SQL in ELT

Prepared by WeCloudData

SQL Syntax

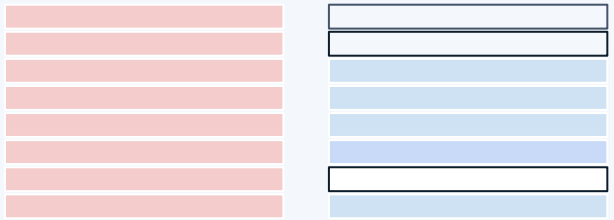
Staging Tables

Query Optimization for Snowflake

Agenda.

JOIN
SQL Clauses

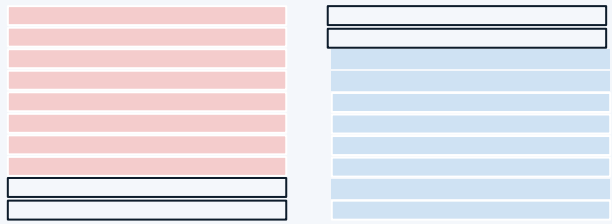
LEFT [OUTER] JOIN



- LEFT JOIN == LEFT OUTER JOIN
- Inner join ALL row of **LEFT TABLE** and rows matches in **RIGHT TABLE**. If no result matches in LEFT TABLE, displayed as **NULL**.
- If we don't want the columns displayed as NULL, we can use **NVL(expr1, expr2)**. This function indicates that if if expr1 is NULL, then use expr2.
    - **Example:**
      SELECT NVL(t2.value, 0)
      FROM t1 LEFT JOIN t2 Using(xxx).
- **Useful Case:**

A calendar table LEFT JOIN a transaction table; if there is no transaction on that date, the value will be 0.

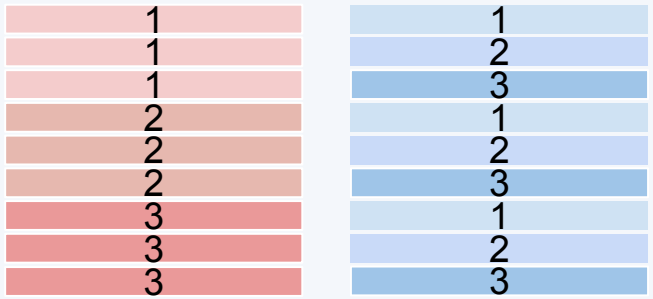
FULL OUTER JOIN



- Include all the rows from both tables. If no result match in LEFT TABLE, displayed as **NULL**.
- **Useful Case:**

If we want to combine all the records from two tables.

CROSS JOIN



- Combine each possible join from two tables.
- Can't use **ON** condition clause.
- Can use a WHERE clause to filter the results.
- **Useful Case:**

Create a table to list all the combination of prod_key and store_key. The table will be used as a reference table for ETL process.

JOIN
SQL Clauses

UNION [ALL]

- Append 2 or more tables together.
- The 2 tables must have the same table schema.
- The difference between **UNION** and **UNION ALL** is that:
  - **UNION** combines with duplicate elimination.
  - **UNION ALL** combines without duplicate elimination.
- **Useful Case:**

Sometimes, we need to work on return several staging tables separately, and UNION the tables together finally.

CASE WHEN

SQL Clauses

```
CASE
    WHEN <condition1> THEN <result1>
  [ WHEN <condition2> THEN <result2> ]
  [ ... ]
  [ ELSE <result3> ]
END
```

- Works like a “if-then-else” statement in SQL.
- **CASE WHEN** are very useful in ETL process, since there are many chances to choose the value conditionally.
- It is very common to combine **CASE WHEN** with **calculation functions**.

Example

```
SELECT
    sum(CASE WHEN c.day_of_wk_num=6 THEN s.STOCK_ON_HAND_QTY ELSE 0 end) AS EOP_STOCK_ON_HAND_QTY,
    sum(CASE WHEN c.day_of_wk_num=6 THEN s.ORDERED_STOCK_QTY ELSE 0 end) AS EOP_ORDERED_STOCK_QTY,
    count(CASE WHEN s.out_of_stock_flg=TRUE THEN 1 ELSE 0 end) AS OUT_OF_STOCK_TIMES,
```

Useful functions

SQL Clauses

CAST

Convert a column from one data type to another.

```
CAST( <source_expr> AS <target_data_type> )
```

```
CAST(submarket_key AS  varchar)
```

