Stat 292 - Recitation 8

SQL in R

Orçun Oltulu

5 / 5 / 2021

Introduction - Motivation:

What is SQL ?

SQL (Structured Query Language) is a special purpose programming language used to manage, extract, and aggregate data stored in large relational database management systems.

In simple words, think of a large machine (rectangular shape) consisting of many, many boxes (again rectangles). Each box comprises a table (dataset). This is a database. A database is an organized collection of data. Now, this database understands only one language, i.e, SQL. No English, Japanese, or Spanish. Just SQL. Therefore, SQL is a language which interacts with the databases to retrieve data.

Following are some important features of SQL:

1. It allows us to create, update, retrieve, and delete data from the database.
2. It works with popular database programs such as Oracle, DB2, SQL Server, etc.
3. As the databases store humongous amounts of data, SQL is widely known for it speed and efficiency.
4. It is very simple and easy to learn.
5. It is enabled with inbuilt string and date functions to execute data-time conversions. Currently, businesses worldwide use both open source and proprietary relational database management systems (RDBMS) built around SQL.

Getting Started with SQL

Let's try to understand SQL commands now. Most of these commands are extremely easy to pick up as they are simple "English words." But make sure you get a proper understanding of their meanings and usage in SQL context. For your ease of understanding, I've categorized the SQL commands in three sections:

1. Data Selection – These are SQL's indigenous commands used to retrieve tables from databases supported by logical statements.

2. Data Manipulation – These commands would allow you to join and generate insights from data.

3. Strings and Dates – These special commands would allow you to work diligently with dates and string variables.

Before we start, you must know that SQL functions recognize majorly four data types. These are:

A. Integers – This datatype is assigned to variables storing whole numbers, no decimals. For example, 123,324,90,10,1, etc.

B. Boolean – This datatype is assigned to variables storing TRUE or FALSE data.

C. Numeric – This datatype is assigned to variables storing decimal numbers. Internally, it is stored as a double precision. It can store up to 15 -17 significant digits.

D. Date/Time – This datatype is assigned to variables storing data-time information. Internally, it is stored as a time stamp.

Data Selection

1. SELECT – It tells you which columns to select.

2. FROM – It tells you columns to be selected should be from which table (dataset)

3. LIMIT – By default, a command is executed on all rows in a table. This commands limits the number of rows. Limiting the rows leads to faster execution of commands.

4. WHERE – This commands specifies a filter condition; i.e., the data retrieval has to be done based on some variable filtering.

5. Comparison Operators – Everyone knows these operators as (= , != , < , > , <= , >=). They are used in conjunction with the WHERE command.

6. Logical Operators – The famous logical operators (AND, OR, NOT) are also being used to specify multiple filtering conditions. Other operators are:

- LIKE – It is used to extract similar values and not exact values.
- IN – It is used to specify the list of values to extract or leave out from a variable.
- BETWEEN – As the names suggests, it activates a condition based on variable(s) in the table.
- IS NULL – It allows you to extract data without missing values from the specified column.

7. ORDER BY – It is used to order a variable in descending or ascending order.

Data Manipulation

1. Aggregate Functions – Originating from statistics, these functions are insanely helpful in generating quick insights from data sets.

A. COUNT – It counts the number of observations.

B. SUM – It calculates the sum of observations.

C. MIN/MAX – It calculates the min/max and, eventually, the range of a numerical distribution.

D. AVG – It calculates the average aka mean.

2. GROUP BY – For categorical variables, it calculates the above stats based on their unique levels.

3. HAVING – It is mostly used for strings to specify a particular string or combination while retreiving data.

4. DISTINCT – It returns the unique number of observations.

5. CASE – It is used to create rules using if/else conditions.

JOINS – It is a popular function in SQL as individual tables are often required to merged to create more meaningful data. It can implement the following variations of join. To understand these joins, let's say we have two tables: Table A and Table B. Both tables have seven variables. Two variables of Table A are also available in Table B (Shown below as an image). Based on a specified criteria:

