dplyr <–> SQL

Statistical Programming

Fall 2021

Dr. Colin Rundel

Creating a database

```
(db = DBI::dbConnect(RSQLite::SQLite(), "flights.sqlite"))
```

```
## <SQLiteConnection>
##   Path: flights.sqlite
##   Extensions: TRUE
```

```
flight_tbl = dplyr::copy_to(db, nycflights13::flights, name = "flights", temporary = FALSE)
flight_tbl
```

```
## # Source:   table<flights> [?? x 19]
## # Database: sqlite 3.37.0 [flights.sqlite]
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # … with more rows, and 11 more variables: arr_delay <dbl>, carrier <chr>,
```

What have we created?

All of this data now lives in the database on the filesystem not in memory,

```
pryr::object_size(db)
```

```
## 2,456 B
```

```
pryr::object_size(flight_tbl)
```

```
## 6,192 B
```

```
pryr::object_size(nycflights13::flights)
```

```
## 40,650,048 B
```

```
ls -lah *.sqlite
```

```
## -rw-r--r-- 1 root root 22M Dec 30 16:31 flights.sqlite
```

What is `flight_tbl`?

```
class(nycflights13::flights)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
class(flight_tbl)
```

```
## [1] "tbl_SQLiteConnection" "tbl_dbi"              "tbl_sql"
## [4] "tbl_lazy"             "tbl"
```

```
str(flight_tbl)
```

```
## List of 2
##  $ src:List of 2
##   ..$ con  :Formal class 'SQLiteConnection' [package "RSQLite"] with 8 slots
##   .. .. ..@ ptr                :<externalptr>
##   .. .. ..@ dbname             : chr "flights.sqlite"
##   .. .. ..@ loadable.extensions: logi TRUE
##   .. .. ..@ flags              : int 70
##   .. .. ..@ vfs                : chr ""
##   .. .. ..@ ref                :<environment: 0x55ed1b64fbe0>
##   .. .. ..@ bigint             : chr "integer64"
##   .. .. ..@ extended_types     : logi FALSE
##   ..$ disco: NULL
##   ..- attr(*, "class")= chr [1:4] "src_SQLiteConnection" "src_dbi" "src_sql" "src"
```

Accessing existing tables

```
(dplyr::tbl(db, "flights"))
```

```
## # Source:   table<flights> [?? x 19]
## # Database: sqlite 3.37.0 [flights.sqlite]
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # … with more rows, and 11 more variables: arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
## #   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dbl>
```

Using dplyr with sqlite

```r
(oct_21 = flight_tbl %>%
   filter(month == 10, day == 21) %>%
   select(origin, dest, tailnum)
)
```

```
## # Source:   lazy query [?? x 3]
## # Database: sqlite 3.37.0 [flights.sqlite]
##    origin dest  tailnum
##    <chr>  <chr> <chr>
##  1 EWR    CLT   N152UW
##  2 EWR    IAH   N535UA
##  3 JFK    MIA   N5BSAA
##  4 JFK    SJU   N531JB
##  5 JFK    BQN   N827JB
##  6 LGA    IAH   N15710
##  7 JFK    IAD   N825AS
##  8 EWR    TPA   N802UA
##  9 LGA    ATL   N996DL
## 10 JFK    FLL   N627JB
## # … with more rows
```

```r
dplyr::collect(oct_21)
```

```
## # A tibble: 991 × 3
##    origin dest  tailnum
##    <chr>  <chr> <chr>
##  1 EWR    CLT   N152UW
##  2 EWR    IAH   N535UA
##  3 JFK    MIA   N5BSAA
##  4 JFK    SJU   N531JB
##  5 JFK    BQN   N827JB
##  6 LGA    IAH   N15710
##  7 JFK    IAD   N825AS
##  8 EWR    TPA   N802UA
##  9 LGA    ATL   N996DL
## 10 JFK    FLL   N627JB
## # … with 981 more rows
```

Laziness

dplyr / dbplyr uses lazy evaluation as much as possible, particularly when working with non-local backends.

