Basic, Intermediate, and Advanced SQL in R

Abdulaziz AlYaqout

10/19/2018

Introduction to SQL

Structured Query Language (or SQL) is a language that is used to communicate with databases. It is the primary language used for relational database systems.

SQL in R

This document walks through basic, intermediate, and advanced uses of SQL.

For ease of use, I demonstrate all the SQL code in R. R contains packages that allow you to write queries in SQL. In some cases (like a work setting) you can connect directly to a database in R (e.g. through an ODBC connection). But for the sake of simplicity, in this document, rather than connecting directly to a database, we will simply import data into R data frames and show how to run SQL queries over it.

The R dataframes will act as our database tables.

We will use the sqldf package in R to do this.

```r
library(sqldf)
```

```
## Loading required package: gsubfn

## Warning: package 'gsubfn' was built under R version 3.4.4

## Loading required package: proto

## Loading required package: RSQLite
```

Data

We will use data related to the real estate market in Kansas City, MO.

```r
library(readxl)
kc_housing <- read_excel("KC_House_Data.xlsx")
```

```
## Warning in strptime(x, format, tz = tz): unknown timezone 'default/America/
## New_York'
```

Explore structure of the data

```r
str(kc_housing)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    21613 obs. of  21 variables:
##  $ id           : num  7.13e+09 6.41e+09 5.63e+09 2.49e+09 1.95e+09 ...
##  $ date         : chr  "20141013T000000" "20141209T000000" "20150225T000000" "20141209T000000" ...
##  $ price        : num  221900 538000 180000 604000 510000 ...
##  $ bedrooms     : num  3 3 2 4 3 4 3 3 3 3 ...
##  $ bathrooms    : num  1 2.25 1 3 2 4.5 2.25 1.5 1 2.5 ...
##  $ sqft_living  : num  1180 2570 770 1960 1680 ...
##  $ sqft_lot     : num  5650 7242 10000 5000 8080 ...
##  $ floors       : num  1 2 1 1 1 1 2 1 1 2 ...
##  $ waterfront   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ view         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ condition    : num  3 3 3 5 3 3 3 3 3 3 ...
##  $ grade        : num  7 7 6 7 8 11 7 7 7 7 ...
##  $ sqft_above   : num  1180 2170 770 1050 1680 ...
##  $ sqft_basement: num  0 400 0 910 0 1530 0 0 730 0 ...
##  $ yr_built     : num  1955 1951 1933 1965 1987 ...
##  $ yr_renovated : num  0 1991 0 0 0 ...
##  $ zipcode      : num  98178 98125 98028 98136 98074 ...
##  $ lat          : num  47.5 47.7 47.7 47.5 47.6 ...
##  $ long         : num  -122 -122 -122 -122 -122 ...
##  $ sqft_living15: num  1340 1690 2720 1360 1800 ...
##  $ sqft_lot15   : num  5650 7639 8062 5000 7503 ...
```

Looks like most of our variables are of numeric type.

We can also aggregate by some factors (e.g. bedoroom size, waterfront, etc.)

Basic SQL

Simple Selects

To see all available fields in a table, run a ’SELECT *‘command. In the FROM clause, specify the dataframe name as the ’table’ name. And we will limit our results to 10 records to save space.

```
sqldf("SELECT *
      FROM kc_housing
      LIMIT 10")
```

```
##            id            date   price bedrooms bathrooms sqft_living
## 1  7129300520 20141013T000000  221900        3      1.00        1180
## 2  6414100192 20141209T000000  538000        3      2.25        2570
## 3  5631500400 20150225T000000  180000        2      1.00         770
## 4  2487200875 20141209T000000  604000        4      3.00        1960
## 5  1954400510 20150218T000000  510000        3      2.00        1680
## 6  7237550310 20140512T000000 1225000        4      4.50        5420
## 7  1321400060 20140627T000000  257500        3      2.25        1715
## 8  2008000270 20150115T000000  291850        3      1.50        1060
## 9  2414600126 20150415T000000  229500        3      1.00        1780
## 10 3793500160 20150312T000000  323000        3      2.50        1890
##    sqft_lot floors waterfront view condition grade sqft_above
## 1      5650      1          0    0         3     7       1180
## 2      7242      2          0    0         3     7       2170
## 3     10000      1          0    0         3     6        770
## 4      5000      1          0    0         5     7       1050
## 5      8080      1          0    0         3     8       1680
## 6    101930      1          0    0         3    11       3890
## 7      6819      2          0    0         3     7       1715
## 8      9711      1          0    0         3     7       1060
## 9      7470      1          0    0         3     7       1050
## 10     6560      2          0    0         3     7       1890
##    sqft_basement yr_built yr_renovated zipcode     lat     long
## 1              0     1955            0   98178 47.5112 -122.257
## 2            400     1951         1991   98125 47.7210 -122.319
## 3              0     1933            0   98028 47.7379 -122.233
## 4            910     1965            0   98136 47.5208 -122.393
## 5              0     1987            0   98074 47.6168 -122.045
## 6           1530     2001            0   98053 47.6561 -122.005
## 7              0     1995            0   98003 47.3097 -122.327
## 8              0     1963            0   98198 47.4095 -122.315
## 9            730     1960            0   98146 47.5123 -122.337
## 10             0     2003            0   98038 47.3684 -122.031
##    sqft_living15 sqft_lot15
## 1           1340       5650
## 2           1690       7639
## 3           2720       8062
## 4           1360       5000
## 5           1800       7503
## 6           4760     101930
## 7           2238       6819
## 8           1650       9711
## 9           1780       8113
## 10          2390       7570
```

