From subqueries to CTEs

Ian D. Gow

6 January 2025

Take a query with subqueries and turn it into one with CTEs.

In this brief note, I use a query from Tanimura (2021) to illustrate how one can re-write a query that makes use of subqueries as a query using common table expressions (CTEs). I then show how a query written with CTEs is easily translated into a query using `dbplyr`. Finally, I do the analysis again from scratch, but using `dbplyr` expressions. In this analysis I find that the SQL query Tanimura (2021) does not quite meet the verbal specification it was intended to meet, while the `dbplyr` query does. I conjecture that the "building blocks" approach to SQL facilitated by `dbplyr` may enhance the accuracy of queries for many users.

1 The data

The example query I study below comes from Chapter 4 of Tanimura (2021). The following packages will be used and should be installed.

```
library(DBI)
library(tidyverse)
library(dbplyr)
```

We first get the data, which requires an internet connection.

```
download_data <- function(filename) {
  if (!dir.exists("data")) dir.create("data")

  url <- paste0("https://raw.githubusercontent.com/cathytanimura/",
                "sql_book/master/Chapter%204%3A%20Cohorts/")

  local_filename <- paste0("data/", filename)
  if (!file.exists(local_filename)) {
    download.file(url = paste0(url, filename),
```

```
                     destfile = local_filename)
  }
}

download_data("legislators.csv")
download_data("legislators_terms.csv")
```

I start by creating an in-memory DuckDB database.

```
db <- dbConnect(duckdb::duckdb())
```

We can read the two downloaded data files into our database.

```
legislators <-
  tbl(db, "read_csv_auto('data/legislators.csv')") |>
  compute(name = "legislators")
legislators_terms <-
  tbl(db, "read_csv_auto('data/legislators_terms.csv')") |>
  compute(name = "legislators_terms")
```

2 The query

Chapter 4 of Tanimura (2021) contains the query shown in Listing 1.

Table 1: Output from original SQL query in Listing 1

cohort_century	pct_5_yrs	pct_10_yrs	pct_15_yrs
18	0.0502	0.0970	0.1438
19	0.0088	0.0244	0.0409
20	0.0100	0.0348	0.0478
21	0.0400	0.0764	0.0873

This query is quite complex. Can we simplify it using CTEs?

First, notice that the subqueries labelled `a` and `b` are identical. Let's clean that up. We'll put `a` as a CTE at the beginning of the query (after `WITH`) and refer to that in both in the place where we currently have it, and also in place of `b`. All references to `b` are changed to references to `a`. We can then run the query—shown in Listing 2—to check that we are still getting the same results.

Next, let's do the same for `aa` and `bb`. Note that there are commas after the CTEs defining `a` and `aa`, but not after `bb`, as it is the last CTE before the body of the query. Again we run the query to check that we still have the same results.

Now we can delete the `aa` and `bb` labels and references to them. We can also tidy up the main query a little, including using the more elegant (in my view) `USING` syntax.

Also notice that we have `age(c.term_start, a.first_term)` three times in the query above. We can split `bb` into two, as we do in Listing 4 (now `bbb` is followed by `bb`).

We are still left with the meaningless labels (e.g., `a` and `bb`). We can give the CTEs more meaningful labels. We also clean up `ages` a little (e.g., `USING`) and move `cohort` to `cohorts`. The resulting query is shown in Listing 5. Again we check that results are the same.

Finally, let's put the "main" query in a CTE too resulting in Listing 6.

