Tuesday 8 October 2013

General Assembly

Practical SQL

Operations

We can do simple arithmetic math

```
SELECT
    score / 100 as pct_score,
    CAST(score AS decimal(8, 2)) / 100 as pct_score_2
FROM
    sampdb.score
LIMIT 5;
```

Clear Run

```
+-----------+-------------+
| pct_score | pct_score_2 |
+-----------+-------------+
|    0.2000 |    0.200000 |
|    0.2000 |    0.200000 |
|    0.1800 |    0.180000 |
|    0.1300 |    0.130000 |
|    0.1800 |    0.180000 |
+-----------+-------------+
```

Clear Run

-- Note that MySQL will automatically convert the value returned as a decimal.

-- Other SQL versions will not. Postgres requires a type conversion of the INT to a DECIMAL type.

More complex math

We can do more complex math

Clear Run

```
SELECT
    student_id,
    score,
    ( score - 15.1 ) / 3.8 AS z_score,
    ( CAST(score AS DECIMAL(8, 2) ) - 15.1 ) / 3.8 AS z_score
    -- ( point - mean ) / std. dev
FROM
    sampdb.score
WHERE
    event_id = 1
LIMIT 5
;

```

Clear Run

```
+------------+-------+----------+-----------+
| student_id | score | z_score  | z_score   |
+------------+-------+----------+-----------+
|          1 |    20 |  1.28947 |  1.289474 |
|          3 |    20 |  1.28947 |  1.289474 |
|          4 |    18 |  0.76316 |  0.763158 |
|          5 |    13 | -0.55263 | -0.552632 |
|          6 |    18 |  0.76316 |  0.763158 |
+------------+-------+----------+-----------+

```

We are 'hard coding' the mean and standard deviation values

We'll discuss how to get these on the fly

Strings

String functions manipulate the appearance of text

Clear Run

```
SELECT
    first_name,
    UPPER(first_name),
    LOWER(first_name)

FROM
    sampdb.president

LIMIT 5;

```

Clear Run

```
+------------+-------------------+-------------------+
| first_name | UPPER(first_name) | LOWER(first_name) |
+------------+-------------------+-------------------+
| George     | GEORGE            | george            |
| John       | JOHN              | john              |
| Thomas     | THOMAS            | thomas            |
| James      | JAMES             | james             |
| James      | JAMES             | james             |
+------------+-------------------+-------------------+

```

String extraction

String functions can "reach into" strings

Clear Run

```
SELECT
    last_name,
    LENGTH(last_name) AS len,
    LEFT(last_name, 3) AS lft,
    RIGHT(last_name, 3) AS rght,
    SUBSTRING(last_name, 3, 2) AS sub,
    SUBSTRING(last_name, LENGTH(last_name) - 2, 2) AS penultimate_2
FROM
    sampdb.president
LIMIT 5;
```

Clear Run

```
+------------+-----+-----+------+-----+---------------+
| last_name  | len | lft | rght | sub | penultimate_2 |
+------------+-----+-----+------+-----+---------------+
| Washington |  10 | Was | ton  | sh  | to            |
| Adams      |   5 | Ada | ams  | am  | am            |
| Jefferson  |   9 | Jef | son  | ff  | so            |
| Madison    |   7 | Mad | son  | di  | so            |
| Monroe     |   6 | Mon | roe  | nr  | ro            |
+------------+-----+-----+------+-----+---------------+
```

Finding strings allows more interesting usages

String manipulation becomes more interesting when we start looking for strings

```
SELECT
    email,
    -- LOCATE('@', email),
    position('@' in email)
FROM
    sampdb.member
LIMIT 5;

```

Clear Run

```
+---------------------------+------------------------+
| email                     | position('@' in email) |
+---------------------------+------------------------+
| jeanne_s@earth.com        |                      9 |
| august.lundsten@pluto.com |                     16 |
| NULL                      |                   NULL |
| arbogast.ruth@mars.net    |                     14 |
| c.dorfman@uranus.net      |                     10 |
+---------------------------+------------------------+

```

Clear Run

Here we get the position of the @ sign

We can use that, with our other functions, to get a list of domains

Putting string functions together

With the location of the '@', we can extract the domain name

```
SELECT
    email,
    RIGHT(email, LENGTH(email) - LOCATE('@', email)) AS domain
FROM
    sampdb.member
LIMIT 5;
```

