



SQLPlotter : Developing a web-interface for a physics analysis database

August 2014

Author:

Sneha

Sneha.snehaindia@gmail.com

Supervisor:

Dr. Maaike Limper

CERN openlab Summer Student Report 2014



CERNopenlab

Project Specification

SQLPlotter is a C++ macro written using ROOT classes which has previously been developed to allow the possibility of making histograms from the results of SQL-queries running on physics analysis data stored in a relational database.

This project is aimed at developing a web – interface for this SQLPlotter with a ROOT-installation running locally on the web-server, eliminating the requirement of learning ROOT in depth and providing the users with an easy to use interface that can be accessed anytime, anywhere to do physics analysis inside a database.

Abstract

As part of the CERN openlab collaboration a study was made into the possibility of performing analysis of the data collected by the experiments at the Large Hadron Collider (LHC)^[1] through SQL-queries on data stored in a relational database^[2]. To show how SQL-queries can be used to perform basic physics analysis tasks, a C++ macro called “SQLPlotter” was developed to make ROOT-plots using the output-data from SQL-queries.

This project, a web-interface to SQLPlotter basically addresses two major objectives. One objective is to make particle physics analysis more accessible, especially to students who do not have the software skills currently needed to analyse LHC data. Another important aim is to demonstrate the underlying principle of SQLPlotter that SQL queries can be used to perform complex analysis tasks. The web-interface has been implemented using basic web-technologies, including PHP, HTML5 and JavaScript.

In future, this web-interface can be envisaged to be brought into production as an educational tool for students interested in physics analysis.

Table of Contents

Abstract	3
1 Introduction	5
1.1 File-based physics analysis	5
1.2 SQL-based physics analysis.....	5
1.3 SQLPlotter: the C++ Macro.....	6
2 Project work : The Interface	8
2.1 Description	8
2.2 Technologies Involved	8
2.3 Implementation.....	10
3 Conclusion	17
4 Future scope.....	18
5 References	19

1 Introduction

1.1 File-based physics analysis

Presently, data analysis at CERN is done using ROOT^[3], the dedicated C++ framework developed by the High Energy Physics community which provides all the necessary tools for plotting, fitting and statistical analysis. This framework uses as input so-called ROOT-ntuples, that are centrally produced by physics groups from previously reconstructed event summary data. Each physics group can determine the content of ntuple needed for their type of analysis, such as the type of physics objects to include, the level of detail to be stored per physics object and whether to apply any event filter and/or pre-analysis steps.

The analysis in this manner is I/O intensive, requires a lot of files and is time consuming. If the user only requires a relatively small dataset, he/she can copy the data and run the analysis locally. For larger datasets, which is often the case, a user needs to send a job to the LHC Computing Grid (LCG) where the analysis is split into multiple jobs each running on a subset of data which can sometimes takes several days to finish.

1.2 SQL-based physics analysis

In order to address the above mentioned challenges, an openlab project was started to explore the possibility of replacing this file analysis system with a centrally accessible Oracle database.

There are several pros and cons to moving to a SQL based analysis, the main ones are listed here.

Analysis via SQL on DB Advantages:

1. No need to recompile macro while tweaking your analysis
2. The DB can hold intermediate query-results to be studied (no need to store filtered results in ntuples)
3. User can analyze the data anywhere with ROOT and a connection to DB - no need for large local storage and/or powerful machine

Analysis via SQL on DB disadvantages:

1. Complex functions from C++ to be made available centrally via separate schema
2. Administrator needs to prevent users from tying up all resources in badly-written SQL
3. Analysis SQL is not always that easy to read/write/debug

If the disadvantages can be addressed sufficiently, SQL-based physics analysis inside a database could be an interesting alternative to the current file-based analysis method.

The figure below shows an example of pseudo-code illustrating the difference between a typical ROOT-macro and a SQL statement that counts the number of electrons in an event that passed specific selection criteria.