Table 2: Output from final SQL query in Listing 6

cohort	pct_5_yrs	pct_10_yrs	pct_15_yrs
18	0.0502	0.0970	0.1438
19	0.0088	0.0244	0.0409
20	0.0100	0.0348	0.0478
21	0.0400	0.0764	0.0873

What's the value of this last step? Well it means we can easily edit the query to debug the CTEs that are used. For example, we could put the following at then end of the query about to look into `cohorts`.

```
SELECT *
FROM cohorts;
```

3 Translating to dbplyr

Now that we have a query based on CTEs, it is *much* easier to translate to `dbplyr`.

```
cohorts <-
  legislators_terms |>
  filter(term_type == 'rep') |>
  group_by(id_bioguide) |>
  summarize(first_term = min(term_start, na.rm = TRUE),
            .groups = "drop") |>
  mutate(cohort = century(first_term))
```

```
cohort_sizes <-
  cohorts |>
  filter(first_term <= '2009-12-31') |>
  group_by(cohort) |>
  summarize(reps = n())
```

```
ages <-
  cohorts |>
  inner_join(legislators_terms, by = "id_bioguide") |>
  filter(term_type == 'sen', term_start > first_term) |>
  mutate(age = age(term_start, first_term)) |>
  select(cohort, id_bioguide, age)
```

```
age_cuts <-
  ages |>
  mutate(num_5_yrs = if_else(age <= years(5), id_bioguide, NA),
         num_10_yrs = if_else(age <= years(10), id_bioguide, NA),
         num_15_yrs = if_else(age <= years(15), id_bioguide, NA)) |>
  group_by(cohort) |>
  summarize(across(starts_with("num_"),
            \(x) n_distinct(x, na.rm = TRUE)))
```

```
cohort_sizes |>
  left_join(age_cuts, by = "cohort") |>
  mutate(across(starts_with("num_"), \(x) round(x * 1.0 / reps, 4))) |>
  rename_with(\(x) str_replace(x, "^num_", "pct_")) |>
  select(cohort, starts_with("pct_")) |>
  arrange(cohort) |>
  collect()
```

cohort	pct_5_yrs	pct_10_yrs	pct_15_yrs
18	0.0502	0.0970	0.1438
19	0.0088	0.0244	0.0409
20	0.0100	0.0348	0.0478
21	0.0400	0.0764	0.0873

4 Doing it again in dplyr

Now let's do it again more or less from scratch using `dbplyr`. In this version, I will build up block by block in a way that is easier (at least for me) to reason about.

For convenience, I reproduce the code we used to create `cohorts` above.

```
cohorts <-
    legislators_terms |>
    filter(term_type == 'rep') |>
    group_by(id_bioguide) |>
    summarize(first_rep_term = min(term_start, na.rm = TRUE),
              .groups = "drop")
```

Next, I produce an equivalent table for senate terms.

```
first_sen_terms <-
    legislators_terms |>
    filter(term_type == 'sen') |>
    group_by(id_bioguide) |>
    summarize(first_sen_term = min(term_start, na.rm = TRUE),
              .groups = "drop")
```

We are interested in [legislators] "start as representatives" as our cohort. We want to exclude those who start as senators and (to match the query in the book), we also want to exclude those who first term is after `'2009-12-31'`.

```
cohort_members <-
  cohorts |>
  left_join(first_sen_terms, by = "id_bioguide") |>
  filter(first_rep_term < first_sen_term | is.na(first_sen_term)) |>
  select(id_bioguide, first_rep_term) |>
  mutate(century = century(first_rep_term)) |>
  filter(first_rep_term <= '2009-12-31')
```

We can now calculate the sizes of the cohorts that we have formed.[1]

```
cohort_sizes <-
  cohort_members |>
  group_by(century) |>
  summarize(reps = n(), .groups = "drop")
```

Now let's turn to the "go on to become senators" part of our remit. This is a simple `INNER JOIN` with an inequality `first_rep_term < first_sen_term` condition. We can calculate the gap using the SQL function `age()`.

