1-> zone mapping query in postgresql (solving all cases)

**CREATE** **OR** **REPLACE** **FUNCTION** getZone(p\_State **varchar**(100),p\_district **varchar**(100))

**returns** **varchar**(100)

**language** plpgsql

**as**

**$$**

**declare** p\_Zone **varchar**(100);

**begin**

**Select** **into** p\_Zone (

**SELECT**

(**CASE**

**WHEN** p\_State = 'DELHI' **THEN** 'NORTH1'

**WHEN** ((p\_State != ' ') **AND** (p\_State **IS** **NOT** **NULL**)) **THEN**(

**SELECT** **COALESCE**((**SELECT** "ZONE"

**FROM** public."SCL\_ZONE\_MAPPING\_csv" **AS** szm1

**WHERE** szm1."State" = p\_State

**fetch** **first** 1 **rows** **only**),

( **CASE**

**WHEN** ((p\_district != ' ') **AND** (p\_district **IS** **NOT** **NULL**)) **THEN**(

**SELECT** "ZONE"

**FROM** public."SCL\_ZONE\_MAPPING\_csv" **AS** szm2

**WHERE** szm2."DISTRICT" = p\_district

**fetch** **first** 1 **rows** **only**

)

**ELSE** 'UNKNOWN'

**END**

)

)

)

**ELSE**

**COALESCE**(( **CASE**

**WHEN** ((p\_district != ' ') **AND** (p\_district **IS** **NOT** **NULL**)) **THEN**(

**SELECT** szm2."ZONE"

**FROM** public."SCL\_ZONE\_MAPPING\_csv" **AS** szm2

**WHERE** szm2."DISTRICT" = p\_district

**fetch** **first** 1 **rows** **only**

)

**ELSE** 'UNKNOWN'

**END**

),'UNKNOWN')

**END**

)

**FROM** public."SCL\_ZONE\_MAPPING\_csv" szm

**fetch** **first** 1 **rows** **only**);

**if** (p\_Zone **is** **NULL**)

**THEN**

**SET** p\_Zone = 'UNKNOWN';

**END** **IF**;

**return** p\_Zone;

**end** ;

**$$** ;

2-> To get first date of current month in date and time

select date\_trunc('month', current\_date)

3->To get first date of current month

select cast(date\_trunc('month', current\_date) as date)

4-> to get current date with time

select now()

5-> to get current date without time

select cast(now() as date)

6-> to get first date of previous month with time

select date\_trunc('month', current\_date - interval '1' month)

7->to get last date of previous month without time

select cast(date\_trunc('month', current\_date)- interval '1' day as date)

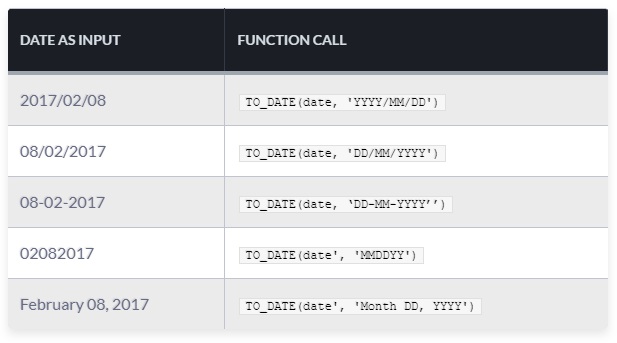
8-> to get first date of previous month without time

select cast(date\_trunc('month', current\_date - interval '1' month)as date)

9-> to get first date of current month of previous year

select cast(date\_trunc('month', current\_date - interval '1' year)as date)

10-> to convert string to date



11->

SELECT TO\_DATE('2017 Feb 20','YYYY Mon DD'); RESULT -> 2017-02-10

12-> TO CONVERT A VARCHAR TO DATE

SELECT TO\_DATE('08-02-2017','DD-MM-YYYY') RESULT -> 2017-02-08

13-> FOR DATE\_DIFF IN POSTGRES

SELECT DATE\_PART('day', "INVOICE\_DATE" - ("INVOICE\_DATE" -interval '5' day ))

from etl\_zone."T\_OEBS\_SCL\_AR\_NCR\_ADVANCE\_CALC\_TAB"

13-> TO CONVERT A VARCHAR TO TIMESTAMP

SELECT TO\_TIMESTAMP(

'2017-03-31 9:30:20',

'YYYY-MM-DD HH:MI:SS'

);

14-> conversion of varchar to time

SELECT CAST(TO\_TIMESTAMP(

'2017-03-31 9:30:20',

'YYYY-MM-DD HH:MI:SS'

)as time);

15-> to get date diff in days hour:minute:second

SELECT TO\_TIMESTAMP(

'2017-03-31 9:30:20',

'YYYY-MM-DD HH:MI:SS'

) - TO\_TIMESTAMP(

'2017-03-6 9:45:20',

'YYYY-MM-DD HH:MI:SS'

)

16-> etl\_zone.ar\_customers

CREATE OR REPLACE VIEW etl\_zone.ar\_customers

AS

SELECT cust."CUST\_ACCOUNT\_ID" AS customer\_id,

substr(party."PARTY\_NAME"::text, 1, 50) AS customer\_name,

cust."ACCOUNT\_NUMBER" AS customer\_number,

party."CUSTOMER\_KEY" AS customer\_key,

cust."STATUS" AS status,

cust."ORIG\_SYSTEM\_REFERENCE" AS orig\_system\_reference,

'CUSTOMER'::text AS customer\_prospect\_code,

party."CATEGORY\_CODE" AS customer\_category\_code,

cust."CUSTOMER\_CLASS\_CODE" AS customer\_class\_code,

cust."CUSTOMER\_TYPE" AS customer\_type,

cust."PRIMARY\_SALESREP\_ID" AS primary\_salesrep\_id,

CASE

WHEN party."PARTY\_TYPE"::text = 'ORGANIZATION'::text THEN party."SIC\_CODE"