<pre>vector<float> el_pt; vector<float> el_eta; tree->getBranch("el_pt",&el_pt); tree->getBranch("el_eta",&el_eta); //etc. for (ievent = 0 ; ievent<nevents ; ievent++){ //find good electrons tree->NextEvent(); for(i=0; i<nelectrons; i++){ if(el_pt[i] > 25. && fabs(el_eta[i])<2.5 etc.) ngodelectron++; } }</pre>	<pre>with sel_electron as (select * from "electron" where "pt">25. and abs("eta")<2.5 ...) select count(*) from sel_event group by EventNumber;</pre>
---	--

Figure 1: Physics Analysis C++ v/s SQL Analysis (Pseudo-code)

1.3 SQLPlotter: the C++ Macro

SQLPlotter is a C++ Macro written by M. Limper using ROOT classes to create histograms from the output of a SQL-query running on data stored in the database. The class currently only produces TH1F and TH2F histogram-types but it could be extended in future to accommodate more features from the ROOT framework.

The three main functions of the class that are used for analysis include:

```
void SQLPlotter::runSQL(const char* sql_query_file )
```

This function takes the SQL query file as an argument and executes the query to store results into a temporary table called 'QUERY_RESULT'.

```
void SQLPlotter::makeTH1F(const char* column_name, int nbin, float
min_value, float max_value, const char* option)
```

This function takes as input column name, number of bins, a range and optionally a plot-style option to create a one dimensional histogram (TH1F).

```
void SQLPlotter::makeTH2F(const char* column_name1,int nbins1, float  
min_value1, float max_value1,const char* column_name2, int nbins2, float  
min_value2, float max_value2, const char* option)
```

This function, similar to the previous one, plots one variable against another to create a two dimensional plot (TH2F):

The general workflow of using the SQLPlotter class is briefly explained by the chart below:

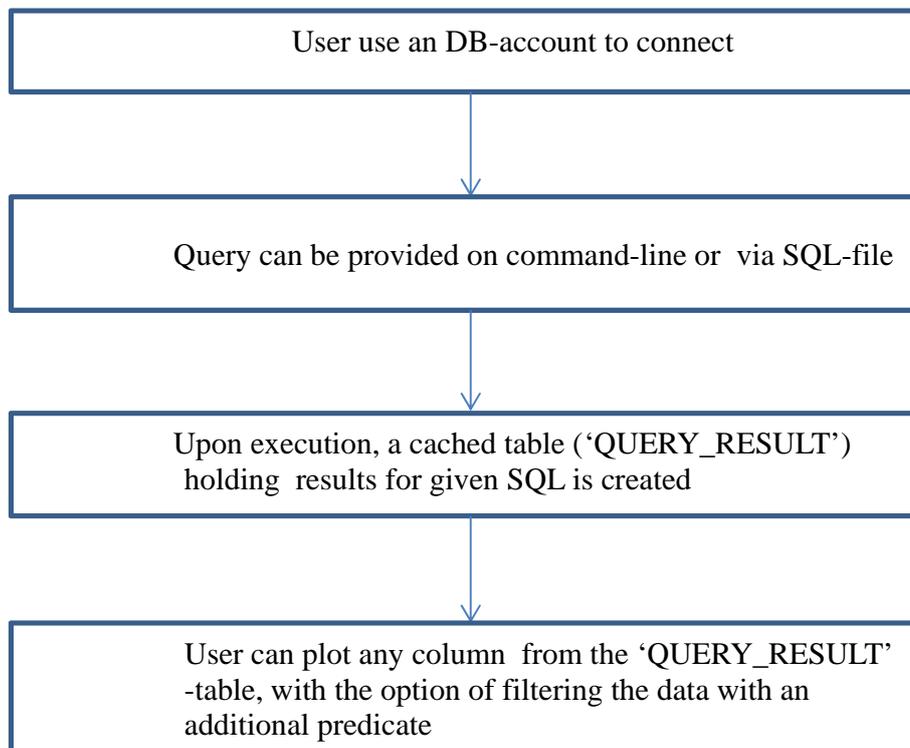


Figure 2: General Workflow of the SQLPlotter class

2 Project work : The Interface

2.1 Description

The project aims at developing an easy to use web-interface to the aforementioned SQLPlotter class with basically two significant purposes:

1. To show how LHC data can be analyzed to find exotic particles. The web-interface could be used as an educational tool in future for the physics students or anyone interested in this domain wherein they can simply use the pre-defined queries on a subset of data to see how analysis can be done to find particles. The text editor allows them to create their own queries and run on the dataset.
2. The project provides a live demonstration of how SQL can be used to perform more complex analysis tasks.

