Advanced SQL

Week 7

Marilson Campos

UCSC Extension - 2022

Common Table Expressions

Complex queries can get messy.

```
SELECT
   state,
   sum(order_total) AS total_amount
FROM (
      SELECT
       user.user_uid,
       user.user_name,
       user.state as state,
       inv.total AS order_total,
   FROM tb_orders inv
   LEFT JOIN tb_user user
      ON  users.user_uid = inv.user_id
   WHERE
      users.user_uid IS NOT NULL
      AND
      inv.order_date = '2020-01-01'
) q1
GROUP BY q1.state
ORDER BY q1.state;
```

Common Table Expressions (CTE)

Complex queries can get messy – Cleaned version

```
WITH orders_today AS
(
   SELECT
      user.user_uid,
      user.user_name,
      user.state as state,
      inv.total AS order_total
   FROM tb_orders inv
   LEFT JOIN tb_user user
      ON  users.user_uid = inv.user_id
   WHERE
      users.user_uid IS NOT NULL
      AND
      inv.order_date = '2020-01-01'
)
SELECT
   state,
   sum(order_total) AS total_amount
FROM orders_today
GROUP BY state
ORDER BY state;
```

1 – Advanced Hive SQL

Temporary tables & Hive config variables

Sometimes you want to materialize an intermediary result.

```
-- This command creates a variable called temp_users.
SET temp_users=tmp_users_for_day_${hiveconf:dt_date};

DROP TABLE IF EXISTS ${hiveconf:temp_users};

CREATE TEMPORARY TABLE ${hiveconf:temp_users}
AS
  SELECT
     user_id,
     user_name
  FROM
     real_table r_table
  WHERE
     r_ table.dt = '${hiveconf:dt_date}'
  ;
```

Macros

Reuse operations by declaring Hive macros.

```
CREATE TEMPORARY MACRO valid_id(user_id STRING)
(
  CASE WHEN cast(user_id as int) is NULL THEN false ELSE true
  end
);

select valid_id('1a');
false

select valid_id('1');
true

select valid_id(NULL);
false
```

1 – SQL Windows functions

Why we need them?

Answer questions like:

How to get the top 2 salaries of each department?

How to calculate the moving average of the last
3 measurements at each point in time?

List the cumulative sales for each customer by order time?

1 – Anatomy of SQL Windows functions

SELECT
  <WindowFunction> **OVER** <WindowSpec>
FROM <table_name>

WindowFunction =

ROW_NUMBER -> sequence of numbers 1,2,3,4
RANK, DENSE_RANK, -> ranking based on field
NTILE -> split the records percentile groups.
SUM -> if contains order by = cumulative sum
        if not then just the sum of elements on the group
COUNT -> count the records on the group
AVG -> the average
LAG -> the value from previous record
LEAD -> the value from next record
FIRST_VALUE -> First record on partition
LAST_VALUE -> Last record on partition

1 – Anatomy of SQL Windows functions

SELECT
  <WindowFunction> **OVER** <WindowSpec>
FROM <table_name>

WindowSpec =
PARTITION BY <field>
ORDER BY <field>
ROWS BETWEEN <boundary> AND <boundary>

<boundary> =
  <x> PRECEDING
  UNBOUNDED PRECEDING
  CURRENT ROW
  <x> FOLLOWING
  UNBOUNDED FOLLOWING

2 – SQL Windows function example

```
The bike_rides table:

  duration INT
  start_time STRING
  end_time STRING
  start_station_number INT
  start_station STRING -- Name of the station
  end_station_number INT
  end_station STRING -- Name of the station
  bike_number STRING
  member_type STRING
```

2 – Window spec with just ‘order’

```
SELECT
  duration,
  SUM(duration) OVER (ORDER BY start_time) AS running_total
FROM bike_rides
LIMIT 25
;
```

```
1407    1407.0
202     1609.0 = 1407 + 202
519     2128.0 = 1609 + 519
522     2650.0 . . .
756     3406.0
551     3957.0
7797    11754.0
```

2 – Window spec with ‘order and partition’

```
SELECT
  start_station,
  duration,
  SUM(duration) OVER
    (PARTITION BY start_station ORDER BY start_time) AS running_total
FROM bike_rides;
```

```
10th & E St NW          942     942.0  = 942 + 0
10th & E St NW          447     1389.0 = 447 + 942
10th & E St NW          1054    2443.0
. . .
10th & E St NW          1211    32987.0 = 31776 + 1211
10th & Florida Ave NW   472     472.0  = 472 + 0
10th & Florida Ave NW   821     1293.0 = 821 + 472
10th & Florida Ave NW   751     2044.0
10th & Florida Ave NW   1315    3359.0
10th & Florida Ave NW   1059    4418.0
```