- When building a query, we don't want the entire table, often we want just enough to check if our query is working / makes sense.
- Since we would prefer to run one complex query over many simple queries, laziness allows for verbs to be strung together.
- Therefore, by default `dplyr`
  - won't connect and query the database until absolutely necessary (e.g. show output),
  - and unless explicitly told to, will only query a handful of rows to give a sense of what the result will look like.
  - we can force evaluation via `compute()`, `collect()`, or `collapse()`

A crude benchmark

```
system.time({
  (oct_21 = flight_tbl %>%
    filter(month == 10, day == 21) %>%
    select(origin, dest, tailnum)
  )
})
```

```
##    user  system elapsed 
##   0.004   0.000   0.005
```

```
system.time({
  print(oct_21) %>%
    capture.output() %>%
    invisible()
})
```

```
##    user  system elapsed 
##   0.039   0.000   0.038
```

```
system.time({
  dplyr::collect(oct_21) %>%
    capture.output() %>%
    invisible()
})
```

```
##    user  system elapsed 
##   0.063   0.000   0.063
```

dplyr -> SQL - dplyr::show_query()

```
class(oct_21)
```

```
## [1] "tbl_SQLiteConnection" "tbl_dbi"              "tbl_sql"
## [4] "tbl_lazy"             "tbl"
```

```
show_query(oct_21)
```

```
## <SQL>
## SELECT `origin`, `dest`, `tailnum`
## FROM `flights`
## WHERE ((`month` = 10.0) AND (`day` = 21.0))
```

More complex queries

```
oct_21 %>% group_by(origin, dest) %>% summarize(n=n())
```

```
## `summarise()` has grouped output by 'origin'. You can override using the `.groups` argument.
```

```
## # Source:   lazy query [?? x 3]
## # Database: sqlite 3.37.0 [flights.sqlite]
## # Groups:   origin
##    origin dest      n
##    <chr>  <chr> <int>
##  1 EWR    ATL      15
##  2 EWR    AUS       3
##  3 EWR    AVL       1
##  4 EWR    BNA       7
##  5 EWR    BOS      17
##  6 EWR    BTV       3
##  7 EWR    BUF       2
##  8 EWR    BWI       1
##  9 EWR    CHS       4
## 10 EWR    CLE       4
## # … with more rows
```

```
oct_21 %>% group_by(origin, dest) %>% summarize(n=n()) %>% show_query()
```

```
## `summarise()` has grouped output by 'origin'. You can override using the `.groups` argument.
```

```
oct_21 %>% count(origin, dest) %>% show_query()
```

```
## <SQL>
## SELECT `origin`, `dest`, COUNT(*) AS `n`
## FROM (SELECT `origin`, `dest`, `tailnum`
## FROM `flights`
## WHERE ((`month` = 10.0) AND (`day` = 21.0)))
## GROUP BY `origin`, `dest`
```

SQL Translation

In general, dplyr / dbplyr knows how to translate basic math, logical, and summary functions from R to SQL. dbplyr has a function, `translate_sql`, that lets you experiment with how R functions are translated to SQL.

```
dbplyr::translate_sql(x == 1 & (y < 2 | z > 3))
```

```
## <SQL> `x` = 1.0 AND (`y` < 2.0 OR `z` > 3.0)
```

```
dbplyr::translate_sql(x ^ 2 < 10)
```

```
## <SQL> POWER(`x`, 2.0) < 10.0
```

```
dbplyr::translate_sql(x %% 2 == 10)
```

```
## <SQL> `x` % 2.0 = 10.0
```

```
dbplyr::translate_sql(mean(x))
```

```
## Warning: Missing values are always removed in SQL.
## Use `AVG(x, na.rm = TRUE)` to silence this warning
## This warning is displayed only once per session.
```

```
## <SQL> AVG(`x`) OVER ()
```

```r
dbplyr::translate_sql(sd(x))
```

```
## <SQL> sd(`x`)
```

```r
dbplyr::translate_sql(paste(x,y))
```

```
## <SQL> CONCAT_WS(' ', `x`, `y`)
```

```r
dbplyr::translate_sql(cumsum(x))
```

```
## Warning: Windowed expression 'SUM(`x`)' does not have explicit order.
## Please use arrange() or window_order() to make determinstic.
```

```
## <SQL> SUM(`x`) OVER (ROWS UNBOUNDED PRECEDING)
```

```r
dbplyr::translate_sql(lag(x))
```

```
## <SQL> LAG(`x`, 1, NULL) OVER ()
```

Dialectic variations?