1. Here I am using the term "cohort" in a way that I object to below. Seems like another term is needed for precision here.

```
rep_then_sen <-
  cohort_members |>
  inner_join(first_sen_terms,
             join_by(id_bioguide,
                     first_rep_term < first_sen_term)) |>
  mutate(gap = age(first_sen_term, first_rep_term))
```

It is easy to check that `id_bioguide` is a valid key for `rep_then_sen`. This fact makes it easy to building up the curve of subsequent senate terms using a window function. Within each `cohort` we sum up the number of rows leading up to each value of age. This is equivalent to `count(DISTINCT id_bioguide) OVER (PARTITION BY cohort ORDER BY gap)`, but we do not need the `DISTINCT` here because each value of `id_bioguide` is unique here.

Note that there will be ties in terms of `age` here, and we deal wtih these using `max()`. That is if we had 432 representatives with `gap` of less than 5 years and 6 with `gap` of exactly 5 years, we want to step from 432 to 438 immediately.

```
rep_then_sen_gaps <-
  rep_then_sen |>
  group_by(century) |>
  window_order(gap) |>
  mutate(cum_ids = cumsum(1)) |>
  group_by(century, gap) |>
  mutate(cum_ids = max(cum_ids, na.rm = TRUE)) |>
  ungroup()
```

I can now combine the "start as representatives" table (`cohort_sizes`) with the "go on to become senators" table (`rep_then_sen_gaps`) to calculate the percentages by `century`.

```
pct_rep_then_sen <-
  rep_then_sen_gaps |>
  inner_join(cohort_sizes, by = "century") |>
  mutate(pct = cum_ids / reps)
```

Rather than writing complicated `CASE` statements, I can make a little table in R with the three cutoff values and sent that to DuckDB and turn the rows into intervals.

```
gap_cutoffs <-
  tibble(cutoff = c(5, 10, 15)) |>
  copy_to(db, df = _, name = "gap_cutoffs",
          overwrite = TRUE) |>
  mutate(cutoff = years(cutoff))
```

Now I `CROSS JOIN` `pct_rep_then_sen` and `gap_cutoffs` and calculate the `pct` values for each before using `pivot_wider` to rearrange the table to match what is shown in Tanimura (2021).

```
pct_rep_then_sen |>
  cross_join(gap_cutoffs) |>
  filter(gap <= cutoff) |>
  group_by(century, cutoff) |>
  summarize(pct = max(pct, na.rm = TRUE),
            .groups = "drop") |>
  mutate(cutoff = year(cutoff)) |>
  pivot_wider(names_from = "cutoff",
              names_prefix = "pct_",
              values_from = "pct") |>
  arrange(century) |>
  collect()
```

century	pct_5	pct_15	pct_10
18	0.0505	0.1448	0.0976
19	0.0087	0.0407	0.0244
20	0.0101	0.0478	0.0349
21	0.0109	0.0473	0.0364

But we now see different numbers. What has happened?

It turns out that there are two situations I addressed in the code that produced the table just above. First, I only included in our cohort members of Congress who *started* as representatives. The original SQL query included cohort members of Congress who had served as representatives, but who had previously served as senators (so had not *started as* representatives). Second, I was consistent in application of the `first_rep_term <= '2009-12-31'` criterion throughout. The original SQL query imposed this requirement in calculating `cohort_sizes`, but not in constructing the cohorts themselves.

I would argue that the answer I produced better matches the original question from Tanimura (2021): "What share of [legislators] start as representatives and go on to become senators? (Some senators later become representatives, but that is much less common.) Since relatively few make this transition, we'll cohort legislators by the century in which they first became a representative."[2]

Because we have access to the underlying tables as lazy data frames in R (effectively subqueries available for use to inspect), it is easy to dig into the steps of the two queries to check our reasoning.

2. Minor edits here. I am not a fan of this use of the term "cohort" as the cohorts are formed based on `rep_first_date` and only aggregation of statistics happens by `cohort`.