The machine running the web-server has a local installation of the ROOT framework along with the SQLPlotter class, so the user only needs to provide a SQL query to the web-interface, as illustrated by the figure below.

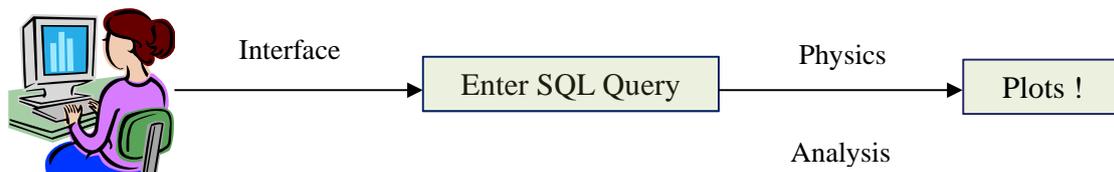


Figure 3: Illustration of working of Interface

2.2 Technologies Involved

The interface is developed and is enabled to execute the query and make plots using very simple technologies such as:

PHP

PHP which stands for the recursive acronym PHP: Hypertext Preprocessor, is a server-side scripting language designed for web-development but also used as a general purpose programming language^[4]. It allows web developers to create dynamic content that interacts with databases. PHP code can be simply mixed with HTML, or it can be used in combination with various templating engines and web frameworks. PHP has been widely ported and can be deployed on most web servers on almost every operating system and platform, free of charge.

SQLPlotter deploys PHP Version 5.1.6 on Apache Web Server.

HTML5 & CSS3

HTML5 is a core technology mark-up language of the Internet used for structuring and presenting content for the World Wide Web. It is the fifth revision of the HTML standard. In particular, HTML5 adds many new syntactic features^[5]. The interface makes use of HTML5 form validation features for the login form as well as other forms used in the web-interface.

Cascading Style Sheets (CSS) is a style sheet language used for describing the look and formatting of a document written in a mark-up language. The interface uses CSS3 transitions, animations and other new style attributes.

Javascript

JavaScript (JS) is a dynamic computer programming language. It is most commonly used as part of web browsers, whose implementations allow client-side scripts to interact with the user, control the browser, communicate asynchronously, and alter the document content that is displayed^[6]. The interface uses JS for client side interaction.

It also uses **codemirror.js** and **sql.js**. Codemirror^[7] is a versatile text editor implemented in JavaScript for the browser. It is specialized for editing code, and comes with a number of language modes and addons that implement more advanced editing functionality. Sql.js is an online SQL interpreter written in JS.

Bash Shell scripts

PHP script interacts with the shell scripts to run ROOT locally, setting up required libraries, establish connection with the database and run the various C++ macros including the SQLPlotter class.

The PHP script runs on Apache Web Server, version 2.2.3 (Red Hat).

2.3 Implementation

The workflow of the Interface is as indicated below:

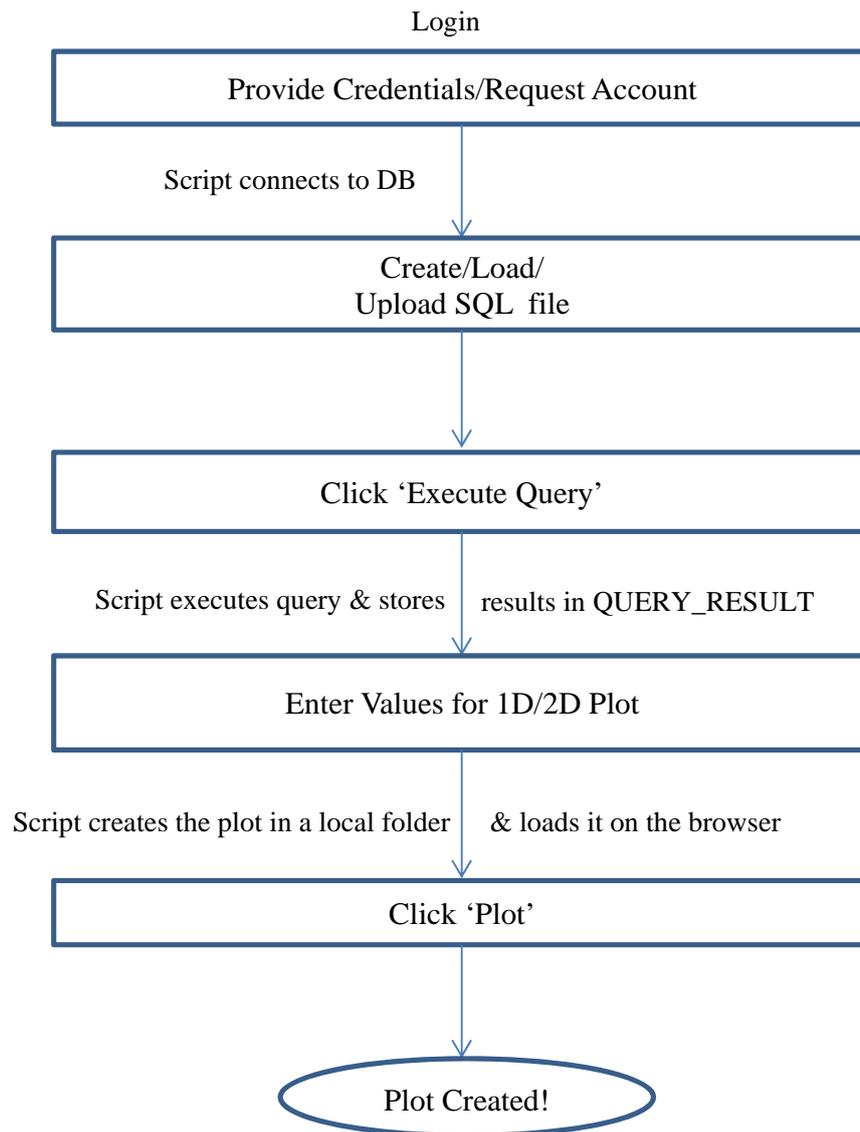


Figure 4: Workflow of Interface

The interface has the following main screens:

Login Screen

Login Form

Fill out the form below to login to my super awesome SQLPlotter.

user01

Password

Register Login

Please fill out this field.

To create an account, kindly contact Database Administrator at SQLPlotter@cern.ch.

Figure 5: Snapshot-Login Form

This is the splash screen of the interface where user needs to enter the database account credentials for connecting to the database. The login form uses HTML5 form validation feature to ensure that user fills both the form fields before submitting the form. The authentication is done via PHP script which tries to connect to the database using given values and gives an error message if invalid/wrong values are entered.

In case the user doesn't have a database account, he can click on the register button which displays the database administrator's info to request an account.