By default `dbplyr::translate_sql()` will translate R / dplyr code into ANSI SQL, if we want to see results specific to a certain database we can pass in a connection object,

```
dbplyr::translate_sql(sd(x), con = db)
```

```
## Warning: Missing values are always removed in SQL.
## Use `STDEV(x, na.rm = TRUE)` to silence this warning
## This warning is displayed only once per session.
```

```
## <SQL> STDEV(`x`) OVER ()
```

```
dbplyr::translate_sql(paste(x,y), con = db)
```

```
## <SQL> `x` || ' ' || `y`
```

```
dbplyr::translate_sql(cumsum(x), con = db)
```

```
## Warning: Windowed expression 'SUM(`x`)' does not have explicit order.
## Please use arrange() or window_order() to make determinstic.
```

```
## <SQL> SUM(`x`) OVER (ROWS UNBOUNDED PRECEDING)
```

```
dbplyr::translate_sql(lag(x), con = db)
```

Complications?

```
oct_21 %>% mutate(tailnum_n_prefix = grepl("^N", tailnum))
```

```
## Error: no such function: grepl
```

```
oct_21 %>% mutate(tailnum_n_prefix = grepl("^N", tailnum)) %>% show_query()
```

```
## <SQL>
## SELECT `origin`, `dest`, `tailnum`, grepl('^N', `tailnum`) AS `tailnum_n_prefix`
## FROM `flights`
## WHERE ((`month` = 10.0) AND (`day` = 21.0))
```

SQL -> R / dplyr

Running SQL queries against R objects

There are two packages that implement this in R which take very different approaches,

- `tidyquery` - this package parses your SQL code using the `queryparser` package and then translates the result into R / dplyr code.
- `sqldf` - transparently creates a database with teh data and then runs the query using that database. Defaults to SQLite but other backends are available.

tidyquery

```
data(flights, package = "nycflights13")

tidyquery::query(
  "SELECT origin, dest, COUNT(*) AS n
   FROM flights
   WHERE month = 10 AND day = 21
   GROUP BY origin, dest"
)
```

```
## # A tibble: 181 × 3
##    origin dest      n
##    <chr>  <chr> <int>
##  1 EWR    ATL      15
##  2 EWR    AUS       3
##  3 EWR    AVL       1
##  4 EWR    BNA       7
##  5 EWR    BOS      17
##  6 EWR    BTV       3
##  7 EWR    BUF       2
##  8 EWR    BWI       1
##  9 EWR    CHS       4
## 10 EWR    CLE       4
## # … with 171 more rows
```

```
flights %>%
  tidyquery::query(
    "SELECT origin, dest, COUNT(*) AS n
     WHERE month = 10 AND day = 21
     GROUP BY origin, dest"
  ) %>%
  arrange(desc(n))
```

```
## # A tibble: 181 × 3
##    origin dest      n
##    <chr>  <chr> <int>
##  1 JFK    LAX      32
##  2 LGA    ORD      31
##  3 LGA    ATL      30
##  4 JFK    SFO      24
##  5 LGA    CLT      22
##  6 EWR    ORD      18
##  7 EWR    SFO      18
##  8 EWR    BOS      17
##  9 LGA    MIA      17
## 10 EWR    LAX      16
## # … with 171 more rows
```

