Tuesday 17 October 2013

General Assembly

# Practical SQL

# Referring to SQL results in SQL queries -- Subqueries

# Subqueries embed one query within another

## These embedded queries are 'subqueries'

We have to watch the table returned by the subquery carefully

The returned table has to have the fields and row expected of it by the calling query

Two varieties, 'correlated' and 'uncorrelated'

'correlated' is cooler but harder, we'll do 'uncorrelated' first

# Where we left off:

## Say we want a list of those presidents from the three states that have sent the most presidents

We can get the list by a query:

```
SELECT
    state,
    COUNT(*) AS state_count
FROM
    sampdb.president
GROUP BY
    state
ORDER BY
    state_count DESC
LIMIT 3
;
```

```
+-------+-------------+
| state | state_count |
+-------+-------------+
| VA    |           8 |
| OH    |           7 |
| MA    |           4 |
+-------+-------------+
```

# First, an easier example

## Say we want a list of students scoring above on grade event 3

### Getting the score is easy:

```
SELECT
    event_id,
    AVG(score)
FROM
    sampdb.score
WHERE
        event_id = 3
GROUP BY
    event_id
LIMIT 5
;
```

### Results

```
+---------+-------+-------+
| student | event | score |
+---------+-------+-------+
|       1 |     3 |    88 |
|       2 |     3 |    84 |
|       5 |     3 |    97 |
|       6 |     3 |    83 |
|       7 |     3 |    88 |
+---------+-------+-------+
```

### Filtering is easy

```
SELECT
    student_id AS student,
    event_id AS event,
    score
FROM
    sampdb.score
WHERE
        event_id = 3
    AND score > 78.2258
;
```

### But:

What if the data change?

We want results for another event?

# Better: refer directly to the average

```
SELECT
    student_id,
    event_id,
    score
FROM
    sampdb.score
WHERE
        event_id = 3
    AND score > (
        SELECT
            AVG(score)
        FROM
            sampdb.score
        WHERE
            event_id = 3
        GROUP BY
            event_id
    )
LIMIT 5;

```

```
+------------+-------+
| student_id | score |
+------------+-------+
|          1 |    88 |
|          2 |    84 |
|          5 |    97 |
|          6 |    83 |
|          7 |    88 |
+------------+-------+

```

**Now:**

- syntax is more complex
- but query automatically changes output for any data change
- we can get a similar list simply by changing the event_id we are seeking
- BUT -- what if we change event_id in one place, and not the other . . . ?

# Better still: Remove duplication

```sql
SELECT
  student_id,
  score
FROM
  sampdb.score AS scr
WHERE
  event_id = 3
  AND score > (
    SELECT
      AVG(score)
    FROM
      sampdb.score
    WHERE
      event_id=scr.event_id
    GROUP BY
      event_id
  )
LIMIT 5;

```

```
+------------+-------+
| student_id | score |
+------------+-------+
|          1 |    88 |
|          2 |    84 |
|          5 |    97 |
|          6 |    83 |
|          7 |    88 |
+------------+-------+

```

**Now:**

- The subquery's event_id filter looks to each particular record for the value of event_id used in its filter
- Thus our query only refers to the desired event_id once
- e.g., we cannot make the 'event_id values out of step' mistake
- This is a 'correlated subquery', and we will spend more time on them later

# Back to the presidents problem

## Revise our condition for state inclusion

This will automatically update values for changes in the data, or application to another data set

It is also easier to change the logic -- if we want the top 5 states, we change one value and the implications flow

```
SELECT
  MIN(st_count)
FROM
(
  SELECT
    state,
    COUNT(*) AS st_count
  FROM
    sampdb.president
  GROUP BY
    state
  ORDER BY
    state_count DESC
  LIMIT 3
) AS tmp
;
```

```
+-----------------+
| MIN(st_count) |
+-----------------+
|               4 |
+-----------------+
```

