Last edited by **Isabel F Freitas** 1 year ago

Functions

SQL Pattern Matching

- https://learnsql.com/blog/using-like-match-patterns-sql/
- https://www.essentialsql.com/get-ready-to-learn-sql-5-query-results-using-pattern-matching/

SQL Lag function

- This is used to analyse if the values of rows have changed
- Examples:

```
with yoo as (
SELECT
unique_key,
logo,
LAG (logo, 1) OVER (PARTITION BY unique_key ORDER BY postingdate) as prev_balance,
logo = LAG (logo, 1) OVER (PARTITION BY unique_key ORDER BY postingdate) as curr_prev_balance
FROM raw.i2gtgeneratedtransactions)

select distinct(curr_prev_balance)
from yoo
```

- Or:

```
with yoo as (
SELECT
unique_key,
logo,
LAG (logo, 1) OVER (PARTITION BY unique_key ORDER BY postingdate) as prev_balance,
logo = LAG (logo, 1) OVER (PARTITION BY unique_key ORDER BY postingdate) as curr_prev_balance
FROM raw.i2gtgeneratedtransactions)

select *
from yoo
```

- https://www.sqlshack.com/sql-lag-function-overview-and-examples/

SQL Stats Function

```
select * from pg_stats where tablename = 'base_applicationdata' and attname = 'primarybrand'
```

SQL CREATE STATISTICS

- https://www.postgresql.org/docs/10/sql-createstatistics.html

SQL SEQUENCE And NEXTVAL

- Very good source: https://www.postgresqltutorial.com/postgresql-sequences/#:~:text=If%20a%20sequence%20is%20associated,%5B%20CASCADE%20%7C%20RESTRICT%20%5D%3B
- Examples:

```
CREATE SEQUENCE SEQUENCE_NAME
[START WITH {Initial_Value}]
[INCREMENT BY {interval}]

CREATE SEQUENCE SEQ_USER START WITH 5 INCREMENT BY 5
```

- https://www.1keydata.com/sql/sequence-nextval.html
- https://www.postgresqltutorial.com/postgresql-serial/

SQL DISTINCT AND SQL GROUP BY

DISTINCT vs. GROUP BY

```
SELECT
        city,
        state,
        zip_code
FROM
        sales.customers
GROUP BY
        city, state, zip_code
ORDER BY
        city, state, zip_code
```

It is equivalent to the following query that uses the DISTINCT operator:

```
SELECT
        DISTINCT
       city,
       state,
       zip_code
FROM
        sales.customers;
```

- **Both `DISTINCT` and `GROUP BY` clause reduces the number of returned rows in the result set by removing the duplicates**
- **However, you should use the `GROUP BY` clause when you want to apply an aggregate function on one or more columns**
- Example `stg_acc_unique_i2bsdaily_values` :

```
with i2bsdaily_values as (

    select

        unique_key,
        logo,
        openeddate

    from {{ ref('base_i2bsdaily') }}
    group by unique_key, logo, openeddate

)

select *
from i2bsdaily_values
```

- https://www.sqlservertutorial.net/sql-server-basics/sql-server-select-distinct/

IS NULL Condition

- Example:

```
SELECT *
FROM dbt_isabel_data.base_codaledger
WHERE el2 IS NULL
```

SQL IN with a subquery

```
select
        *
from dbt_isabel_data.base_i2bsdaily
where
        unique_key IN (
        SELECT
                DISTINCT(unique_key)
        FROM
                dbt_isabel_data.base_i2bsdaily
        ORDER BY unique_key
        LIMIT 1000
        )
```

- Example:

```
SELECT
        first_name,
        last_name
FROM
        customer
WHERE
        customer_id IN (
                SELECT
                        customer_id
                FROM
                        rental
                WHERE
                        CAST (return_date AS DATE) = '2005-05-27'
        );
```