If you want to select just a few fields, specify those in the SELECT clause instead of the *.

```
sqldf("SELECT id, price, bedrooms
      FROM kc_housing
      LIMIT 10")
```

```
##            id   price bedrooms
## 1  7129300520  221900        3
## 2  6414100192  538000        3
## 3  5631500400  180000        2
## 4  2487200875  604000        4
## 5  1954400510  510000        3
## 6  7237550310 1225000        4
## 7  1321400060  257500        3
## 8  2008000270  291850        3
## 9  2414600126  229500        3
## 10 3793500160  323000        3
```

Select count

Another common SQL command is 'select count()' which returns a count of records.

The following query returns a count of all records in the table.

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

```
##   COUNT(*)
## 1    21613
```

Select distinct

Sometimes you may just want the unique values in a field. The following query will show how many unique floor counts there are in the KC houses.

```
sqldf("SELECT DISTINCT floors
      FROM kc_housing")
```

```
##   floors
## 1    1.0
## 2    2.0
## 3    1.5
## 4    3.0
## 5    2.5
## 6    3.5
```

Integrating our knowledge of count() function, we can find the number of distinct floors in the KC housing data.

```
sqldf("SELECT COUNT(DISTINCT floors)
      FROM kc_housing")
```

```
##   COUNT(DISTINCT floors)
## 1                      6
```

Note: this is useful, but the field name is a little unclean. Let's clean it up. You can rename a field directly using SQL code using 'as' for a cleaner output. We'll name it 'NumberOfDistinctFloors'

```
sqldf("SELECT COUNT(DISTINCT floors) as NumberOfDistinctFloors
      FROM kc_housing")
```

```
##   NumberOfDistinctFloors
## 1                      6
```

WHERE Clause

In the WHERE clause, we filter the data. The SELECT tells us what fields to return. The WHERE clause puts a filter on the data.

Lets return the count (number) of all houses with more than 2 bathrooms

```
sqldf("SELECT COUNT(*) as HighBathroomHouses
      FROM kc_housing
      WHERE bathrooms > 2")
```

```
##   HighBathroomHouses
## 1              11242
```

And compare it to the 1-bathroom houses

```
sqldf("SELECT COUNT(*) as OneBathroomHouses
      FROM kc_housing
      WHERE bathrooms =  1")
```

```
##   OneBathroomHouses
## 1              3852
```

As you can see, the WHERE clause takes logical operators (<, >, =, and !=)

You can link multiple conditions together using AND and OR as well.

Let's return 15 records of 2 bedroom houses whos living rooms are greater than 2000 square feet.

```
sqldf("SELECT id, bathrooms, sqft_living
      FROM kc_housing
      WHERE bathrooms = 2 AND sqft_living > 2000
      LIMIT 15")
```

```
##            id bathrooms sqft_living
## 1  2768000400         2        2360
## 2  8820901275         2        2750
## 3  8075400570         2        2260
## 4  2617300160         2        2020
## 5  4058000060         2        2220
## 6  3021059276         2        2010
## 7  9189700045         2        2290
## 8  7771300125         2        2590
## 9  1959700550         2        2050
## 10  191100045         2        2490
## 11 2883200160         2        2020
## 12 9808650060         2        2350
## 13 1723049270         2        2270
## 14 4432600075         2        2110
## 15 6909200575         2        2060
```

