Running SQL Query in R

Cheng Peng

Contents

1 Introducion

To run SQL clauses in R, we need to use several R libraries (installed and loaded in the above R setup code chunk). There are different ways to run SQL query in R. We only introduce one methods that is close to the authentic SQL code that can be run a DBMS.

1.1 Connect R to Existing Database

If there is an existing database, the following code connects R to the database.

```
con <- DBI::dbConnect(drv = odbc::odbc(),
                      Driver = "driver_name",
                      Server = "server_url",
                      Database = "database_name",
                      user = "user", #optional
                      password = "password") #optional
```

1.2 Create A Database to Run SQL Queries in R

This short note shows the three basic steps to run SQL in R using R Markdown starting with a set of relational tables.

1. Load relational data tables as usual to R.
2. Create a SQLite (relational) database that contain these relational table.
3. Create R code chunk and connect to the created database using Chunk options.

2 Create SQLite Database with R

If modeling requires a data set that contains information from multiple relational data tables, we need to perform data management to aggregate the required information from different data tables. We can load the different data sets in different formats using appropriate R functions.

As an example, We use three ecological survey data sets to create a database.

```
#Load the sample data
plots <- read.csv("https://pengdsci.github.io/datasets/AnimalSurvey/plots.csv")
species <- read.csv("https://pengdsci.github.io/datasets/AnimalSurvey/species.csv")
surveys <- read.csv("https://pengdsci.github.io/datasets/AnimalSurvey/surveys.csv")
```

Next, we create a SQLit database using several R libraries.

```
#Create database
con <- dbConnect(drv = SQLite(),
                 dbname = ":memory:")

#store sample data in database
dbWriteTable(conn = con,
             name = "plots",
             value = plots)

dbWriteTable(conn = con,
             name = "species",
             value = species)

dbWriteTable(conn = con,
             name = "surveys",
             value = surveys)

#remove the local data from the environment
rm(plots, species, surveys)
```

We can use table view function `tbl()` to explore the information of relational data tables in the database. Note that, we

```
tbl(src = con, #the source if the database connection profile
    c("surveys")) #the name of the table to preview
```

```
## # Source:   table<surveys> [?? x 10]
## # Database: sqlite 3.45.2 [:memory:]
##        X record_id month   day  year plot_id species_id sex   hindfoot_length
##    <int>     <int> <int> <int> <int>   <int> <chr>      <chr>           <int>
##  1     1         1     7    16  1977       2 NL         M                  32
##  2     2         2     7    16  1977       3 NL         M                  33
##  3     3         3     7    16  1977       2 DM         F                  37
##  4     4         4     7    16  1977       7 DM         M                  36
##  5     5         5     7    16  1977       3 DM         M                  35
##  6     6         6     7    16  1977       1 PF         M                  14
##  7     7         7     7    16  1977       2 PE         F                  NA
##  8     8         8     7    16  1977       1 DM         M                  37
##  9     9         9     7    16  1977       1 DM         F                  34
## 10    10        10     7    16  1977       6 PF         F                  20
## # i more rows
## # i 1 more variable: weight <int>
```

3 Running SQL Queries in R Code chunks

To use SQL in RMarkdown, we need the following chunk options:

- sql
- connection = "database-name"
- output.var = "output-dataset-name"

If we create a data view only, we simply ignore option `output.var =`

Following are few examples of SQL queries based on the animal survey data tales in the database.

3.1 Subsetting and Duplicating Data

1. Extract year, month and day from `survey` table

```
SELECT
  surveys.year, surveys.month, surveys.Day
FROM
 surveys /* pointer is not needed since it is in the database */
WHERE
  surveys.species_id IN ('NL', 'DM') AND
  surveys.sex = 'M'
```

2. Duplicate a data and rename it

```
SELECT
  surveys.*
FROM
 surveys
```

3. Create a table view (i.e., no data set will be created and saved)

```
SELECT
  surveys.year, surveys.month, surveys.Day
FROM
 surveys
WHERE
  surveys.species_id = 'NL' AND
  surveys.sex = 'M'
```

3.2 Define A New Variable

1. Define a new variable with simple arithmetic operations

```
SELECT
    surveys.plot_id,
    surveys.species_id,
    surveys.sex,
    surveys.weight,
    surveys.weight/100 AS wt_kilo  /*should not the pointer in front of the name of the new variable*/
FROM
   surveys
```

2. Define new variables using string functions in SQL

```
SELECT surveys.*,
       surveys.species_id||'-'||surveys.sex AS newKey
FROM surveys
```

3. Define new variables with aggregated information

```
SELECT surveys.species_id,
       COUNT(surveys.species_id) AS species_ctr
FROM surveys
GROUP BY surveys.species_id
HAVING species_ctr > 10
```

3.3 Sorting Variables

1. Sort data based on the summarized statistics of a variable

Summary functions are restricted to the SELECT and HAVING clauses only;

```
SELECT surveys.species_id
FROM surveys
GROUP BY surveys.species_id
ORDER BY COUNT(surveys.species_id);
```

2. Sort data based on a new variable defined using summarized statistics of a variable.

```
/* create a table view*/
SELECT surveys.species_id AS subtotal,
       COUNT(*)
FROM surveys
GROUP BY surveys.species_id
ORDER BY subtotal;
```

3.4 Join Tables

This section introduce commonly used join operation to merge tables using the common key(s).

1. Inner Join

```
SELECT *
FROM surveys AS A
JOIN species AS B
ON A.species_id = B.species_id;
```

2. Left Join

```
SELECT *
FROM surveys AS A
LEFT JOIN species AS B
ON A.species_id = B.species_id;
```

3. Right Join

```
SELECT *
FROM surveys AS A
RIGHT JOIN species AS B
ON A.species_id = B.species_id;
```

4. Full Join

```
SELECT *
FROM surveys AS A
FULL JOIN species AS B
ON A.species_id = B.species_id;
```

5. Join sub-tables

```
SELECT A.species_id,
       A.sex,
       AVG(A.weight) as mean_wgt
FROM surveys AS A
JOIN species AS B
ON A.species_id=B.species_id
WHERE taxa = 'Rodent' AND A.sex IS NOT NULL
GROUP BY A.species_id, A.sex;   /* sorted by two variables */
```

3.5 Subqueries

1. Sample size

```
SELECT COUNT(*)
FROM surveys
```

2. Relative Frequency with sub-query

```
SELECT B.taxa,
       100.0*COUNT(*)/(SELECT COUNT(*) FROM surveys)  AS Percentage
FROM surveys AS A
JOIN species AS B
ON A.species_id = B.species_id
GROUP BY taxa;
```