ELSE NULL::character varying

END AS sic\_code,

party."TAX\_REFERENCE" AS tax\_reference,

cust."TAX\_CODE" AS tax\_code,

cust."FOB\_POINT" AS fob\_point,

cust."SHIP\_VIA" AS ship\_via,

CASE

WHEN party."PARTY\_TYPE"::text = 'ORGANIZATION'::text THEN party."GSA\_INDICATOR\_FLAG"

ELSE 'N'::character varying

END AS gsa\_indicator,

cust."SHIP\_PARTIAL" AS ship\_partial,

party."JGZZ\_FISCAL\_CODE" AS taxpayer\_id,

cust."PRICE\_LIST\_ID" AS price\_list\_id,

cust."FREIGHT\_TERM" AS freight\_term,

cust."ORDER\_TYPE\_ID" AS order\_type\_id,

cust."SALES\_CHANNEL\_CODE" AS sales\_channel\_code,

cust."WAREHOUSE\_ID" AS warehouse\_id,

CASE

WHEN party."PARTY\_TYPE"::text = 'ORGANIZATION'::text THEN party."MISSION\_STATEMENT"

ELSE NULL::character varying

END AS mission\_statement,

CASE

WHEN party."PARTY\_TYPE"::text = 'ORGANIZATION'::text THEN party."EMPLOYEES\_TOTAL"

ELSE NULL::numeric

END AS num\_of\_employees,

CASE

WHEN party."PARTY\_TYPE"::text = 'ORGANIZATION'::text THEN party."CURR\_FY\_POTENTIAL\_REVENUE"

ELSE NULL::numeric

END AS potential\_revenue\_curr\_fy,

CASE

WHEN party."PARTY\_TYPE"::text = 'ORGANIZATION'::text THEN party."NEXT\_FY\_POTENTIAL\_REVENUE"

ELSE NULL::numeric

END AS potential\_revenue\_next\_fy,

CASE

WHEN party."PARTY\_TYPE"::text = 'ORGANIZATION'::text THEN party."FISCAL\_YEAREND\_MONTH"

ELSE NULL::character varying

END AS fiscal\_yearend\_month,

CASE

WHEN party."PARTY\_TYPE"::text = 'ORGANIZATION'::text THEN party."YEAR\_ESTABLISHED"

ELSE NULL::numeric

END AS year\_established,

CASE

WHEN party."PARTY\_TYPE"::text = 'ORGANIZATION'::text THEN party."ANALYSIS\_FY"

ELSE NULL::character varying

END AS analysis\_fy,

party."COMPETITOR\_FLAG" AS competitor\_flag,

party."REFERENCE\_USE\_FLAG" AS reference\_use\_flag,

party."THIRD\_PARTY\_FLAG" AS third\_party\_flag,

cust."ATTRIBUTE\_CATEGORY" AS attribute\_category,

cust."ATTRIBUTE1" AS attribute1,

cust."ATTRIBUTE2" AS attribute2,

cust."ATTRIBUTE3" AS attribute3,

cust."ATTRIBUTE4" AS attribute4,

cust."ATTRIBUTE5" AS attribute5,

cust."ATTRIBUTE6" AS attribute6,

cust."ATTRIBUTE7" AS attribute7,

cust."ATTRIBUTE8" AS attribute8,

cust."ATTRIBUTE9" AS attribute9,

cust."ATTRIBUTE10" AS attribute10,

cust."ATTRIBUTE11" AS attribute11,

cust."ATTRIBUTE12" AS attribute12,

cust."ATTRIBUTE13" AS attribute13,

cust."ATTRIBUTE14" AS attribute14,

cust."ATTRIBUTE15" AS attribute15,

cust."LAST\_UPDATED\_BY" AS last\_updated\_by,

cust."LAST\_UPDATE\_DATE" AS last\_update\_date,

cust."LAST\_UPDATE\_LOGIN" AS last\_update\_login,

cust."CREATED\_BY" AS created\_by,

cust."CREATION\_DATE" AS creation\_date,

CASE

WHEN party."PARTY\_TYPE"::text = 'ORGANIZATION'::text THEN party."ORGANIZATION\_NAME\_PHONETIC"

