, and it can be changed at run time. There are two following ways of creating a new column:

1. add_column(const char *group, const char *colname),

   where the "group" is the group name that column will be associated with and "colname" is the name of the column.

   NOTE: The column names do not have 4 character length limitation. They can be declared within 64 character length.

2. init_column(const char *group, const char *colname, const char *type, int itype, int mode)

This call will create a new column with within group "group" with name "colname" with given type and mode. The variable "type" is a character string according to mySQL syntax (see documentation of mysql) defining the type of the variable that will be used for the column in the database. Type commonly used by mySQL are "INT","FLOAT" and CHAR(32) (means character string with 32 length)

As mentioned above, the groups and columns can be created from a file. The file must be written in DDL format. By calling

n_columns = read_ddlfile2(const char *filename)

the package will parse "filename" and create group with a name specified in the file, and initialize all columns contained in the file within the created group.

The results of this can be seen by calling

print_group(char *grpname)
which will produce table type output on the display with content (columns) of the
group with its types and values.

Example 2:

```c
#include "stdio.h"
#include "csq1.h"
int main(){
    init_sql();
    add_group("BEAM");
    init_column("BEAM","Energy",FLOAT,COL_TYPE_FLOAT,COL_CW_TBL);
    init_column("BEAM","RunNo",INT,COL_TYPE_INT,COL_CW_TBL);
    init_column("BEAM","Comment",CHAR(32),32,COL_CW_TBL);
    print_group("BEAM");
}
```

4.4 Storing results in columns

The simplest way to set the column value is to call one of these three functions

```c
set_column_int(char *group, char *col, int value)
set_column_float(char *group, char *col, float value)
set_column_char(char *group, char *col, char *value)
```

It is important to remember the type of declared variables. If one declares it as
an INTEGER, then tries to assign it value with `set_column_float(...)`, the value of
the column will not be changed.

If one is using CSQ1 package with BOS files (like RECSIS) one can also assign
values to all columns within a "group" if the group has been initialized from a DDL
file (see `read_ddlfile2(...)`). One needs to pass a pointer to the first element of
the array containing data. This works only if there are INT and FLOAT columns in
the group.

Example 3:

```c
/* This example will create a group and columns in the group
then assign values to the columns and print them on the screen
*/
#include "stdio.h"
#include "csq1.h"

int main(){
    init_sql();
```
add_group(‘BEAM’);
init_column(‘BEAM’, ‘Energy’, FLOAT, COL_TYPE_FLOAT, COL_CW_TBL);
init_column(‘BEAM’, ‘RunNo’, INT, COL_TYPE_INT, COL_CW_TBL);
init_column(‘BEAM’, ‘Comment’, CHAR(32), 32, COL_CW_TBL);

/* after creating columns let’s assign them values */

set_column_int(‘BEAM’, ‘RunNo’, 32456);
set_column_float(‘BEAM’, ‘Energy’, 12.05); // in couple of years
set_column_char(‘BEAM’, ‘Comment’, ‘Upgraded CLAS setup’);
print_group(‘BEAM’);
}

Here is an example how to initialize a group from DDL file, then fill it from an
array or BOS bank. It is not required to use a BOS bank as source of data, one can
also define an array which contains the given structure.

Example 4:
Here is an example of DDL file (/home/joe/csql.ddl)

<table>
<thead>
<tr>
<th></th>
<th>BANKname</th>
<th>BANKtype</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE</td>
<td>CSQL</td>
<td>B32</td>
<td>Data bank for mySQL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>!COL</td>
<td>ATT-name</td>
<td>FMT</td>
<td>Min</td>
</tr>
<tr>
<td>1</td>
<td>EVID</td>
<td>I</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>NPRC</td>
<td>I</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>CPU</td>
<td>F</td>
<td>0.</td>
</tr>
<tr>
<td>4</td>
<td>FC</td>
<td>F</td>
<td>0.</td>
</tr>
<tr>
<td>5</td>
<td>FCG</td>
<td>F</td>
<td>0.</td>
</tr>
<tr>
<td>6</td>
<td>TG</td>
<td>F</td>
<td>0.</td>
</tr>
<tr>
<td>7</td>
<td>IBEAM</td>
<td>F</td>
<td>0.</td>
</tr>
</tbody>
</table>

The lines with ! at the first column are ignored by the parser. The third line
defines GROUP with name CSQL (B32 tells BOS that 32 bit format to be used).
Then variables are listed with following “I” or “F” (INTEGER or FLOAT). Min and
Max columns are ignored.