Clear Run

```
+--------------------------+------------+
| email                    | domain     |
+--------------------------+------------+
| jeanne_s@earth.com       | earth.com  |
| august.lundsten@pluto.com | pluto.com  |
| NULL                     | NULL       |
| arbogast.ruth@mars.net   | mars.net   |
| c.dorfman@uranus.net     | uranus.net |
+--------------------------+------------+
```

Clear Run

Now, for each record, we can extract the domain name

Might use this to count members by domain our list use, or check for correlation with other factors

Date Functions

Clear Run

```
SELECT
    birth,
    death,
    YEAR(birth) AS yr,
    MONTH(birth) AS mnt,
    MONTHNAME(birth) AS mnt_name,
    DATEDIFF(death, birth) AS days,
    DATEDIFF(death, birth) / 365 AS years
FROM
    sampdb.president
LIMIT 5;
```

Clear Run

birth	death	yr	mnt	mnt_name	days	years
1732-02-22	1799-12-14	1732	2	February	24767	67.8548
1735-10-30	1826-07-04	1735	10	October	33119	90.7370
1743-04-13	1826-07-04	1743	4	April	30397	83.2795
1751-03-16	1836-06-28	1751	3	March	31150	85.3425
1758-04-28	1831-07-04	1758	4	April	26729	73.2301

Putting it together

We can combine records, operations and functions to calculate and format output

Clear Run

```
SELECT
    CONCAT(
        first_name,
        " ",
        last_name,
        " lived ",
        ROUND(DATEDIFF(death, birth) / 365, 1),
        " years"
    ) AS sentence
FROM
    sampdb.president
LIMIT 5;
```

Clear Run

```
+-----------------------------------+
| sentence                          |
+-----------------------------------+
| George Washington lived 67.9 years |
| John Adams lived 90.7 years       |
| Thomas Jefferson lived 83.3 years |
| James Madison lived 85.3 years    |
| James Monroe lived 73.2 years     |
+-----------------------------------+
```

More on Aggregate Functions

We attack this through dataset exploration

SELECT DISTINCT gives us the set of values in an attribute -- no repetitions

```
SELECT DISTINCT
    state
FROM
    sampdb.president
ORDER BY
    state
;

```

Clear Run

```
+-------+
| state |
+-------+
| AR    |
| CA    |
| CT    |
| GA    |
| IA    |
| IL    |
| KY    |
| MA    |
| MO    |
| NC    |
| NE    |
| NH    |
| NJ    |
| NY    |
| OH    |
| PA    |
| SC    |
| TX    |
| VA    |
| VT    |
+-------+

```

Clear Run

This is basically what we get from GROUP BY

GROUP BY subdivides the dataset into subsets

Each subset is characterized by having the same value for the GROUP BY attribute

Clear Run

```
SELECT
    state,
    COUNT(state) AS count
FROM
    sampdb.president
GROUP BY
    state
ORDER BY
    state
;

```

Clear Run

```
+-------+-------+
| state | count |
+-------+-------+
| AR    |     1 |
| CA    |     1 |
| CT    |     1 |
| GA    |     1 |
| IA    |     1 |
| IL    |     1 |
| KY    |     1 |
| MA    |     4 |
| MO    |     1 |
| NC    |     2 |
| NE    |     1 |
| NH    |     1 |
| NJ    |     1 |
| NY    |     4 |
| OH    |     7 |
| PA    |     1 |
| SC    |     1 |
| TX    |     2 |
| VA    |     8 |
| VT    |     2 |
+-------+-------+

```

Same analysis, against member

```
SELECT
    state,
    COUNT(state) AS count
FROM
    sampdb.member
GROUP BY
    state
ORDER BY
    state
LIMIT 15;
```

Clear Run

```
+-------+-------+
| state | count |
+-------+-------+
| AK    |     1 |
| AL    |     3 |
| AZ    |     1 |
| CA    |     6 |
| CO    |     3 |
| CT    |     1 |
| FL    |     5 |
| GA    |     3 |
| HI    |     1 |
| IA    |     2 |
| ID    |     1 |
| IL    |     6 |
| IN    |     3 |
| KS    |     2 |
| KY    |     2 |
+-------+-------+
```

Clear Run

DISTINCT also works by _tuples_

```
SELECT DISTINCT
    city,
    state
FROM
    sampdb.member
ORDER BY
    state,
    city
LIMIT 15
;

