# Using R with PostgreSQL

ACTIVITY 21

Now that the connection between PostgreSQL and R has been established, access to the remaining data sets is available. More significantly, the data from PostgreSQL can be queried and aggregated as needed rather than requiring R to process through all the data. This saves programming and processing time, an important consideration for projects that, even when narrowed, contain substantial amounts of data.

## Purpose

The purpose of this activity is to introduce some applications of sqlQuery() and allow you to explore the connection between R and PostgreSQL with emphasis in querying and aggregating information.

## Learning Objective

Use SQL queries to pull data from SQL based resources into R.

## Required Resources

- R, R Studio
- ODBC connection to PostgreSQL

## Time Allocated

60 minutes in class

## Tasks

### A. Install and Load the RODBC Package

To make a call to RODBC we use this set of commands:

```
channel <- odbcConnect("ce510", uid="ce510")
qry <- " SELECT * FROM wim.wimdata WHERE timestamp >= '08-08-2009' AND
timestamp < '08-09-2009'"
wim <-sqlQuery(channel, qry)
```

`Channel` opens the connection to the ODBC driver you set up in Activity 20. The first "`ce510`" in the `odbconnect()` call is the name of the driver in your windows ODBC library. This is the "Datasource" you named. The uid="`ce510`" is the name of the user that is authorized to access the PostgreSQl database. This is the user name you set in Activity 20. The variable `qry <-` is just a holder of the text string to pass to the database. The line wim `<-sqlQuery(channel, qry)` calls the `sqlQuery ()` function, passes the text in `qry` to the `channel`, then stores the return in a dataframe that will be named `wim`. For the most part, the datatypes will be handled much better in this rather than reading. For the most part, the datatypes will be handled much better using RODBC than when reading the files in from CSV or other text file.

### B. R Operations or SQL Operations

You can be ambidextrous when deciding whether to use R or SQL. In this activity, we will show you some simple comparisons of R and SQL operations. Before we start, it is helpful to narrow the date range in the WIM data frame since we may not want to query in all records (there are 1.4 million!)

```
qry <- " SELECT min(timestamp), max (timestamp) FROM wim.wimdata "
range <-sqlQuery(channel, qry)
```

So, for the purposes which follow, lets read in the data from the wim schema for the WIM records for just one day

```
qry <- " SELECT * FROM wim.wimdata WHERE timestamp >= '08-08-2009' AND
timestamp < '08-09-2009'"
wim <-sqlQuery(channel, qry)


plot (wim$timestamp, wim$gvw)
```

We can easily add another criteria to the SELECT statement to be only type 11 trucks (5-axle semis)

```
qry <- " SELECT * FROM wim.wimdata WHERE timestamp >= '08-08-2009' AND
timestamp < '08-09-2009' AND type='11'"
wim.11 <-sqlQuery(channel, qry)
```

You could also do this in R from the original wim dataframe

```
wim.11R <- subset(wim, type==11)
```

In R Studio workspace, using the number of records, confirm that the subsets are the same. Alternatively, use plots:

```
par (mfrow=c(1,2))
plot (wim.11$timestamp, wim.11$gvw, main=paste("Num Recs", nrow(wim.11),
"\nAvg GVW", mean(wim.11$gvw)))
plot    (wim.11R$timestamp,    wim.11R$gvw,    main=paste("Num    Recs",
nrow(wim.11R), "\nAvg GVW", mean(wim.11R$gvw)))
```

