Writing better SQL with `dbplyr`

Ian D. Gow

6 January 2025

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.

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. 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 <- db_get_csv(db, "legislators")
legislators_terms <- db_get_csv(db, "legislators_terms")
```

2 Converting the SQL to a CTE-based query

Table 1: Output from original SQL query (Listing 2)

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

Listing 2 shows the SQL query taken from Chapter 4 of Tanimura (2021, pp. 161–162).[1] Note that the output shown in Table 1 matches the output shown in Tanimura (2021, p. 162).

The query shown in Listing 2 is quite complex and I will show how it can be simplified using **common-table expressions** (**CTEs**). According to the documentation for PostgreSQL: CTEs "can be thought of as defining temporary tables that exist just for one query." CTEs are created using the SQL keyword `WITH`. In converting to a CTE-based query, I proceed in a series of small steps to better illustrate just how such a conversion can be done.

In Listing 3, I start by cleaning that up the two identical subqueries labelled `a` and `b`. I 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 refer to `a` and also in place of `b`. All references to `b` are changed to references to `a`. If you run the query in Listing 3, you will see it produces the same results as seen in Table 1.

In Listing 4, I convert `aa` and `bb` from subqueries to CTEs. 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, if you run the SQL, you will see that the results are unchanged.

In Listing 5, I switch to `USING` syntax for the join of `aa` and `bb`. My view is that `USING` produces more elegant SQL when it can be used and also facilitates the omission of the `aa` and `bb` prefixes for the variables in the final query. I also notice that `age(c.term_start, a.first_term)` appears three

1. I edited the query slightly to reflect code style guidelines I use in later queries. I also rename the column `cohort_century` to `cohort` throughout.

times in the query in Listing 4. By splitting `bb` into two, as in Listing 5 (now `bbb` is followed by `bb`), I can write a query in which `age(c.term_start, a.first_term)` appears just once.

In Listing 6, I replace the meaningless labels for the CTEs (e.g., `a` and `bb`) with more meaningful labels (e.g., `cohorts` and `age_cuts`). We also clean up `ages` a little (e.g., `USING`) and move `cohort` to `cohorts`. If you run the SQL, you will see that the results are unchanged.

In Listing 7, I also put the "main" query in a CTE. Te value of doing so it that it means we can easily edit the query to debug the CTEs that are used. For example, we could put `SELECT * FROM cohorts` at the end of the query in Listing 7 to look into `cohorts` if we are concerned about the output we are seeing from `cohort_retention`.

Table 2: Output from final translated SQL query (Listing 7)

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

3 Translating from SQL to dbplyr

Now that I have an SQL query based on CTEs, it is *much* easier to translate from SQL to `dbplyr`. In effect, I can translate each CTE in turn. I start with `cohort`. Note that I could have created `cohort` as part of the `summarize()` step using `century(min(term_start, na.rm = TRUE))`, but my view is that the code is easier to read if `min(term_start, na.rm = TRUE)` appears just once and `cohort` is created as a function of `first_term` in a separate `mutate()` step.

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

Translating `cohort_sizes` is straightforward. Note that `.groups = "drop"` is strictly optional in this case. Nonetheless I generally include `.groups = "drop"` in all my queries to prevent issues that can arise due to unintended grouping.[2]

2. My view is that the choice made early in the development `dplyr` of a de facto default of `.groups = "drop_last"` is a rather unfortunate one.

```
cohort_sizes <-
  cohorts |>
  filter(first_rep_term <= '2009-12-31') |>
  group_by(cohort) |>
  summarize(reps = n(), .groups = "drop")
```

Translating ages is also straightforward. The main difference here is that `filter()` seems to go more naturally earlier in the query rather than at the end. It would make no difference if the order of the `mutate()` and `filter()` steps in the query below we flipped. However, we cannot put `filter()` at the end of the query (i.e., after `select()`) because the fields would no longer be available at that point.

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

Blah.[3]

```
age_cuts <-
  ages |>
  mutate(num_5_yrs = case_when(age <= years(5) ~ id_bioguide),
         num_10_yrs = case_when(age <= years(10) ~ id_bioguide),
         num_15_yrs = case_when(age <= years(15) ~ id_bioguide)) |>
  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()
```

3. Note that `if_else(age <= years(5), id_bioguide, NA)` is an alternative way to get the same result as `case_when(age <= years(5) ~ id_bioguide)` gives.

Table 3: Output from query translated to `dbplyr`

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 Building the query from scratch using `dbplyr`

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.

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'`.