# Revise listing query

## Uses presidents count criteria for state inclusion

```
SELECT
  state, COUNT(*) as state_count
FROM
  sampdb.president
GROUP BY
  state
HAVING
  COUNT(*) >=
  (
    SELECT
      MIN(state_count)
    FROM
      (
        SELECT
          state,
          COUNT(*) AS state_count
        FROM
          sampdb.president
        GROUP BY
          state
        ORDER BY
          state_count DESC
        LIMIT 3
      ) AS t3
  )
ORDER BY state_count DESC
;

```

```
+-------+-------------+
| state | state_count |
+-------+-------------+
| VA    |           8 |
| OH    |           7 |
| MA    |           4 |
| NY    |           4 |
+-------+-------------+

```

- This gives us a 'dynamic' derivation of the state list we need
- HAVING
- - similar to WHERE
- - appears after GROUP BY
- - refers to, and filters on, the grouped records and aggregated fields
- So WHERE filters the records that are aggregated into groups, and HAVING filters the groups themselves

# Putting all this together provides our answer

```
SELECT last_name, state
FROM sampdb.president
WHERE
  state IN
  (
    SELECT state
    FROM sampdb.president
    GROUP BY state
    HAVING
      COUNT(*) >=
      (
        SELECT
          MIN(state_count)
        FROM
          (
            SELECT COUNT(*) AS state_count
            FROM sampdb.president
            GROUP BY state
            ORDER BY state_count DESC
            LIMIT 3
          ) AS t3
      )
  )
ORDER BY
  state, last_name, first_name
;

```

```
+------------+-------+
| last_name  | state |
+------------+-------+
| Adams      | MA    |
| Adams      | MA    |
| Bush       | MA    |
| Kennedy    | MA    |
| Fillmore   | NY    |
| Roosevelt  | NY    |
| Roosevelt  | NY    |
| Van Buren  | NY    |
| Garfield   | OH    |
| Grant      | OH    |
| Harding    | OH    |
| Harrison   | OH    |
| Hayes      | OH    |
| McKinley   | OH    |
| Taft       | OH    |
| Harrison   | VA    |
| Jefferson  | VA    |
| Madison    | VA    |
| Monroe     | VA    |
| Taylor     | VA    |
| Tyler      | VA    |
| Washington | VA    |
| Wilson     | VA    |
+------------+-------+

```

- Inner query lists the top three states and their counts
- Next extracts the minimum value of those counts
- Next out lists the states with that count or higher
- Last query takes presidents whose states fall in that list

# Subqueries in SELECT field expressions

```
SELECT
( SELECT
    ROUND(AVG(score),1)
  FROM sampdb.score
  WHERE event_id = 1
  GROUP BY event_id
)   AS 1_scr,
( SELECT
    ROUND(AVG(score),1)
  FROM sampdb.score
  WHERE event_id = 2
  GROUP BY event_id
)   AS 2_scr,
( SELECT
    ROUND(AVG(score),1)
  FROM sampdb.score
  WHERE event_id = 3
  GROUP BY event_id
)   AS 3_scr

-- Notice the SELECT
-- expression is
-- just a field list

;



