Cookbook Recipes

- Baby Names - SSA database

Baby Names - SSA database

Social Security Administration's baby names database

In this cookbook recipe, you will:

1. Download the full `babynames_yob` data from the Social Security Administration website.
2. ingest all the CSV files at once using DuckDB.
3. Create and populate rank and popularity for each year/sex combination.
4. See how to export the data to popular file formats CSV and parquet
5. See a few analyses queries
6. See the data in action in a R Shiny app and a basic html page using an API

0. Prerequisites - Install DuckDB

If you don't already have DuckDB installed, see the How to install DuckDB page.

1. Download the Data Set

> Download the names.zip file from catalog.data.gov website

I saved mine into `C:\DuckDB\babynames\`. Just change line 12 below if you save it elsewhere.

2. Ingest all the files into a single table at once using DuckDB

Since none of the text files have headers, we are explicitly creating the headers. The reserved word `filename` extracts the file name as a column with that same name. In our case we only want the 4-digit year part of the name, so we use `regexp_extract` to get just that part. I am using abbreviated names `cnt` for count and later `rnk` for rank to avoid the use of keywords in the column names.

```
-- DROP TABLE IF EXISTS babynames;


-- Ingest SSA baby names, compute rank & popularity index
```

```
CREATE TABLE babynames AS
SELECT
    name
  , sex
  , cnt
  , regexp_extract(filename, 'yob([0-9]{4})', 1)::INT AS year

FROM read_csv(
    'C:/DuckDB/babynames/yob*.txt'
  , columns = { 'name': 'VARCHAR'
              , 'sex': 'VARCHAR'
              , 'cnt': 'INT'
              }
  , header = FALSE
  , delim = ','
  , auto_detect = FALSE
  , filename = TRUE
)
;
```

3. Create and populate rank and popularity for each year/sex combination.

```
--
================================================================================
-- Add rnk and popularity_index
ALTER TABLE main.babynames ADD COLUMN rnk INT NULL;
ALTER TABLE main.babynames ADD COLUMN popularity_index DOUBLE NULL;



--
================================================================================
-- populate rank
UPDATE babynames AS b
SET rnk = x.rnk
FROM (
    SELECT name, sex, year,
```

```
        ROW_NUMBER() OVER (PARTITION BY year, sex ORDER BY cnt DESC) AS rnk
    FROM babynames
) AS x
WHERE b.name = x.name
  AND b.sex = x.sex
  AND b.year = x.year;


--
============================================================================
=============
-- populate popularity_index
UPDATE babynames AS b
SET popularity_index = sub.popularity_index
FROM (
    SELECT name, sex, year,
        100.0 * cnt / MAX(cnt) OVER (PARTITION BY sex, year) AS popularity_index
    FROM babynames
) AS sub
WHERE b.name = sub.name
  AND b.sex  = sub.sex
  AND b.year = sub.year;
```

4. Export the data to CSV and parquet

CSV files give you maximum portability and can work with practically everything - although at a cost of file size

Parquet files give you great compression and columnar internal storage which is optimized for analytical applications

```
-- Export to csv
COPY babynames TO 'C:/temp/babynames.csv' (FORMAT CSV, HEADER TRUE, DELIMITER ',', QUOTE '"');


-- Export to parquet format
COPY babynames TO 'C:/temp/babynames.parquet' (FORMAT PARQUET, COMPRESSION 'ZSTD');
```

5. Look at Some Analysis Queries

Query 1: Top ranked names for both male and female over the years

```
--
================================================================================
-- #1 M & F names over the years
SELECT * FROM babynames WHERE sex = 'F' AND rnk = 1 ORDER BY YEAR DESC;
```

```
SELECT * FROM babynames WHERE sex = 'M' AND rnk = 1 ORDER BY YEAR DESC;
```

Query 2: Number of unique names per year/sex

One row per year with an 'M' and 'F' column showing how many unique names there were for that year and sex

```
--
================================================================================
-- Unique name counts by year/sex
SELECT *
FROM (SELECT YEAR, sex, COUNT(*) AS N FROM babynames GROUP BY ALL)
PIVOT (
   SUM(N) FOR sex IN ('M', 'F')
)
ORDER BY year;
```