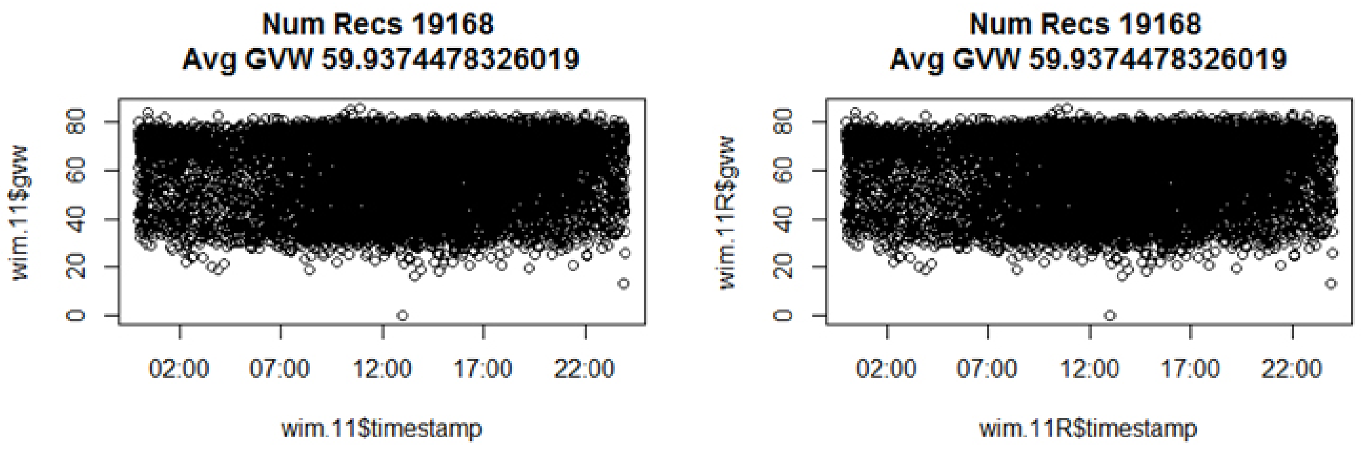

**Figure 39**

Note that since each record is a truck, we can count up records to get flow

```
qry <- " SELECT hour, COUNT(hour) FROM wim.wimdata WHERE timestamp >=
'08-08-2009' AND timestamp < '08-09-2009' GROUP BY hour"
wim5 <-sqlQuery(channel, qry)
barplot (wim5$count, names.arg=wim5$hour) # hours not in order!
```

```
qry <- " SELECT hour, COUNT(hour) FROM wim.wimdata WHERE timestamp >=
'08-08-2009' AND timestamp < '08-09-2009' GROUP BY hour ORDER BY hour"
wim5 <-sqlQuery(channel, qry)
```

In R, we can use the table function to count

```
table (wim$hour)
```

If you want to turn the table return into a data frame, use the following syntax

```
wim5R <- as.data.frame( table (wim$hour))
names(wim5R) <- c("hour","count")
str(wim5R)
```

Show plots

```
par (mfrow=c(1,3))
barplot (wim5$count, names.arg=wim5$hour)
barplot (wim5R$count, col="dodgerblue") #use the dataframe
barplot (table (wim$hour), col="green") #just use the table function
return
```