ELSE NULL::character varying

END AS customer\_name\_phonetic,

cust."TAX\_HEADER\_LEVEL\_FLAG" AS tax\_header\_level\_flag,

cust."TAX\_ROUNDING\_RULE" AS tax\_rounding\_rule,

cust."GLOBAL\_ATTRIBUTE\_CATEGORY" AS global\_attribute\_category,

cust."GLOBAL\_ATTRIBUTE1" AS global\_attribute1,

cust."GLOBAL\_ATTRIBUTE2" AS global\_attribute2,

cust."GLOBAL\_ATTRIBUTE3" AS global\_attribute3,

cust."GLOBAL\_ATTRIBUTE4" AS global\_attribute4,

cust."GLOBAL\_ATTRIBUTE5" AS global\_attribute5,

cust."GLOBAL\_ATTRIBUTE6" AS global\_attribute6,

cust."GLOBAL\_ATTRIBUTE7" AS global\_attribute7,

cust."GLOBAL\_ATTRIBUTE8" AS global\_attribute8,

cust."GLOBAL\_ATTRIBUTE9" AS global\_attribute9,

cust."GLOBAL\_ATTRIBUTE10" AS global\_attribute10,

cust."GLOBAL\_ATTRIBUTE11" AS global\_attribute11,

cust."GLOBAL\_ATTRIBUTE12" AS global\_attribute12,

cust."GLOBAL\_ATTRIBUTE13" AS global\_attribute13,

cust."GLOBAL\_ATTRIBUTE14" AS global\_attribute14,

cust."GLOBAL\_ATTRIBUTE15" AS global\_attribute15,

cust."GLOBAL\_ATTRIBUTE16" AS global\_attribute16,

cust."GLOBAL\_ATTRIBUTE17" AS global\_attribute17,

cust."GLOBAL\_ATTRIBUTE18" AS global\_attribute18,

cust."GLOBAL\_ATTRIBUTE19" AS global\_attribute19,

cust."GLOBAL\_ATTRIBUTE20" AS global\_attribute20

FROM etl\_zone."T\_OEBS\_HZ\_CUST\_ACCOUNTS" cust,

etl\_zone."T\_OEBS\_HZ\_PARTIES" party

WHERE cust."PARTY\_ID" = party."PARTY\_ID";

17->To get start of current date

select date\_trunc('day', current\_date)

18->to get current date

select current\_date

19->FUNCTION TO GET REGION

**CREATE** **OR** **REPLACE** **FUNCTION** ey\_shree\_cement.getRegion\_test(p\_district **varchar**(100))

**returns** **varchar**(100)

**language** plpgsql

**as**

**$$**

**declare** p\_Region **varchar**(500);

**begin**

**Select** **into** p\_Region(

**select** **coalesce** ((

**case**

**WHEN** ((p\_district != '') **AND** (p\_district **IS** **NOT** **NULL**)) **then** (

**select** (**case**

**when** (szm1."Region" **IS** **NULL**) **then** 'Region is Null'

**else** szm1."Region"

**end**)

**FROM** ey\_shree\_cement."SCL\_ZONE\_MAPPING\_csv" **AS** szm1

**WHERE** szm1."DISTRICT" = p\_district

**fetch** **first** 1 **rows** **only**)

**else** 'DISTRICT IS NULL'

**end**),'Invalid District')

**FROM** ey\_shree\_cement."SCL\_ZONE\_MAPPING\_csv" szm

**fetch** **first** 1 **rows** **only**);

**if** (p\_Region **is** **null** )

**then**

**SET** p\_Region = 'REGION IS NULL';

**END** **IF**;

**return** p\_Region;

**end** ;

**$$** ;

20-> to number function in postgres

21->To convert varchar to date using substring(created and inserted values for test)

CREATE TABLE public."FOR\_TEST" (

"DATE\_1" varchar(5000) null

);

INSERT INTO public."FOR\_TEST"("DATE\_1")

VALUES ('03-02-19');

INSERT INTO public."FOR\_TEST"("DATE\_1")

VALUES ('05-07-19');

INSERT INTO public."FOR\_TEST"("DATE\_1")

VALUES ('05-02-21');

select \* from public."FOR\_TEST"

## IMPORTANT PART

SELECT to\_date(concat('20',substr("DATE\_1"::text,7,2),'-', substr("DATE\_1"::text,4,2) ,'-',substr("DATE\_1"::text,1,2)), 'yyyy-mm-dd') as jsaj

from public."FOR\_TEST"

**22->TO CHECK DATATYPE IN POSTGRESQL**

**SELECT**

column\_name,

data\_type,

character\_maximum\_length **AS** max\_length,

character\_octet\_length **AS** **octet\_length**

**FROM**

information\_schema.**columns**

**WHERE**

table\_schema = 'ey\_shree\_cement' **AND**

table\_name = 'T\_OEBS\_SCL\_ROUTE\_MASTER' **AND**

column\_name = 'ROUTE\_ID';

23-> to select timezone of india in postgresql

select now() at time zone 'Asia/Kolkata'

24-> to convert interval in hh:mm

**SELECT** **concat**(**cast**((**EXTRACT**(EPOCH **FROM** '12 hours 65 minutes'::**INTERVAL**)/3600)**as** **integer**) ,':', **cast**(((**Extract**(EPOCH **FROM** '12 hours 65 minutes'::**INTERVAL**)%3600)/60)**as** **integer**))

25-> to get difference in hours

**select** **cast**((**EXTRACT**(EPOCH **FROM** (a."GATE\_EXIT\_TIME" - a."GATE\_ENTRY\_TIME")::**interval**)/3600)**as** **integer**) TOTAL\_TIME

**from** etl\_zone."T\_OEBS\_SCL\_GATE\_ENTRY" a

26-> to get diffrerrence in seconds

**EXTRACT**(EPOCH **FROM** (a."GATE\_EXIT\_TIME" - a."GATE\_ENTRY\_TIME")) TIME

**from** etl\_zone."T\_OEBS\_SCL\_GATE\_ENTRY" a

27-> comparing time with difference of hours without using epoch from

**CASE**

**WHEN** ((**cast**(dd."SHIPPED\_QTY"**as** **numeric**)= tt."SHIPPED\_QUANTITY") **AND** (((((**cast**(tt."DILINK" **as** **timestamp**) - **interval** '30' **minute**) - ((**CASE**

**WHEN** tt."TAX\_INVOICE\_DATE" < '20-MAR-2022' **THEN** ((**cast**(tt."GATE\_EXIT\_TIME" **as** **timestamp**))+ **interval** '40'**hour**)

