SQLDF: SQL für R-Einsteiger leicht gemacht

30-Mar-2023

www.mainanalytics.de

Julia Psenner
julia.psenner@mainanalytics.de

Content

Introduction

The package SQLDF

Connection to data base

Coding differences and similarities

Timing comparison

PROC SQL vs. **SQLDF**

SQLDF vs. traditional R

Conclusion

Introduction

Effective data processing → SQL queries

Include SQL queries in other languages

- SAS (**PROC SQL**)
  - Most common way to use SQL queries in clinical context
- R (**SQLDF**)
  - Growing popularity due to shift to open source programming
  - Easy way for traditional SAS users to perform SQL queries in R

The package `sqldf`

R-package

Install with **install.packages('sqldf')**

Functions included in Package:

read.csv.sql	→ Read file filtered by SQL
read.csv2.sql	→ Read file filtered by SQL
sqldf	→ SQL select on data frames

Reference: https://www.rdocumentation.org/packages/sqldf/versions/0.4-11

Connection to database

When SQL query is executed, a data base (DB) is built in the backend

Default:

- PROC SQL: internal DB
- SQLDF: builds SQLite DB

Pay attention on which DB the connection is built, because there are some differences in coding.

Coding differences and similarities

SAS

```
proc sql;
        create table new_table as
        select * from old_table;
quit;
```

- No need for package
- Create new table with statement "`create table … as`"
- SQL statement in one data step

R

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

new_table <- sqldf("select * from old_table")
```

- Need to install and load package "`sqldf`"
- Create new table with assignment ("`<-`" or "`=`")
- SQL statement in parenthesis and quotation mark

Select, where, order by

randomdf

	fruit	number	name	years
1	banana	4	Voldemort	2
2	apple	2	Hermione	16
3	orange	5	Harry	73
4	banana	2	Ron	78
5	apple	3	Hermione	47
6	apple	5	Dumbledore	57
7	banana	6	Ron	76
8	apple	2	Hermione	26
9	orange	3	Harry	60
10	orange	1	Ron	14

```
proc sql;
          select fruit, number, years
          from randomdf
          where number < 5
          order by fruit, years desc;
quit;
```

```
sqldf("select fruit, number, years
          from randomdf
          where number < 5
          order by fruit, years desc")
```

	fruit	number	years
1	apple	3	47
2	apple	2	26
3	apple	2	16
4	banana	2	78
5	banana	4	2
6	orange	3	60
7	orange	1	14

(inner) join

countries

	ID	country_name
1	1	Germany
2	2	Italy
3	3	France

cities

	zip	city_name	country_id
1	10713	Berlin	1
2	60329	Frankfurt	1
3	75001	Paris	3
4	1030	Vienna	4

```
proc sql;
          select a.ID, a.country_name, b.city_name, b.zip
          from countries as a
          inner join cities as b
          on a.ID = b.country_id;
quit;
```

```
sqldf("select a.ID, a.country_name, b.city_name, b.zip
          from countries as a
          inner join cities as b
          on a.ID = b.country_id")
```

	ID	country_name	city_name	zip
1	1	Germany	Berlin	10713
2	1	Germany	Frankfurt	60329
3	3	France	Paris	75001

Full (outer) join

countries

	ID	country_name
1	1	Germany
2	2	Italy
3	3	France

cities

	zip	city_name	country_id
1	10713	Berlin	1
2	60329	Frankfurt	1
3	75001	Paris	3
4	1030	Vienna	4

```
proc sql;
          select a.ID, a.country_name, b.city_name, b.zip
          from countries as a
          full join cities as b
          on a.ID = b.country_id;
quit;
```

```
sqldf("select a.ID, a.country_name, b.city_name, b.zip
          from countries as a
          full join cities as b
          on a.ID = b.country_id")
```

	ID	country_name	city_name	zip
1	1	Germany	Berlin	10713
2	1	Germany	Frankfurt	60329
3	2	Italy	*NA*	*NA*
4	3	France	Paris	75001
5	*NA*	*NA*	Vienna	1030

Missing values

	ID	country_name	city_name	zip
1	1.000000000000000	Germany	Berlin	10713.00000000000
2	1.000000000000000	Germany	Frankfurt	60329.00000000000
3	2.000000000000000	Italy		.
4	3.000000000000000	France	Paris	75001.00000000000
5	.		Vienna	1030.00000000000

```
proc sql;
            select country_name, city_name
            from city_country
            where zip is not null and country_name is not null;
quit;
```

	country_name	city_name
1	Germany	Berlin
2	Germany	Frankfurt
3	France	Paris

	ID	country_name	city_name	zip
1	1	Germany	Berlin	10713
2	1	Germany	Frankfurt	60329
3	2	Italy	*NA*	*NA*
4	3	France	Paris	75001
5	*NA*	*NA*	Vienna	1030

```
sqldf("select country_name, city_name
            from city_country
            where zip is not null and country_name is not null")
```

Although missing values are displayed differently,
"`is not NULL`" leads to the same outcome

Text operators

randomdf

	fruit
1	banana
2	apple
3	orange
4	banana
5	apple
6	apple
7	banana
8	apple
9	orange
10	orange

```
proc sql;
            select fruit
            from randomdf
            where fruit like 'B%'
            or fruit like '%le';
quit;
```

	fruit
1	apple
2	apple
3	apple
4	apple

```
sqldf("select fruit
            from randomdf
            where fruit like 'B%'
            or fruit like '%le'")
```

	fruit
1	banana
2	apple
3	banana
4	apple
5	apple
6	banana
7	apple

Note: SQLDF is not case sensitive!

Integer division leads to integer value in R

int_table

	a	b
1	1	2
2	2	1

```
proc sql;
            create table integer_division as
            select a/b as quotient
            from int_table;
quit;
```

	quotient
1	0.5
2	2

```
integer_division <- sqldf("select a/b as quotient from int_table")
```

	quotient
1	0
2	2

In R we need to typecast integer to float to perform a real division

```
integer_division <- sqldf("select (a+0.0)/b as quotient from int_table")
```

	quotient
1	0.5
2	2.0

Aggregat functions

Aggregate functions work the same way, but unlike in PROC SQL, the new variables do not need to be renamed in SQLDF.
Caution: the function calculating standard deviation is called differently.