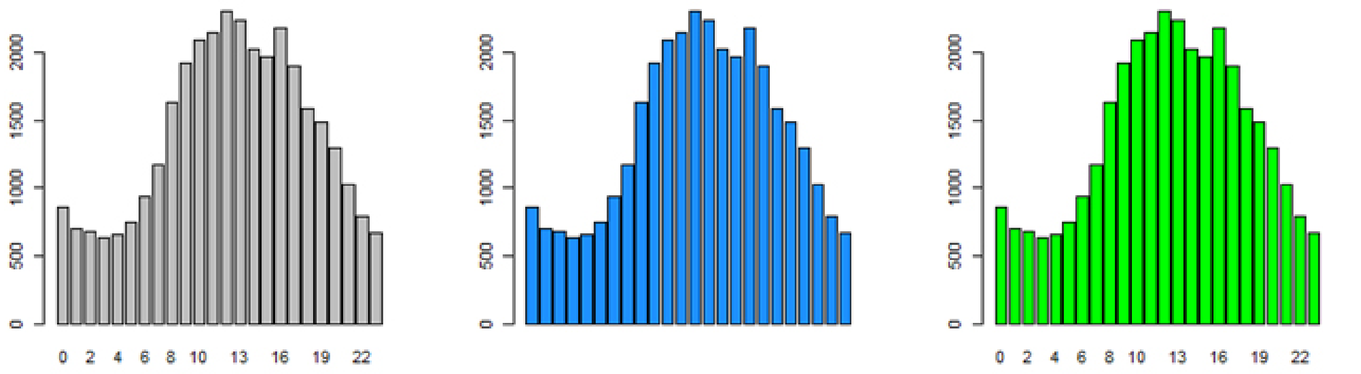

**Figure 40**

Use SQL to calculate average GVW by station

```
qry <- " SELECT stationnum, avg(gvw) AS avggvw FROM wim.wimdata WHERE
timestamp >= '08-08-2009' AND timestamp < '08-09-2009'
GROUP BY stationnum
ORDER BY stationnum"
gvwbysta <-sqlQuery(channel, qry)
barplot (gvwbysta$avggvw, names.arg=gvwbysta$stationnum)
```

Doing aggregate operations in R is also easy (see Dalgaard, page 75 for more tips). A built-in function called `tapply` does the grouping operation over any function that you can apply.

```
tapply (wim$gvw, wim$stationnum, mean)
```

It takes the variable gvw, groups it by *stationnum*, then applies whatever function you specify (here, the mean). There is no limit to the types of operations that can be applied using `tapply` hence it is more flexible than the SQL aggregate options.

```
par (mfrow=c(1,2))
barplot (gvwbysta$avggvw, names.arg=gvwbysta$stationnum)
barplot(tapply (wim$gvw, wim$stationnum, mean), col="dodgerblue")
```

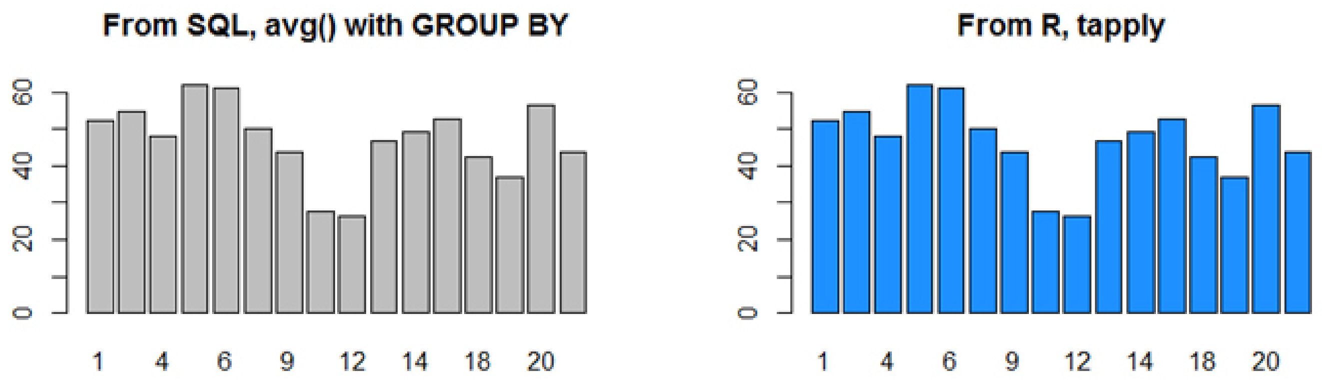

Figure 41

## C. Group Collaborations

Now it is your turn to try to show your skills working with SQL and R selection and grouping methods. Working with a partner, you will assigned by the instructor one of the data sets to explore further. We will post these codes to a Google doc so that the instructor can share your code with the class.

First, after being assigned the data frame select the variable that appears most interesting to you. Write a SQL statement to select "ALL" of the records. You will then show how these records could be subset in SQL and R, how the GROUP BY and table functions return the same values, and how the GROUP BY aggregate functions (`avg`, `stddev`, `sum`, `variance`, `min`, `max`) and match with the tapply options in R. To "prove" they match you are required to make the comparison plots in R as shown in the script examples.