```

Clear Run

```
+---------------+-------+
| city          | state |
+---------------+-------+
| Fairbanks     | AK    |
| Dothan        | AL    |
| Huntsville    | AL    |
| Mobile        | AL    |
| Kayenta       | AZ    |
| Los Angeles   | CA    |
| Oakland       | CA    |
| San Francisco | CA    |
| Stockton      | CA    |
| Trona         | CA    |
| Denver        | CO    |
| Durango       | CO    |
| Waterbury     | CT    |
| Coral Gables  | FL    |
| Fort Myers    | FL    |
+---------------+-------+

```

Clear Run

Here we are getting a list of city / state value pairs that occur at least once in the data

This is not _permutations_, e.g. the possible combinations of these values -- each of these _does_ appear at least once

As does GROUP BY

Grouping by _several_ attributes subdivides counts

```
SELECT
    city,
    state,
    COUNT(*)
FROM
    sampdb.member
GROUP BY
    state,
    city
ORDER BY
    state,
    city
LIMIT 15
;

```

Clear Run

```
+---------------+-------+----------+
| city          | state | COUNT(*) |
+---------------+-------+----------+
| Fairbanks     | AK    |        1 |
| Dothan        | AL    |        1 |
| Huntsville    | AL    |        1 |
| Mobile        | AL    |        1 |
| Kayenta       | AZ    |        1 |
| Los Angeles   | CA    |        2 |
| Oakland       | CA    |        1 |
| San Francisco | CA    |        1 |
| Stockton      | CA    |        1 |
| Trona         | CA    |        1 |
| Denver        | CO    |        2 |
| Durango       | CO    |        1 |
| Waterbury     | CT    |        1 |
| Coral Gables  | FL    |        1 |
| Fort Myers    | FL    |        1 |
+---------------+-------+----------+

```

Clear Run

All the Alabama records are still in the output.

But the sum of Alabama records, which was 3, is now subdivided over three different cities

ORDER BY shows the grouping of the records

Clear Run

```
SELECT
    last_name,
    first_name,
    member_id,
    city,
    state
FROM
    sampdb.member
ORDER BY
    state,
    city
LIMIT 15
;
```

Clear Run

last_name	first_name	member_id	city	state
Matthews	Bill	56	Fairbanks	AK
Edwards	John	82	Dothan	AL
Hughes	Max	37	Huntsville	AL
Schauer	Alma	65	Mobile	AL
Kirby	Timothy	48	Kayenta	AZ
Puntillo	Cheryl	59	Los Angeles	CA
Pierson	Stanley	88	Los Angeles	CA
Sprague	Earl	99	Oakland	CA
Smith	Laura	98	San Francisco	CA
Feit	Daniel	19	Stockton	CA
Simmons	David	49	Trona	CA
Garner	Steve	89	Denver	CO
Sawyer	Dennis	7	Denver	CO
Bookstaff	Barbara	47	Durango	CO
Schenk	Cindy	42	Waterbury	CT

Referring to SQL results in SQL queries -- Subqueries

Subqueries allow us to 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

Presidents from the most presidential states

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
;

```

Clear Run

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

```

Clear Run

Hard-code the values

We can put those values into a query

```
SELECT
    last_name,
    first_name,
    state
FROM
    sampdb.president
WHERE
    state IN
    (
        'VA',
        'OH',
        'MA'
    )
ORDER BY
    state
;
```

Clear Run

```
+------------+---------------+-------+
| last_name  | first_name    | state |
+------------+---------------+-------+
| Bush       | George H.W.   | MA    |
| Adams      | John Quincy   | MA    |
| Kennedy    | John F.       | MA    |
| Adams      | John          | MA    |
| Harding    | Warren G.     | OH    |
| Taft       | William H.    | OH    |
| McKinley   | William       | OH    |
| Harrison   | Benjamin      | OH    |
| Garfield   | James A.      | OH    |
| Hayes      | Rutherford B. | OH    |
| Grant      | Ulysses S.    | OH    |
| Taylor     | Zachary       | VA    |
| Tyler      | John          | VA    |
| Harrison   | William H.    | VA    |
| Monroe     | James         | VA    |
| Wilson     | Woodrow       | VA    |
| Madison    | James         | VA    |
| Jefferson  | Thomas        | VA    |
| Washington | George        | VA    |
+------------+---------------+-------+
```

Clear Run

But, Kludgy!

What if the data changes?

```
1
```
Clear Run

```
1
```
Clear Run

```
1
```

Clear Run

```
1
```

Clear Run

```
1
```
Clear Run

```
1
```
Clear Run

Contact

gordon@practicalhorseshoeing.com