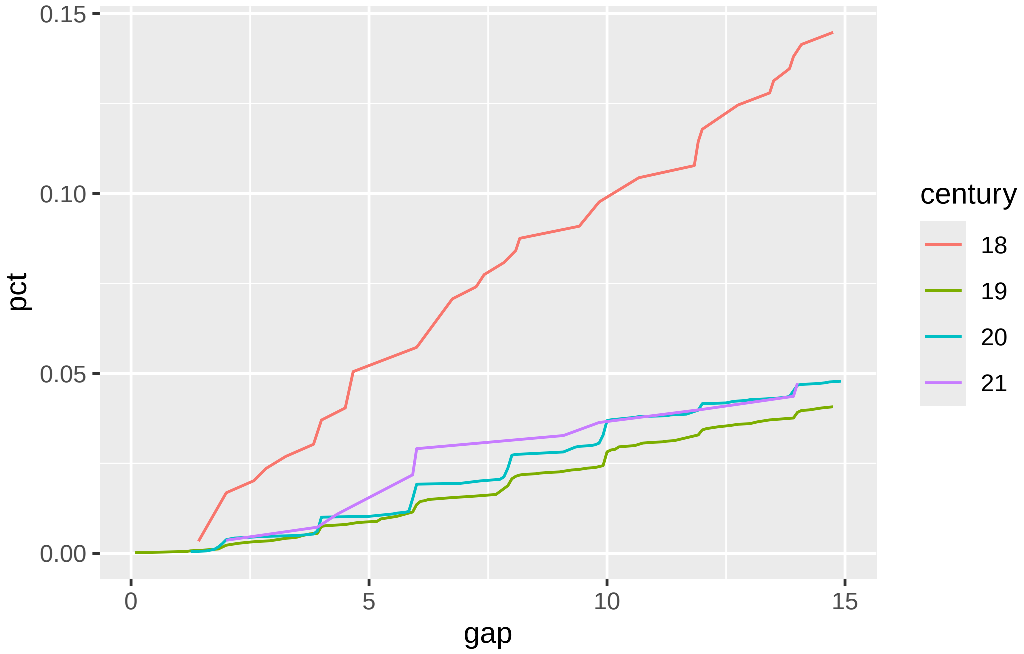
```r
last_sen_terms <-
    legislators_terms |>
    filter(term_type == 'sen') |>
    group_by(id_bioguide) |>
    summarize(last_sen_term = max(term_start, na.rm = TRUE),
              .groups = "drop")

cohorts |>
  anti_join(cohort_members, by = "id_bioguide") |>
  inner_join(first_sen_terms, by = "id_bioguide") |>
  inner_join(last_sen_terms, by = "id_bioguide") |>
  mutate(rep_to_sen = last_sen_term > first_rep_term,
         sen_to_rep = first_sen_term < first_rep_term,
         too_late = first_rep_term > '2009-12-31') |>
  arrange(desc(rep_to_sen), first_sen_term) |>
  collect()
```

