ClinBAY
Statistical solutions to drug development

Using SQL for cross-platform statistical programming (for SAS, R and more)

ClinBAY Solution Series

Christos Stylianou, PhD

November 2020

ClinBAY Ltd
182 Agias Fylaxeos street
Office 101, Kofteros Business center
3083 Limassol, Cyprus

Table of Contents

Introduction

Despite the different advantages and capabilities offered by different statistical software, clinical trials are usually reported using one package, this is done to ensure a consistent presentation and to ease the review. Our company has used SAS for statistical reporting since our conception, but has used a number of other software like R for a similar duration for different purposes like complex Bayesian models or analyses not available in SAS or for Shiny applications. We have also created and maintain our own Data visualization tool Datasly, that cannot be used for reporting, but is used extensively for QC purposes and data exploration. We also use LINQPad, a free, lightweight and powerful tool to execute SQL query on databases.

As one can imagine, maintaining code for 4 different software can be a burden for the statistical programming team. This issue was also further increased with the preparation of modules to fully analyze and report a study using R instead of SAS. This white paper presents the path explored to reduce the development time.

The proposed solution presented in this paper is to use SQL as a cross-software statistical language, by reusing validated SQL programs with no changes in the 4 software presented.

Methodology

The vast majority of the programming activities for any project are the data manipulation aspects. Whilst working on generating QC code within Datasly's SQL package, our programming team has spotted that most of the code could be made to work in SAS using PROC SQL with little to no edits. When the preparation of the R modules started, the complexity of the task to perform the analysis in R native coding was explored but was deemed as a huge task. With the SQLite code already available for Datasly, the task in R was therefore quickly switched to identifying a working SQL package. After a thorough investigation the "sqldf" library was selected in R due to its simplicity.

Transferring the code between the packages was quite straight forward in all tests. Almost the whole code worked without any issue, with some notable exceptions:

- **Different operators used:** SAS especially enables the use of its own operators like "ne" but these needed to be replaced in other packages with their correct operators.
- **Converting of data types** (Cast function)**:** Different data types were allowed between SAS and SQLite (R and Datasly)
- **Different mathematical functions:** When generating summary statistics (e.g. mean, standard deviation etc) directly from SQL code, although some statements like COUNT were accepted for all of them, some others had different naming and some were not available for all packages.

It is noted though that individual SQL statements on large datasets were over 20% more time consuming in both SAS and R, compared to native functions. An example was the calculation of estimating the average value for each patient at each timepoint for each parameter, where PROC SQL required 20% more time than PROC MEANS in SAS and in R the dbSendQuery required 25% more time than using the aggregate function. Therefore,

using SQL could cause considerable delays in extremely large datasets when compared to native functions. Finally, it is note that in R there is also the extra step(s) to load the datasets into the database, which could add some time in the execution.

An example of SQL code for adverse events that can be used with no edits in SAS, R and Datasly is included in Appendix 1. The code can then be reported with either PROC REPORT in SAS or in R using Markdown or whatever library is preferred. Some steps for converting numeric to test formats and combining columns could have been performed as well in SQL, as these steps required editing across software they were omitted. Appendix 2 has a snippet of how the code SQL can be used in R.

Conclusion

This project demonstrated the potential to use SQL code to reduce the development time by using the same data manipulation code across different statistical software. Using this approach a huge time saving in development time was observed for our Shiny applications and our reporting in R, by allowing us to borrow validated SQL code already generated for Datasly and SAS and simply updating the reporting coding.

Key Takeaways

1. Using SQL in lieu of native functions can decrease considerably the development time of code in a new software.
2. More time is needed the execution of SQL code than native functions in both SAS and R, this could be a burden for considerably large datasets, if a number of data manipulations are needed.
3. In most cases tested the minor delays in execution were out-weighted by the considerable decrease in development time.

ClinBAY is a company that provides biometrics solutions to decision makers. Feel free to contact us (info@clinbay.com) if you are interested in our services or products.

Appendix 1: Sample SQL Code to generate an AE table from ADaM data

```
/*Load the ADSL data with the subset*/
  create table tempadsl as
      select *
      from adsl where SAFFL='Y';

/*Load the AE data with the subset */
  create table tempads as
      select *
           from ADAE where SAFFL='Y';

  /*Add a total column in ADSL*/
   create table adsl2 as
    SELECT USUBJID,'Total' as TRT01P,99 as TRT01PN FROM
tempadsl
    UNION
    SELECT USUBJID,TRT01P,TRT01PN FROM tempadsl;

/*Calculate N from ADSL*/
 create table bigN as
      select count(USUBJID) as bigN, TRT01P,TRT01PN
           from adsl2
      group by TRT01P,TRT01PN;


/*Add total SOC in AE dataset*/
```

```
    create table tempads2 as
        SELECT USUBJID, 'Total' as AESOC,TRT01P,TRT01PN FROM
tempads
        UNION
        SELECT  USUBJID, AESOC, TRT01P,TRT01PN FROM tempads;

    /*Add total treatment in AE dataset*/
     create table adsall as
        SELECT USUBJID, AESOC,'Total' as TRT01P,99 as TRT01PN FROM
tempads2
        UNION
        SELECT  USUBJID, AESOC, TRT01P,TRT01PN FROM tempads2;

    /*Obtain unique SoC AE's within each subject*/
     create table ads as
          select distinct USUBJID, AESOC, TRT01P,TRT01PN
               from adsall
          group by USUBJID,AESOC,TRT01P,TRT01PN;

    /*Calculate the unique AE count*/
     create table ads2 as
          select  AESOC, TRT01P,TRT01PN,count(*) as count
               from ads
          group by AESOC,TRT01P,TRT01PN;

    /*Merge big N with AE count data*/
    CREATE TABLE ADS3 AS
        SELECT * FROM ADS2 as AE, bigN as bigN WHERE
AE.TRT01P=bigN.TRT01P ;
```

```
/*Calculate the proporions*/
CREATE TABLE ADS4 AS
    select *, 100*COUNT/bigN as percent
        from ads3
      order by AESOC,TRT01P,TRT01PN;
```

Appendix 2: R code snipet to run the Adverse events SQL code above

```
library(sqldf)
#Open SQL database
db <- dbConnect(SQLite(), dbname=filename)

#load data into SQL database
dbWriteTable(db, "adae", ads);
dbWriteTable(db, "adsl", adsl);

#Run SQL code, each query needs to be sent separately in dbSendQuery, they cant be run in bulk
#Load the ADSL data with the subset
dbSendQuery(conn = db,"create table tempadsl as select * from adsl where SAFFL='Y' ;");
#Load the ADAE data with the subset
dbSendQuery(conn = db,"create table tempads as select * from ADAE where SAFFL='Y' and TRTEMFL='Y';");
#Statements could be loaded instead one at a line from a text file containing the SQL code rather than entering the code one query line at a time
#con <- file("Adverse events by SOC.sql", "rt")
#for (i in 0:100) {dbSendQuery(conn = db,test[i]);}
…
```

```
#read data from SQL database back to R
Ads4<-dbReadTable(db, "ads4");
#Close SQL database
dbDisconnect(db)
```