About Screen

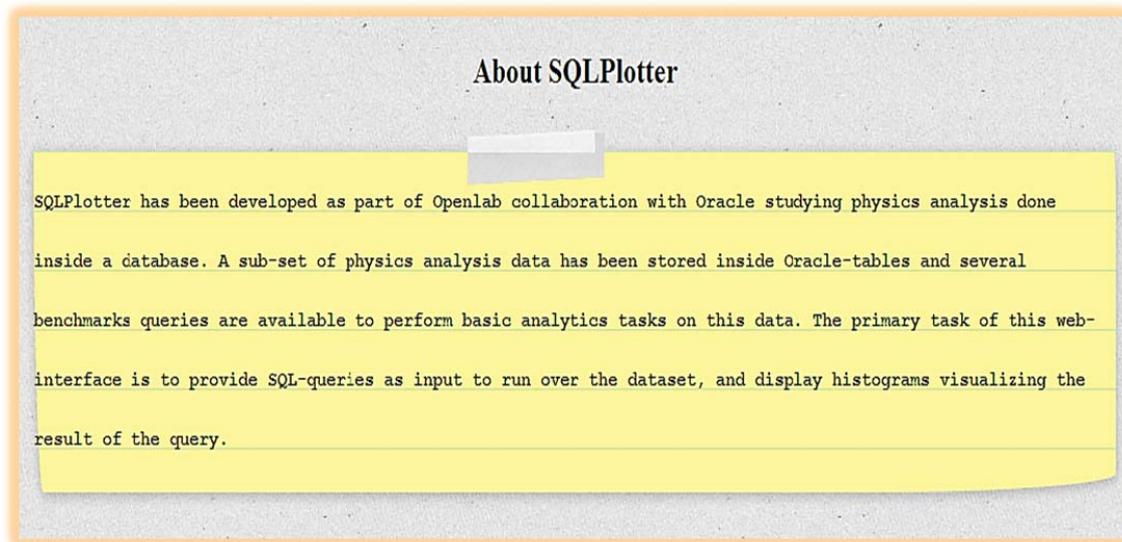


Figure 6: Snapshot-About screen

This is the About Screen that is displayed when the user logs into the application for the first time. This shows information regarding what SQLPlotter is and what are the main functions being used in the C++ macro.

Load/Create SQL

This is the main screen which allows user to create/ load /upload a SQL file. The screen has a text editor that uses codemirror.js and sql.js to provide high-lighting and make the editing easy. Once the query has been created or loaded in the editor, the user can click on 'Execute Query' button to execute the query on the database, which is done by calling the local installation of the SQLPlotter class via PHP script. Once the query is executed, the user is directed to the 'Create plot' screen where the user can enter values for variables to finally make a plot.

The figure below shows an example of a complex SQL query loaded in the editor and ready to be executed. Upon execution, the query results are stored into a temporary database table named 'QUERY_RESULT'. This table is then used to perform analysis and create plots.

```

1 with
2 "sel_muon" as (select "muon_i", "RunNumber", "EventNumber", "E", "px", "py", "pz", "charge", "pt", "phi", "eta"
3 from DATA12_STEV.muon where "pt" > 10000. and abs("eta") < 2.7 and "tight" = 1 and ("id_d0" < 10. or abs("eta") > 2.5) and "ptcone2
4 "sel_2muon" as ( select "RunNumber", "EventNumber",
5 FIRST_VALUE("muon_i") OVER (PARTITION BY "RunNumber", "EventNumber" ORDER BY "muon_i" RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
6 FIRST_VALUE("charge") OVER (PARTITION BY "RunNumber", "EventNumber" ORDER BY "muon_i" RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
7 FIRST_VALUE("E") OVER (PARTITION BY "RunNumber", "EventNumber" ORDER BY "muon_i" RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FO
8 FIRST_VALUE("px") OVER (PARTITION BY "RunNumber", "EventNumber" ORDER BY "muon_i" RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED F
9 FIRST_VALUE("py") OVER (PARTITION BY "RunNumber", "EventNumber" ORDER BY "muon_i" RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
10 FIRST_VALUE("pz") OVER (PARTITION BY "RunNumber", "EventNumber" ORDER BY "muon_i" RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
11 FIRST_VALUE("phi") OVER (PARTITION BY "RunNumber", "EventNumber" ORDER BY "muon_i" RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
12 FIRST_VALUE("eta") OVER (PARTITION BY "RunNumber", "EventNumber" ORDER BY "muon_i" RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
13 FIRST_VALUE("pt") OVER (PARTITION BY "RunNumber", "EventNumber" ORDER BY "muon_i" RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
14 LAST_VALUE("muon_i") OVER (PARTITION BY "RunNumber", "EventNumber" ORDER BY "muon_i" RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
15 LAST_VALUE("charge") OVER (PARTITION BY "RunNumber", "EventNumber" ORDER BY "muon_i" RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
16 LAST_VALUE("E") OVER (PARTITION BY "RunNumber", "EventNumber" ORDER BY "muon_i" RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FO
17 LAST_VALUE("px") OVER (PARTITION BY "RunNumber", "EventNumber" ORDER BY "muon_i" RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED F
18 LAST_VALUE("py") OVER (PARTITION BY "RunNumber", "EventNumber" ORDER BY "muon_i" RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED F
19 LAST_VALUE("pz") OVER (PARTITION BY "RunNumber", "EventNumber" ORDER BY "muon_i" RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED F
20 LAST_VALUE("phi") OVER (PARTITION BY "RunNumber", "EventNumber" ORDER BY "muon_i" RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
21 LAST_VALUE("eta") OVER (PARTITION BY "RunNumber", "EventNumber" ORDER BY "muon_i" RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
22 LAST_VALUE("pt") OVER (PARTITION BY "RunNumber", "EventNumber" ORDER BY "muon_i" RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED F
23 ROW_NUMBER() OVER (PARTITION BY "RunNumber", "EventNumber" ORDER BY "muon_i") rn.