**ELSE** **NULL**

**END**))) )) <= **interval** '45'**hour** )) **then** 'TRUE'

**ELSE** 'FALSE'

**END** **as** OTIF\_STATUS,

28-> To change format of date according to canonical date

**SELECT** **TO\_CHAR**(**NOW**() :: **DATE**, 'dd-Mon-yy');

29-> to extract month in name format from a date

TO\_CHAR(dd."OE\_CREATION\_DT"::date, 'Month')

30-> to add a column which generates serially

ALTER TABLE test1 ADD COLUMN id SERIAL PRIMARY KEY;

31->For table having pk but not with a auto generated sequence

ALTER TABLE test1 ADD COLUMN id INTEGER;

CREATE SEQUENCE test\_id\_seq OWNED BY test1.id;

ALTER TABLE test ALTER COLUMN id SET DEFAULT nextval('test\_id\_seq');

UPDATE test1 SET id = nextval('test\_id\_seq');

32-> WORKS FOR MAKING PRESENT ID IN TABLE AS SEQUENTIAL

CREATE SEQUENCE IF NOT EXISTS etl\_zone."GODOWN\_MASTER\_SEQ"

INCREMENT 1

START 1

MINVALUE 1

MAXVALUE 9223372036854775807

CACHE 1;

alter table etl\_zone."GODOWN\_MASTER"

alter column "ID" SET DEFAULT nextval('etl\_zone."GODOWN\_MASTER\_SEQ"');

update etl\_zone."GODOWN\_MASTER"

set "ID" = nextval('etl\_zone."GODOWN\_MASTER\_SEQ"');

alter sequence etl\_zone."GODOWN\_MASTER\_SEQ" restart with 1;

33->TO GET LAST DATE OF ANY MONTH

CREATE OR REPLACE FUNCTION etl\_zone."LAST\_DAY"(date)

RETURNS date AS

$$

SELECT (date\_trunc('MONTH', $1) + INTERVAL '1 MONTH - 1 day')::date;

$$ LANGUAGE 'sql';

select etl\_zone."LAST\_DAY"('2022-10-11')

34->To check uniqueness of column

select unique\_no, count(\*)

from unique\_test

group by unique\_no

having count(\*) > 1;

35-> for using auto increment at first column using select query

**create** **table** etl\_zone."DIM\_PRODUCT\_TEST"

**as**

**select** **distinct**

1 **as** "PRODUCT\_ID",

ncr."INVENTORY\_ITEM\_ID" **as** "PRODUCT\_KEY",

ncr."PRODUCT" **as** "PRODUCT" ,

pm."ITEM\_DESC1" **as** "DESCRIPTION",

ncr."ORG\_ID" **as** "ORG\_ID",

**case**

**when** ncr."ORG\_ID" = 101 **then** 'SCL'

**when** ncr."ORG\_ID" = 102 **then** 'SHREE'

**when** ncr."ORG\_ID" = 103 **then** 'BANGUR'

**when** ncr."ORG\_ID" = 104 **then** 'ROCKSTRONG'

**else** 'NULL'

**end** **as** "BRAND",

**CASE**

**WHEN** ncr."PRODUCT"::**text** = 'PPC'::**text** **OR** ncr."PRODUCT"::**text** = 'PPC\_ROOFON'::**text** **THEN** 'PPC'::**text**

**WHEN** ncr."PRODUCT"::**text** = 'OPC\_43'::**text** **OR** ncr."PRODUCT"::**text** = 'OPC\_53'::**text** **OR** ncr."PRODUCT"::**text** = 'OPC\_PREMIUM'::**text** **THEN** 'OPC'::**text**

**WHEN** ncr."PRODUCT"::**text** = 'PSC'::**text** **THEN** 'PSC'::**text**

**WHEN** ncr."PRODUCT"::**text** = 'CC'::**text** **THEN** 'CC'::**text**

**ELSE** 'OTHERS'::**text**

**END** **AS** "PRODUCT\_CATEGORY",

ncr."PACKING\_TYPE" **as** "PACKING\_TYPE",

ncr."PACKING\_BAG" **as** "PACKING\_BAG",

**concat**(

**CASE**

**WHEN** ncr."ORG\_ID" = 102::**numeric** **THEN** 'SHREE'::**text**

**WHEN** ncr."ORG\_ID" = 103::**numeric** **THEN** 'BANGUR'::**text**

**WHEN** ncr."ORG\_ID" = 104::**numeric** **THEN** 'ROCKSTRONG'::**text**

**ELSE** **NULL**::**text**

**END**, '-', ncr."PRODUCT", '-', ncr."PACKING\_TYPE") **AS** "SKU",

1 **as** "CREATED\_BY",

(**now**() **at** **time** **zone** 'Asia/Kolkata') **as** "CREATION\_DATE",

1 **as** "LAST\_UPDATED\_BY",

(**now**() **at** **time** **zone** 'Asia/Kolkata') **as** "LAST\_UPDATE\_DATE",

1 **as** "LAST\_UPDATE\_LOGIN"

**from** etl\_zone."T\_OEBS\_SCL\_AR\_NCR\_ADVANCE\_CALC\_TAB" ncr,

etl\_zone."T\_OEBS\_XXSCL\_PRODUCT\_MASTER" pm

**where** ncr."PRODUCT" = pm."ITEM\_NO"

**CREATE** **SEQUENCE** **IF** **NOT** **EXISTS** etl\_zone."DIM\_PRODUCT\_TEST\_SEQ"

**INCREMENT** 1

**START** 10001

**MINVALUE** 1

**MAXVALUE** 9223372036854775807

**CACHE** 1;

**alter** **table** etl\_zone."DIM\_PRODUCT\_TEST"

**alter** **column** "PRODUCT\_ID" **set** **default** **nextval**(etl\_zone."DIM\_PRODUCT\_TEST\_SEQ");

**update** etl\_zone."DIM\_PRODUCT\_TEST"

**set** "PRODUCT\_ID" = **nextval**('etl\_zone."DIM\_PRODUCT\_TEST\_SEQ"');

36-> to add now() as mon-yyyy format while creating table as default

create table etl\_zone."DATE\_TEST"(

"ID" int generated always as identity(start 1

increment 1),

"NAME" text,

"PERIOD" text default TO\_CHAR(NOW() :: DATE, 'Mon-yyyy')