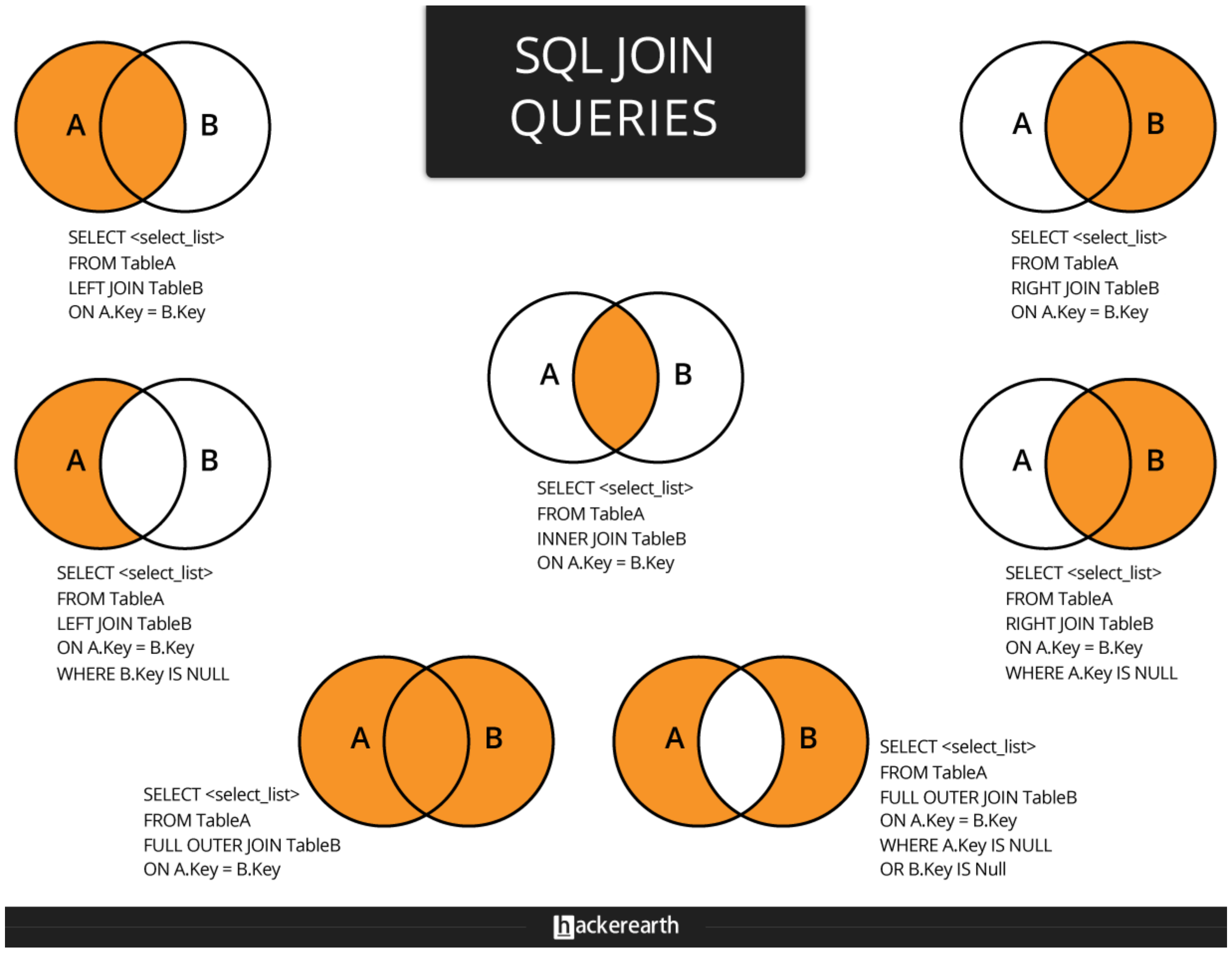
A. INNER JOIN – It returns the common rows specifying the joining criteria from A and B.

