this code snippet returns,

ID_VAR	NULL_VAR	COALESCE_NULL_VAR
19017	(null)	MISSING
19017	(null)	MISSING
19017	(null)	MISSING
19017	(null)	MISSING
19017	(null)	MISSING
19064	(null)	MISSING
19064	(null)	MISSING
19064	(null)	MISSING
19064	(null)	MISSING
19064	(null)	MISSING
19228	(null)	MISSING
19228	(null)	MISSING
19228	(null)	MISSING
19272	(null)	MISSING

COALESCE( ) to recode NULL

One important note, however, is that in databases, ***missing values*** can be encoded in various ways besides NULL. For instance, they could be empty string/blank space (e.g., EMPTY_STR_VAR in our table), or a character string 'NA' (e.g., NA_STR_VAR in our table). In these cases, COALESCE( ) would not work, but they can be handled with the CASE WHEN statement,

```sql
--- However, COALESCE() NOT WORK for Empty or NA string, instead, use CASE WHEN
SELECT
  ID_VAR,
  EMPTY_STR_VAR,
  COALESCE(EMPTY_STR_VAR, 'MISSING') AS COALESCE_EMPTY_STR_VAR,
  CASE WHEN EMPTY_STR_VAR = ' ' THEN 'EMPTY_MISSING' END AS CASEWHEN_EMPTY_STR_VAR,

  NA_STR_VAR,
  CASE WHEN NA_STR_VAR = 'NA' THEN 'NA_MISSING' END AS CASEWHEN_NA_STR_VAR
FROM
  CURRENT_TABLE
ORDER BY ID_VAR
```

coalesce_casewhen.sql hosted with ♡ by GitHub view raw

CASE WHEN to re-code empty or NA

ID_VAR	EMPTY_STR_VAR	COALESCE_EMPTY_STR_VAR	CASEWHEN_EMPTY_STR_VAR	NA_STR_VAR	CASEWHEN_NA_STR_VAR
19228	S	S		NA	NA_MISSING
19228	S	S		NA	NA_MISSING
19228	S	S		NA	NA_MISSING
19272			EMPTY_MISSING	NA	NA_MISSING

Output from CASE WHEN

2. Compute running total and cumulative frequency

Running total can be useful when we are interested in the total sum (but not individual value) at a given point for potential analysis population segmentation and outlier identification.

The following showcases how to calculate the running total and cumulative frequency for the variable NUM_VAR,

```sql
--- 2) Running total/frequency
SELECT
  DAT.NUM_VAR,
  SUM(NUM_VAR) OVER (PARTITION BY JOIN_ID) AS TOTAL_SUM,
  ROUND(CUM_SUM / SUM(NUM_VAR) OVER (PARTITION BY JOIN_ID), 4) AS CUM_FREQ
FROM
(
  SELECT
        T.*,
        SUM(NUM_VAR) OVER (ORDER BY NUM_VAR ROWS UNBOUNDED PRECEDING) AS CUM_SUM,
        CASE WHEN ID_VAR IS NOT NULL THEN '1' END AS JOIN_ID
  FROM CURRENT_TABLE    T
) DAT
ORDER BY CUM_FREQ
```

running_total.sql hosted with ♡ by GitHub view raw

NUM_VAR	TOTAL_SUM	CUM_FREQ
62.71	24458.96	0.0026
62.71	24458.96	0.0051
62.74	24458.96	0.0077
83.05	24458.96	0.0111
99.66	24458.96	0.0152
135.13	24458.96	0.0207
135.13	24458.96	0.0262
135.13	24458.96	0.0317
198.2	24458.96	0.0398
212.35	24458.96	0.0485
302.44	24458.96	0.0609
318.53	24458.96	0.0739
424.99	24458.96	0.0913
22226.19	24458.96	1

Output for cumulative frequency

Here is our output (on the left).

Two tricks here, (1) SUM over ROWS UNBOUNDED PRECEDING will calculate the sum of all prior values to this point; (2) create a JOIN_ID to calculate the total sum.

We use the window function for this calculation, and from the cumulative frequency, it is not hard to spot the last record as an outlier.

3. Find the record(s) with extreme values without self joining

So our task is to return the row(s) with the largest NUM_VAR value for each unique ID. An intuitive query is to first find the max value for each ID using group by, and then self join on ID and the max value. Yet a more concise way would be,

```
--- 3) Find the record having a number calculated by analytic functions (e.g., MAX) without sel
SELECT *
FROM
(
  SELECT
    DAT.*,
    CASE WHEN (NUM_VAR = MAX(NUM_VAR) OVER (PARTITION BY ID_VAR)) THEN 'Y' ELSE 'N' END AS MAX_
  FROM
    CURRENT_TABLE     DAT
) DAT2
WHERE MAX_NUM_IND = 'Y'
```

Records with the max value

this query should give us the following output, showing rows having the max NUM_VAR grouped by ID,

ID_VAR	SEQ_VAR	EMPTY_STR_VAR	NULL_VAR	NA_STR_VAR	NUM_VAR	DATE_VAR1	DATE_VAR2	MAX_NUM_IND
19017	3		(null)	NA	424.99	11/2/2018	11/30/2018	Y
19064	1		(null)	NA	135.13	1/28/2019	1/28/2019	Y
19064	2		(null)	NA	135.13	1/30/2019	1/30/2019	Y
19064	3		(null)	NA	135.13	1/26/2019	1/26/2019	Y
19228	3	S	(null)	NA	62.74	3/29/2019	3/29/2019	Y
19272	1		(null)	NA	22226.19	2/24/2019	3/22/2019	Y

Output for records with the max NUM_VAR value

4. Conditional WHERE clause

Everyone knows the WHERE clause in SQL for subsetting. In fact, I find myself using conditional WHERE clause more often. With the toy table, for instance, we want only

2