2 – Window spec with only ‘partition’

```
SELECT
  start_station,
  duration,
  SUM(duration) OVER (PARTITION BY start_station) AS start_station_total
FROM bike_rides;

10th & E St NW           860 32987.0
10th & E St NW           412 32987.0
10th & E St NW           987 32987.0
10th & E St NW           942 32987.0 = sum of values in partition
10th & Florida Ave NW    517 13321.0 = sum of values in partition
10th & Florida Ave NW    672 13321.0
10th & Florida Ave NW    751 13321.0
10th & Florida Ave NW    893 13321.0
```

2 – Window with percentage of partition total

```
SELECT
  start_station,
  duration,
  SUM(duration) OVER (PARTITION BY start_station) AS start_terminal_total,
  round((duration/SUM(duration)
    OVER (PARTITION BY start_station)) * 100, 2) AS pct_of_time
FROM bike_rides
ORDER BY start_station, pct_of_time
;
```

```
10th & E St NW          2191    32987.0 6.64
10th & E St NW          2526    32987.0 7.66
10th & E St NW          3335    32987.0 10.11
10th & E St NW          7084    32987.0 21.48
10th & Florida Ave NW   118     13321.0 0.89
10th & Florida Ave NW   294     13321.0 2.21
10th & Florida Ave NW   419     13321.0 3.15
10th & Florida Ave NW   472     13321.0 3.54
10th & Florida Ave NW   517     13321.0 3.88
```

2 – Row number on each start_station

```
SELECT
  start_station,
  start_time,
  duration,
  ROW_NUMBER() OVER
    (PARTITION BY start_station ORDER BY start_time) AS row_number
FROM bike_rides
;
```

```
10th & E St NW          2018-01-28 17:17:56  791     25
10th & E St NW          2018-01-29 15:46:25  630     26
10th & E St NW          2018-01-30 17:34:53  798     27
10th & E St NW          2018-01-31 15:38:12  1211    28
10th & Florida Ave NW   2018-01-01 14:55:34  472     1
10th & Florida Ave NW   2018-01-02 07:13:39  821     2
10th & Florida Ave NW   2018-01-10 07:48:49  751     3
10th & Florida Ave NW   2018-01-10 13:14:26  1315    4
10th & Florida Ave NW   2018-01-12 08:15:30  1059    5
```

2 – First 2 rides for each_station

```
SELECT *
FROM (
  SELECT
    start_station, start_time, duration,
    ROW_NUMBER() OVER
      (PARTITION BY start_station ORDER BY start_time) AS row_number
  FROM bike_rides
) q1
WHERE row_number < 3;
```

```
10th & Monroe St NE              2018-01-02 18:16:02  157     1
10th & Monroe St NE              2018-01-07 16:19:17  894     2
10th & U St NW                   2018-01-05 20:19:50  412     1
10th & U St NW                   2018-01-08 23:22:07  290     2
10th St & Constitution Ave NW    2018-01-02 09:49:08  1937    1
10th St & Constitution Ave NW    2018-01-02 15:50:59  761     2
10th St & L'Enfant Plaza SW      2018-01-02 18:11:25  1318    1
10th St & L'Enfant Plaza SW      2018-01-05 17:07:49  1616    2
11th & F St NW                   2018-01-04 07:12:44  196     1
11th & F St NW                   2018-01-07 09:12:32  134     2
```

2 – Ranking by minutes

```
SELECT
  start_station,
  duration,
  SUBSTRING(start_time, 1, 13) as time_slot,
  RANK() OVER
   (PARTITION BY start_station ORDER BY SUBSTRING(start_time, 1, 13)) AS
rank
FROM bike_rides;

10th & E St NW   942 2018-01-03 00    1
10th & E St NW   447 2018-01-06 16    2
10th & E St NW   412 2018-01-08 14    3
10th & E St NW   10542018-01-08 14    3
10th & E St NW   503 2018-01-11 00    5 <- there is no rank 4.
10th & E St NW   21912018-01-11 15    6
10th & E St NW   25262018-01-12 13    7
10th & E St NW   390 2018-01-12 16    8
10th & E St NW   334 2018-01-15 15    9
10th & E St NW   563 2018-01-16 18    10
10th & E St NW   19252018-01-17 18    11
```