id_bioguide	first_rep_term	first_sen_term	last_sen_term	rep_to_sen	sen_to_rep	too_late
C000482	1811-11-04	1806-01-01	1849-12-03	TRUE	TRUE	FALSE
J000137	1833-12-02	1818-01-01	1844-01-01	TRUE	TRUE	FALSE
M000519	1833-12-02	1826-01-01	1837-09-04	TRUE	TRUE	FALSE
S001184	2011-01-05	2013-01-03	2017-01-03	TRUE	FALSE	TRUE
C001095	2013-01-03	2015-01-06	2015-01-06	TRUE	FALSE	TRUE
L000575	2011-01-05	2015-01-06	2017-01-03	TRUE	FALSE	TRUE
G000562	2011-01-05	2015-01-06	2015-01-06	TRUE	FALSE	TRUE
D000618	2013-01-03	2015-01-06	2015-01-06	TRUE	FALSE	TRUE
Y000064	2011-01-05	2017-01-03	2017-01-03	TRUE	FALSE	TRUE
D000622	2013-01-03	2017-01-03	2017-01-03	TRUE	FALSE	TRUE
R000608	2017-01-03	2019-01-03	2019-01-03	TRUE	FALSE	TRUE
M001197	2015-01-06	2019-01-03	2019-01-03	TRUE	FALSE	TRUE
C001096	2013-01-03	2019-01-03	2019-01-03	TRUE	FALSE	TRUE
S001191	2013-01-03	2019-01-03	2019-01-03	TRUE	FALSE	TRUE
W000633	1793-12-02	1789-03-04	1789-03-04	FALSE	TRUE	FALSE
S000805	1801-12-07	1790-01-01	1790-01-01	FALSE	TRUE	FALSE
S000941	1813-05-24	1796-01-01	1796-01-01	FALSE	TRUE	FALSE
D000069	1799-12-02	1798-12-05	1798-12-05	FALSE	TRUE	FALSE
P000354	1819-12-06	1798-12-06	1799-12-02	FALSE	TRUE	FALSE
N000086	1807-10-26	1799-12-02	1799-12-02	FALSE	TRUE	FALSE
M000221	1817-12-01	1800-01-01	1800-01-01	FALSE	TRUE	FALSE
W000768	1809-05-22	1801-12-07	1801-12-07	FALSE	TRUE	FALSE
A000041	1831-12-05	1803-10-17	1803-10-17	FALSE	TRUE	FALSE
P000324	1813-05-24	1803-10-17	1805-12-02	FALSE	TRUE	FALSE

id_bioguide	first_rep_term	first_sen_term	last_sen_term	rep_to_sen	sen_to_rep	too_late
A000026	1831-12-05	1805-12-02	1805-12-02	FALSE	TRUE	FALSE
R000125	1817-12-01	1806-11-25	1807-10-26	FALSE	TRUE	FALSE
P000431	1837-09-04	1807-10-26	1807-10-26	FALSE	TRUE	FALSE
C000325	1853-12-05	1813-05-24	1825-12-05	FALSE	TRUE	FALSE
C000912	1861-07-04	1817-12-01	1855-12-03	FALSE	TRUE	FALSE
B000398	1853-12-05	1821-12-03	1845-12-01	FALSE	TRUE	FALSE
B000763	1831-12-05	1823-12-01	1823-12-01	FALSE	TRUE	FALSE
C000703	1849-12-03	1842-01-01	1842-01-01	FALSE	TRUE	FALSE
R000063	1851-03-04	1851-02-01	1851-02-01	FALSE	TRUE	FALSE
M000596	1857-12-07	1851-12-01	1851-12-01	FALSE	TRUE	FALSE
W000476	1869-03-04	1859-12-05	1859-12-05	FALSE	TRUE	FALSE
N000050	1873-12-01	1861-07-04	1861-07-04	FALSE	TRUE	FALSE
B001019	1887-12-05	1863-12-07	1863-12-07	FALSE	TRUE	FALSE
P000406	1867-03-04	1865-12-04	1865-12-04	FALSE	TRUE	FALSE
K000069	1883-12-03	1868-01-01	1877-10-15	FALSE	TRUE	FALSE
N000160	1885-12-07	1871-03-04	1871-03-04	FALSE	TRUE	FALSE
E000028	1883-12-03	1875-12-06	1875-12-06	FALSE	TRUE	FALSE
P000557	1883-12-03	1880-01-01	1880-01-01	FALSE	TRUE	FALSE
W000012	1933-03-09	1915-12-06	1921-04-11	FALSE	TRUE	FALSE
J000161	1933-03-09	1923-12-03	1923-12-03	FALSE	TRUE	FALSE
M000993	1943-01-06	1930-12-13	1930-12-13	FALSE	TRUE	FALSE
P000218	1963-01-09	1936-01-01	1945-01-03	FALSE	TRUE	FALSE
B001061	1945-01-03	1940-11-27	1941-01-03	FALSE	TRUE	FALSE
M000814	1949-01-03	1945-01-10	1945-01-10	FALSE	TRUE	FALSE
W000658	1952-08-02	1949-01-20	1949-01-20	FALSE	TRUE	FALSE
L000240	1957-01-03	1953-07-10	1953-07-10	FALSE	TRUE	FALSE