For the FORTRAN code examples see Appendix.
4.5 Writing Data to Database

Once all necessary GROUPS and COLUMNS have been created and data is assigned to COLUMNS, one can transfer the data into a database table. The CSQL package provides routines to write data into existing table in the database as well as to create a new one according to GROUP information defined run-time. If one tries to write data into an existing table only columns which names coincide with ones defined in the table will be exported into database, other will be omitted.

```c
fill_table(const char *tablename, const char *group_list)
```

exports data from GROUPS given by ‘‘group_list’’ into a table ‘‘tablename’’.

groups_list is a list of groups (i.e. “BEAMCSQLEPC”).

Then fill_table(...) is called it first looks up the database to see weather table with a given name exists or not. If not, it will create one using column names and the types from the specified groups. If table exists it will extract column names from existing table and will compare with ones in the given GROUPS. Only those column values will be transfered to the existing table which have corresponding column name in the already existing table.

Example 5:

```c
/* add the following lines in the end of Example 3 and the program will
write the output into database
*/
...
...
...
fill_table(‘‘mytable’’, ‘‘BEAM’’);
}
```

4.6 Monitoring tables

As mentioned above the CSQL package can also be used for monitoring running processes to extract dynamic data for long lasting jobs. In the package the is a default table name defined constructed from node name and process id of currecnt job. There is a prefix “mon_” added to default table name. Example: If is is running a job on host “farm12.min.edu” the default table name will be “mon_farm12_3456” (where 3456 is the process id). Since there can never be 2 jobs running on the same machine with the same process id the default table name is always unique. This tables must be deleted after job finished to aviod later confusions.

The monitoring tables are usefull if one has long lasting job running and would like to monitor its progress. The table must be updated from the code.
The

\texttt{fill\_mon\_table(char *group\_list)}

function creates a table with default name (if it does not exist) and then fill content of groups given by \texttt{“group\_list”} into the table.

To delete created monitoring table one may use:

\texttt{delete\_mon\_table()}

This will delete table with default name from the database if such exists.

5 \textbf{Function Reference}

- \texttt{void add\_column(const char *group, const char *colname)}
  Adds a column with name \texttt{“colname”} to the group \texttt{“group”} and initializes it with default parameters \texttt{type=“INT”, mode=COL\_CW\_TBL}

- \texttt{void init\_column(const char *group, const char *colname, const char *type, int itype, int mode)}
  Adds column to the Group and sets up the type and the mode of the column.

- \texttt{void add\_group(const char *name)}
  Creates a new \texttt{GROUP} with given name. Group name follow 4 character length convention. If \texttt{“name”} is more than 4 characters only first 4 will be used as a group name.

- \texttt{int connect\_mSQL\_server(MYSQL *mysql, char *DB\_hostname, char *DB\_username, char *DB\_name, char *DB\_passwd)}
  Creates a connection with a database and returns 1 if successful, -1 otherwise.

- \texttt{void disconnect\_mSQL\_server(MYSQL *mysql)}
  Disconnects from the database.

- \texttt{void create\_table(const char *tablename, const char *group\_list)}
  Creates table in the database already defined runtime (see \texttt{set\_database(…)}) with given name.
• int is_table(const char *tablename)
  Checks default database for existence of a table with name "tablename". Returns 1 if table exists and -1 otherwise.

• void delete_table(const char *tablename)
  Deletes table with given name from default database.