Now let's return 15 records of 2 or 3 bedroom houses whos living rooms are greater than 2000 square feet and whose condition is at least a rating of 5.

```
sqldf("SELECT id, bathrooms, sqft_living, condition
      FROM kc_housing
      WHERE (bathrooms = 2 OR bathrooms = 3) AND sqft_living > 2000 AND condition >= 5
      LIMIT 15")
```

```
##            id bathrooms sqft_living condition
## 1  8820901275         2        2750         5
## 2  3127200041         3        2440         5
## 3  4058000060         2        2220         5
## 4  9189700045         2        2290         5
## 5  7424700045         3        3830         5
## 6  7771300125         2        2590         5
## 7   191100045         2        2490         5
## 8  1373800295         3        4380         5
## 9  6909200575         2        2060         5
## 10 2021200370         2        3010         5
## 11 6665800060         2        2920         5
## 12 2301400640         2        2330         5
## 13 7578200310         2        2208         5
## 14 9368700223         3        2010         5
## 15 9554200105         2        2020         5
```

Intermediate SQL

The two intermediate functions in SQL are to group by and to join tables.

Group by

Group by allows you to run the query aggregating the results on some factor.

Here we'll return the count of records in the table, grouping by condition

```
sqldf("SELECT condition, COUNT(*) as Volume
      FROM kc_housing
      GROUP BY condition")
```

```
##   condition Volume
## 1         1     30
## 2         2    172
## 3         3  14031
## 4         4   5679
## 5         5   1701
```

Suppose we want to order the results by Volume. We can order our results by typing 'ORDER BY' and the variable to order by.

```
sqldf("SELECT condition, COUNT(*) as Volume
      FROM kc_housing
      GROUP BY condition
      ORDER BY Volume")
```

```
##   condition Volume
## 1         1     30
## 2         2    172
## 3         5   1701
## 4         4   5679
## 5         3  14031
```

Group by is also useful when performing functions on data and comparing groups.

To perform functions in SQL, you type the function name and call it on the field of interest. For example, to find the minimum yr built, type min(yr_built).

```
sqldf("SELECT MIN(yr_built)
      FROM kc_housing")
```

```
##   MIN(yr_built)
## 1          1900
```

Now let's find the most recent year a house was built, grouping by condition

```
sqldf("SELECT condition, MAX(yr_built) as MostRecentYear
      FROM kc_housing
      GROUP BY condition
      ORDER BY condition")
```

```
##   condition MostRecentYear
## 1         1           1966
## 2         2           1995
## 3         3           2015
## 4         4           2009
## 5         5           2005
```

The worst quality houses (1 and 2) not surprisingly are older than the better quality houses.

Inner Joins

A key aspect of SQL is joining tables together. This is central to the relational structure of databases.

To illustrate, let's make some data that we can join to the housing data. We'll assume that each house owner owns a car. We'll take the cars from the classic mtcars dataset and randomly assign them to each id in the KC housing data.

```
kc_cars <- data.frame(id = kc_housing$id,
                      cartype = sample(rownames(mtcars), nrow(kc_housing),
                                       replace=TRUE))
```

```r
head(kc_cars)
```

```
##           id             cartype
## 1 7129300520 Lincoln Continental
## 2 6414100192       Maserati Bora
## 3 5631500400 Lincoln Continental
## 4 2487200875          Duster 360
## 5 1954400510      Ford Pantera L
## 6 7237550310          Volvo 142E
```

For each ID in KC Housing, there is an associated car.

Suppose we want to find the cars in houses whose lots exceed 5000 square feet.

We can join the KC Housing data with the KC Cars data via a common key called a primary key. This binds the two tables together. In our case it is id.