TO_DATE

Converts an input expression to a date.

```
TO_DATE( <string_expr> [, <format> ] )
```

```
UPDATE trans_raw SET orderdate=TO_DATE(orderdate, 'YYYY-MM-DD');
```

|| = CONCAT

Concatenates one or more strings.

```
<expr1> || <expr2>
```

```
'category'||'-'||CAST (category_key AS varchar)
```

DATEADD

Adds the specified value for the specified date.

```
DATEADD( <date_or_time_part>, <value>, <date_or_time_expr> )
```

```
dateadd(DAY, uniform(1,20, random()), trans_dt)
```

Useful link: https://docs.snowflake.com/en/sql-reference/functions-all.html

Useful functions

SQL Clauses

DIV0

Performs division like the division operator (/), but returns 0 when the divisor is 0 (rather than reporting an error).

```
DIV0( <dividend> , <divisor> )
```

```
SELECT div0(1, 0);
```

TRIM

Removes leading and trailing characters from a string. Sometimes there are extra '"' when loading csv.

```
TRIM( <expr> [, <characters> ] )
```

```
to_date(trim(orderdate, '"'))
```

CURRENT_TIMESTAMP

Returns the current timestamp for the system. Used for recording the table updating time.

```
CURRENT_TIMESTAMP( [ <fract_sec_precision> ] )
```

```
CREATE OR REPLACE TABLE walsup.store_dim
(
    update_time timestamp default CURRENT_TIMESTAMP()
```

NVL(expr1, expr2)

If expr1 is null, then use expr2. This is useful after you left join, some column value is null.

```
select s.tore_id,
       nvl(sales_amont, 0) as sales_amount
from store_dim s
left join sales sl using(store_id);
```

Useful link: https://docs.snowflake.com/en/sql-reference/functions-all.html

Window Functions

SQL Clauses

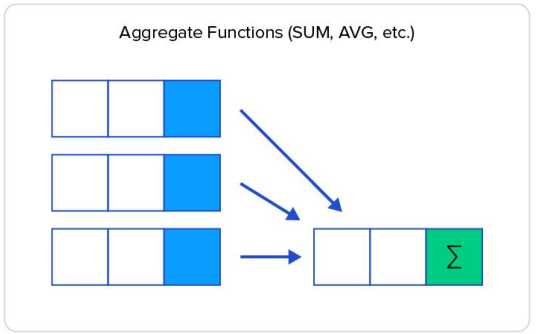
Why name Window Function?

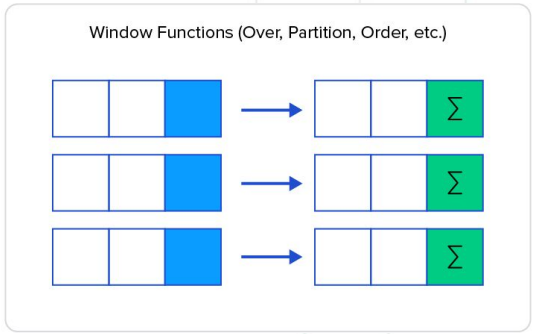
A window is a group of related rows. A window is a group. So, a window function is a function operate in a window.

In ETL process, Window Functions are usually used in staging tables to label some rows we need.

fiscal_year	sales_employee	sale	total_sales
2016	Alice	150.00	450.00
2016	Bob	100.00	450.00
2016	John	200.00	450.00
2017	Alice	100.00	400.00
2017	Bob	150.00	400.00
2017	John	150.00	400.00
2018	Alice	200.00	650.00
2018	Bob	200.00	650.00
2018	John	250.00	650.00

window





Useful link: https://docs.snowflake.com/en/sql-reference/functions-analytic.html#what-is-a-window-function

Window Functions

SQL Clauses

Syntax

RANK () **OVER** (**PARTITION BY** *col2* [**ORDER BY** *col3*])

Function — **window function clause** — **group by** — **order by**

col2

col3

student_name	dept_name	subject	marks	rank_num
Alicia	English	History	60	1
Sofia	English	History	70	2
Sofia	English	Grammar	70	2
John	English	Literature	75	3
Fred	CS	Programming	80	1
Jason	CS	Programming	80	1
Tom	CS	Programming	90	2
Rob	CS	Programming	95	3
Jason	CS	Database	98	4
Tom	CS	Database	100	5
Alter	CS	Database	100	5

Function Results

Useful Window Functions

SQL Clauses

RANK

the rank the value in a group.If two values are the same, they have the same rank. It is useful when we want to get top N rows from each group.

```
RANK() OVER (PARTITION BY state ORDER BY bushels DESC)
```

Aggregations (sum, avg, etc.)