• void fill_table(const char *tablename, const char *group_list)
  fills table from groups given by "group_list". If table does not exist, one will be created with column names corresponding to columns in the groups. If table already exists, only column from "group_list" that have corresponding column names in the table will be inserted into table.

• void fill_mon_table(const char *group_list)
  fills a defualt monitoring table from groups given by "group_list". The table name will be mon_[node name]_[process id].

• void init_csql()
  Initializes CSQL package enviroment. This function must be called before you refer to any other function in this package.

• void set_database(char *hostname, char *username, char *password, char *DB_name)
  This function is used to set default database host, user name, password and database name for the package. NOTE: Most of the function in the package use this parameters to connect to database and read and write from it. So this must be probably the second call after init_csql() before one starts to read from database or write into one.

• void print_all_params()
  Prints on the display all internal parameter variables. Very usefull to have it printed out before connecting to database to make sure you have set them up properly.

• int read_ddlfile2(const char *filename)
  Reads DDL file and creates a "GROUP" with a group name and columns specified in the DDL file.
• void set_column_int(const char *group, const char *col, int value)
  Sets the value of column "col" in the group "group" to "value". NOTE: The
  package does not check if you call this function for a variable of INT type so
  you have to take care to not make a mistake there.

• void set_column_float(const char *group, const char *col, float value)
  Sets the value of column "col" in the group "group" to "value". NOTE: The
  package does not check if you call this function for a variable of FLOAT type
  so you have to take care to not make a mistake there.

• void set_column_char(const char *group, const char *col, const char *
  value)
  Sets the value of column "col" in the group "group" to "value". NOTE: The
  package does not check if you call this function for a variable of CHAR type so
  you have to take care to not make a mistake there.

• void set_group(char *grpname, char *buffer)
  This function is used in conjunction with BOS. First a group must be initialized
  with read_dd1file2(...) then the pointer to the first element of BOS data
  should be passed to the routine. It will automatically fill the columns in the
  group with values from a buffer. Only INT and FLOAT types are supported
  for this operation.

• void get_user_name(char *u_name)
  This function returns user name of the process owner. The "u_name" must be
  initialized before this call and has to have at least 9 characters in length (one
  might use even longer string to be safe).

• void get_table_name(char *tbl_name, char *nodename, int max_len)
  This function is used for monitoring purposes. It returns a unique table name
  constructed from node name, user name and process id. So if one's running
  different jobs on same or different machines and would like to monitor them
  runtime by creating a table for each job to view progress of the programs this
  routine might come handy in defining different table names for each process.
  Input parameters are: "nodename" - character string representing the host,
  "max_len" maximum length of the table name. Output: "tbl_name" - table
  name to be used. NOTE: variable "tbl_name" must be initialized with length
  "max_len" before calling this function.
6 Appendix A: Column Types and Modes

Here is the list of column types and various modes used in the CSQDL package with their numerical values and descriptions. This constants are used when initializing column parameters (see init_column(...)). The names of this variables are defined in the C include file hence can not be used from fortran code. One has to use numerical values from FORTRAN code.

Types:

<table>
<thead>
<tr>
<th>Constant name</th>
<th>Num Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COL_TYPE_INT</td>
<td>1</td>
<td>column of an integer type</td>
</tr>
<tr>
<td>COL_TYPE_FLOAT</td>
<td>3</td>
<td>column of a float type</td>
</tr>
</tbody>
</table>

NOTE!: If CHARACTER type column is defined the type must be set to be equal to length of the CHAR string. See examples for more details.