)

37-> Alter column type in postgresql

alter table etl\_zone."TGT\_CUSTOMER\_OUTSTANDING\_MASTER"

alter column "PERIOD" type date using "PERIOD"::date

38-> Altering a column to foreign key

ALTER TABLE etl\_zone."FACT\_NT\_SALES\_PLANNING"

ADD CONSTRAINT "FACT\_NT\_SALES\_PLANNING\_fk1" FOREIGN KEY ("ACCOUNT\_KEY") REFERENCES etl\_zone."DIM\_ACCOUNT\_TYPE"("ID");

39-> To add auto generated Id in select query

select

row\_number() over () as id

from etl\_zone."XXSCL\_AGEWISE\_SETUP"

40-> to drop foreign key of a table

Alter table table\_name drop constraint key name;

**EX - ALTER** **TABLE** target."AUGMENTATION\_OUTPUT\_TABLE"

**drop** **constraint** "AUGMENTATION\_OUTPUT\_TABLE\_pkey"

41-> Using describe in postgresql

SELECT

table\_name,

column\_name,

data\_type

FROM

information\_schema.columns

WHERE

table\_name = 'T\_OEBS\_SCL\_AR\_NCR\_ADVANCE\_CALC\_TAB';

42-> DROP NOT NULL CONSTRAINT

alter table etl\_zone."DIM\_CUSTOMERS\_TEST"

alter "CUST\_ACCOUNT\_ID" drop not null

43-> to convert numeric value to interval

(tt."TAX\_INVOICE\_DATE" )::timestamp + (((((totd."route\_distance")\*3600)/25)::text)::interval)

44-> to get random value between range where b is higher and a is lower range

SELECT floor(random()\*(b-a+1))+a;

**SELECT** **CEIL**(10.33943 \* 1000) / 1000; RESULT -> 10.34

45-> ----No. of queries running-----

SELECT datname, pid, state, query, age(clock\_timestamp(), query\_start) AS age ,\*

FROM pg\_stat\_activity

WHERE state <> 'idle'

AND query NOT LIKE '% FROM pg\_stat\_activity %'

ORDER BY age;

-----Kill a query----

SELECT pg\_terminate\_backend(27677)

46->to add primary key on a table

**ALTER TABLE [Table\_Name] ADD PRIMARY KEY (ID);**

**47->Query to get size of tables**

SELECT relkind,

relname,

pg\_catalog.pg\_namespace.nspname,

pg\_size\_pretty(pg\_relation\_size(pg\_catalog.pg\_class.oid))

FROM pg\_catalog.pg\_class

INNER JOIN pg\_catalog.pg\_namespace

ON relnamespace = pg\_catalog.pg\_namespace.oid

ORDER BY pg\_catalog.pg\_namespace.nspname,

pg\_relation\_size(pg\_catalog.pg\_class.oid) DESC;

48-> QUERY TO GET SIZE OF SCHEMA

SELECT schema\_name,

pg\_size\_pretty(sum(table\_size)::bigint),

(sum(table\_size) / pg\_database\_size(current\_database())) \* 100

FROM (

SELECT pg\_catalog.pg\_namespace.nspname as schema\_name,

pg\_relation\_size(pg\_catalog.pg\_class.oid) as table\_size

FROM pg\_catalog.pg\_class

JOIN pg\_catalog.pg\_namespace ON relnamespace = pg\_catalog.pg\_namespace.oid

) t

GROUP BY schema\_name

ORDER BY schema\_name

49-> IMPORTANT FOR LAST UPDATED ON

CREATE FUNCTION etl\_zone.sync\_lastmod() RETURNS trigger AS $$

BEGIN

NEW."LAST\_UPDATE\_DATE" := NOW() AT TIME ZONE 'Asia/Kolkata';

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER

sync\_lastmod

BEFORE UPDATE ON

etl\_zone."TEMP\_TEST"

FOR EACH ROW EXECUTE PROCEDURE

etl\_zone.sync\_lastmod();

Just to complement this very good answer: change "NEW.lastmodified" to the respective name of this field in your table. For example, if I have a column with name "updated", the FUNCTION would be "NEW.updated".

50->TO CHECK INVENTORY ITEM ID

select "LOOKUP\_CODE" from etl\_zone."T\_OEBS\_FND\_LOOKUP\_VALUES" oflv,

target."TGT\_DEPO\_INVENTORY\_STK\_F" disf

where oflv."MEANING" = disf."ITEM"

and oflv."LOOKUP\_TYPE" = 'SCL\_CEMENT\_ITEMS'

and oflv."ZD\_EDITION\_NAME" = 'SET2';

51->UPDATE A TABLE USING OTHER TABLE

update target."DIM\_CUSTOMERS\_TEST" a

set "CITY\_ID" = (select "CITY\_ID\_ERP"

from etl\_zone."SCL\_HIERARCHY\_MASTER" b

where b."ERP\_CITY\_SCL" = a."CITY"

and b."DISTRICT\_SCL" = a."DISTRICT" limit 1)