Because of the form of our second analysis, it is easy to make a plot.

```
pct_rep_then_sen |>
  mutate(gap = (year(gap) * 12 + month(gap)) / 12) |>
  filter(gap <= 15) |>
  group_by(century, gap) |>
  summarize(pct = max(pct),
            .groups = "drop") |>
  collect() |>
  mutate(century = factor(century)) |>
  ggplot(aes(x = gap, y = pct, group = century,
             colour = century)) +
  geom_line()
```



Note that it would be easy to translate our `dbplyr` code—arguably a more precise solution than the original SQL query—back into SQL (probably using CTEs) if that were desired.

References

Tanimura, C., 2021. SQL for data analysis. O'Reilly Media.

Listing 1 Original SQL code

```
SELECT aa.cohort_century::int AS cohort_century,
  round(bb.rep_and_sen_5_yrs * 1.0 / aa.reps, 4) AS pct_5_yrs,
  round(bb.rep_and_sen_10_yrs * 1.0 / aa.reps, 4) AS pct_10_yrs,
  round(bb.rep_and_sen_15_yrs * 1.0 / aa.reps, 4) AS pct_15_yrs
FROM
(
  SELECT date_part('century', a.first_term) AS cohort_century,
    count(id_bioguide) as reps
  FROM
  (
    SELECT id_bioguide, min(term_start) AS first_term
    FROM legislators_terms
    WHERE term_type = 'rep'
    GROUP BY 1
  ) a
  WHERE first_term <= '2009-12-31'
  GROUP BY 1
) aa
LEFT JOIN
(
  SELECT date_part('century', b.first_term) AS cohort_century,
    count(dISTINCT CASE WHEN age(c.term_start, b.first_term) <=
      INTERVAL '5 years'
      THEN b.id_bioguide END) AS rep_and_sen_5_yrs,
    count(DISTINCT CASE WHEn age(c.term_start, b.first_term) <=
      INTERVAL '10 years'
      THEN b.id_bioguide END) AS rep_and_sen_10_yrs,
    count(DISTINCT CASE WHEN age(c.term_start, b.first_term) <=
      INTERVAL '15 years'
      THEN b.id_bioguide END) AS rep_and_sen_15_yrs
  FROM
  (
    SELECT id_bioguide, min(term_start) AS first_term
    FROM legislators_terms
    WHERE term_type = 'rep'
    GROUP BY 1
  ) b
  JOIN legislators_terms c ON b.id_bioguide = c.id_bioguide
    AND c.term_type = 'sen' AND c.term_start > b.first_term
  GROUP BY 1
) bb ON aa.cohort_century = bb.cohort_century
ORDER BY 1;
```

Listing 2 SQL with first CTE

```
WITH a AS (
  SELECT id_bioguide, min(term_start) AS first_term
  FROM legislators_terms
  WHERE term_type = 'rep'
  GROUP BY 1)

SELECT aa.cohort::int AS cohort,
  round(bb.rep_and_sen_5_yrs * 1.0 / aa.reps, 4) AS pct_5_yrs,
  round(bb.rep_and_sen_10_yrs * 1.0 / aa.reps, 4) AS pct_10_yrs,
  round(bb.rep_and_sen_15_yrs * 1.0 / aa.reps, 4) AS pct_15_yrs
FROM
(
  SELECT date_part('century', a.first_term) AS cohort,
    count(id_bioguide) AS reps
  FROM a
  WHERE first_term <= '2009-12-31'
  GROUP BY 1
) aa
LEFT JOIN
(
  SELECT date_part('century', a.first_term) AS cohort,
    count(DISTINCT CASE WHEN age(c.term_start, a.first_term) <=
      INTERVAL '5 years'
      THEN a.id_bioguide END) AS rep_and_sen_5_yrs,
    count(DISTINCT CASE WHEN age(c.term_start, a.first_term) <=
      INTERVAL '10 years'
      THEN a.id_bioguide END) AS rep_and_sen_10_yrs,
    count(DISTINCT CASE WHEN age(c.term_start, a.first_term) <=
      INTERVAL '15 years'
      THEN a.id_bioguide END) AS rep_and_sen_15_yrs
  FROM a
  JOIN legislators_terms c ON a.id_bioguide = c.id_bioguide
  AND c.term_type = 'sen' AND c.term_start > a.first_term
  GROUP BY 1
) bb ON aa.cohort = bb.cohort
ORDER BY 1;
```