```

Execute Query

Figure 7: Snapshot-Load/Create Plot Screen

Create Plot

This is another important screen in the interface which creates the desired plots. The user is currently provided two options, to create a one dimensional plot or a two dimensional plot. The user can choose either of these and based on the choice certain form fields are enabled & others disabled for him.

In case of a one dimensional plot, the user needs to enter one column name, followed by number of bins and a minimum and maximum value to specify the range. The form again uses HTML5 form validation feature to ensure that all required values are filled in for submitting the form. There is also a select field to select the drawing-option for the plot. The default drawing-option is 'E' (display error-bars) and there are other options to choose as defined in the ROOT framework.

The snapshot below shows a plot for INV_MASS (the column name representing the invariant mass of two muons) for 400 bins and a range of 1-105. The query shows peaks which are actually physics particles. For example, the largest peak around INV_MASS=90, shows that LHC collisions produced many Z-bosons (Z-bosons have an invariant mass of 90 GeV and can decay to muon-pairs). Similarly, we can re-run the query for different values and smaller ranges without executing the query again as long as we are in the same session.

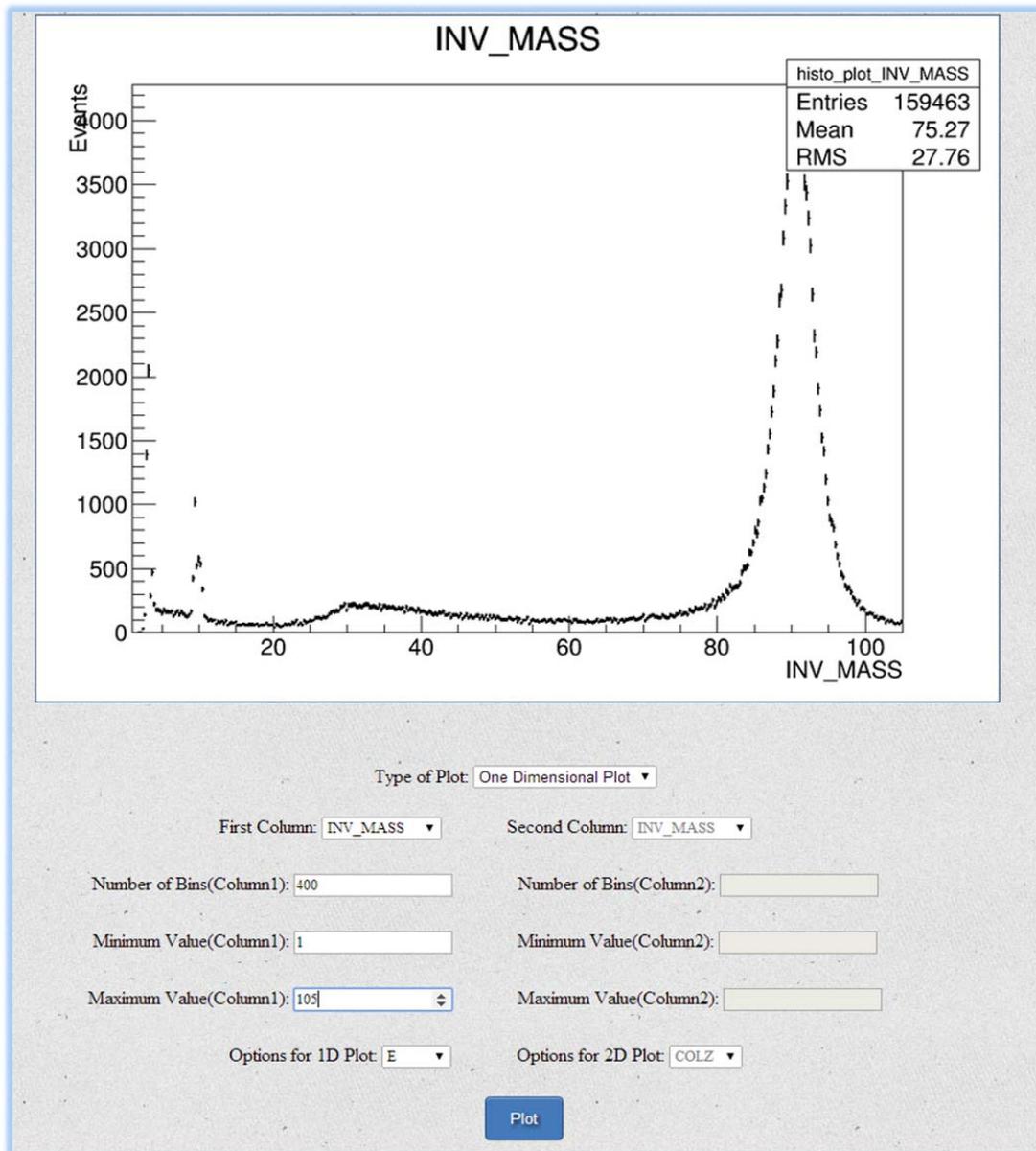


Figure 8: Snapshot-One dimensional plot

For a two-dimensional plot the user needs to do a similar process, except that the fields are now required to be filled in for two variables or columns against each other. The default option is 'COLZ'