52-> SO\_OFFICER IN DIM\_CUSTOMERS

update etl\_zone."DIM\_CUSTOMERS\_TEST" dct

set "SO\_OFFICER" = (select "FIELD\_OFFICER\_NAME"

from etl\_zone."T\_OEBS\_HZ\_CUST\_ACCOUNTS" hca,

etl\_zone."T\_OEBS\_XXSCL\_MKT\_LINK" ml

where hca."ACCOUNT\_NUMBER" = dct."ACCOUNT\_NUMBER"

and hca."ATTRIBUTE3" = ml."REC\_LINK\_ID"::varchar

and ml."Active" = 1 )

53-> QUERY TO GET INDEX DETAILS IN A SCHEMA

**SELECT**

tablename,

indexname,

indexdef

**FROM**

pg\_indexes

**WHERE**

schemaname = 'etl\_zone'

**ORDER** **BY**

tablename,

indexname;

54->RIGHT JOIN QUERY(fetch data related to active dealer with outstanding data for last 12 months)

select t."CUSTOMER\_NUMBER",

t."CUSTOMER\_NAME",

t."ADDRESS1",

t."ADDRESS2",

t."ADDRESS3",

t."ADDRESS4",

t."CITY",

t."TALUKA",

t."DISTRICT",

t."STATE",

t."BRAND",

t."SECURITY\_AMOUNT",

coalesce(t."OUTSTANDING\_BALANCE",0) "OUTSTANDING\_BALANCE",

coalesce(t."NOD\_AMT",0) "NOD\_AMT",

coalesce(t."DSO",0) "DSO"

from (

(select fgo."ACCOUNT\_NUMBER",

fgo."SECURITY\_AMOUNT",

coalesce(fgo."OUTSTANDING\_BALANCE",0) "OUTSTANDING\_BALANCE",

coalesce(fgo."NOD\_AMT",0) "NOD\_AMT",

coalesce(fgo."DSO",0) "DSO"

from etl\_zone."FACT\_CUST\_GRP\_OUTSTANDING\_ONLY" fgo

where fgo."PERIOD"::date = '2022-12-01'::date

and fgo."AGE\_ID" = 14) a

right join

(

select xcm."ACCOUNT\_NUMBER" as "CUSTOMER\_NUMBER",

xcm."PARTY\_NAME" as "CUSTOMER\_NAME",

xcm."ADDRESS1",

xcm."ADDRESS2",

xcm."ADDRESS3",

xcm."ADDRESS4",

xcm."CITY",

xcm."TALUKA",

xcm."DISTRICT",

xcm."STATE",

(CASE

WHEN xcm."ORG\_ID" = 101 AND substr(xcm."PARTY\_NAME",1,4) = ':' AND xcm."PARTY\_NAME" LIKE 'S%' THEN 'Shree'

WHEN xcm."ORG\_ID" = 101 AND substr(xcm."PARTY\_NAME",1,4) = ':' AND xcm."PARTY\_NAME" LIKE 'B%' THEN 'Bangur'

WHEN xcm."ORG\_ID" = 101 AND substr(xcm."PARTY\_NAME",1,4) = ':' AND xcm."PARTY\_NAME" LIKE 'T%q' THEN 'Cemento'

WHEN xcm."ORG\_ID" = 102 THEN 'Shree'

WHEN xcm."ORG\_ID" = 103 THEN 'Bangur'

WHEN xcm."ORG\_ID" = 104 THEN 'Cemento'

ELSE 'SCL'

END) "BRAND"

from etl\_zone."T\_OEBS\_XXSCL\_CUSTOMER\_MASTER" xcm

where xcm."ACCOUNT\_NUMBER"::numeric in

(select distinct "CUSTOMER\_NUMBER"::numeric

from etl\_zone."T\_OEBS\_SCL\_AR\_NCR\_ADVANCE\_CALC\_TAB2"

where "INVOICE\_DATE"::date between '2022-12-23'::date - '12 month'::interval and '2022-12-23'::date

and "CUST\_CATEG" = 'TR'

and "CUST\_SUBCATEG" = 'TR')

)b

on a."ACCOUNT\_NUMBER" = b."CUSTOMER\_NUMBER"

)t;

55-> TO GET ALL TABLE NAMES OF SCHEMA IN POSTGRES

SELECT table\_name FROM information\_schema.tables

WHERE table\_schema in ('etl\_zone', 'target')

56-> Using nextval in select query

SELECT nextval('foo."SQ\_ID"');

# 57-> [SQL Update with row\_number()](https://stackoverflow.com/questions/13648898/sql-update-with-row-number)

With UpdateData As

(

SELECT RS\_NOM,

ROW\_NUMBER() OVER (ORDER BY [RS\_NOM] DESC) AS RN

FROM DESTINATAIRE\_TEMP

)

58-> Create foreign key example

CREATE TABLE customers(

customer\_id INT GENERATED ALWAYS AS IDENTITY,

customer\_name VARCHAR(255) NOT NULL,

PRIMARY KEY(customer\_id)

);

CREATE TABLE contacts(

contact\_id INT GENERATED ALWAYS AS IDENTITY,

customer\_id INT,

contact\_name VARCHAR(255) NOT NULL,

phone VARCHAR(15),

email VARCHAR(100),

PRIMARY KEY(contact\_id),

CONSTRAINT fk\_customer

FOREIGN KEY(customer\_id)

REFERENCES customers(customer\_id)

);

59-> TO set off read only transaction

SET default\_transaction\_read\_only = OFF;