```
first_terms <-
  legislators_terms |>
  group_by(id_bioguide) |>
  summarize(first_rep_term = min(case_when(term_type == 'rep' ~ term_start),
                                 na.rm = TRUE),
            first_sen_term = min(case_when(term_type == 'sen' ~ term_start),
                                 na.rm = TRUE),
            last_sen_term = max(case_when(term_type == 'sen' ~ term_start),
                                na.rm = TRUE),
            first_term = min(term_start, na.rm = TRUE))
```

```
population <-
  first_terms |>
  filter(!is.na(first_rep_term),
         first_term == first_rep_term,
         first_term <= "2009-12-31") |>
  select(id_bioguide, first_rep_term, first_sen_term) |>
  mutate(cohort = century(first_rep_term))
```

We can now calculate the sizes of the cohorts that we have formed.[4]

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

```
cohort_sizes <-
  population |>
  group_by(cohort) |>
  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()`.

It is easy to check that `id_bioguide` is a valid key for `population`. 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 in this query.

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 <-
  population |>
  mutate(gap = age(first_sen_term, first_rep_term)) |>
  group_by(cohort) |>
  window_order(gap) |>
  mutate(cum_ids = cumsum(1)) |>
  group_by(cohort, 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 = "cohort") |>
  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 send that to DuckDB and turn the rows into intervals using `years()`.

```
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(cohort, 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",
              names_sort = TRUE) |>
  arrange(cohort) |>
  collect()
```

Table 4: Output of from-scratch `dbplyr` query

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

Because of the form of our second analysis, it is easy to make the plot seen in Figure 1.

```
pct_rep_then_sen |>
  mutate(gap = (year(gap) * 12 + month(gap)) / 12) |>
  filter(gap <= 15) |>
  group_by(cohort, gap) |>
  summarize(pct = max(pct, na.rm = TRUE),
            .groups = "drop") |>
  # collect() |>
  mutate(cohort = as.character(cohort)) |>
```

```
ggplot(aes(x = gap, y = pct, group = cohort,
           colour = cohort)) +
geom_line()
```