Defined modes:

<table>
<thead>
<tr>
<th>Constant name</th>
<th>Num Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COL_C_TBL</td>
<td>1</td>
<td>Column will be created in the table but will not the value of it will not be written in the table (Used for TIMESTAMP variables)</td>
</tr>
<tr>
<td>COL_CW_TBL</td>
<td>2</td>
<td>Column will be created and filled into table</td>
</tr>
<tr>
<td>COL_PRIKEY</td>
<td>3</td>
<td>Column will be declared as a PRIMARY KEY</td>
</tr>
<tr>
<td>COL_PRIKEY_AUTO</td>
<td>4</td>
<td>Column will be declared as a PRIMARY KEY and will be assigned values automatically</td>
</tr>
</tbody>
</table>

7 Appendix B: Example of C code

A slightly more complicated example of C code that includes all basic types of GROUP-COLUMN operations.

```
#include <stdio.h>
#include "csq1.h"

int main()
{
    // This part defines an array of integers that can
```
// store integers and floats at the same time.
// This is same as FORTRAN's EQUIVALENT command

int    data[4];
float  *data_f = (float *) &data[0];

// Now we set values in the array. Of course one needs to know
// structure of the DDL file that will be used to initialize this group.
//
data[0]  = 12000;
data_f[1] = 4.53;
data[2]  = 8000;
data_f[3] = 145.678;

init_csql();
set_database("clasdb","offline_e1d","","e1d_offline");

add_group("TORS"); // group to describe Torus run-tim parameters
add_group("BEAM"); // Beam characteristics
add_group("SYST"); // Group containing run-time system information

// Adding columns in the TORS group
add_column("TORS","Setting");
add_column("TORS","Status");

// Adding columns in the BEAM group
add_column("BEAM","Energy");
add_column("BEAM","Current");
add_column("BEAM","Comment");

// Adding columns in the SYST group
add_column("SYST","system");

// now initializing columns with types and modes
init_column("TORS","Setting","FLOAT",COL_TYPE_FLOAT,COL_CW_TBL);
```fortran
init_column("TORS","Status","INT",COL_TYPE_INT,COL_CW_TBL);
init_column("BEAM","Energy","FLOAT",COL_TYPE_FLOAT,COL_CW_TBL);
init_column("BEAM","Current","FLOAT",COL_TYPE_FLOAT,COL_CW_TBL);
init_column("BEAM","Comment","CHAR(20)",20,COL_CW_TBL);
init_column("SYST","system","CHAR(32)",32,COL_CW_TBL);

set_column_float("TORS","Setting",2550.3);
set_column_int("TORS","Status",4);

set_column_float("BEAM","Energy",4.182);
set_column_float("BEAM","Current",7.5);
set_column_char("BEAM","Comment","eild run period");

set_column_char("SYST","system","Red Hat 7.3");

// Example of defining a group from the DDL file and then setting // values of it from an already defined array
read_ddlfile2("test.ddl");
set_group("DDL","(char *) &data[0]");

// Printing all group onto screen
print_all_groups();

// and finally let as write a table into datavase that we have // already selected in the beginning of the code
fill_table("my_first_table","SYSTBEAMTORSDDLF");

return 0;
}
```

### 8 Appendix C: Example of FORTRAN code

Example in FORTRAN that reads group from the DDL file (see appendix D), fills columns with values then exports them into a database table.

*NOTE!* when compiling a FORTRAN code on Linux one needs to use “-fno-second-underscore” flag.
PROGRAM CSQL_TEST

REAL RW(4)
INTEGER IW(4)
EQUIVALENCE (IW(1),RW(1))

C- Declaring two arrays INT and FLOAT in the same memory block
WRITE(*,*)’Starting CSQL test program’

C- Setting initial values

IW(1) = 45670
IW(3) = 32890
RW(2) = 45.6
RW(4) = 134.567

C- Firrst the CSQL package needs to be initialized
CALL INIT_CSQl

C- Reading DDL file. See Appendix for DDL file printout
CALL READ_DDLFILE2(’test.ddl’)

C- At this point all variables in the group will have
C- values set to 0
CALL PRINT_ALL_GROUPS()

C- now let’s copy values from array into columns
CALL SET_GROUP(’DDLF’,IW(1))

C- Print it again to make sure values were transfered OK.
CALL PRINT_ALL_GROUPS()
CALL FILL_TABLE(’my_fort_table’,’DDLF’) END