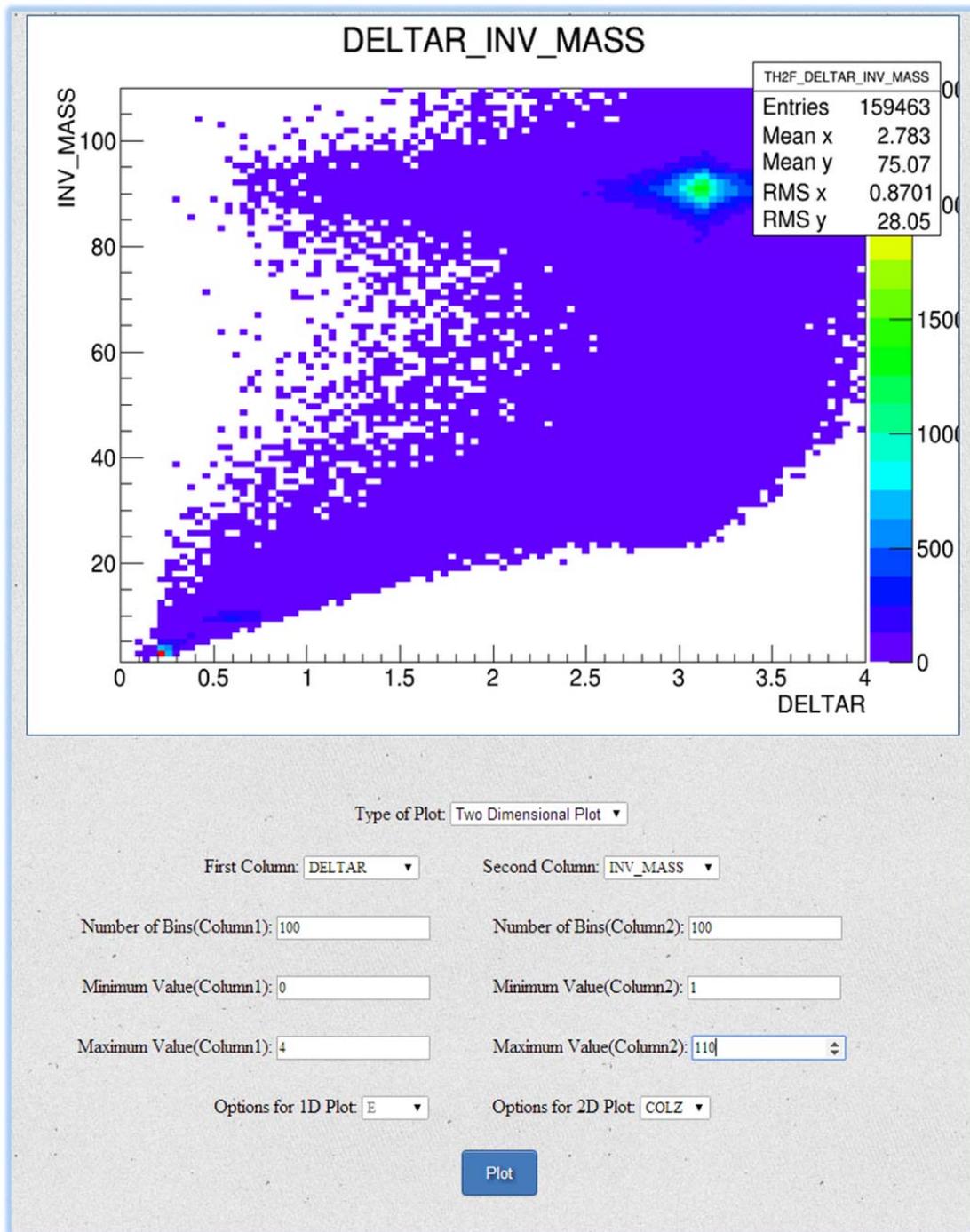


Figure 9: Snapshot-Two dimensional plot

The above plot, plots 'DELTA' and 'INV_MASS' against each other for 100 bins each and a range of 0-4 and 1-110 respectively with option 'COLZ'.

Contact Us Screen



Figure 10: Snapshot- Contact Us Screen

This is the usual contact screen that displays the contact information for help and support.

3 Conclusion

SQLPlotter shows how SQL can be used for complex analysis tasks, and how it might provide an alternative to the current file-based method of physics analysis.

The interface has been built using really simple technologies and provides support to the existing class in the following ways:

1. Easy to use interface which provides the user ability to create/load or upload SQL files (to server from local user PC)
2. The text-editor emulates the stand-alone editors using the javascript framework. Which has features like hard and soft tabs, auto open/close brackets, paranthesis, double and single quotes etc.
3. No need to setup libraries or install ROOT locally, the PHP script does everything using bash scripts.
4. No need to learn ROOT in depth, one just needs to know the appropriate values to be used to create useful plots!

Further, SQLPlotter can be used as an educational tool for students who are interested in data analysis. They can use the pre-defined queries illustrating how to find exotic particles and they can use the examples as a basis to build their own queries.

4 Future scope