B. OUTER JOIN – It returns the rows which are not common to A and B.

C. LEFT JOIN – It returns the rows which are in A but not in B.

D. RIGHT JOIN – It returns the rows which are in B but not in A.

E. FULL OUTER JOIN – It returns all the rows from both tables. It often leads to NULL values in the resultant data set.



7. ON – It is used to specify a column used for filtering while joining tables.

8. UNION – It is similar to rbind() in R. Use it to combine two tables where both the tables have identical variable names.

Practising SQL in R

For writing SQL queries, we'll use sqldf package. It is one of the most versatile package packages available these days which activate SQL in R. It uses SQLite (default) as the underlying database and is often faster than performing the same manipulations in base R. Besides SQLite, it also supports H2 Java database, PostgreSQL database, and MySQL.

Yes, you can easily connect database servers using this package and query data. For more details on this package, I suggest you read this github repo created by its author.

When using SQL in R, think of R as the database storage machine. The process is simple. You load the data set either using read.csv or read.csv.sql and start querying data. Ready to get your hands dirty? Let's begin! I request you to code every line as you scroll the page.

The more you write, the more confident you'll become at writing SQL queries.

We'll be using multiple data sets to understand different SQL functions. If you haven't installed R yet, I request you to download here. For now, we'll use the babynames data set.

Install and Load sqldf package

```
# install.packages("sqldf")
library(sqldf)
```

Exercise 1:

In this exercise we are going to use 'babynames' dataset. First, install 'babynames' package and then load that data into R. When your data set is ready, check the structure of the data set.

```
# install.packages("babynames")
library(babynames)
```

```
data(babynames)

str(babynames)
```

```
## tibble [1,924,665 x 5] (S3: tbl_df/tbl/data.frame)
##  $ year: num [1:1924665] 1880 1880 1880 1880 1880 1880 1880 1880 1880 1880 ...
##  $ sex : chr [1:1924665] "F" "F" "F" "F" ...
##  $ name: chr [1:1924665] "Mary" "Anna" "Emma" "Elizabeth" ...
##  $ n   : int [1:1924665] 7065 2604 2003 1939 1746 1578 1472 1414 1320 1288 ...
##  $ prop: num [1:1924665] 0.0724 0.0267 0.0205 0.0199 0.0179 ...
```

Part A:

str() function gives the number of rows, but now let's check the number of rows in babynames data.

```
sqldf("SELECT COUNT(*) FROM babynames")
```

```
##   COUNT(*)
## 1  1924665
```

Part B:

Print the first 8 rows of the data set. (**Hint:** * sign is used to select all the data available)

```
sqldf("SELECT * FROM babynames LIMIT 8")
```

```
##   year sex      name    n       prop
## 1 1880   F      Mary 7065 0.07238359
## 2 1880   F      Anna 2604 0.02667896
## 3 1880   F      Emma 2003 0.02052149
## 4 1880   F Elizabeth 1939 0.01986579
## 5 1880   F    Minnie 1746 0.01788843
## 6 1880   F  Margaret 1578 0.01616720
## 7 1880   F       Ida 1472 0.01508119
## 8 1880   F     Alice 1414 0.01448696
```

Part C:

Now, select ‘year’, ‘name’, ‘prop’ variables and print first 12 rows.

```
sqldf("SELECT year, name, prop FROM babynames LIMIT 12")
```

```
##    year      name       prop
## 1  1880      Mary 0.07238359
## 2  1880      Anna 0.02667896
## 3  1880      Emma 0.02052149
## 4  1880 Elizabeth 0.01986579
## 5  1880    Minnie 0.01788843
## 6  1880  Margaret 0.01616720
## 7  1880       Ida 0.01508119
## 8  1880     Alice 0.01448696
## 9  1880    Bertha 0.01352390
## 10 1880     Sarah 0.01319605
## 11 1880     Annie 0.01288868
## 12 1880     Clara 0.01256083
```