- https://www.postgresqltutorial.com/postgresql-in/

SQL LEFT JOIN

```
WITH i2bsdaily as (

        select
        distinct(unique_key) as i2bsdaily_key
        from raw.i2bsdaily

), i2bsbasesegment2 as (

        select
        distinct(unique_key) as i2bsbasesegment2_key
        from raw.i2bsbasesegment2

), i2gtgeneratedtransactions as (

        select
        distinct(unique_key) as i2gtgeneratedtransactions_key
        from raw.i2gtgeneratedtransactions

), applicationdata as (

        select
        distinct(unique_key) as applicationdata_key
        from raw.applicationdata

), i2ptpostedtransactions as (

        select
        distinct(unique_key) as i2ptpostedtransactions_key
        from raw.i2ptpostedtransactions

), i2sbstatementbasesegment as (

        select
        distinct(unique_key) as i2sbstatementbasesegment_key
        from raw.i2sbstatementbasesegment

), ifrs9 as (

        select
        distinct(unique_key) as ifrs9_key
        from raw.ifrs9

), rptcd1750 as (

        select
        distinct(unique_key) as rptcd1750_key
        from raw.rptcd1750

), allcalls_2 as (

        select
        distinct(unique_key) as allcalls_2_key
        from raw.allcalls_2
```

```
)


select *
from i2bsdaily
left join i2bsbasesegment2                      on i2bsbasesegment2.i2bsbasesegment2_key
left join i2gtgeneratedtransactions on i2gtgeneratedtransactions.i2gtgeneratedtransactions_key  = i2
left join applicationdata                       on applicationdata.applicationdata_key
left join i2ptpostedtransactions        on i2ptpostedtransactions.i2ptpostedtransactions_key
left join i2sbstatementbasesegment      on i2sbstatementbasesegment.i2sbstatementbasesegment_key
left join ifrs9                                 on ifrs9.ifrs9_key
left join rptcd1750                             on rptcd1750.rptcd1750_key
left join allcalls_2                            on allcalls_2.allcalls_2_key




where
i2bsbasesegment2.i2bsbasesegment2_key                                                   = i2bsdaily
and i2gtgeneratedtransactions.i2gtgeneratedtransactions_key     = i2bsdaily.i2bsdaily_key
and applicationdata.applicationdata_key                                                 = i2bsdaily
and i2ptpostedtransactions.i2ptpostedtransactions_key                   = i2bsdaily.i2bsdaily_key
and i2sbstatementbasesegment.i2sbstatementbasesegment_key               = i2bsdaily.i2bsdaily_key
and ifrs9.ifrs9_key
and rptcd1750.rptcd1750_key
and allcalls_2.allcalls_2_key                                                           = i

ORDER BY random()
LIMIT 1000
```

```
WITH i2bsdaily as (

        select
                distinct(unique_key),
                snapshotdate
        from raw.i2bsdaily


), i2gtgeneratedtransactions as (

        select
                distinct(unique_key),
                postingdate
        from raw.i2gtgeneratedtransactions

), i2ptpostedtransactions as (

        select
                distinct(unique_key),
                postingdate
        from raw.i2ptpostedtransactions

), i2sbstatementbasesegment as (

        select
                distinct(unique_key),
                statementdate
        from raw.i2sbstatementbasesegment

), ifrs9 as (

        select
                distinct(unique_key),
                monthendposition
        from raw.ifrs9

)


select i2.unique_key
from i2bsdaily i2
```

```
left join i2gtgeneratedtransactions g on g.unique_key = i2.unique_key
        and left(i2.snapshotdate, 7) LIKE '2019-04'
        and left(g.postingdate, 7) LIKE '2019-04'

left join i2ptpostedtransactions p on p.unique_key = i2.unique_key
        and left(p.postingdate, 7) LIKE '2019-04'


ORDER BY random()
LIMIT 1000
```

- https://www.postgresql.org/message-id/20040118215714.GA11708@guests.deus.net
- https://www.postgresqltutorial.com/postgresql-left-join/