Note: an inner join will only return records with ids that are common to both tables.

```r
sqldf("SELECT kc_housing.id, price, bedrooms, cartype
      FROM kc_housing
      INNER JOIN kc_cars
      ON kc_housing.id = kc_cars.id
      WHERE kc_housing.sqft_lot > 5000
      LIMIT 15")
```

```
##            id   price bedrooms             cartype
## 1  7129300520  221900        3 Lincoln Continental
## 2  6414100192  538000        3       Maserati Bora
## 3  5631500400  180000        2 Lincoln Continental
## 4  1954400510  510000        3      Ford Pantera L
## 5  7237550310 1225000        4          Volvo 142E
## 6  1321400060  257500        3   Hornet Sportabout
## 7  2008000270  291850        3           Mazda RX4
## 8  2414600126  229500        3          Datsun 710
## 9  3793500160  323000        3       Porsche 914-2
## 10 1736800520  662500        3          Merc 450SE
## 11 9212900260  468000        2            Merc 230
## 12  114101516  310000        3          Merc 450SE
## 13 6054650070  400000        3            Merc 230
## 14 1875500060  395000        3       Maserati Bora
## 15   16000397  189000        2   Hornet Sportabout
```

Now you can join the two tables. The associated car type will be joined to the KC Housing on the records with a matching ID. This is essentiall the idea of joins.

Left Joins and Right Joins

Suppose your cars data only contains a subset of the IDs in the housing data. But you still want to retain all the records in the housing data in your query.

A left join will allow you to do that.

It retains all records of the left table, while performing a join with columns on the right table where appropriate.

Subset the kc_cars to just half the data and do a left join with kc_housing

```
set.seed(42)
kc_cars_subset <- kc_cars[sample(1:nrow(kc_housing), nrow(kc_housing)*.5),]
```

```
sqldf("SELECT kc_housing.id, price, bedrooms, cartype
      FROM kc_housing
      LEFT JOIN kc_cars_subset
      ON kc_housing.id = kc_cars_subset.id
      LIMIT 10")
```

```
##            id   price bedrooms          cartype
## 1  7129300520  221900        3             <NA>
## 2  6414100192  538000        3             <NA>
## 3  5631500400  180000        2             <NA>
## 4  2487200875  604000        4             <NA>
## 5  1954400510  510000        3             <NA>
## 6  7237550310 1225000        4       Volvo 142E
## 7  1321400060  257500        3 Hornet Sportabout
## 8  2008000270  291850        3        Mazda RX4
## 9  2414600126  229500        3       Datsun 710
## 10 3793500160  323000        3             <NA>
```

As you can see many of the rows contain NA. Left joins retain all records from the left table, and only matching records of the right table. Left join fills in NA or NULL where there is no matching key in the right table.

Right join does the opposite to this (reversing left and right tables).

There is one more join called outer join. This returns all records in both tables. It fills in NA on the left table where there is no match to the right table key. And NA on the right table where there is no match to the left table key. Sometimes this is called a full outer join.

Advanced SQL

There are many advanced SQL techniques. Advanced SQL includes building nested queries, doing multiple joins, and building case statements to create variables in your table using if/else logic.

Nested queries

The simple idea behind a nested query is to return records using SQL statement, and then query from those results as if the results were its own table.

Here is a simple example. Suppose we want to find the maximum average square footage of a lot size grouping by condition.

We could just do the following query which groups the data by condition and finds the max square footage for each of these groups. We can then order by the MaxSqFootage in descending order.

```
sqldf("SELECT condition, MAX(sqft_lot) as MaxSqFootage
      FROM kc_housing
      GROUP BY condition
      ORDER BY MaxSqFootage DESC")
```

```
##   condition MaxSqFootage
## 1         4      1651359
## 2         2      1164794
## 3         5      1074218
## 4         3      1024068
## 5         1       209959
```

While this may work to give us our answer, technically we don't have the max. We have a table with 5 rows. If we want a single maximum, we can use nested queries for this.

Now that we have aggregated by groups, we can return the maximum of these aggregates by nesting the previous query in another select max() statement.

```
sqldf("SELECT grouped.condition, max(grouped.MaxSqFootage) as HighestMax
       FROM
           (SELECT condition, MAX(sqft_lot) as MaxSqFootage
           FROM kc_housing
           GROUP BY condition
           ORDER BY MaxSqFootage DESC) as grouped")
```

```
##   condition HighestMax
## 1         4    1651359
```

Here you can see we actually encased our previous SQL statement in parentheses. Afterwards it is common practice to give our table results a name. We named it 'grouped.' Now in the outer layer of this nested query we can refer to these records as a table named 'grouped'. So in the outer layer of the query we select the condition and find the max of our MaxSqFootage field.

In practice, nested queries can allow you to do quite flexible analyses.