```

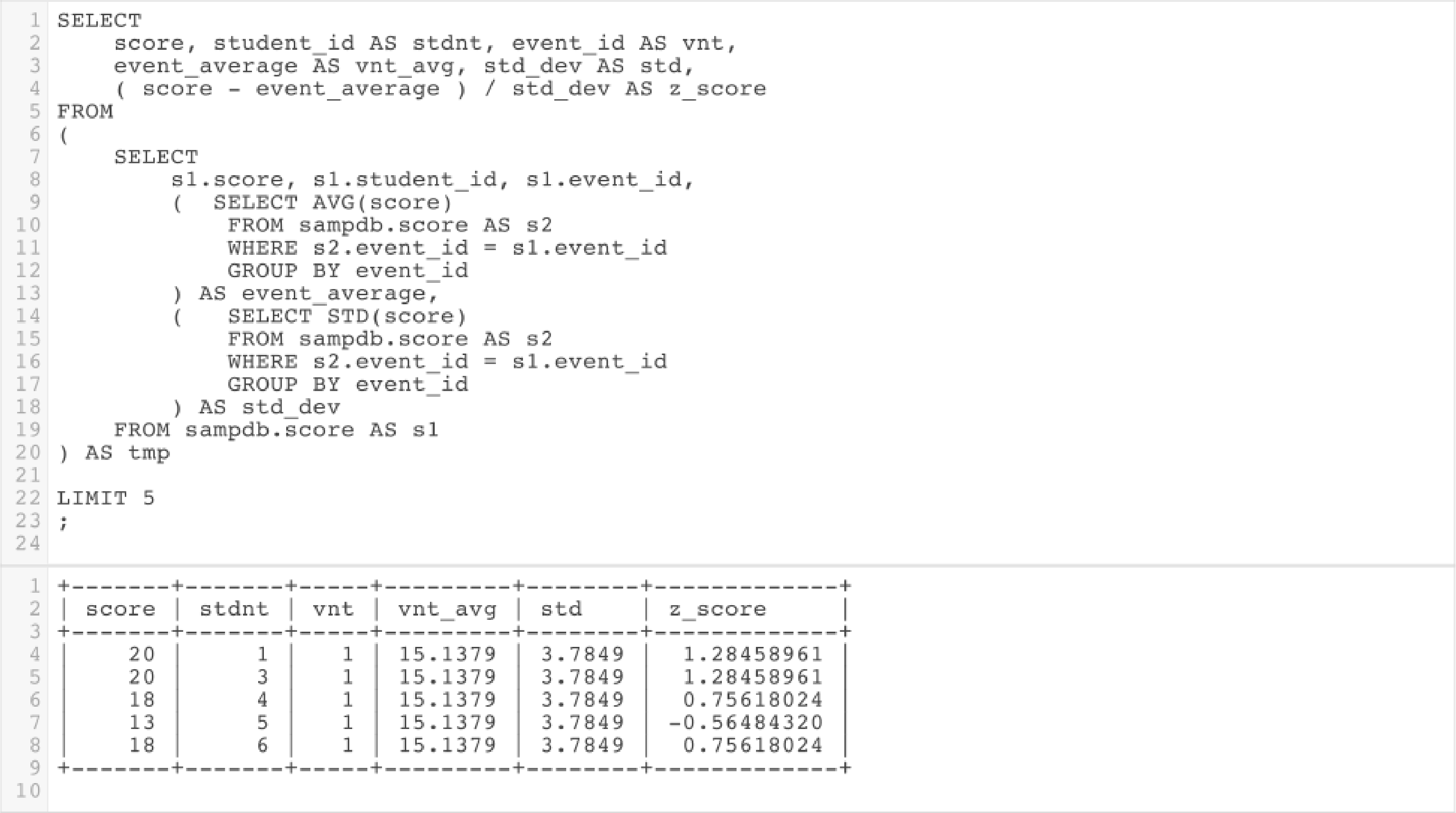
```
+-------+-------+-------+
| 1_scr | 2_scr | 3_scr |
+-------+-------+-------+
|  15.1 |  14.2 |  78.2 |
+-------+-------+-------+

```

### Subqueries can be used in field expressions

- Here we use them to replicate an Excel pivot table
- Notice that this doesn't handle "arbitrary" columns
- We have to designate a field in our output for each column
- The fundamental problem is we can specify sets of records, but not fields
- There are hacks around this, but they're complicated

# Field Subqueries used for calculations



```
SELECT
    score, student_id AS stdnt, event_id AS vnt,
    event_average AS vnt_avg, std_dev AS std,
    ( score - event_average ) / std_dev AS z_score
FROM
(
    SELECT
        s1.score, s1.student_id, s1.event_id,
        (  SELECT AVG(score)
            FROM sampdb.score AS s2
            WHERE s2.event_id = s1.event_id
            GROUP BY event_id
        ) AS event_average,
        (   SELECT STD(score)
            FROM sampdb.score AS s2
            WHERE s2.event_id = s1.event_id
            GROUP BY event_id
        ) AS std_dev
    FROM sampdb.score AS s1
) AS tmp

LIMIT 5
;