Listing 3 SQL with CTEs for aa and bb

```sql
WITH

a AS (
  SELECT id_bioguide, min(term_start) as first_term
  FROM legislators_terms
  WHERE term_type = 'rep'
  GROUP BY 1),

aa AS (
  SELECT date_part('century',a.first_term) AS cohort,
      count(id_bioguide) AS reps
  FROM a
  WHERE first_term <= '2009-12-31'
    GROUP BY 1),

bb AS (
  SELECT date_part('century', a.first_term) AS cohort,
    count(DISTINCT CASE WHEN age(c.term_start, a.first_term) <=
      INTERVAL '5 years'
      THEN a.id_bioguide END) AS rep_and_sen_5_yrs,
    count(DISTINCT CASE WHEN age(c.term_start, a.first_term) <=
      INTERVAL '10 years'
      THEN a.id_bioguide END) AS rep_and_sen_10_yrs,
    count(DISTINCT CASE WHEN age(c.term_start, a.first_term) <=
      INTERVAL '15 years'
      THEN a.id_bioguide END) AS rep_and_sen_15_yrs
  FROM a
  JOIN legislators_terms c ON a.id_bioguide = c.id_bioguide
    AND c.term_type = 'sen' AND c.term_start > a.first_term
  GROUP BY 1)

SELECT aa.cohort::int AS cohort,
  round(bb.rep_and_sen_5_yrs * 1.0 / aa.reps, 4) AS pct_5_yrs,
  round(bb.rep_and_sen_10_yrs * 1.0 / aa.reps, 4) AS pct_10_yrs,
  round(bb.rep_and_sen_15_yrs * 1.0 / aa.reps, 4) as pct_15_yrs
FROM aa
LEFT JOIN bb ON aa.cohort = bb.cohort
ORDER BY 1;
```

Listing 4 SQL with bbb

```
WITH

a AS (
  SELECT id_bioguide, min(term_start) as first_term
  FROM legislators_terms
  WHERE term_type = 'rep'
  GROUP BY 1),

aa AS (
  SELECT date_part('century',a.first_term) AS cohort,
      count(id_bioguide) AS reps
  FROM a
  WHERE first_term <= '2009-12-31'
    GROUP BY 1),

bb AS (
  SELECT date_part('century', a.first_term) AS cohort,
    count(DISTINCT CASE WHEN age(c.term_start, a.first_term) <=
      INTERVAL '5 years'
      THEN a.id_bioguide END) AS rep_and_sen_5_yrs,
    count(DISTINCT CASE WHEN age(c.term_start, a.first_term) <=
      INTERVAL '10 years'
      THEN a.id_bioguide END) AS rep_and_sen_10_yrs,
    count(DISTINCT CASE WHEN age(c.term_start, a.first_term) <=
      INTERVAL '15 years'
      THEN a.id_bioguide END) AS rep_and_sen_15_yrs
  FROM a
  JOIN legislators_terms c ON a.id_bioguide = c.id_bioguide
    AND c.term_type = 'sen' AND c.term_start > a.first_term
  GROUP BY 1)

SELECT aa.cohort::int AS cohort,
  round(bb.rep_and_sen_5_yrs * 1.0 / aa.reps, 4) AS pct_5_yrs,
  round(bb.rep_and_sen_10_yrs * 1.0 / aa.reps, 4) AS pct_10_yrs,
  round(bb.rep_and_sen_15_yrs * 1.0 / aa.reps, 4) as pct_15_yrs
FROM aa
LEFT JOIN bb ON aa.cohort = bb.cohort
ORDER BY 1;
```