Query 3: Top 5 M & F names - pivoted to 1 row / year

```
-- top 5 names of each sex each year
SELECT
     year

   -- Female Top 5 (names)
   --, MAX(name) FILTER (WHERE sex = 'F' AND rnk = 1) AS F_1    -- FILTER supported by DuckDB
   , MAX(CASE WHEN sex = 'F' AND rnk = 1 THEN name END) AS F_1  -- CASE is more portable
   , MAX(CASE WHEN sex = 'F' AND rnk = 2 THEN name END) AS F_2
   , MAX(CASE WHEN sex = 'F' AND rnk = 3 THEN name END) AS F_3
```

```
    , MAX(CASE WHEN sex = 'F' AND rnk = 4 THEN name END) AS F_4
    , MAX(CASE WHEN sex = 'F' AND rnk = 5 THEN name END) AS F_5

    -- Male Top 5 (names)
    , MAX(CASE WHEN sex = 'M' AND rnk = 1 THEN name END) AS M_1
    , MAX(CASE WHEN sex = 'M' AND rnk = 2 THEN name END) AS M_2
    , MAX(CASE WHEN sex = 'M' AND rnk = 3 THEN name END) AS M_3
    , MAX(CASE WHEN sex = 'M' AND rnk = 4 THEN name END) AS M_4
    , MAX(CASE WHEN sex = 'M' AND rnk = 5 THEN name END) AS M_5

    -- Female Top 5 (counts)
    , SUM(CASE WHEN sex = 'F' AND rnk = 1 THEN cnt ELSE 0 END) AS F1n
    , SUM(CASE WHEN sex = 'F' AND rnk = 2 THEN cnt ELSE 0 END) AS F2n
    , SUM(CASE WHEN sex = 'F' AND rnk = 3 THEN cnt ELSE 0 END) AS F3n
    , SUM(CASE WHEN sex = 'F' AND rnk = 4 THEN cnt ELSE 0 END) AS F4n
    , SUM(CASE WHEN sex = 'F' AND rnk = 5 THEN cnt ELSE 0 END) AS F5n

    -- Male Top 5 (counts)
    , SUM(CASE WHEN sex = 'M' AND rnk = 1 THEN cnt ELSE 0 END) AS M1n
    , SUM(CASE WHEN sex = 'M' AND rnk = 2 THEN cnt ELSE 0 END) AS M2n
    , SUM(CASE WHEN sex = 'M' AND rnk = 3 THEN cnt ELSE 0 END) AS M3n
    , SUM(CASE WHEN sex = 'M' AND rnk = 4 THEN cnt ELSE 0 END) AS M4n
    , SUM(CASE WHEN sex = 'M' AND rnk = 5 THEN cnt ELSE 0 END) AS M5n

FROM babynames
WHERE rnk <= 5
  AND sex IN ('F', 'M')
GROUP BY year
ORDER BY year DESC;
```

Query 4: Top 10 # 1 Streaks for both Male and Female

This multi-level CTE query tackles the classic "gaps and islands" problem. When a name is #1 it is often #1 for multiple years. This identifies how many years a name is at #1 before dropping rank.

```
--
============================================================
=============
-- Top 10 #1 streaks for M and F
```

```
WITH ranked AS (
    SELECT name,sex,year,rnk
    FROM babynames
    WHERE rnk = 1
),

islands AS (
    SELECT name,sex,year,
        /* grouping key for consecutive years */
        year - ROW_NUMBER() OVER (
            PARTITION BY name, sex
            ORDER BY year
        ) AS grp
    FROM ranked
),

streaks AS (
    SELECT name, sex,
        MIN(year) AS start_year,
        MAX(year) AS end_year,
        COUNT(*) AS streak_length
    FROM islands
    GROUP BY name, sex, grp
),

top20_f AS (
    SELECT * FROM streaks WHERE sex = 'F' ORDER BY streak_length DESC LIMIT 20
),
top20_m AS (
    SELECT * FROM streaks WHERE sex = 'M' ORDER BY streak_length DESC LIMIT 20
)

SELECT * FROM top20_f
UNION ALL
SELECT * FROM top20_m
ORDER BY sex, streak_length DESC;
```