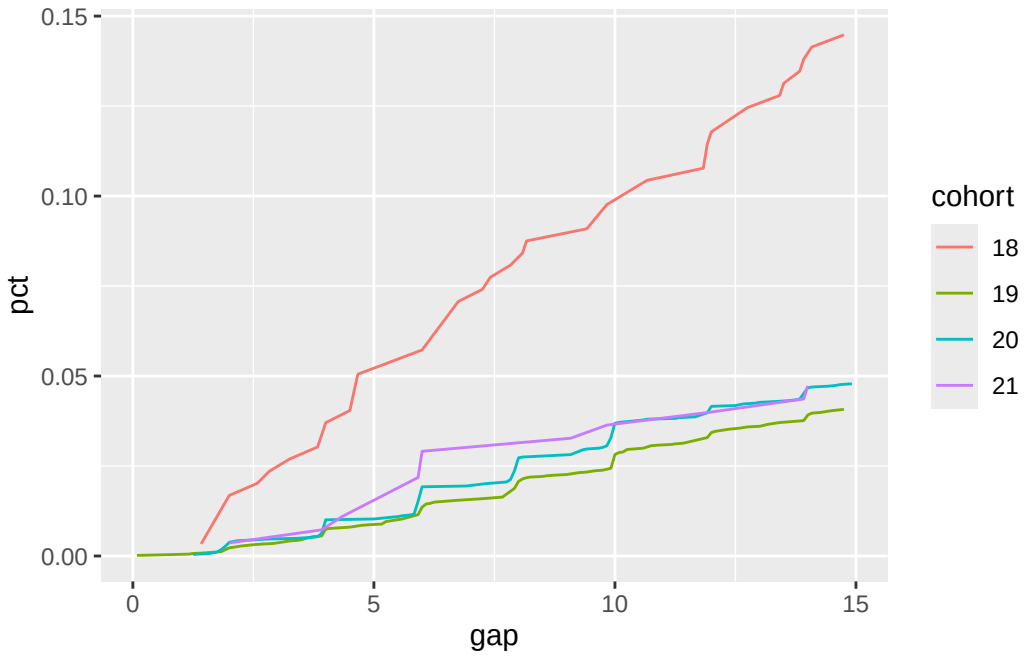

Figure 1: Representatives becoming senators over time by century

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.

5 Reconciling differences between queries

A careful reader might have noticed differences in the numbers in Table 8 and Table 4 and wonder why these differences exist.

I would argue that the answer I produced better matches the original question from Tanimura (2021, p. 158): "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."[5]

It turns out that some representatives were incorrectly included in the original analysis, but correctly excluded in producing Table 4. I compile the "problem cases" in a single data frame `problem_cases`.

5. I made minor punctuation edits here.

Table 5: Problematic cases: Senators first, later senators again

id_bioguide	first_sen_term	first_rep_term	last_sen_term
C000482	1806-01-01	1811-11-04	1849-12-03
M000519	1826-01-01	1833-12-02	1837-09-04
J000137	1818-01-01	1833-12-02	1844-01-01

```
problem_cases <-
  first_terms |>
  semi_join(cohorts, by = "id_bioguide") |>
  anti_join(population, by = "id_bioguide") |>
  filter(!is.na(first_sen_term) & !is.na(first_rep_term)) |>
  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')
```

The first set of problem cases comprises those legislators who were senators *before* they were representatives. These legislators should have been excluded on the based that they do not meet the "start as representatives" criterion. Note that this set can be view as comprising two subsets. The first subset are those legislators who started as senators, then became representatives, then became senators again. Cases from the first subset are shown in Table 5.

The second subset are those legislators who started as senators, then became representatives, but did *not* became senators again after serving as a representative. Cases from the second subset are shown in Table 6.

The second set of problem cases are representatives whose terms began after `'2009-12-31``. The original SQL query imposed this requirement in calculatingcohort_sizes, but not in constructing the cohorts themselves. Thus, while the SQL in @lst-original does exclude these cases from the denominator, these legislators are included in the numerator if they go on to serve as senators. These cases are shown in @tbl-problems-too-late. In contrast I was consistent in application of thefirst_rep_term <= '2009-12-31'' criterion throughout. The members of this set of problem cases are shown in Table 7.

It is easy to show that these issues explain the differences between Table 8 and Table 4. To do need to recapitulate some steps our analysis above, starting with a revised version of `ages` in which we `anti_join()` the `problem_cases` data frame.

```
ages <-
  cohorts |>
  anti_join(problem_cases, by = "id_bioguide") |>
  inner_join(legislators_terms, by = "id_bioguide") |>
```