2 – Dense ranking by minutes

```
SELECT
  start_station,
  duration,
  SUBSTRING(start_time, 1, 13) as time_slot,
  DENSE_RANK() OVER
   (PARTITION BY start_station ORDER BY SUBSTRING(start_time, 1, 13)) AS
rank
FROM bike_rides;

10th & E St NW   942 2018-01-03 00    1
10th & E St NW   447 2018-01-06 16    2
10th & E St NW   412 2018-01-08 14    3
10th & E St NW   10542018-01-08 14    3
10th & E St NW   503 2018-01-11 00    4 <- Dense does not skip.
10th & E St NW   21912018-01-11 15    5
10th & E St NW   25262018-01-12 13    6
10th & E St NW   390 2018-01-12 16    7
10th & E St NW   334 2018-01-15 15    8
10th & E St NW   563 2018-01-16 18    9
10th & E St NW   19252018-01-17 18    10
10th & E St NW   10682018-01-18 14    11
```

2 – The variation of duration between rides.

```
SELECT
  start_station,
  duration,
  duration - LAG(duration, 1) OVER
    (PARTITION BY start_station ORDER BY start_time) AS delta
FROM bike_rides;
```

```
10th & E St NW   1054    NULL <- first row does not have lag.
10th & E St NW   1068    14.0
10th & E St NW   1211    143.0
10th & E St NW   1925    714.0
10th & E St NW   2191    266.0
10th & E St NW   2526    335.0
10th & E St NW   3335    809.0
10th & E St NW   334     -3001.0
10th & E St NW   381     47.0
10th & E St NW   381     0.0
10th & E St NW   390     9.0
10th & E St NW   401     11.0
10th & E St NW   412     11.0
```

2 – Collecting previous and next record.

```
SELECT
  start_station, duration,
  LAG(duration) OVER window_spec AS prev,
  LEAD(duration) OVER window_spec AS next
FROM bike_rides
ORDER BY start_station, duration
WINDOW window_spec AS (PARTITION BY start_station ORDER BY duration)
```

Station	Curr	Last	Next	
10th & E St NW	***818***	798	823	
10th & E St NW	823	***818***	**860**	
10th & E St NW	**860**	823	**942**	
10th & E St NW	**942**	860	983	
10th & E St NW	983	942	987	
10th & E St NW	987	983	NULL	<- No next on last rec
10th & Florida Ave NW	1039	NULL	1059	<- No previous on first rec.
10th & Florida Ave NW	1059	1039	1068	
10th & Florida Ave NW	1068	1059	118	
10th & Florida Ave NW	118	1068	1315	
10th & Florida Ave NW	1315	118	294	
10th & Florida Ave NW	294	1315	419	

2 – Specifying custom window.

```
SELECT
  start_station, duration,
  SUM(duration) OVER window_spec AS tot
FROM bike_rides
ORDER BY start_station, duration
WINDOW window_spec AS (PARTITION BY start_station ORDER BY duration
    ROWS BETWEEN UNBOUNDED PRECEDING AND 1 PRECEDING);
    -- Default ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT);
```

```
10th & E St NW   1054    NULL
10th & E St NW   1068    1054.0 = 1054
10th & E St NW   1211    2122.0 = 1054 + 1068
10th & E St NW   1925    3333.0 = 1054 + 1068 + 1211
10th & E St NW   2191    5258.0
10th & E St NW   2526    7449.0
10th & E St NW   3335    9975.0
10th & E St NW   334     13310.0
10th & E St NW   381     13644.0
10th & E St NW   381     14025.0
10th & E St NW   390     14406.0
```

2 – Sum of the 2 previous records.

```
SELECT
  start_station, duration,
  SUM(duration) OVER window_spec AS tot
FROM bike_rides
ORDER BY start_station, duration
WINDOW window_spec AS (PARTITION BY start_station ORDER BY duration
    ROWS BETWEEN 2 PRECEDING AND 1 PRECEDING)
    -- 2 days + today = ROWS BETWEEN 2 PRECEDING AND CURRENT)
10th & E St NW   1054    NULL
10th & E St NW   1068    1054.0 = 1054
10th & E St NW   1211    2122.0 = 1054 + 1068
10th & E St NW   1925    2279.0.= 1068 + 1211
10th & E St NW   2191    3136.0 = 1211 + 1925
10th & E St NW   2526    4116.0
10th & E St NW   3335    4717.0
10th & E St NW   334     5861.0
10th & E St NW   381     3669.0
10th & E St NW   381     715.0
10th & E St NW   390     762.0
10th & E St NW   401     771.0
```