RPAD() and LPAD() function

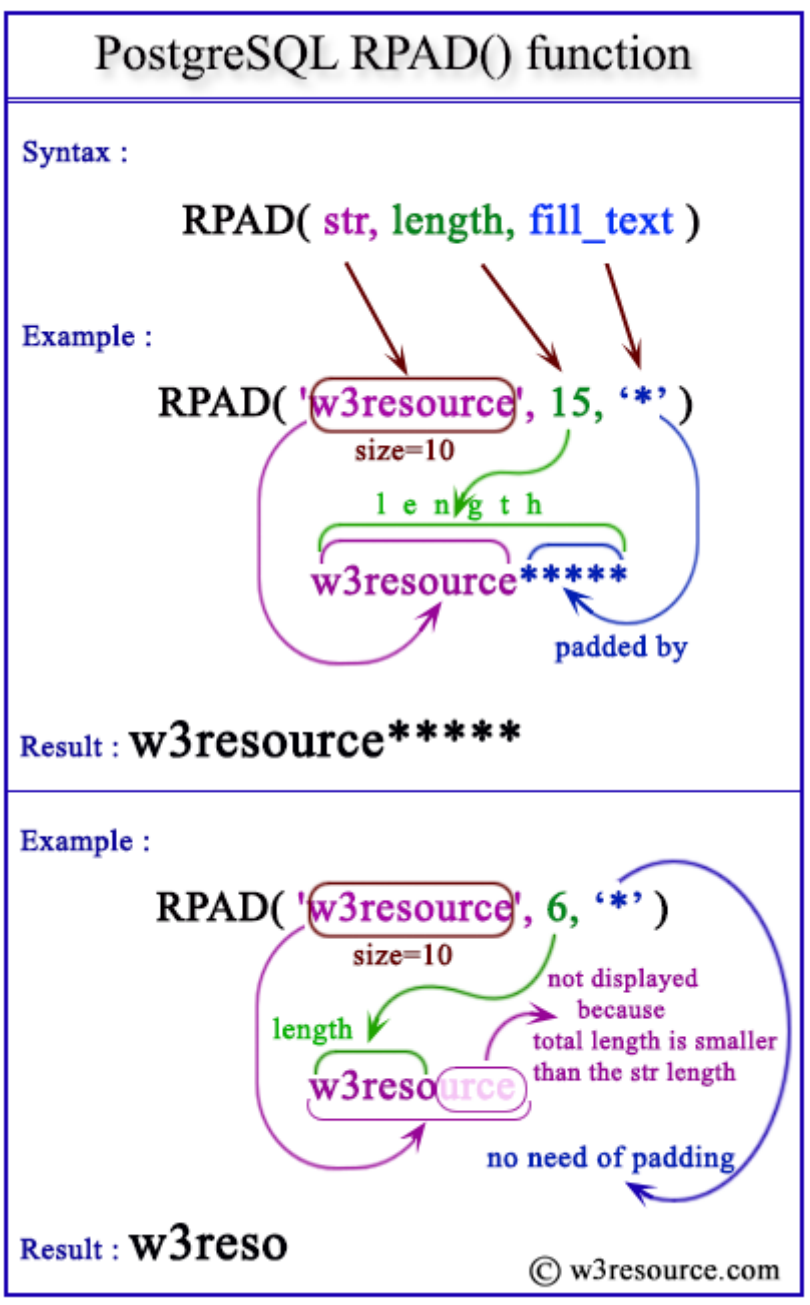
- The PostgreSQL rpad() function is used to fill up a string of specific length by a substring. If the length of the substring is equal to the remaining length of main string it will fill up properly, if less than the remaining length, the substring will repeat until it is not filling up, if longer than the remaining length or specified length it will be truncated on the left.

```
rpad(<string>, length, [<fill_text >])
```

- Example:

```
SELECT rpad('w3r', 10, 'esource');

    rpad
------------
 w3resource
```



- https://w3resource.com/PostgreSQL/rpad-function.php

LEFT() and RIGHT() function

- The PostgreSQL left() function is used to extract n number of characters specified in the argument from the left of a given string. When the value of n is negative, the extraction will happen from left except last n characters

```
left(string,n)
```

- Example:

```
SELECT left('w3resource',3)
AS "Extract 3 characters from the left";

 Extract 3 characters from the left
------------------------------------
 w3r
```