```

```
+-------+-------+-----+---------+--------+-------------+
| score | stdnt | vnt | vnt_avg | std    | z_score     |
+-------+-------+-----+---------+--------+-------------+
|    20 |     1 |   1 | 15.1379 | 3.7849 |  1.28458961 |
|    20 |     3 |   1 | 15.1379 | 3.7849 |  1.28458961 |
|    18 |     4 |   1 | 15.1379 | 3.7849 |  0.75618024 |
|    13 |     5 |   1 | 15.1379 | 3.7849 | -0.56484320 |
|    18 |     6 |   1 | 15.1379 | 3.7849 |  0.75618024 |
+-------+-------+-----+---------+--------+-------------+

```

# Correlated Subqueries work record by record

## The 'inner' queries are run for each record in the 'outer' query

### Definitions

- 'Outer' query is the query calling on result
- 'Inner' query supplies result
- Records in separate are referenced by aliases

## Outer query checks inner query for each of its own records

1. Outer gets record
2. Outer consults inner, 'correlated' query
3. Inner query takes values from record in process for outer query
4. Inner passes result out to outer query

## Are duplicative inner queries cached?

- It depends
- Performance is complex, requires first mastering syntax

# HAVING

## Seen earlier in subquery

### HAVING applies criteria to output of aggregations

- Calculated after WHERE and GROUP BY
- WHERE filters input records to initial query
- HAVING filters resultant aggregations

# HAVING Example

## States with more than one President whose last name doesn't begin with a vowel

```
SELECT
  state,
  COUNT(*) AS state_count
FROM
  sampdb.president
WHERE
  LEFT(last_name, 1)
  NOT IN
  ('A', 'E', 'I', '
    O', 'U')
GROUP BY
  state
HAVING
  state_count >= 2
;
```

```
+-------+-------------+
| state | state_count |
+-------+-------------+
| MA    |           2 |
| NC    |           2 |
| NY    |           4 |
| OH    |           7 |
| VA    |           8 |
+-------+-------------+
```

- WHERE clause strips out the vowel last names
- HAVING gets the states with more than one (now not-vowel) president
- Splitting up the WHERE and HAVING criteria gives a finer control over the output set

# Queries for Analysis

# Using queries for analysis

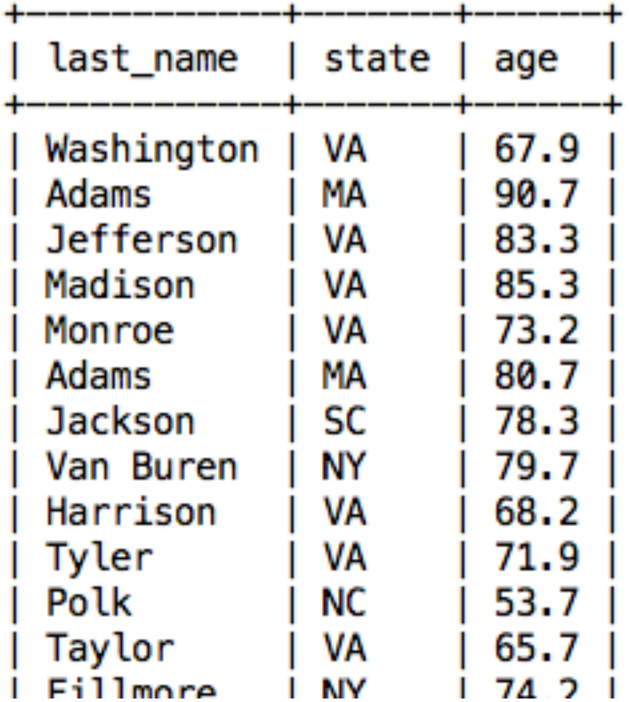
- Notice that all of our queries return (e.g., output) tables
  - The output is a set of records, with defined fields
- We can perform further operations on those returned tables, and join them to one another
- By cascading queries, we can filter and massage our data to get where we want
  - "views" give us a tool for holding queries "in place"

# Create a “base table”

```
DROP VIEW IF EXISTS sampdb.president_age;

CREATE VIEW
   sampdb.president_age
AS
   SELECT
      last_name,
      state,
      ROUND(DATEDIFF(death, birth) / 365, 1) as AGE
   FROM
      sampdb.president;
```