Translating to dplyr

```
tidyquery::show_dplyr(
  "SELECT origin, dest, COUNT(*) AS n
   FROM flights
   WHERE month = 10 AND day = 21
   GROUP BY origin, dest"
)
```

```
## flights %>%
##   filter(month == 10 & day == 21) %>%
##   group_by(origin, dest) %>%
##   summarise(n = dplyr::n()) %>%
##   ungroup()
```

sqldf

```r
sqldf::sqldf(
  "SELECT origin, dest, COUNT(*) AS n
   FROM flights
   WHERE month = 10 AND day = 21
   GROUP BY origin, dest"
)
```

```
## Warning: no DISPLAY variable so Tk is not avail
```

```
##      origin dest  n
## 1       EWR  ATL 15
## 2       EWR  AUS  3
## 3       EWR  AVL  1
## 4       EWR  BNA  7
## 5       EWR  BOS 17
## 6       EWR  BTV  3
## 7       EWR  BUF  2
## 8       EWR  BWI  1
## 9       EWR  CHS  4
## 10      EWR  CLE  4
## 11      EWR  CLT 15
## 12      EWR  CMH  3
## 13      EWR  CVG  9
## 14      EWR  DAY  4
```

```r
sqldf::sqldf(
  "SELECT origin, dest, COUNT(*) AS n
   FROM flights
   WHERE month = 10 AND day = 21
   GROUP BY origin, dest"
) %>%
  as_tibble() %>%
  arrange(desc(n))
```

```
## # A tibble: 181 × 3
##    origin dest      n
##    <chr>  <chr> <int>
##  1 JFK    LAX      32
##  2 LGA    ORD      31
##  3 LGA    ATL      30
##  4 JFK    SFO      24
##  5 LGA    CLT      22
##  6 EWR    ORD      18
##  7 EWR    SFO      18
##  8 EWR    BOS      17
##  9 LGA    MIA      17
## 10 EWR    LAX      16
## # … with 171 more rows
```

Query performance

Setup

To give us a bit more variety, I am going to add one more table to our SQLite database - `nycflights13::planes` which has details on the characteristics of the planes in the dataset as identified by their tail numbers.

```
dplyr::copy_to(db, nycflights13::planes, name = "planes", temporary = FALSE)
```

All of the following code will be run in the SQLite commandline interface, to make sure you have the database make sure you've created the database and copied both the flights and planes tables into the db.

The database can then be opened from the terminal tab using,

```
sqlite3 flights.sqlite
```

As before we should set a could of configuration options so that our output is readable, we include `.timer on` so that we get time our queries.

```
sqlite> .headers on
sqlite> .mode column
sqlite> .timer on
```

```
sqlite> select * from flights limit 10;
year  month  day  dep_time  sched_dep_time  dep_delay  arr_time  sched_arr_time  arr_delay  carrier  flight  tailnum  origin  dest  air
----  -----  ---  --------  --------------  ---------  --------  --------------  ---------  -------  ------  -------  ------  ----  ---
2013  1      1    517       515             2.0        830       819             11.0       UA       1545    N14228   EWR     IAH   227
2013  1      1    533       529             4.0        850       830             20.0       UA       1714    N24211   LGA     IAH   227
2013  1      1    542       540             2.0        923       850             33.0       AA       1141    N619AA   JFK     MIA   160
2013  1      1    544       545             -1.0       1004      1022            -18.0      B6       725     N804JB   JFK     BQN   183
2013  1      1    554       600             -6.0       812       837             -25.0      DL       461     N668DN   LGA     ATL   116
2013  1      1    554       558             -4.0       740       728             12.0       UA       1696    N39463   EWR     ORD   150
2013  1      1    555       600             -5.0       913       854             19.0       B6       507     N516JB   EWR     FLL   158
2013  1      1    557       600             -3.0       709       723             -14.0      EV       5708    N829AS   LGA     IAD   53.
2013  1      1    557       600             -3.0       838       846             -8.0       B6       79      N593JB   JFK     MCO   140
2013  1      1    558       600             -2.0       753       745             8.0        AA       301     N3ALAA   LGA     ORD   138
Run Time: real 0.051 user 0.000258 sys 0.000126
```

```
sqlite> select * from planes limit 10;
tailnum  year  type                     manufacturer      model      engines  seats  speed  engine
-------  ----  -----------------------  ----------------  ---------  -------  -----  -----  ---------
N10156   2004  Fixed wing multi engine  EMBRAER           EMB-145XR  2        55            Turbo-fan
N102UW   1998  Fixed wing multi engine  AIRBUS INDUSTRIE  A320-214   2        182           Turbo-fan
N103US   1999  Fixed wing multi engine  AIRBUS INDUSTRIE  A320-214   2        182           Turbo-fan
N104UW   1999  Fixed wing multi engine  AIRBUS INDUSTRIE  A320-214   2        182           Turbo-fan
N10575   2002  Fixed wing multi engine  EMBRAER           EMB-145LR  2        55            Turbo-fan
```

