LAB: dplyr and SQL

2024-01-31

- **M1 MIDS & MFA**
- **Université Paris Cité**
- Année 2023-2024
- Course Homepage

- Moodle

```
to_be_loaded <- c("tidyverse",
                  "glue",
                  "cowplot",
                  "patchwork",
                  "nycflights13",
                  "DBI",
                  "RSQLite",
                  "RPostgreSQL",
                  "dtplyr",
                  "dbplyr"
)

for (pck in to_be_loaded) {
  if (!require(pck, character.only = T)) {
    install.packages(pck, repos="http://cran.rstudio.com/")
    stopifnot(require(pck, character.only = T))
  }
}
```

```
old_theme <- theme_set(theme_minimal())
```

Objectives

Loading `nycflights`

In memory

```
flights <- nycflights13::flights
weather <- nycflights13::weather
```

```
airports <- nycflights13::airports
airlines <- nycflights13::airlines
planes <- nycflights13::planes
```

```
con <- DBI::dbConnect(RSQLite::SQLite(), ":memory:")
flights_lite <- copy_to(con, nycflights13::flights)
airports_lite <- copy_to(con, nycflights13::airports)
planes_lite <- copy_to(con, nycflights13::planes)
weather_lite <- copy_to(con, nycflights13::weather)
airlines_lite <- copy_to(con, nycflights13::airlines)
```

Pure relational algebra $\sigma, \pi, \bowtie, \cap, \cup, \setminus$

Projection, π, `select(...)`

Projection of a table R on columns $A_{i_1}, \ldots, A_{i_k}$ results in a table with schema $A_{i_1}, \ldots, A_{i_k}$ and one row for each row of R. Projection is denoted by $\pi(R, A_{i_1}, \ldots, A_{i_k})$.

In SQL this reads as

```
SELECT Ai1, , Aik
FROM R
```

In the sequel, we illustrate operations on the next two toy tables

Table `R`

A	B	C
2024-02-10	6	q
2024-02-01	7	y
2024-02-07	1	a
2024-02-01	5	l
2024-02-10	10	d
2024-02-05	2	e
2024-02-05	5	h
2024-02-02	4	l
2024-02-04	8	l
2024-02-06	6	l

Table `S`

A	D	F
2024-02-06	21	u
2024-02-07	29	v
2024-02-05	24	q
2024-02-01	22	s
2024-02-06	23	j
2024-02-05	23	z
2024-02-04	26	k
2024-02-04	20	e

In Relational Algebra, tables are sets rather than multisets, there are no duplicates. In SQL we are handling multisets of rows, duplicates need to be removed explicitly

```
SELECT DISTINCT Ai1, ..., Aik
FROM R
```

`dplyr` has one verb `select(...)` for π or `SELECT`, and verb `distinct()` for `SELECT DISTINCT ...`.

If we have no intention to remove duplicates:

```
select(R, Ai1, ..., Aik)
# or
R |>
  select(Ai1, ..., Aik)
```

If we want to remove deplicates

```
distinct(R, Ai1, ..., Aik)
# or
R |>
  distinct(Ai1, ..., Aik)
```

$\pi(R, B, C)$ (`SELECT B, C FROM R`) leads to

B	C
6	q
7	y
1	a
5	l
10	d
2	e
5	h
4	l
8	l
6	l

$\pi(R, B)$ and `SELECT DISTINCT B FROM R` lead to

B
6
7
1
5
10
2
4
8

> For each departure airport (denoted by `origin'), each day of the year, list the codes (denoted by`carrier‘) of the airlines that have one or more planes taking off from that airport on that day.