Table 6: Problematic cases: Senators first, not later senators again

id_bioguide	first_sen_term	last_sen_term	first_rep_term
P000406	1865-12-04	1865-12-04	1867-03-04
E000028	1875-12-06	1875-12-06	1883-12-03
A000026	1805-12-02	1805-12-02	1831-12-05
S000805	1790-01-01	1790-01-01	1801-12-07
M000814	1945-01-10	1945-01-10	1949-01-03
R000125	1806-11-25	1807-10-26	1817-12-01
P000218	1936-01-01	1945-01-03	1963-01-09
S000941	1796-01-01	1796-01-01	1813-05-24
B000398	1821-12-03	1845-12-01	1853-12-05
C000703	1842-01-01	1842-01-01	1849-12-03
P000557	1880-01-01	1880-01-01	1883-12-03
N000160	1871-03-04	1871-03-04	1885-12-07
D000069	1798-12-05	1798-12-05	1799-12-02
L000240	1953-07-10	1953-07-10	1957-01-03
P000324	1803-10-17	1805-12-02	1813-05-24
N000050	1861-07-04	1861-07-04	1873-12-01
B000763	1823-12-01	1823-12-01	1831-12-05
A000041	1803-10-17	1803-10-17	1831-12-05
P000431	1807-10-26	1807-10-26	1837-09-04
P000354	1798-12-06	1799-12-02	1819-12-06
W000768	1801-12-07	1801-12-07	1809-05-22
K000069	1868-01-01	1877-10-15	1883-12-03
W000658	1949-01-20	1949-01-20	1952-08-02
N000086	1799-12-02	1799-12-02	1807-10-26
W000476	1859-12-05	1859-12-05	1869-03-04
R000063	1851-02-01	1851-02-01	1851-03-04
C000912	1817-12-01	1855-12-03	1861-07-04
C000325	1813-05-24	1825-12-05	1853-12-05
M000221	1800-01-01	1800-01-01	1817-12-01
J000161	1923-12-03	1923-12-03	1933-03-09
M000993	1930-12-13	1930-12-13	1943-01-06
W000012	1915-12-06	1921-04-11	1933-03-09
M000596	1851-12-01	1851-12-01	1857-12-07
B001019	1863-12-07	1863-12-07	1887-12-05
W000633	1789-03-04	1789-03-04	1793-12-02
B001061	1940-11-27	1941-01-03	1945-01-03

Table 7: Problematic cases: Too late

id_bioguide	first_rep_term	first_sen_term	last_sen_term
S001191	2013-01-03	2019-01-03	2019-01-03
Y000064	2011-01-05	2017-01-03	2017-01-03
C001096	2013-01-03	2019-01-03	2019-01-03
M001197	2015-01-06	2019-01-03	2019-01-03
L000575	2011-01-05	2015-01-06	2017-01-03
G000562	2011-01-05	2015-01-06	2015-01-06
D000622	2013-01-03	2017-01-03	2017-01-03
R000608	2017-01-03	2019-01-03	2019-01-03
S001184	2011-01-05	2013-01-03	2017-01-03
D000618	2013-01-03	2015-01-06	2015-01-06
C001095	2013-01-03	2015-01-06	2015-01-06

```
  filter(term_type == 'sen', term_start > first_rep_term) |>
  mutate(age = age(term_start, first_rep_term)) |>
  select(cohort, id_bioguide, age)
```

We then pass this revised `ages` to create a new version of `age_cuts`.

```
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)))
```

Finally we `left_join()` this revised `age_cuts` to produce Table 8. We now see that the output lines up with that in Table 4, confirming that the observations in `problem_cases` are the source of the differences between Table 4 and Table 3.

```
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()
```

Table 8: Output from query translated to `dbplyr` omitting problem cases

cohort	pct_5_yrs	pct_10_yrs	pct_15_yrs
18	0.0505	0.0976	0.1448
19	0.0087	0.0244	0.0407
20	0.0101	0.0349	0.0478
21	0.0109	0.0364	0.0473

Listing 1 Utility functions

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

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

  local_filename <- file.path(data_dir, paste0(filename, ".csv"))
  if (!file.exists(local_filename)) {
    download.file(url = paste0(url, filename), destfile = local_filename)
  }
}

db_read_csv <- function(db, table, data_dir = "data") {
  csv_file <- file.path(data_dir, paste0(table, ".csv"))
  tbl(db, paste0("read_csv_auto('", csv_file, "')")) |>
    compute(name = table)
}

db_get_csv <- function(db, table, data_dir = "data") {
  download_data(table, data_dir = data_dir)
  db_read_csv(db, table, data_dir = data_dir)
}
```

References

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

Listing 2 Original SQL code

```
SELECT aa.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
  (
    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,
    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 = bb.cohort
ORDER BY 1;
```

Listing 3 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,
  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 4 SQL with CTEs for `aa` and `bb`

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

bbb AS (
    SELECT date_part('century', a.first_term) AS cohort,
      a.id_bioguide,
      age(c.term_start, a.first_term) AS age
    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),

bb 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 bbb
    GROUP BY 1)

SELECT cohort as 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 aa
LEFT JOIN bb
USING (cohort)
ORDER BY 1;
```

Listing 6 SQL with meaningful labels

```
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 7 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;
```