Exercise 1

Write a query that determines the total number of seats available on all of the planes that flew out of New York in 2013.

Options

- incorrect

```
sqlite> select sum(seats) from flights natural left join planes;

sum(seats)
----------
614366
Run Time: real 0.148 user 0.139176 sys 0.007804
```

- join and select

```
sqlite> select sum(seats) from flights left join planes using (tailnum);

sum(seats)
----------
38851317
Run Time: real 0.176 user 0.167993 sys 0.007354
```

- select then join

EXPLAIN QUERY PLAN

```
sqlite> explain query plan select sum(seats) from flights left join planes using (tailnum);

QUERY PLAN
|--SCAN flights
`--SEARCH planes USING AUTOMATIC COVERING INDEX (tailnum=?)
```

```
sqlite> explain query plan select sum(seats) from (select tailnum from flights) left join (select tailnum

QUERY PLAN
|--MATERIALIZE SUBQUERY 2
|  `--SCAN planes
|--SCAN flights
`--SEARCH SUBQUERY 2 USING AUTOMATIC COVERING INDEX (tailnum=?)
```

Key things to look for:

- `SCAN` - indicates that a full table scan is occurring
- `SEARCH` - indicates that only a subset of the table rows are visited
- `AUTOMATIC COVERING INDEX` - indicates that a temporary index has been created for this query

Adding indexes

```
sqlite> create index flight_tailnum on flights (tailnum);
Run Time: real 0.241 user 0.210099 sys 0.027611
```

```
sqlite> create index plane_tailnum on planes (tailnum);
Run Time: real 0.003 user 0.001407 sys 0.001442
```

```
sqlite> .indexes
flight_tailnum  plane_tailnum
```

Improvements?

```
sqlite> select sum(seats) from flights left join planes using (tailnum);
sum(seats)
----------
38851317
Run Time: real 0.118 user 0.115899 sys 0.001952
```

```
sqlite> select sum(seats) from (select tailnum from flights) left join (select tailnum, seats from planes
sum(seats)
----------
38851317
Run Time: real 0.131 user 0.129165 sys 0.001214
```

```
sqlite> explain query plan select sum(seats) from flights left join planes using (tailnum);
QUERY PLAN
|--SCAN flights USING COVERING INDEX flight_tailnum
`--SEARCH planes USING INDEX plane_tailnum (tailnum=?)
```

```
sqlite> explain query plan select sum(seats) from (select tailnum from flights) left join (select tailnum
QUERY PLAN
|--MATERIALIZE SUBQUERY 2
|  `--SCAN planes
|--SCAN flights USING COVERING INDEX flight_tailnum
```

Filtering

```
sqlite> select origin, count(*) from flights where origin = "EWR";
origin  count(*)
------  --------
EWR     120835
Run Time: real 0.034 user 0.028124 sys 0.005847
```

```
sqlite> explain query plan select origin, count(*) from flights where origin = "EWR";
QUERY PLAN
`--SCAN flights
```

```
sqlite> select origin, count(*) from flights where origin != "EWR";
origin  count(*)
------  --------
LGA     215941
Run Time: real 0.036 user 0.029798 sys 0.006171
```

```
sqlite> explain query plan select origin, count(*) from flights where origin != "EWR";
QUERY PLAN
`--SCAN flights
```

```
sqlite> create index flights_orig_dest on flights (origin, dest);
Run Time: real 0.267 user 0.232886 sys 0.030270
```