In SQL, we can phrase this query like this:

```
SELECT DISTINCT f.origin, f.year, f.month, f.day, f.carrier
FROM nycflights.flights f ;
```

Using `dplyr` and chaining with standard pipe `|>` or `%>%` from `magrittr`, we can write.

```
q1 <- . %>%                                                    ①
  distinct(origin, year, month, day, carrier)

q1(flights) |>
  head()
```

① To define a unary function implementing the short pipeline, we have to use `%>%`.

```
# A tibble: 6 x 5
  origin  year month   day carrier
  <chr>  <int> <int> <int> <chr>
1 EWR     2013     1     1 UA
2 LGA     2013     1     1 UA
3 JFK     2013     1     1 AA
4 JFK     2013     1     1 B6
5 LGA     2013     1     1 DL
6 EWR     2013     1     1 B6
```

We can reuse the pipeline to query the lazy tables., we can even

```
q1(flights_lite) |>
  head()
```

```
# Source:   SQL [6 x 5]
# Database: sqlite 3.41.2 [:memory:]
  origin  year month   day carrier
  <chr>  <int> <int> <int> <chr>
1 EWR     2013     1     1 UA
2 LGA     2013     1     1 UA
3 JFK     2013     1     1 AA
4 JFK     2013     1     1 B6
5 LGA     2013     1     1 DL
6 EWR     2013     1     1 B6
```

```
q1(flights_lite) |>
  show_query()
```

```
<SQL>
SELECT DISTINCT `origin`, `year`, `month`, `day`, `carrier`
FROM `nycflights13::flights`
```

```
q1(flights_lite) |>
  explain()
```

```
<SQL>
SELECT DISTINCT `origin`, `year`, `month`, `day`, `carrier`
FROM `nycflights13::flights`

<PLAN>
  id parent notused                        detail
```

Selection, σ, `filter(...)`

Selection of a table R according to condition `expr` is an expression that can be evaluated on each row of R results in a table with the same schema as R and all rows of R where `expr` evaluates to `TRUE`. Selection is denoted by $\sigma(R, \texttt{expr})$.

In SQL this reads as

```
SELECT R.*
FROM R
WHERE expr
```

`dplyr` has one verb `filter(...)` for σ.

$\sigma(R, A < \text{2024-02-06} \wedge \text{2024-02-02} \leq A)$ (`SELECT * FROM R WHERE A < CAST('2024-02-06' AS DATE) AND A >= CAST('2024-02-02' AS DATE)`) leads to

A	B	C
2024-02-05	2	e
2024-02-05	5	h
2024-02-02	4	l
2024-02-04	8	l

`SELECT DISTINCT B FROM R` leads to

B
6
7
1
5
10
2
4
8

List all the planes built by a manufacturer named like `AIRBUS` between `2005` and `2010`

```
qx <- . %>%
  filter(str_like(manufacturer, 'AIRBUS%') &  year >= 2005 & year <= 2010)

qx(planes)
```

```
# A tibble: 143 x 9
   tailnum  year type              manufacturer model engines seats speed engine
   <chr>   <int> <chr>             <chr>        <chr>   <int> <int> <int> <chr>
 1 N125UW   2009 Fixed wing multi~ AIRBUS       A320~       2   182    NA Turbo~
 2 N126UW   2009 Fixed wing multi~ AIRBUS       A320~       2   182    NA Turbo~
 3 N127UW   2010 Fixed wing multi~ AIRBUS       A320~       2   182    NA Turbo~
 4 N128UW   2010 Fixed wing multi~ AIRBUS       A320~       2   182    NA Turbo~
 5 N193UW   2008 Fixed wing multi~ AIRBUS       A321~       2   199    NA Turbo~
 6 N195UW   2008 Fixed wing multi~ AIRBUS       A321~       2   199    NA Turbo~
 7 N196UW   2009 Fixed wing multi~ AIRBUS       A321~       2   199    NA Turbo~
 8 N197UW   2009 Fixed wing multi~ AIRBUS       A321~       2   199    NA Turbo~
 9 N201FR   2008 Fixed wing multi~ AIRBUS       A320~       2   182    NA Turbo~
10 N202FR   2008 Fixed wing multi~ AIRBUS       A320~       2   182    NA Turbo~
# i 133 more rows
```

```
qx(planes_lite) |>
  show_query()
```

```
<SQL>
SELECT *
FROM `nycflights13::planes`
WHERE (`manufacturer` LIKE 'AIRBUS%' AND `year` >= 2005.0 AND `year` <= 2010.0)
```

Joins, $\bowtie$, `xxx_join(...)`

In relational algebra, a θ-join boils down to a selection according to expression θ over a cross product (possibly after renaming some columns)

$$\bowtie (R, S, \theta) \approx \sigma(R \times S, \theta)$$

`dplyr` does not (yet?) offer such a general join (which anyway can be very expensive on a cutting edge RDBMS) several variants of *equijoin*.