Multiple joins

Here let's join some metadata about the cars from mtcars to the kc_cars_subset table. Then we can left join the resulting table to the kc_housing table.

```
mtcars$cartype <- row.names(mtcars)
sqldf("SELECT kc_housing.id, kc_housing.price, kc_housing.bathrooms, temp.cartype, temp.mpg
       FROM kc_housing
       LEFT JOIN
           (SELECT kc_cars_subset.id, mtcars.cartype, mtcars.mpg
            FROM kc_cars_subset
            INNER JOIN mtcars
            ON kc_cars_subset.cartype = mtcars.cartype) as temp
       ON kc_housing.id = temp.id
       LIMIT 30")
```

```
##            id   price bathrooms             cartype  mpg
## 1  7129300520  221900      1.00                <NA>   NA
## 2  6414100192  538000      2.25                <NA>   NA
## 3  5631500400  180000      1.00                <NA>   NA
## 4  2487200875  604000      3.00                <NA>   NA
## 5  1954400510  510000      2.00                <NA>   NA
## 6  7237550310 1225000      4.50          Volvo 142E 21.4
## 7  1321400060  257500      2.25   Hornet Sportabout 18.7
## 8  2008000270  291850      1.50           Mazda RX4 21.0
## 9  2414600126  229500      1.00          Datsun 710 22.8
## 10 3793500160  323000      2.50                <NA>   NA
## 11 1736800520  662500      2.50                <NA>   NA
## 12 9212900260  468000      1.00            Merc 230 22.8
## 13  114101516  310000      1.00          Merc 450SE 16.4
## 14 6054650070  400000      1.75                <NA>   NA
## 15 1175000570  530000      2.00               <NA>   NA
## 16 9297300055  650000      3.00               <NA>   NA
## 17 1875500060  395000      2.00      Maserati Bora 15.0
## 18 6865200140  485000      1.00      Mazda RX4 Wag 21.0
## 19   16000397  189000      1.00  Hornet Sportabout 18.7
## 20 7983200060  230000      1.00               <NA>   NA
## 21 6300500875  385000      1.75     Toyota Corolla 33.9
## 22 2524049179 2000000      2.75               <NA>   NA
## 23 7137970340  285000      2.50               <NA>   NA
## 24 8091400200  252700      1.50     Ford Pantera L 15.8
## 25 3814700200  329000      2.25  Hornet Sportabout 18.7
## 26 1202000200  233000      2.00         Merc 450SE 16.4
## 27 1794500383  937000      1.75               <NA>   NA
## 28 3303700376  667000      1.00               <NA>   NA
## 29 5101402488  438000      1.75  Hornet Sportabout 18.7
## 30 1873100390  719000      2.50 Lincoln Continental 10.4
```

Case Statements

We will wrap up with CASE statements, which allow you to create new variables using if/else logic inside of SQL.

Suppose we wanted to create a new binary variable called 'BigLot' based on whether the lot size square footage exceeds 10,000 feet.

We can use the following statement to create the variable.

```
sqldf("SELECT id, price, sqft_lot,
       CASE WHEN sqft_lot >= 10000 THEN 1
            ELSE 0 END AS BigLot
       FROM kc_housing
       LIMIT 15")
```

```
##            id   price sqft_lot BigLot
## 1  7129300520  221900     5650      0
## 2  6414100192  538000     7242      0
## 3  5631500400  180000    10000      1
## 4  2487200875  604000     5000      0
## 5  1954400510  510000     8080      0
## 6  7237550310 1225000   101930      1
## 7  1321400060  257500     6819      0
## 8  2008000270  291850     9711      0
## 9  2414600126  229500     7470      0
## 10 3793500160  323000     6560      0
## 11 1736800520  662500     9796      0
## 12 9212900260  468000     6000      0
## 13  114101516  310000    19901      1
## 14 6054650070  400000     9680      0
## 15 1175000570  530000     4850      0
```

Finally, we can use case statements to make new variables that we can use in nested SQL statements, group by statements, etc.

Suppose we wanted to take the previous query and now perform aggregation based on the BigLot status.

```
sqldf("SELECT avg(price) as AvgPrice,
       CASE WHEN sqft_lot >= 10000 THEN 1
```

```
            ELSE 0 END AS BigLot
       FROM kc_housing
      GROUP BY BigLot")
```

```
##   AvgPrice BigLot
## 1 494176.1      0
## 2 653812.1      1
```

As you can see case statements can provide you with some flexibile analysis.

Summary

In this guide, we demonstrated beginner, intermediate, and more advanced uses of Structured Query Language (SQL).