Filtering w/ indexes

```
sqlite> select origin, count(*) from flights where origin = "EWR";
origin  count(*)
------  --------
EWR     120835
Run Time: real 0.007 user 0.006419 sys 0.000159

sqlite> select origin, count(*) from flights where origin != "EWR";
origin  count(*)
------  --------
JFK     215941
Run Time: real 0.020 user 0.019203 sys 0.000497
```

```
sqlite> explain query plan select origin, count(*) from flights where origin = "EWR";
QUERY PLAN
`--SEARCH flights USING COVERING INDEX flights_orig_dest (origin=?)

sqlite> explain query plan select origin, count(*) from flights where origin != "EWR";
QUERY PLAN
`--SCAN flights USING COVERING INDEX flights_orig_dest
```

!= alternative

```
sqlite> select origin, count(*) from flights where origin > "EWR" OR origin < "EWR";
origin  count(*)
------  --------
JFK     215941
Run Time: real 0.022 user 0.021148 sys 0.001290
```

```
sqlite> explain query plan select origin, count(*) from flights where origin > "EWR" OR origin < "EWR";
QUERY PLAN
`--MULTI-INDEX OR
   |--INDEX 1
   |  `--SEARCH flights USING COVERING INDEX flights_orig_dest (origin>?)
   `--INDEX 2
      `--SEARCH flights USING COVERING INDEX flights_orig_dest (origin<?)
```

What about dest?

```
sqlite> select dest, count(*) from flights where dest = "LAX";
dest  count(*)
----  --------
LAX   16174
Run Time: real 0.017 user 0.016513 sys 0.000237
```

```
sqlite> explain query plan select dest, count(*) from flights where dest = "LAX";
QUERY PLAN
`--SCAN flights USING COVERING INDEX flights_orig_dest
```

```
sqlite> select dest, count(*) from flights where dest = "LAX" AND origin = "EWR";
dest  count(*)
----  --------
LAX   4912
Run Time: real 0.003 user 0.000729 sys 0.000778
```

```
sqlite> explain query plan select dest, count(*) from flights where dest = "LAX" AND origin = "EWR";
QUERY PLAN
`--SEARCH flights USING COVERING INDEX flights_orig_dest (origin=? AND dest=?)
```

Group bys

```
sqlite> select carrier, count(*) from flights group by carrier;
carrier  count(*)
-------  --------
9E       18460
AA       32729
AS       714
B6       54635
DL       48110
EV       54173
F9       685
FL       3260
HA       342
MQ       26397
OO       32
UA       58665
US       20536
VX       5162
WN       12275
YV       601
Run Time: real 0.172 user 0.114274 sys 0.018946

sqlite> explain query plan select carrier, count(*) from flight
QUERY PLAN
|--SCAN flights
`--USE TEMP B-TREE FOR GROUP BY
```

```
sqlite> create index flight_carrier on flights (c
```

```
sqlite> select carrier, count(*) from flights group by carrier;
carrier  count(*)
-------  --------
9E       18460
AA       32729
AS       714
B6       54635
DL       48110
EV       54173
F9       685
FL       3260
HA       342
MQ       26397
OO       32
UA       58665
US       20536
VX       5162
WN       12275
YV       601
Run Time: real 0.023 user 0.022521 sys 0.000411

sqlite> explain query plan select carrier, count(*) from flight
QUERY PLAN
`--SCAN flights USING COVERING INDEX flight_carrier
```

Why not index all the things?

- As mentioned before, creating an index requires additional storage (memory or disk)
- Additionally, when adding or updating data - indexes also need to be updated, making these processes slower (read vs. write tradeoffs)
- Index order matters - `flights (origin, dest)`, `flights (dest, origin)` are not the same and similarly are not the same as separate indexes on `dest` and `origin`.