Part D:

Select first ‘year’, ‘sex’ and ‘name’ variables, and rename ‘sex’ as ‘gender’, ‘name’ as ‘NAME’, and print first 8 rows. (**Hint:** You can use the AS command to rename a variable)

```
sqldf("SELECT year, sex AS gender, name AS NAME FROM babynames LIMIT 8")
```

```
##   year gender      NAME
## 1 1880      F      Mary
## 2 1880      F      Anna
## 3 1880      F      Emma
## 4 1880      F Elizabeth
## 5 1880      F    Minnie
## 6 1880      F  Margaret
## 7 1880      F       Ida
## 8 1880      F     Alice
```

Part E:

Find how many distinct years in the data set.

```
sqldf("SELECT COUNT (DISTINCT year) AS Distinct_Year
      FROM babynames")
```

```
##   Distinct_Year
## 1           138
```

Part F:

Now, order the babynames data set by 'year'.

```
sqldf("SELECT *
      FROM babynames
      ORDER BY year DESC
      LIMIT 10")
```

```
##    year sex      name     n       prop
## 1  2017   F      Emma 19738 0.01052750
## 2  2017   F    Olivia 18632 0.00993760
## 3  2017   F       Ava 15902 0.00848152
## 4  2017   F  Isabella 15100 0.00805377
## 5  2017   F    Sophia 14831 0.00791029
## 6  2017   F       Mia 13437 0.00716679
## 7  2017   F Charlotte 12893 0.00687664
## 8  2017   F    Amelia 11800 0.00629367
## 9  2017   F    Evelyn 10675 0.00569364
## 10 2017   F   Abigail 10551 0.00562750
```

Part G:

Select 'year', 'name', and 'new_column' which is obtained by dividing 'n' to 'prop', print first 10 rows.

```
sqldf("SELECT year, name, n/prop AS new_column
      FROM babynames
      LIMIT 10")
```

```
##    year      name new_column
## 1  1880      Mary   97605.00
## 2  1880      Anna   97605.00
## 3  1880      Emma   97605.00
## 4  1880 Elizabeth   97604.98
## 5  1880    Minnie   97604.99
## 6  1880  Margaret   97605.03
## 7  1880       Ida   97605.03
## 8  1880     Alice   97605.02
## 9  1880    Bertha   97604.98
## 10 1880     Sarah   97604.97
```

Part H:

Print how many distinct female names you have in 2000.

```
sqldf("SELECT COUNT(DISTINCT name) AS Distinct_Names
      FROM babynames
      WHERE sex == 'F' AND year == 2000")
```

```
##   Distinct_Names
## 1          17653
```

Part I:

Alphabetically order male names in 1900. (There are 3394 distinct names, you better not print them out)

```
sqldf("SELECT DISTINCT name
      FROM babynames
      WHERE year == 1900
      ORDER BY name")
```

Part J:

First, print unique names which start with 'Ben' (first 5 is enough), then find the number of unique names which start with 'Ben'.

```
sqldf("SELECT year, sex, name
      FROM babynames
      WHERE name LIKE 'Ben%'
      LIMIT 5")
```

```
##   year sex     name
## 1 1880   F   Bennie
## 2 1880   F     Bena
## 3 1880   M Benjamin
## 4 1880   M      Ben
## 5 1880   M Benjiman
```

```
sqldf("SELECT COUNT(DISTINCT name)
      FROM babynames
      WHERE name LIKE 'Ben%'")
```

```
##   COUNT(DISTINCT name)
## 1                  151
```

Part K:

First, print unique names which end with 'man'. (first 5 is enough), then find the number of unique names which end with 'man'.

```
sqldf("SELECT year, sex, name
      FROM babynames
      WHERE name LIKE '%man'
      LIMIT 5")
```

```
##   year sex     name
## 1 1880   M   Herman
## 2 1880   M   Norman
## 3 1880   M  Sherman
## 4 1880   M    Lyman
## 5 1880   M Benjiman
```

```
sqldf("SELECT COUNT(DISTINCT name)
      FROM babynames
      WHERE name LIKE '%man'")
```

```
##   COUNT(DISTINCT name)
## 1                  249
```