- https://w3resource.com/PostgreSQL/left-function.php

SPLIT_PART() function

- The PostgreSQL split_part function is used to split a given string based on a delimiter and pick out the desired field from the string, start from the left of the string

```
split_part(<string>,<delimiter>, <field_number>)
```

- Example:

```
SELECT split_part('ordno-#-orddt-#-ordamt', '-#-', 2)
 split_part
------------
 orddt
(1 row)
```

- Another:

```
SELECT (split_part(docdate,':', 1))::date as yoo
FROM "raw".codaledger
order by yoo

or

SELECT distinct((split_part(docdate,':', 1))::date) as yoo
FROM "raw".codaledger
order by yoo
```

- https://w3resource.com/PostgreSQL/split_part-function.php

Unpivot with Postgres

- Sometimes it’s necessary to normalize de-normalized tables - the opposite of a “crosstab” or “pivot” operation. Postgres does not support an UNPIVOT operator like Oracle or SQL Server, but simulating it, is very simple.
- https://blog.sql-workbench.eu/post/unpivot-with-postgres/

GROUPING SETS

- The GROUPING SETS is an option of the GROUP BY clause. The GROUPING SETS defines multiple grouping sets within the same query.
- The following illustrates the general syntax of the GROUPING SETS option:

```
SELECT
    c1,
    c2,
    aggregate (c3)
FROM
    table
GROUP BY
    GROUPING SETS (
        (c1, c2),
```

```
        (c1),
        (c2),
        ()
);
```

- Example:

```
SELECT
        GROUPING(warehouse) grouping_warehouse,
        GROUPING(product) grouping_product,
    warehouse,
    product,
    SUM (quantity) qty
FROM
    inventory
GROUP BY
    GROUPING SETS(
        (warehouse,product),
        (warehouse),
        (product),
        ()
    )
order by 4;
```

- As shown in the screenshot, when the value in the grouping_warehouse is 0, the sum column shows the subtotal of the warehouse.
- When the value in the grouping_product is zero, the sum column shows the subtotal of the product.
- You can use the GROUPING() function in the HAVING clause to find the subtotal of each brand like this:

```
SELECT
        GROUPING(warehouse) grouping_warehouse,
        GROUPING(product) grouping_product,
    warehouse,
    product,
    SUM (quantity) qty
FROM
    inventory
GROUP BY
    GROUPING SETS(
        (warehouse,product),
        (warehouse),
        (product),
        ()
    )
HAVING GROUPING(warehouse) = 0
ORDER BY 4;
```

- https://www.sqltutorial.org/sql-grouping-sets/#:~:text=A%20grouping%20set%20is%20a,inventory%20by%20warehouse%20and%20product.
- https://www.postgresqltutorial.com/postgresql-grouping-sets/

ROLLUP

- The PostgreSQL `ROLLUP` is a subclause of the `GROUP BY` clause that offers a shorthand for defining multiple grouping sets. A grouping set is a set of columns by which you group.
- Different from the `CUBE` subclause, `ROLLUP` does not generate all possible grouping sets based on the specified columns. It just makes a subset of those.
- The `ROLLUP` assumes a hierarchy among the input columns and generates all grouping sets that make sense considering the hierarchy. This is the reason why `ROLLUP` is often used to generate the subtotals and the grand total for reports.
- For example, the `CUBE (c1,c2,c3)` makes all eight possible grouping sets:

```
(c1, c2, c3)
(c1, c2)
(c2, c3)
(c1,c3)
(c1)
(c2)
(c3)
()
```

- However, the `ROLLUP(c1,c2,c3)` generates only four grouping sets, assuming the hierarchy `c1 > c2 > c3` as follows:

```
(c1, c2, c3)
(c1, c2)
(c1)
()
```

- A common use of `ROLLUP` is to calculate the aggregations of data by year, month, and date, considering the hierarchy `year > month > date` .
- The following illustrates the syntax of the PostgreSQL `ROLLUP` :

```
SELECT
    c1,
    c2,
    c3,
    aggregate(c4)
FROM
    table_name
GROUP BY
    ROLLUP (c1, c2, c3);
```