Currently, SQLPlotter uses Oracle tables but experiments are being conducted to store physics analysis data using different technologies such as Hadoop+Impala or PostgreSQL. The interface is database technology independent, only the SQL-query provided by the user might change due to the difference in SQL language support between different technologies. Thus changing the input data source can be easily accommodated in the interface.

SQLPlotter currently implements only some of the ROOT features. In future, it can be extended to use other features, for example to add the capability of fitting the results in a plot to determine the standard deviation and mean of a particular peak.

If brought into production, it would require rigorous testing for multiple database accounts, as presently it has been tested only with a single account. Further, it might require more than a single dedicated machine in case many users are accessing the interface simultaneously. For these tests only a small subset of data has been used but there are many types of physics analysis which would require a much large dataset to yield significant analysis results.

The application has not yet been reviewed from a security point of view. The database accounts used to access the database via SQLPlotter are protected with limited read/write permissions, to prevent mal-intent on the database. In addition the web-application should run on https and behind the single sign-on method used currently for CERN accounts.

5 References

- [1] LHC, <http://home.web.cern.ch/topics/large-hadron-collider>
- [2] An SQL-based approach to physics analysis, Dr Maaïke Limper 2014 *J. Phys.: Conf. Ser.* 513 022022
- [3] ROOT, <http://root.cern.ch/drupal/>
- [4] PHP, <http://php.net/>
- [5] HTML5, <http://www.w3.org/TR/html5/>
- [6] JavaScript, <http://www.w3schools.com/js/>
- [7] Codemirror.js, <http://codemirror.net/>