9 Appendix D: DDL file used in the example codes

!! TABLE DDLF B32 ! Comment ! 1 NPROC I 1 10000 ! 2 CPU F 0. 999. ! 3
NRec I 1 1000 ! 4 FCUP F 0. 999. !! END TABLE
10 Appendix E: Plot generated on the CSQL WEB Page

Figure 5: Plot Run number vs number of reconstructed electron per trigger
## Appendix F: CSQL and CALB DDL files used in RECSYS

This ddl files can be found on JLAB CUE nodes in `$CLAS_PARMS` directory.

### CSQL DDL file:

```sql
!-----------------------------------------------
! BANKname BANKtype !Comments
TABLE    CSQL    B32 ! create write display delete ! Data bank for MySQL

!COL ATT-name FMT Min Max !Comments
1 EVID I 1 100000 ! Event ID (number of triggers)
2 NPROC I 1 100000 ! Number of processed triggers
3 CPU F 0. 99999. ! CPU used (sec)
4 FC F 0. 999. ! Faraday Cup (K)
5 FCN F 0. 999. ! Faraday Cup Gated (K)
6 TG F 0. 999. ! Clock Gated
7 IBEAM F 0. 999. ! Beam current
8 NeS1 I 0 100000 ! Number of electrons in sect 1
9 NeS2 I 0 100000 ! Number of electrons in sect 2
10 NeS3 I 0 100000 ! Number of electrons in sect 3
11 NeS4 I 0 100000 ! Number of electrons in sect 4
12 NeS5 I 0 100000 ! Number of electrons in sect 5
13 NeS6 I 0 100000 ! Number of electrons in sect 6
14 NhN I 0 1000000 ! Number of HB
15 NhT I 0 1000000 ! Number of TB
16 Nprot I 0 1000000 ! Number of protons
17 Npip I 0 1000000 ! Number of pip
18 Ndeut I 0 1000000 ! Number of deuterons
19 Nphot I 0 1000000 ! Number of photons
20 Nelhp I 0 1000000 ! Number of electrons at pos. Helic.
21 Nelhn I 0 1000000 ! Number of electrons at neg. helic.

!-----------------------------------------------
END TABLE
```

### CALB DDL file:

```sql
!-----------------------------------------------
! BANKname BANKtype !Comments
TABLE    CALB    B32 ! create write display delete ! Monhist fit results for MySQL
```
12 Appendix G: RECSIS output with csql switch on

Here is a typical output produced by RECSIS-CSQL

Programs Initialized with following parameters..

PAR=> DATABASE HOST : clasdb.jlab.org
PAR=> DATABASE USER : offline_e2b
PAR=> DATABASE DB : e2b_offline
PAR=> MONITOR TABLE : mon_ifarm11_11150
PAR=> DEFAULT TABLE : e2b_pass0_00
PAR=> DDL FILENAME : /group/clas/builds/PRODUCTION/packages/bankdefs/csdl.ddl
PAR=> CALIB DATABASE : default
PAR=> CALIB RUNINDEX : calib_user.RunIndexte2b
PAR=> CALIB TIMESTAMP : 20370101220926