- “Stores” the result of the SELECT query in a “table” available to other queries
- Uses functions to create an << age >> column from birth and death



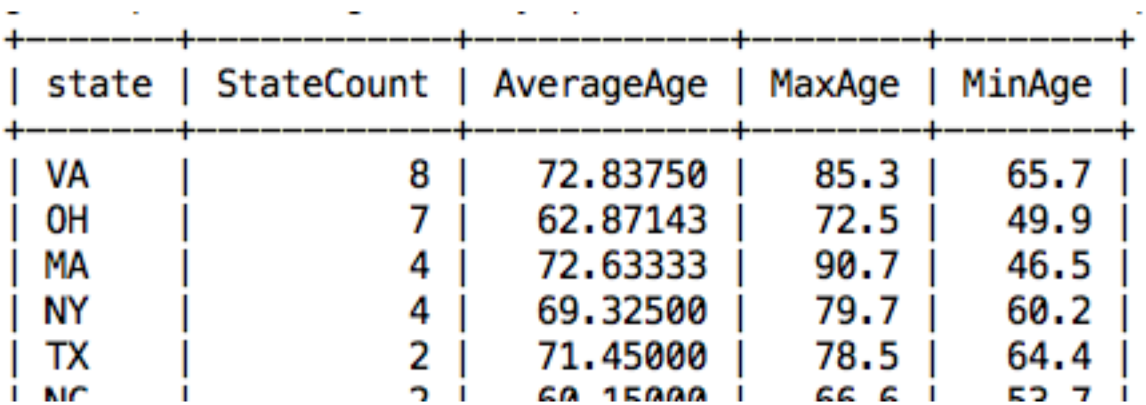
```
+------------+-------+------+
| last_name  | state | age  |
+------------+-------+------+
| Washington | VA    | 67.9 |
| Adams      | MA    | 90.7 |
| Jefferson  | VA    | 83.3 |
| Madison    | VA    | 85.3 |
| Monroe     | VA    | 73.2 |
| Adams      | MA    | 80.7 |
| Jackson    | SC    | 78.3 |
| Van Buren  | NY    | 79.7 |
| Harrison   | VA    | 68.2 |
| Tyler      | VA    | 71.9 |
| Polk       | NC    | 53.7 |
| Taylor     | VA    | 65.7 |
| Fillmore   | NY    | 74.2 |
```

# Derive a second table from our base

```
DROP VIEW IF EXISTS sampdb.president_aggregates;

CREATE VIEW
   sampdb.president_aggregates
AS
   SELECT
      state,
      COUNT(state) as StateCount,
      AVG(age) as AverageAge,
      MAX(age) as MaxAge,
      MIN(age) as MinAge
   FROM
      sampdb.president_age
   GROUP BY
      state
   ORDER BY
      StateCount DESC
;
```

- Uses our prior query to create a table of aggregated values

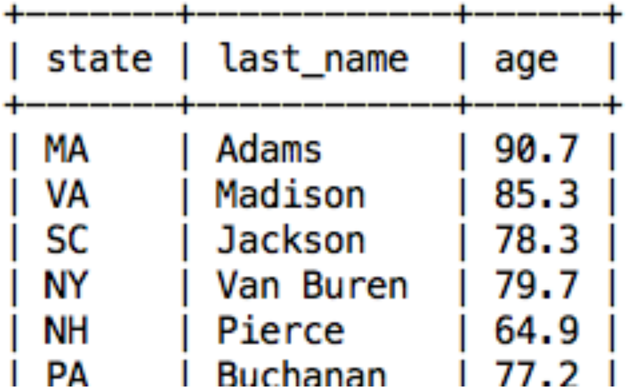


```
+-------+------------+------------+--------+--------+
| state | StateCount | AverageAge | MaxAge | MinAge |
+-------+------------+------------+--------+--------+
| VA    |          8 |   72.83750 |   85.3 |   65.7 |
| OH    |          7 |   62.87143 |   72.5 |   49.9 |
| MA    |          4 |   72.63333 |   90.7 |   46.5 |
| NY    |          4 |   69.32500 |   79.7 |   60.2 |
| TX    |          2 |   71.45000 |   78.5 |   64.4 |
| NC    |          2 |   60.15000 |   66.6 |   53.7 |
```