The instructor will give you the data set (*incident*, *trimet*, *loopdetector*, *weather*, *bicycle*). Your team should browse the tables and select the primary variable to work with and what a logical subset and grouping variable should be. In the example, we picked gvw and subset by type (after limiting to trucks for one day). Your script needs to produce, in a 2 column by 3 row arrangement, the following comparisons (based on the sample R script):

| SQL STATEMENT | R OPERATIONS |
|---|---|
| PLOT 1 : SELECT A SUBSET OF RECORDS | PLOT 2 :subset (df, criteria) |
| PLOT 3 :SELECT COUNT() GROUP BY | PLOT 4 :table (df$col) |
| PLOT 5: SELECT GROUP BY AGGEGATE FUNCTION<br>OPTIONS = avg, stddev, sum, variance, min, max | PLOT 6: tapply (df$variable, df$groupby, fun)<br>OPTIONS FOR fun = mean, sd, sum, var, min, max |

The corresponding R Script for the activity includes sample code of what would this look like for the following options:

1. Data set: wim
2. Selection of "ALL" records= all trucks from August 8, 2009

3. Subsetting variable = trucks where type=11
4. Grouping variable = hour

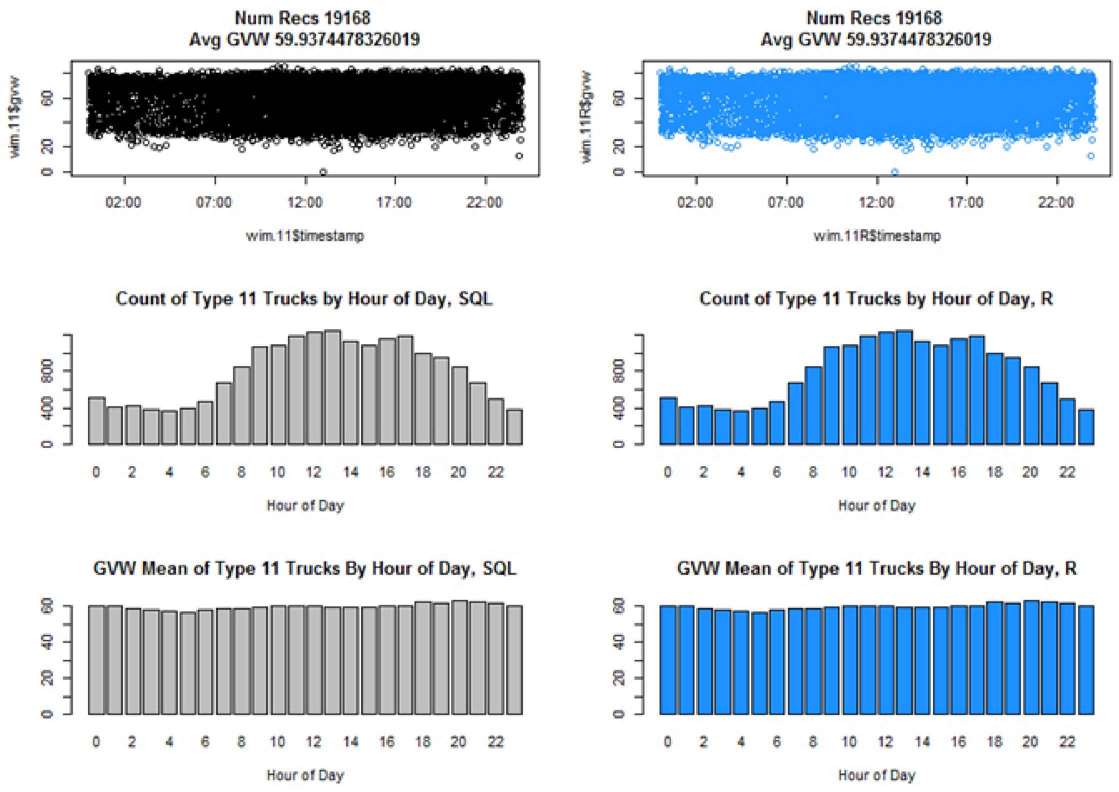

**Figure 42**

## Deliverable

A completed R code posted on the Google doc and your explanation shared with the class.

## Assessment

Participation!

Student Notes