- It is also possible to do a partial roll up to reduce the number of subtotals generated:

```
SELECT
    c1,
    c2,
    c3,
    aggregate(c4)
FROM
    table_name
GROUP BY
    c1,
    ROLLUP (c2, c3);
```

- https://www.postgresqltutorial.com/postgresql-rollup/
- https://www.sqlservertutorial.net/sql-server-basics/sql-server-rollup/

SELECT DISTINCT ON

- PostgreSQL has a really interesting and powerful construct called SELECT DISTINCT ON. No, this is not a typical DISTINCT. This is different. It is perfect when you have groups of data that are similar and want to pull a single record out of each group, based on a specific ordering.
- With DISTINCT ON, You tell PostgreSQL to return a single row for each distinct group defined by the ON clause. Which row in that group is returned is specified with the ORDER BY clause.
- Example for the `stg_account_state` table where a row per account per month (last day of the month is needed). Code can be found at `/mnt/c/alp/alp_data_ops/alp_dbt/models/alp_sample/account_state/stg_account_state_201902.sql` :

```
select distinct on (account_key)
    a.*
from final a
order by account_key, account_state_dt DESC
```

- Select use `account_key` for the `distinct on` clause and then the `order by` clause must include `account_key` and `account_state_dt DESC` specifies which row is selected: last day of the month in this case
- https://www.geekytidbits.com/postgres-distinct-on/
- https://zaiste.net/posts/postgresql-distinct-on/

INDEXES

- Indexes are special lookup tables that the database search engine can use to speed up data retrieval.

```
CREATE INDEX index_name
ON table_name (column_name)
```

- Example:

```
CREATE INDEX index_raw_i2bsdaily_md5
ON raw.i2bsdaily (md5(unique_key))
```

- https://www.tutorialspoint.com/postgresql/postgresql_indexes.htm

FILTERED AGGREGATION

- Instead of `case when "thing to filter on" then id` you can use the `filter` aggregation method
- Eaxmple:

```
select
'A0002' as measure_id,
count (distinct case when a.is_active_acc = true then account_key else NULL end) as measure_value,
count(distinct account_key) filter (where a.is_active_acc = True and a.is_aof = True) as yoo,
'Active accounts' as measure_name,
'accounts' as source_table,
2019 as reporting_year,
2 as reporting_month
from dbt_isabel_marts.latest_mart_3 a
where
reporting_year = 2019
and
reporting_month =2;
```

- Instead of:

```
count (distinct case when a.is_active_acc = true then account_key else NULL end) as measure_value
```

- Use:

```
count(distinct account_key) filter (where a.is_active_acc) as yoo,
```

- `And` can be used too:

```
count(distinct account_key) filter (where a.is_active_acc = True and a.is_aof = True) as yoo,
```

- https://stackoverflow.com/questions/48216935/pl-pgsql-for-all-in-one-dynamic-query
- Can be used with window functions as well:
  - https://www.postgresql.org/docs/9.4/sql-expressions.html

Dynamic Stored Procedure using PL/pgSQL

```
CREATE OR REPLACE PROCEDURE public.trial_3(
    schema_name TEXT,
    table_name TEXT
)
LANGUAGE plpgsql

AS $$

DECLARE

        row RECORD;
        measure_id text;
        measure_value bigint;
        measure_name text;
        source_table text;
        measure_logic text;
        reporting_year int;
        reporting_month int;
        measure_summary_level text;
        measure_calculated_ts timestamp;

BEGIN

FOR row IN
        execute format('SELECT *
                                    FROM  %I.%I',
                    schema_name, table_name)
LOOP
```

```
        measure_id = row.measure_id;
        RAISE NOTICE 'measure_id: %', measure_id;

        measure_value = row.measure_value;
        RAISE NOTICE 'measure_value: %', measure_value;

        measure_name = row.measure_name;
        RAISE NOTICE 'measure_name: %',measure_name;

        source_table = row.source_table;
        RAISE NOTICE 'source_table: %', source_table;

        measure_logic = row.measure_logic;
        Raise notice 'measure_logic: %', measure_logic;

        reporting_year = row.reporting_year;
        RAISE NOTICE 'reporting_year: %', reporting_year;

        reporting_month = row.reporting_month;
        RAISE NOTICE 'reporting_month: %', reporting_month;

        measure_summary_level = row.measure_summary_level;
```

```
        measure_calculated_ts = NOW();
        RAISE NOTICE 'measure_calculated_ts: %', measure_calculated_ts;


        EXECUTE FORMAT('INSERT INTO dbt_isabel_data.yoo
                   select
                   $1 as measure_id,
                   $2 as measure_value,
                   $3 as measure_name,
                   $4 as source_table,
                   $5 as measure_logic,
                   $6 as reporting_year,
                   $7 as reporting_month,
                   $8 as measure_summary_level,
                   $9 as measure_calculated_ts'
                        )

        USING measure_id, measure_value, measure_name, source_table, measure_logic, reporting_year,


END LOOP;
END;
$$;
```