Listing 5 SQL with meaningful labels

```sql
WITH

cohorts AS (
  SELECT id_bioguide, min(term_start) AS first_term,
    date_part('century', min(term_start)) AS cohort,
  FROM legislators_terms
  WHERE term_type = 'rep'
  GROUP BY 1),

cohort_sizes AS (
  SELECT cohort, count(id_bioguide) AS reps
  FROM cohorts
  WHERE first_term <= '2009-12-31'
  GROUP BY 1),

ages AS (
  SELECT cohort, id_bioguide,
    age(term_start, first_term) AS age
  FROM cohorts
  JOIN legislators_terms
  USING (id_bioguide)
  WHERE term_type = 'sen' AND term_start > first_term),

age_cuts AS (
  SELECT cohort,
    count(DISTINCT CASE WHEN age <= INTERVAL '5 years'
      THEN id_bioguide END) AS rep_and_sen_5_yrs,
    count(DISTINCT CASE WHEN age <= INTERVAL '10 years'
      THEN id_bioguide END) AS rep_and_sen_10_yrs,
    count(DISTINCT CASE WHEN age <= INTERVAL '15 years'
      THEN id_bioguide END) AS rep_and_sen_15_yrs
  FROM ages
  GROUP BY 1)

SELECT cohort,
    round(rep_and_sen_5_yrs * 1.0 / reps, 4) AS pct_5_yrs,
    round(rep_and_sen_10_yrs * 1.0 / reps, 4) AS pct_10_yrs,
    round(rep_and_sen_15_yrs * 1.0 / reps, 4) AS pct_15_yrs
FROM cohort_sizes
LEFT JOIN age_cuts
USING (cohort)
ORDER BY 1;
```

Listing 6 SQL with main query in CTE

```
WITH

cohorts AS (
  SELECT id_bioguide, min(term_start) AS first_term,
    date_part('century', min(term_start)) AS cohort,
  FROM legislators_terms
  WHERE term_type = 'rep'
  GROUP BY 1),

cohort_sizes AS (
  SELECT cohort, count(id_bioguide) AS reps
  FROM cohorts
  WHERE first_term <= '2009-12-31'
  GROUP BY 1),

ages AS (
  SELECT cohort, id_bioguide, age(term_start, first_term) AS age
  FROM cohorts
  JOIN legislators_terms
  USING (id_bioguide)
  WHERE term_type = 'sen' AND term_start > first_term),

age_cuts AS (
  SELECT cohort,
    count(DISTINCT CASE WHEN age <= INTERVAL '5 years'
      THEN id_bioguide END) AS rep_and_sen_5_yrs,
    count(DISTINCT CASE WHEN age <= INTERVAL '10 years'
      THEN id_bioguide END) AS rep_and_sen_10_yrs,
    count(DISTINCT CASE WHEN age <= INTERVAL '15 years'
      THEN id_bioguide end) AS rep_and_sen_15_yrs
  FROM ages
  GROUP BY 1),

cohort_retention AS (
  SELECT cohort,
    round(rep_and_sen_5_yrs * 1.0 / reps, 4) AS pct_5_yrs,
    round(rep_and_sen_10_yrs * 1.0 / reps, 4) AS pct_10_yrs,
    round(rep_and_sen_15_yrs * 1.0 / reps, 4) AS pct_15_yrs
  FROM cohort_sizes
  LEFT JOIN age_cuts
  USING (cohort))

SELECT *
FROM cohort_retention
ORDER BY cohort;
```