Integrating SQL queries: from PROC SQL to SQLDF

Julia Psenner
Statistician
mainanalytics GmbH
julia.psenner@mainanalytics.de

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

Basic call to create `new_table` from `old_table`

PROC SQL

```
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

SQLDF

```
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

Database Management System (DBMS): when executing SQL statement -> SQL database (DB) is created in backend

If PROC SQL and SQLDF connect to DB of same DBMS, there is no different behavior between the two languages.

This poster shows the similarities and differences when working on the default DBMS:

- **PROC SQL:** internal database
- **SQLDF:** connects to SQLite database

Similarities

Join two datasets

countries

ID	country_name
1	Germany
2	Italy
3	France

+ cities

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

(inner) join

```
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	Germany	Berlin	10713
1	Germany	Frankfurt	60329
3	France	Paris	75001

full (outer) join

```
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	Germany	Berlin	10713
1	Germany	Frankfurt	60329
2	Italy	*NA*	*NA*
3	France	Paris	75001
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

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

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

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

country_name	city_name
Germany	Berlin
Germany	Frankfurt
France	Paris

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

Select, where, order by

randomdf

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

Text operators

randomdf

fruit
banana
apple
orange
banana
apple
apple
banana
apple
orange
orange

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

fruit
apple
apple
apple
apple

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

fruit
banana
apple
banana
apple
apple
banana
apple

Note: SQLDF is not case sensitive!

Differences

Aggregate functions

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

	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

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

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

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.

Missing grouping

exampledata

value	group1	group2
-0.6264538	1	B
0.1836433	0	B
-0.8356286	1	B
1.5952808	0	C
0.3295078	0	A
-0.8204684	0	B
0.4874291	0	A
0.7383247	0	C
0.5757814	1	A
-0.3053884	0	B

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

	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

group2 missing

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

max_value	group1	group2
1.5952808	0	C
0.5757814	1	A

If not all variables from the „select" statement are added to the „group by" statement, SQLDF only displays the observations containing the value, calculated by the aggregate function, while PROC SQL displays all observations.

Easy column renaming in R

column_A
1
2
3
4
5
6
7
8

```
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
1.000000000000000
2.000000000000000
3.000000000000000
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, ^=, |, ||, ~=.
```

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

Column name changes

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

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

Integer division leads to integer value in R

int_table

a	b
1	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
0
2

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
0.5
2.0

Conclusion

- Learning SQLDF with previous knowledge of PROC SQL is not a big effort
- There are some differences to be considered
- Be aware of which database management system SQLDF is using in the backend

Reproduce R-Code