<table>
<thead>
<tr>
<th>Group [SYST] Columns</th>
<th>Column Type</th>
<th>TYPE</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>TIMESTAMP(14)</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>user</td>
<td>CHAR(12)</td>
<td>2</td>
<td>pass0_cooking</td>
</tr>
<tr>
<td>jobname</td>
<td>CHAR(32)</td>
<td>2</td>
<td>ifarm11</td>
</tr>
<tr>
<td>calibdb</td>
<td>CHAR(64)</td>
<td>2</td>
<td>default</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td>----</td>
<td>---------</td>
</tr>
<tr>
<td>runindex</td>
<td>CHAR(32)</td>
<td>2</td>
<td>calib_user.RunIndexe2b</td>
</tr>
<tr>
<td>timestamp</td>
<td>CHAR(16)</td>
<td>2</td>
<td>20370101220926</td>
</tr>
<tr>
<td>runno</td>
<td>INT</td>
<td>2</td>
<td>32973</td>
</tr>
<tr>
<td>runext</td>
<td>INT</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>GROUP [SYST] has 9 columns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Group [CSQL] Columns</td>
<td>Column Type</td>
<td>TYPE</td>
<td>Value</td>
</tr>
<tr>
<td>--------</td>
<td>--------------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>EVID</td>
<td>INT</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>NPROC</td>
<td>INT</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>CPU</td>
<td>FLOAT</td>
<td>2</td>
<td>0.000000</td>
</tr>
<tr>
<td>FC</td>
<td>FLOAT</td>
<td>2</td>
<td>0.000000</td>
</tr>
<tr>
<td>FCG</td>
<td>FLOAT</td>
<td>2</td>
<td>0.000000</td>
</tr>
<tr>
<td>TG</td>
<td>FLOAT</td>
<td>2</td>
<td>0.000000</td>
</tr>
<tr>
<td>IBEAM</td>
<td>FLOAT</td>
<td>2</td>
<td>0.000000</td>
</tr>
<tr>
<td>NeS1</td>
<td>INT</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>NeS2</td>
<td>INT</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>NeS3</td>
<td>INT</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>NeS4</td>
<td>INT</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>NeS5</td>
<td>INT</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>NeS6</td>
<td>INT</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Nhb</td>
<td>INT</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Ntb</td>
<td>INT</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Nprot</td>
<td>INT</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Npip</td>
<td>INT</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Ndeut</td>
<td>INT</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Nphot</td>
<td>INT</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Nelhp</td>
<td>INT</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Nelhn</td>
<td>INT</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>GROUP [CSQL] has 21 columns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Group [CALB] Columns</td>
<td>Column Type</td>
<td>TYPE</td>
<td>Value</td>
</tr>
<tr>
<td>--------</td>
<td>--------------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>meanRFe</td>
<td>FLOAT</td>
<td>2</td>
<td>0.000000</td>
</tr>
<tr>
<td>sigmaRFe</td>
<td>FLOAT</td>
<td>2</td>
<td>0.000000</td>
</tr>
<tr>
<td>sigmaRFe</td>
<td>FLOAT</td>
<td>2</td>
<td>0.000000</td>
</tr>
<tr>
<td>sigmaEFe</td>
<td>FLOAT</td>
<td>2</td>
<td>0.000000</td>
</tr>
</tbody>
</table>

21
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SFECe</td>
<td>FLOAT</td>
<td>2</td>
<td>0.000000</td>
<td></td>
</tr>
<tr>
<td>sigmaSF</td>
<td>FLOAT</td>
<td>2</td>
<td>0.000000</td>
<td></td>
</tr>
<tr>
<td>ResSL1</td>
<td>FLOAT</td>
<td>2</td>
<td>0.000000</td>
<td></td>
</tr>
<tr>
<td>ResSL2</td>
<td>FLOAT</td>
<td>2</td>
<td>0.000000</td>
<td></td>
</tr>
<tr>
<td>ResSL3</td>
<td>FLOAT</td>
<td>2</td>
<td>0.000000</td>
<td></td>
</tr>
<tr>
<td>ResSL4</td>
<td>FLOAT</td>
<td>2</td>
<td>0.000000</td>
<td></td>
</tr>
<tr>
<td>ResSL5</td>
<td>FLOAT</td>
<td>2</td>
<td>0.000000</td>
<td></td>
</tr>
<tr>
<td>ResSL6</td>
<td>FLOAT</td>
<td>2</td>
<td>0.000000</td>
<td></td>
</tr>
</tbody>
</table>

+------------------------------------------------------------------------+
| GROUP [CALB] has 12 columns                                           |

Booking seb histos

+++++++++++++++++++++ EC1 booking ++++++++++++++++++++

Read EC pedestals from Map - Run 32973