```
proc sql;
          select fruit, avg(years), min(years), max(years),
                    count(fruit), std(years), sum(years)
          from randomdf
          group by fruit;
quit;
```

```
sqldf("select fruit, avg(years), min(years), max(years), count(fruit),
          stdev(years), sum(years)
          from randomdf
          group by fruit")
```

	fruit	_TEMG001	_TEMG002	_TEMG003	_TEMG004	_TEMG005	_TEMG006
1	apple	36.5	16	57	4	18.806027403	146
2	banana	52	2	78	3	43.312815655	156
3	orange	49	14	73	3	31	147

	fruit	avg(years)	min(years)	max(years)	count(fruit)	stdev(years)	sum(years)
1	apple	36.5	16	57	4	18.80603	146
2	banana	52.0	2	78	3	43.31282	156
3	orange	49.0	14	73	3	31.00000	147

Easy column renaming in R

	column_A
1	1
2	2
3	3
4	4
5	1
6	2
7	3
8	4

```
new_df <- sqldf("select column_a as a from df")
```

	a
1	1
2	2
3	3
4	4
5	1
6	2
7	3
8	4

Column name changes

```
new_df <- sqldf("select column_a a from df")
```

```
proc sql;
        create table new_df as
        select column_a as a
        from df;
quit;
```

Only changes column label but not column name

	column_A
1	1.000000000000000
2	2.000000000000000
3	3.000000000000000
4	4.000000000000000
5	1.000000000000000
6	2.000000000000000
7	3.000000000000000
8	4.000000000000000

Can be solved by

```
options nolabel;
```

```
proc sql;
        create table new_df as
        select column_a a
        from df;
quit;
```

Leads to error

```
62   proc sql;
63        create table new_df as
64        select column_a a
                          -
                          22
ERROR 22-322: Syntax error, expecting one of the following: !, !!, &, *, **, +, ',', -, '.', /, <,
              <=, <>, =, >, >=, ?, AND, AS, CONTAINS, EQ, EQT, GE, GET, GT, GTT, LE, LET, LIKE, LT,
              LTT, NE, NET, OR, ^=, |, ||, ~=.
```

Missing grouping

exampledata

	value	group1	group2
1	-0.6264538	1	B
2	0.1836433	0	B
3	-0.8356286	1	B
4	1.5952808	0	C
5	0.3295078	0	A
6	-0.8204684	0	B
7	0.4874291	0	A
8	0.7383247	0	C
9	0.5757814	1	A
10	-0.3053884	0	B

```
proc sql;
            select group1, group2,
            max(value) as max_value
            from exampledata
            group by group1;
quit;
```

group2 missing

	max_value	group1	group2
1	1.5952808021	0	B
2	1.5952808021	0	B
3	1.5952808021	0	C
4	1.5952808021	0	A
5	1.5952808021	0	A
6	1.5952808021	0	B
7	1.5952808021	0	C
8	0.5757813517	1	B
9	0.5757813517	1	B
10	0.5757813517	1	A

```
sqldf("select group1, group2,
            max(value) as max_value
            from exampledata
            group by group1")
```

group2 missing

	max_value	group1	group2
1	1.5952808	0	C
2	0.5757814	1	A

Timing comparison

- Used datasets from CDISC Pilot exampledata
  - LB (23 variables, 59580 observations)
  - SUPPLB (10 variables, 64403 observations)

Reference: phuse-scripts/data/sdtm/cdiscpilot01 at master · phuse-org/phuse-scripts (github.com)

Timing comparison

PROC SQL vs SQLDF

Function	PROC SQL	SQLDF
Full join	23.96 sec	444.83 sec (= 7.4min)
Inner join	24.88 sec	127.79 sec (=2.1min)
Order by	0.11 sec	0.34 sec
Where	0.05 sec	0.33 sec
Aggregate functions	0.05 sec	0.22 sec
Filter by text operator	0.10 sec	0.30 sec

Timing comparison

SQLDF vs traditional R

SQLDF-Function	Time	Traditional R-Function	Time
Full join	444.83 sec	merge {base}	164.40 sec
Inner join	127.79 sec	inner_join {dplyr}	15.85 sec
Order by	0.46 sec	arrange {dplyr}	0.13 sec
Where	0.46 sec	filter {dplyr}	0.28 sec
Aggregate functions	0.25 sec	summarize {dplyr}	0.25 sec
Filter by text operator	0.31 sec	filter/str_detect {stringr}	0.22 sec

Conclusion

- SQLDF facilitates the switch to R for traditional SAS users
- There are some coding differences to be considered
- Consider which database management system SQLDF is using in the backend
- Performance is better in PROC SQL
- There are better ways to process data in R

Thank you for your attention!

julia.psenner@mainanalytics.de