- Use/apply stored procedure:

```
call public.trial_3('schema_name', 'table_name')
```

ID with Prefix

```
CREATE SEQUENCE yourseq INCREMENT 1 START 1 MINVALUE 1;

CREATE TABLE yourtable_2
(
 bill_id   TEXT DEFAULT concat('2018AA', LPAD(NEXTVAL('yourseq'::regclass)::text, 6 , '0')),
 bill_desc TEXT
);

INSERT INTO yourtable_2(bill_desc) VALUES ('Telephone Bill');
INSERT INTO yourtable_2(bill_desc) VALUES ('Water Bill');
```

Measure Definition

```
CREATE OR REPLACE PROCEDURE public.measure_definition(
    schema_name TEXT,
    table_name TEXT
)
```

```
LANGUAGE plpgsql

AS $$

DECLARE

row RECORD;
account_key text;
measure_id text;
measure_value text;
measure_name text;

source_table text;
measure_logic text;
reporting_year int;
reporting_month int;
measure_summary_level text;
measure_calculated_ts timestamp;
mart_schema text;
mart_table text;
account_code text;

BEGIN

FOR row IN
        execute format('SELECT *
                                    From %I.%I',
                                schema_name, table_name)
LOOP
        measure_id = row.measure_id;
        RAISE NOTICE 'measure_id: %', measure_id;

--      measure_value = row.measure_value;
        measure_value = runstatement_4(concat('SELECT ', row.measure_value , 'FROM ', row.mart_scher
        RAISE NOTICE 'measure_value: %', measure_value;

        measure_name = row.measure_name;
        RAISE NOTICE 'measure_name: %',measure_name;

        source_table = row.source_table;
        RAISE NOTICE 'source_table: %', source_table;

        measure_logic = row.measure_logic;
        Raise notice 'measure_logic: %', measure_logic;

        reporting_year = 2019;
        RAISE NOTICE 'reporting_year: %', reporting_year;

        reporting_month = 2;
        RAISE NOTICE 'reporting_month: %', reporting_month;

        measure_summary_level = row.measure_summary_level;


        measure_calculated_ts = NOW();
        RAISE NOTICE 'measure_calculated_ts: %', measure_calculated_ts;

        mart_schema = row.mart_schema;

        mart_table = row.mart_table;

        account_code = replace(replace((SPLIT_PART(row.measure_logic, 'filter ', 2)), '(', ''), ')',

        for row in EXECUTE concat('SELECT distinct(account_key) FROM dbt_isabel_marts.latest_mart_3

        loop

        account_key = row.account_key;

        RAISE NOTICE 'account_key: %', account_key;

                EXECUTE FORMAT('INSERT INTO dbt_isabel_data.yaaaa
                   select
                   $1 as account_key,
                   $2 as measure_id,
                   $3 as measure_value,
```

```
                    $4 as measure_name,
                    $5 as source_table,
                    $6 as measure_logic,
                    $7 as reporting_year,
                    $8 as reporting_month,
                    $9 as measure_summary_level,
                    $10 as measure_calculated_ts')

                USING account_key, measure_id, measure_value, measure_name, source_table, measure_lo



END LOOP;
END LOOP;
-- RETRUNS;
END;
$$;
```