Same as the regular aggregation functions but works on partitions.

ROW_NUMBER

Return a unique row number for each row within a window partition. This is a common function used for top N rows in each partition.

```
row_number() over (partition by exchange order by shares)
```

Useful link: https://docs.snowflake.com/en/sql-reference/functions-all.html

Merge

SQL Clauses

When **Inserts**, **updates**, and **deletes** values in a target table based on values in a source table, **MERGE** is useful if the source table is a change log that contains new rows (*to be inserted*), modified rows (**to be updated**), and/or marked rows (**to be deleted**) in the target table.

MERGE handles 2 cases:

1. match (update and delete source table)
2. not match (insert)

MERGE INTO <target_table>

USING <source> **ON** <join_expr>

WHEN MATCHED [AND <case_predicate> **] THEN**

UPDATE SET <expr>{**DELETE** <expr>}

MERGE INTO <target_table>

USING <source> **ON** <join_expr>

WHEN NOT MATCHED [AND <case_predicate> **]THEN**

INSERT [(<col_name> [, ...]) **] VALUES** (<expr> [, ...])

Merge - Examples

SQL Clauses

MATCH

```
MERGE INTO ENTP.PRODUCT_DIM_VER t1
USING ENTP.PRODUCT_RAW_STG t2
ON t1.prod_key=t2.prod_key
WHEN MATCHED
AND (t1.prod_name!=t2.prod_name
    OR t1.vol!=t2.vol )
THEN UPDATE SET
                end_date = current_date(),
                tlog_active_flg=FALSE
;
```

NOT MATCH

```
MERGE INTO ENTP.PRODUCT_DIM_VER t1
USING ENTP.PRODUCT_RAW_STG t2
ON  t1.prod_key=t2.prod_key
    AND t1.prod_name=t2.prod_name

WHEN NOT MATCHED
THEN INSERT (
    prod_key,
    prod_name)
VALUES (
    t2.prod_key,
    t2.prod_name)
;
```

Subqueries

SQL Clauses

Goal: Get a querying result on the fly, and use it to another query.

Correlated vs. Uncorrelated

```
-- Uncorrelated subquery:
select c1, c2
from table1
where c1 = (select max(x) from table2);

-- Correlated subquery:
select c1, c2
from table1
where c1 = (select x from table2 where y = table1.c2);
```

Scalar vs. Non-scalar

- A scalar subquery returns a single value (one column of one row). *It is used to get a value from another table*.
- A non-scalar subquery returns 0, 1, or multiple rows, each of which may contain 1 or multiple columns. *It can be used in a conditional clause where a column in a range of value.*

Useful link: https://docs.snowflake.com/en/user-guide/querying-subqueries.html#examples

Subquery Use Cases

SQL Clauses

- **Aggregate in multiple stages**

```
SELECT LEFT(sub.date, 2) AS cleaned_month,
       sub.day_of_week,
       AVG(sub.incidents) AS average_incidents
  FROM (
        SELECT day_of_week,
               date,
               COUNT(incidnt_num) AS incidents
          FROM tutorial.sf_crime_incidents_2014_01
         GROUP BY 1,2
       ) sub
 GROUP BY 1,2
 ORDER BY 1,2
```

- **In conditional logic**

```
SELECT *
  FROM tutorial.sf_crime_incidents_2014_01
 WHERE Date = (SELECT MIN(date)
                 FROM tutorial.sf_crime_incidents_2014_01
               )
```

```
SELECT *
  FROM tutorial.sf_crime_incidents_2014_01
 WHERE Date IN (SELECT date
                  FROM tutorial.sf_crime_incidents_2014_01
                 ORDER BY date
                 LIMIT 5
               )
```

Subquery Use Cases

SQL Clauses

- **Joining subqueries**

```
SELECT *
  FROM tutorial.sf_crime_incidents_2014_01 incidents
  JOIN ( SELECT date
           FROM tutorial.sf_crime_incidents_2014_01
          ORDER BY date
          LIMIT 5
       ) sub
    ON incidents.date = sub.date
```

- **Used in *UNION [ALL]***

```
SELECT *
FROM tutorial.crunchbase_investments_part1

 UNION ALL

 SELECT *
FROM tutorial.crunchbase_investments_part2
```

Subquery — Best Practice

SQL Clauses

- Avoid using too many layers of subqueries which will make it is difficult to debug and also my lower the query performance.
- If a SELECT result is used for several queries, consider creating a staging table (transient table) or a materialized view.
- In order to have high query performance, try to avoid Correlated Subqueries, instead, consider JOIN.