# Combine the two to get our result

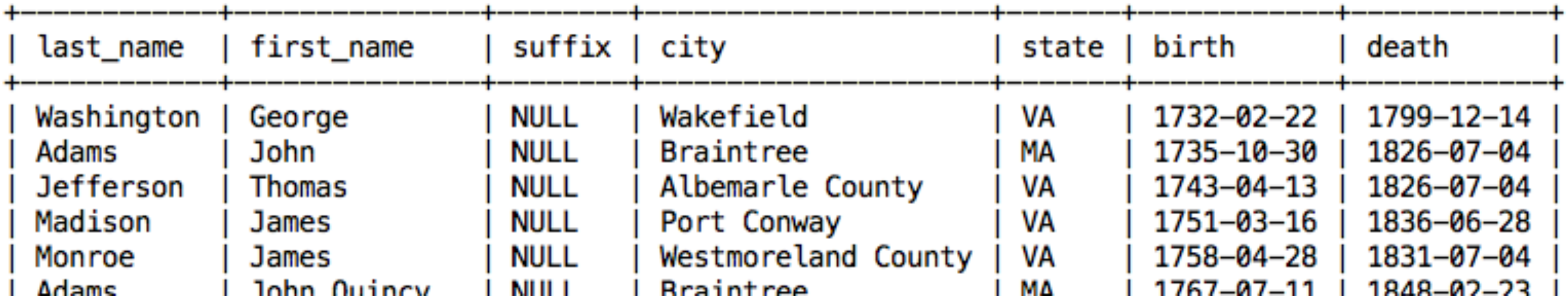
```
SELECT
   sub.state,
   sub.last_name,
   sub.age
FROM
   sampdb.president_age sub
INNER JOIN
   sampdb.president_aggregates sub2
ON
   sub.state = sub2.state
   AND sub.age = sub2.MaxAge
```

- Now we can join our Age table with our Aggregates table to get oldest presidents by State

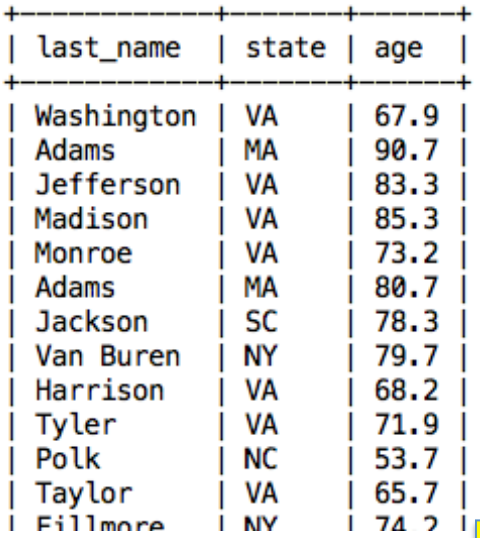


```
+-------+------------+------+
| state | last_name  | age  |
+-------+------------+------+
| MA    | Adams      | 90.7 |
| VA    | Madison    | 85.3 |
| SC    | Jackson    | 78.3 |
| NY    | Van Buren  | 79.7 |
| NH    | Pierce     | 64.9 |
| PA    | Buchanan   | 77.2 |
```