`ChatGPT` asserts:

> An equijoin is a type of join operation in relational databases where the join condition involves an equality comparison between two attributes from different tables. In other words, it's a join operation that combines rows from two tables based on matching values in specified columns. These specified columns are usually called the "join columns" or "join keys."

Joins in `dplyr` documentation

- `inner_join()`

- `left_join()`
- `right_join()`
- `full_join()`

but also

- `semi_join()`
- `anti_join()`

the matching columns are stated using optional argument `by=...`

If argument `by` is omitted, `NATURAL JOIN` is assumed.

$\bowtie$ (R, S) (`SELECT * FROM R NATURAL JOIN S`) leads to

```
Joining with `by = join_by(A)`
Warning in inner_join(R, S): Detected an unexpected many-to-many relationship between `x` and `y`.
i Row 6 of `x` matches multiple rows in `y`.
i Row 4 of `y` matches multiple rows in `x`.
i If a many-to-many relationship is expected, set `relationship =
  "many-to-many"` to silence this warning.
```

A	B	C	D	F
2024-02-01	7	y	22	s
2024-02-07	1	a	29	v
2024-02-01	5	l	22	s
2024-02-05	2	e	24	q
2024-02-05	2	e	23	z
2024-02-05	5	h	24	q
2024-02-05	5	h	23	z
2024-02-04	8	l	26	k
2024-02-04	8	l	20	e
2024-02-06	6	l	21	u
2024-02-06	6	l	23	j

$\bowtie^{\text{Left outer}}$ (R, S) and `SELECT * FROM R LEFT JOIN S ON (R.A=S.A)` lead to

```
Joining with `by = join_by(A)`
Warning in left_join(R, S): Detected an unexpected many-to-many relationship between `x` and `y`.
i Row 6 of `x` matches multiple rows in `y`.
i Row 4 of `y` matches multiple rows in `x`.
i If a many-to-many relationship is expected, set `relationship =
  "many-to-many"` to silence this warning.
```

A	B	C	D	F
2024-02-10	6	q	NA	NA
2024-02-01	7	y	22	s
2024-02-07	1	a	29	v
2024-02-01	5	l	22	s
2024-02-10	10	d	NA	NA
2024-02-05	2	e	24	q
2024-02-05	2	e	23	z
2024-02-05	5	h	24	q
2024-02-05	5	h	23	z
2024-02-02	4	l	NA	NA
2024-02-04	8	l	26	k
2024-02-04	8	l	20	e
2024-02-06	6	l	21	u
2024-02-06	6	l	23	j

List weather conditions at departure for all flights operated by airline named `Delta` ....