SQL Syntax

Staging Tables

Query Optimization for Snowflake

Agenda.

Staging table

Staging Tables

```
CREATE OR REPLACE TRANSIENT TABLE sp_stg0 AS
  SELECT DISTINCT inv.store_key, inv.prod_key, inv.cal_dt
  FROM precedm.inventory inv
    JOIN product_dim p ON inv.prod_key = p.PROD_KEY
    JOIN store_dim s ON inv.store_key = s.STORE_KEY
  WHERE inv.cal_dt BETWEEN to_date('&{v_sel_inv_min_dt}') AND to_date('&{v_max90_dt}')
    AND  EXISTS (SELECT DISTINCT prod_key FROM precedm.trans_line T WHERE  inv.prod_key= T.prod_key)
UNION
  SELECT DISTINCT t.store_key, t.prod_key, t.trans_dt AS cal_dt
  FROM precedm.trans_line t
    JOIN product_dim p ON t.prod_key = p.PROD_KEY
    JOIN store_dim s ON t.store_key = s.STORE_KEY
  WHERE t.trans_dt BETWEEN to_date('&{v_sel_inv_min_dt}') AND to_date('&{v_max90_dt}')
  ORDER BY 1,2,3;
```

Why create Staging table?

- Too many steps to create the final table, use the staging tables to cache the result.
- Can be reused for other table creation. Staging tables can be considered as a spare part of the product, the final table is the final product.
- Easy to debug.
- Staging tables are only visible for DE not for BI.

How to create Staging table

Staging Tables

- *CREATE TRANSIENT TABLE…..*, like creating a regular table.

```
## Creating staging tables
CREATE OR REPLACE TRANSIENT TABLE tb_stg1
.................;


CREATE OR REPLACE TRANSIENT TABLE tb_stg2
.................;


CREATE OR REPLACE TRANSIENT TABLE tb_stg3
.................;


CREATE OR REPLACE TRANSIENT TABLE tb_stg4
.................;


# Put all tables together
CREATE OR REPLACE TABLE final_tb
.......
FROM table_a
JOIN tb_stg1
JOIN tb_stg2
JOIN tb_stg3
JOIN tb_stg4
GROUP BY .....
ORDER BY .....
;
```

SQL Syntax

Staging Tables

Query Optimization for Snowflake

Agenda.

Use the Distinct Clause

Query Optimization

Row explosion happens when a JOIN query retrieves many more rows than expected. One way to reduce row explosion is by using the DISTINCT clause that neglects duplicates.

```
SELECT DISTINCT a.FirstName, a.LastName, v.District
FROM records a
INNER JOIN resources v
ON a.LastName = v.LastName
ORDER BY a.FirstName;
```

Use Temporary or Transient Tables instead of sub-query

Query Optimization

The Temp or Transient table will cach partial result which make it necessary to run some queries again and again.

```
CREATE TEMPORARY TABLE tempList AS
    SELECT a,b,c,d FROM table1
        INNER JOIN table2 USING (c);

SELECT a,b FROM tempList
    INNER JOIN table3 USING (d);
```

Check the Join Order

Query Optimization

When you are using JOIN, be careful about the order of the tables, especially for the outer join.

```
SELECT *
FROM sales
LEFT JOIN entries ON entries.id = orders.id AND entries.id = products.id;


 SELECT *
FROM entries
LEFT JOIN sales ON entries.id = orders.id AND entries.id = products.id;
```

Clustering

Query Optimization

Data clustering can significantly improve the performance. You can cluster a table when you create it or when you alter an existing table. But on the contrary, the clustering in Snowflake take extra cost.

```
CREATE TABLE recordsTable (C1 INT, C2 INT) CLUSTER BY (C1, C2);

ALTER TABLE recordsTable CLUSTER BY (C1, C2);
```

Avoid Correlated Sub-queries

Query Optimization

Avoid correlated sub queries as it searches row by row, impacting the speed of SQL query processing.

```
-- Uncorrelated subquery:
select c1, c2
from table1
where c1 = (select max(x) from table2);

-- Correlated subquery:
select c1, c2
from table1
where c1 = (select x from table2 where y = table1.c2);
```

DEMO

Demo Scripts

Thank you

WeCloudData

500-80 Bloor Street West, Toronto
ON
www.weclouddata.com