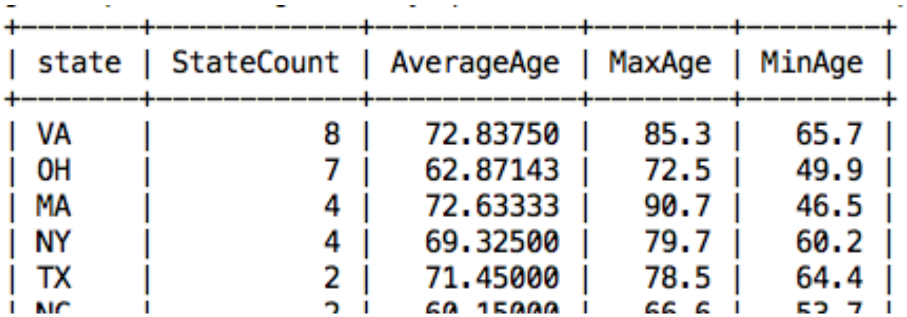
# Cascading through queries to result



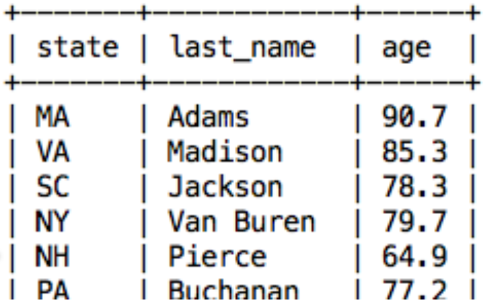
```
+------------+-------------+--------+---------------------+-------+------------+------------+
| last_name  | first_name  | suffix | city                | state | birth      | death      |
+------------+-------------+--------+---------------------+-------+------------+------------+
| Washington | George      | NULL   | Wakefield           | VA    | 1732-02-22 | 1799-12-14 |
| Adams      | John        | NULL   | Braintree           | MA    | 1735-10-30 | 1826-07-04 |
| Jefferson  | Thomas      | NULL   | Albemarle County    | VA    | 1743-04-13 | 1826-07-04 |
| Madison    | James       | NULL   | Port Conway         | VA    | 1751-03-16 | 1836-06-28 |
| Monroe     | James       | NULL   | Westmoreland County | VA    | 1758-04-28 | 1831-07-04 |
| Adams      | John Quincy | NULL   | Braintree           | MA    | 1767-07-11 | 1848-02-23 |
```



```
+------------+-------+------+
| last_name  | state | age  |
+------------+-------+------+
| Washington | VA    | 67.9 |
| Adams      | MA    | 90.7 |
| Jefferson  | VA    | 83.3 |
| Madison    | VA    | 85.3 |
| Monroe     | VA    | 73.2 |
| Adams      | MA    | 80.7 |
| Jackson    | SC    | 78.3 |
| Van Buren  | NY    | 79.7 |
| Harrison   | VA    | 68.2 |
| Tyler      | VA    | 71.9 |
| Polk       | NC    | 53.7 |
| Taylor     | VA    | 65.7 |
| Fillmore   | NY    | 74.2 |
```



```
+-------+------------+------------+--------+--------+
| state | StateCount | AverageAge | MaxAge | MinAge |
+-------+------------+------------+--------+--------+
| VA    |          8 |   72.83750 |   85.3 |   65.7 |
| OH    |          7 |   62.87143 |   72.5 |   49.9 |
| MA    |          4 |   72.63333 |   90.7 |   46.5 |
| NY    |          4 |   69.32500 |   79.7 |   60.2 |
| TX    |          2 |   71.45000 |   78.5 |   64.4 |
| NC    |          2 |   60.15000 |   66.6 |   53.7 |
```



```
+-------+------------+------+
| state | last_name  | age  |
+-------+------------+------+
| MA    | Adams      | 90.7 |
| VA    | Madison    | 85.3 |
| SC    | Jackson    | 78.3 |
| NY    | Van Buren  | 79.7 |
| NH    | Pierce     | 64.9 |
| PA    | Buchanan   | 77.2 |
```

# Can we do without the views?

```
SELECT
   state,
   last_name as oldest,
   ROUND(DATEDIFF(death, p.birth) / 365, 1) as age

FROM
   sampdb.president p

WHERE
  (    state,
      ROUND(DATEDIFF(death, birth) / 365, 1)
 ) IN
   (
      SELECT
         state,
         MAX(ROUND(DATEDIFF(death, birth) / 365, 1)) as age
      FROM
         sampdb.president p2
      GROUP BY
         state
   )
;
```

- Here we use a “subquery”
- We’re still combining tables with one another, but here we generate one of them on the fly