Agregation, `summarize(...)`

Partition, `group_by`

```
SELECT f.origin, f.carrier, COUNT(*) AS n
FROM nycflights.flights f
GROUP BY f.origin, f.carrier
ORDER BY f.carrier, n DESC;
```

```
q4 <- . %>%
  group_by(carrier, origin) %>%
  summarise(n=n(), .groups="drop") %>%
  arrange(desc(carrier))

q4(flights)
```

```
# A tibble: 35 x 3
   carrier origin     n
   <chr>   <chr>  <int>
 1 YV      LGA      601
 2 WN      EWR     6188
 3 WN      LGA     6087
 4 VX      EWR     1566
 5 VX      JFK     3596
 6 US      EWR     4405
 7 US      JFK     2995
 8 US      LGA    13136
 9 UA      EWR    46087
10 UA      JFK     4534
# i 25 more rows
```

```
q4(flights_lite) |>
  show_query()
```

```
<SQL>
SELECT `carrier`, `origin`, COUNT(*) AS `n`
FROM `nycflights13::flights`
GROUP BY `carrier`, `origin`
ORDER BY `carrier` DESC
```

```sql
SELECT f.origin, f.year, f.month, f.day, f.carrier, COUNT(DISTINCT tailnum)
FROM nycflights.flights f
GROUP BY f.origin, f.year, f.month, f.day, f.carrier
ORDER BY f.origin, f.year, f.month, f.day, f.carrier;
```

```r
q2 <- . %>%
  group_by(origin, year, month, day, carrier) %>%
  summarise(n_tailnum=n_distinct(tailnum), .groups = "drop") %>%
  arrange(origin, year, month, day, carrier)

q2(flights) |>
  head()
```

```
# A tibble: 6 x 6
  origin  year month   day carrier n_tailnum
  <chr>  <int> <int> <int> <chr>       <int>
1 EWR     2013     1     1 AA              9
2 EWR     2013     1     1 AS              2
3 EWR     2013     1     1 B6             18
4 EWR     2013     1     1 DL              6
5 EWR     2013     1     1 EV             66
6 EWR     2013     1     1 MQ              7
```

```r
q2(flights_lite) |>
  show_query()
```

```
<SQL>
SELECT
  `origin`,
  `year`,
  `month`,
  `day`,
  `carrier`,
  COUNT(DISTINCT `tailnum`) AS `n_tailnum`
FROM `nycflights13::flights`
GROUP BY `origin`, `year`, `month`, `day`, `carrier`
ORDER BY `origin`, `year`, `month`, `day`, `carrier`
```

Adding/modifying columns, `mutate(...)`

```sql
WITH for_hire AS (
    SELECT f.tailnum, COUNT(DISTINCT f.carrier) AS n_carrier
    FROM nycflights.flights f
    GROUP BY f.tailnum
    HAVING COUNT(DISTINCT f.carrier) >=2
)

SELECT p.*
FROM nycflights.planes p NATURAL JOIN for_hire ;
```

```r
for_hire <- . %>%
  group_by(tailnum) %>%
  summarise(n_carriers=n_distinct(carrier)) %>%
  filter(n_carriers >= 2) %>%
  select(tailnum)

for_hire(flights) |>  head()
```

```
# A tibble: 6 x 1
  tailnum
  <chr>
1 N146PQ
2 N153PQ
3 N176PQ
4 N181PQ
5 N197PQ
6 N200PQ
```

```r
planes %>%
  semi_join(for_hire(flights), by=join_by(tailnum))
```



```
# A tibble: 17 x 9
   tailnum  year type              manufacturer model engines seats speed engine
   <chr>   <int> <chr>             <chr>        <chr>   <int> <int> <int> <chr>
 1 N146PQ   2007 Fixed wing multi~ BOMBARDIER ~ CL-6~       2    95    NA Turbo~
```

Window functions

```sql
WITH f_delayed AS (
  SELECT f.*, RANK() OVER w AS rnk
  FROM nycflights.flights f
  WHERE f.dep_time IS NOT NULL
  WINDOW w AS (PARTITION BY f.origin, f.year, f.month, f.day ORDER BY f.dep_delay DESC)
)

SELECT fd.origin, fd.year, fd.month, fd.day, fd.tailnum
FROM f_delayed fd
WHERE fd.rnk <= 10;
```

```r
f_delayed <- . %>%
  filter(!is.na(dep_time)) %>%
  group_by(origin, year, month, day) %>%
  mutate(rnk=min_rank(desc(dep_delay))) %>%
  ungroup()

f_delayed(flights) |>
  filter(rnk <= 10) |>
  head()
```

```
# A tibble: 6 x 20
   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      811            630       101     1047            830
2  2013     1     1      848           1835       853     1001           1950
3  2013     1     1      957            733       144     1056            853
4  2013     1     1     1114            900       134     1447           1222
5  2013     1     1     1301           1150        71     1518           1345
6  2013     1     1     1400           1250        70     1645           1502
# i 12 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 <dttm>, rnk <int>
```

Tidy selection

Use the cheatsheet