Query 5: Androgynous Names

Names that are (nearly) equal for both sexes.

```
-- Androgynous names
-- Base aggregation (name-level totals)
WITH totals AS (
    SELECT
          name
        , SUM(cnt)                                  AS total_cnt
        , SUM(CASE WHEN sex = 'M' THEN cnt ELSE 0 END) AS male_cnt
        , SUM(CASE WHEN sex = 'F' THEN cnt ELSE 0 END) AS female_cnt
    FROM babynames
    GROUP BY name
)
SELECT
      name
    , total_cnt
    , male_cnt
    , female_cnt
    , ROUND(male_cnt * 1.0 / total_cnt, 4)       AS male_share      -- notice reuse of male_share
    , ROUND(ABS(male_share - 0.5), 4)            AS androgyny_score -- Lower androgyny_score = more
androgynous
    , ROUND( 1 - (2 * ABS(male_share - 0.5)), 4) AS androgyny_index -- 1 = 100% androgynous
    , CONCAT('https://shiny.briandill.net/Babynames?names=', name, ',', name, ',%20&sexes=F,M,M') AS
shiny_url
FROM totals
WHERE male_cnt > 0
  AND female_cnt > 0
  AND total_cnt >= 15000      -- volume threshold
ORDER BY androgyny_index DESC, total_cnt DESC
LIMIT 50;
```

Query 6: Top Birth Count

List top birth count by name/sex. Also return the year, rank, popularity index, etc. Using three different methods

1. CTE
2. QUALIFY
3. MAX_BY

```
-- CTE ============================================================
WITH foo AS (
    SELECT *, RANK() OVER (PARTITION BY name, sex ORDER BY cnt DESC, year) AS rn
```

```
    FROM main.babynames
)
SELECT name
    , sex
    , YEAR
    , rnk
    , popularity_index
 , YEAR(CURRENT_DATE()) - YEAR AS years_ago
    , cnt AS largest_count
FROM foo
WHERE rn = 1
ORDER BY largest_count DESC
LIMIT 10;

-- QUALIFY ============================================================
SELECT name
    , sex
    , YEAR
    , YEAR(CURRENT_DATE()) - YEAR AS years_ago
    , rnk
    , popularity_index
    , cnt AS largest_count
FROM main.babynames
QUALIFY RANK() OVER (PARTITION BY name, sex ORDER BY cnt DESC, year) = 1
ORDER BY largest_count DESC
LIMIT 10;

-- MAX_BY() ===========================================================
SELECT name
    , sex
    , MAX_BY(year, cnt) AS year_of_max
    , MAX_BY(rnk, cnt) AS rnk
    , MAX_BY(popularity_index, cnt) AS popularity_index
    , YEAR(CURRENT_DATE()) - MAX_BY(year, cnt) AS years_ago
    , MAX(cnt) AS largest_count
FROM main.babynames
GROUP BY ALL
ORDER BY largest_count DESC
LIMIT 10;
```

Query 7 - Popular names that never reached #1

```
-- popular names that never reached #1
SELECT name
, sex
, SUM(cnt) AS tot_cnt
, MIN(rnk) AS min_rnk
, COUNT(*) FILTER (WHERE rnk = 1) AS yrs_at_1
, COUNT(*) FILTER (WHERE rnk = 2) AS yrs_at_2
, COUNT(*) FILTER (WHERE rnk = 3) AS yrs_at_3
, COUNT(*) FILTER (WHERE rnk = 4) AS yrs_at_4
, COUNT(*) FILTER (WHERE rnk = 5) AS yrs_at_5
, COUNT(*) AS years_active
FROM main.babynames
GROUP BY name, sex
HAVING MIN(rnk) >= 2 -- never reached #1
AND SUM(cnt) > 10000 -- have at least 10000 people with that name/sex
ORDER BY SUM(cnt) DESC
```

6. See the data in action

See the baby names data set in action

- in a R Shiny app
- in basic html page using an API