Part L:

First, print unique names which contain 'ert'. (first 5 is enough), then find the number of unique names which contain 'ert'.

```
sqldf("SELECT year, sex, name
      FROM babynames
      WHERE name LIKE '%ert%'
      LIMIT 5")
```

```
##   year sex     name
## 1 1880   F   Bertha
## 2 1880   F Gertrude
## 3 1880   F   Bertie
## 4 1880   F  Alberta
## 5 1880   F   Gertie
```

```
sqldf("SELECT COUNT(DISTINCT name)
      FROM babynames
      WHERE name LIKE '%ert%'")
```

```
##   COUNT(DISTINCT name)
## 1                  361
```

Part M:

Find the number of distinct names whose ‘prop’ is between 0.005 and 0.01 in 2005.

```
sqldf("SELECT COUNT(DISTINCT name)
      FROM babynames
      WHERE year == 2005
      AND prop BETWEEN 0.005 AND 0.01")
```

```
##   COUNT(DISTINCT name)
## 1                   46
```

Part N:

Print the ‘prop’ for the names Coleman, Emily and Victoria in 1907.

```
sqldf("SELECT name, prop
      FROM babynames
      WHERE name IN ('Coleman','Emily','Victoria')
      AND year == 1907")
```

```
##       name       prop
## 1    Emily 0.00211597
## 2 Victoria 0.00118542
## 3  Coleman 0.00015134
```

Part O:

Find the most popular girl and boy names in 1996.

```
sqldf("SELECT sex, name, n
      FROM babynames
      WHERE year == 1996
      GROUP BY sex")
```

```
##   sex    name     n
## 1   F   Emily 25151
## 2   M Michael 38365
```

Part P:

List the most common girl and boy names between 2007 and 2017.

```
sqldf("SELECT year, sex, name, n
      FROM babynames
      WHERE year BETWEEN 2007 AND 2017
      GROUP BY year, sex")
```

```
##    year sex     name     n
## 1  2007   F    Emily 19355
## 2  2007   M    Jacob 24273
## 3  2008   F     Emma 18809
## 4  2008   M    Jacob 22591
## 5  2009   F Isabella 22298
## 6  2009   M    Jacob 21169
## 7  2010   F Isabella 22905
## 8  2010   M    Jacob 22117
## 9  2011   F   Sophia 21837
## 10 2011   M    Jacob 20365
## 11 2012   F   Sophia 22304
## 12 2012   M    Jacob 19069
## 13 2013   F   Sophia 21213
## 14 2013   M     Noah 18241
## 15 2014   F     Emma 20924
## 16 2014   M     Noah 19286
## 17 2015   F     Emma 20435
## 18 2015   M     Noah 19613
## 19 2016   F     Emma 19471
## 20 2016   M     Noah 19082
## 21 2017   F     Emma 19738
## 22 2017   M     Liam 18728
```

Part Q:

Find the minimum and maximum 'prop' for all babynames.

```
sqldf("SELECT MIN(prop) AS min_prop, MAX(prop) AS max_prop
      FROM babynames")
```

```
##   min_prop   max_prop
## 1 2.26e-06 0.08154561
```

Part R:

Find the number of births in each year (sum(n)) and then print the first 10 highest.

```
sqldf("SELECT year, SUM(n) AS total_births
      FROM babynames
      GROUP BY year
      ORDER BY total_births DESC
      LIMIT 10")
```

```
##    year total_births
## 1  1957      4200007
## 2  1959      4156434
## 3  1960        4154377
## 4  1961        4140244
## 5  1958        4131657
## 6  1956        4121070
## 7  1962        4035234
## 8  1955        4014041
## 9  2007        3994007
## 10 1954        3979884
```