60-> Query to check whether a table exists or not in particular schema

**SELECT** 1 **FROM** pg\_catalog.pg\_class c **JOIN** pg\_catalog.pg\_namespace n **ON** c.relnamespace = n.**oid** **WHERE** n.nspname = 'etl\_zone' **AND** c.relname = 'RCA\_EVENT\_TREE\_A1' **AND** c.relkind = 'r'

* For table c.relkind = 'r'
* For materialized view c.relkind = 'm'

61-> Function to create table and mv dynamically

CREATE OR REPLACE FUNCTION semantic."GET\_RCA\_SUMMARY\_DATA"(p\_id bigint)

RETURNS numeric

LANGUAGE plpgsql

AS $function$

declare

v\_query text;

v\_query1 text := 'select \* from target."TGT\_RCA\_SUMMARY\_DATA"';

v\_table\_schema TEXT := 'target';

v\_mv\_schema text := 'etl\_zone';

v\_mv\_name text := 'MV\_SUMMARY\_DATA';

v\_table\_name TEXT ;

v\_id bigint;

C1 cursor for

select "ID" from etl\_zone."RCA\_EVENT\_TREE"

where ("ID" = p\_id or p\_id is null)

order by "ID";

begin

v\_id := 0;

for i in C1 loop

v\_query:= NULL;

v\_table\_name:= NULL;

v\_id := i."ID";

select "QUERY\_SUMMARY",upper("MV\_SUMMARY") into v\_query,v\_table\_name

from etl\_zone."RCA\_EVENT\_TREE"

where "ID" = v\_id;

IF EXISTS (SELECT 1 FROM pg\_catalog.pg\_class c JOIN pg\_catalog.pg\_namespace n ON c.relnamespace = n.oid WHERE n.nspname = v\_table\_schema AND c.relname = upper(v\_table\_name) AND c.relkind = 'r') THEN

-- Refresh materialized view

EXECUTE 'DELETE FROM ' || v\_table\_schema || '."' || upper(v\_table\_name)||'"'|| ' WHERE "EVENT\_TREE\_ID" = ' || v\_id;

EXECUTE 'INSERT INTO ' || v\_table\_schema || '."' || upper(v\_table\_name)||'" '|| v\_query;

ELSE

-- Create materialized view

EXECUTE 'CREATE TABLE ' || v\_table\_schema || '."' || upper(v\_table\_name) ||'"'|| ' AS ' || v\_query;

END IF;

IF EXISTS (SELECT 1 FROM pg\_catalog.pg\_class c JOIN pg\_catalog.pg\_namespace n ON c.relnamespace = n.oid WHERE n.nspname = v\_mv\_schema AND c.relname = upper(v\_mv\_name) AND c.relkind = 'm') THEN

-- Refresh materialized view

EXECUTE 'REFRESH MATERIALIZED VIEW ' || v\_mv\_schema || '."' || upper(v\_mv\_name)||'"';

ELSE

-- Create materialized view

EXECUTE 'CREATE MATERIALIZED VIEW ' || v\_mv\_schema || '."' || upper(v\_mv\_name) ||'"'|| ' AS ' || v\_query1;

END IF;

v\_id := 0;

end loop;

return 0;

END;

$function$

;

62-> **Query to get tablename and column count**

**SELECT**

t.table\_name **as** "TABLE\_NAME",

(**select** **count**(t1.column\_name) **from** information\_schema.**columns** t1

**WHERE** t1.table\_schema = 'etl\_zone'

**and** t1.table\_name = t.table\_name

) **as** "COLUMN\_COUNT"

**FROM**

information\_schema.**columns** t

**WHERE**

t.table\_schema = 'etl\_zone'

**and** t.table\_name **like** 'T\_OEBS%'

**group** **by** t.table\_name

**order** **by** t.table\_name

63-> UPDATE in DO\_LEAGUE\_DATE

update etl\_zone."DO\_LEAGUE\_DATA" dld

set "ZONE" = (select distinct zm."ZONE"

from etl\_zone."ZONE\_MAPPING\_NEW" zm,

etl\_zone."SO\_TALUKA\_MAPPING\_DELHI\_HYD" std

where UPPER(zm."STATE") = UPPER(std."STATE")

and UPPER(zm."TALUKA") = UPPER(std."TALUKA")

and std."EMP\_CODE" = dld."SO\_ID"

)ss

64-> QUERY to get sequence name alongwith last value of sequence

**select** s.sequence\_name **as** "SEQUENCE\_NAME",**coalesce**(a.**last\_value**,a.start\_value) **as** "LAST\_VALUE",

**UPPER**(**regexp\_replace**(s.sequence\_name, '\_seq$|\_id\_seq$|\_SEQ$|\_ID\_$|\_ID$', '')) **AS** "TABLE\_NAME"

**from** information\_schema.**sequences** s,

pg\_catalog.pg\_sequences a

**where** a.schemaname = s.sequence\_schema

**and** a.sequencename = s.sequence\_name

**and** a.schemaname = 'etl\_zone'

65->To convert bigint into interval

**select** **make\_interval**(hours => 5)

**select** **make\_interval**(days => 5)

**select** **make\_interval**(secs => 5)

**select** **make\_interval**(mins => 5)

**select** **make\_interval**(months => 5)

**select** **make\_interval**(years => 5)

66-> Extract integer month from character month

**select** **EXTRACT**(**month** **FROM** **to\_date**(**upper**('JuNe'::**text**), 'Month'::**text**))

67-> FUNCTION TO INSERT OR UPDATE ON TABLE WHEN DATA IS INSERTED OR UPDATED IN ANOTHER TABLE

CREATE OR REPLACE FUNCTION etl\_zone."UPDATE\_OR\_INSERT\_AI\_TEMPLATE"()

RETURNS TRIGGER

LANGUAGE plpgsql

AS $function$

DECLARE

C1 CURSOR FOR

SELECT DISTINCT tosal."STATE"

FROM etl\_zone."T\_OEBS\_SCL\_ADDRESS\_LINK" tosal,

(select distinct "DISTRICT" from etl\_zone."CRM\_PRICING"

where substr("LAST\_UPDATE\_DATE"::varchar,1,23) = substr(NEW."LAST\_UPDATE\_DATE"::varchar,1,23))cp

WHERE tosal."ACTIVE" = 'Y'

and tosal."DISTRICT" = cp."DISTRICT"

;

v\_wsp\_price numeric(22,2);

v\_rsp\_price numeric(22,2);

BEGIN

raise notice 'DISTRICT is -> %', NEW."DISTRICT";

FOR I IN C1 LOOP

v\_wsp\_price := 0;

v\_rsp\_price := 0;

select AVG("WSP\_PRICE") into v\_wsp\_price

from etl\_zone."CRM\_PRICING"

where substr("LAST\_UPDATE\_DATE"::varchar,1,23) = substr(NEW."LAST\_UPDATE\_DATE"::varchar,1,23)

AND "DISTRICT" = NEW."DISTRICT"

AND "BRAND" = NEW."BRAND"

AND "DATE" = NEW."DATE"

;

select AVG("RSP\_PRICE") into v\_rsp\_price

from etl\_zone."CRM\_PRICING"

where substr("LAST\_UPDATE\_DATE"::varchar,1,23) = substr(NEW."LAST\_UPDATE\_DATE"::varchar,1,23)

AND "DISTRICT" = NEW."DISTRICT"

AND "BRAND" = NEW."BRAND"

AND "DATE" = NEW."DATE"

;

raise notice 'STATE is -> %', I."STATE";

raise notice 'DISTRICT is -> %', NEW."DISTRICT";

IF EXISTS (

SELECT 1

FROM etl\_zone."PRICING\_INPUT\_TEMPLATE\_ALL\_INDIA"

WHERE "STATE" = I."STATE"

AND "DISTRICT" = NEW."DISTRICT"

AND "BRAND" = NEW."BRAND"

AND "DATE" = NEW."DATE"

AND "PRICE\_TYPE" = 'WSP'

) THEN

-- Record exists, update it

UPDATE etl\_zone."PRICING\_INPUT\_TEMPLATE\_ALL\_INDIA"

SET "PRICE" = v\_wsp\_price,

"LAST\_UPDATED\_BY" = NEW."LAST\_UPDATED\_BY",

"LAST\_UPDATE\_DATE" = (now() AT TIME ZONE 'Asia/Kolkata'::text),

"LAST\_UPDATE\_LOGIN" = NEW."LAST\_UPDATE\_LOGIN"

WHERE "STATE" = I."STATE"

AND "DISTRICT" = NEW."DISTRICT"

AND "BRAND" = NEW."BRAND"

AND "DATE" = NEW."DATE"

AND "PRICE\_TYPE" = 'WSP';

UPDATE etl\_zone."PRICING\_INPUT\_TEMPLATE\_ALL\_INDIA"

SET "PRICE" = v\_rsp\_price,

"LAST\_UPDATED\_BY" = NEW."LAST\_UPDATED\_BY",

"LAST\_UPDATE\_DATE" = (now() AT TIME ZONE 'Asia/Kolkata'::text),

"LAST\_UPDATE\_LOGIN" = NEW."LAST\_UPDATE\_LOGIN"

WHERE "STATE" = I."STATE"

AND "DISTRICT" = NEW."DISTRICT"

AND "BRAND" = NEW."BRAND"

AND "DATE" = NEW."DATE"

AND "PRICE\_TYPE" = 'RSP';

ELSE

-- No record exists, insert new record for the first day of the current month

INSERT INTO etl\_zone."PRICING\_INPUT\_TEMPLATE\_ALL\_INDIA"

("STATE", "DISTRICT", "BRAND", "PRICE\_TYPE", "DATE", "PRICE", "CREATED\_BY", "CREATION\_DATE", "LAST\_UPDATED\_BY", "LAST\_UPDATE\_DATE", "LAST\_UPDATE\_LOGIN")

VALUES

(I."STATE", NEW."DISTRICT", NEW."BRAND", 'WSP', NEW."DATE", v\_wsp\_price, NEW."CREATED\_BY", NEW."CREATION\_DATE", NEW."LAST\_UPDATED\_BY", NEW."LAST\_UPDATE\_DATE", NEW."LAST\_UPDATE\_LOGIN");

INSERT INTO etl\_zone."PRICING\_INPUT\_TEMPLATE\_ALL\_INDIA"

("STATE", "DISTRICT", "BRAND", "PRICE\_TYPE", "DATE", "PRICE", "CREATED\_BY", "CREATION\_DATE", "LAST\_UPDATED\_BY", "LAST\_UPDATE\_DATE", "LAST\_UPDATE\_LOGIN")

VALUES

(I."STATE", NEW."DISTRICT", NEW."BRAND", 'RSP', NEW."DATE", v\_rsp\_price, NEW."CREATED\_BY", NEW."CREATION\_DATE", NEW."LAST\_UPDATED\_BY", NEW."LAST\_UPDATE\_DATE", NEW."LAST\_UPDATE\_LOGIN");

END IF;

END LOOP;

RETURN NULL;

END;

$function$;

-- drop trigger UPDATE\_OR\_INSERT\_AI\_TEMPLATE on etl\_zone."CRM\_PRICING"

CREATE TRIGGER UPDATE\_OR\_INSERT\_AI\_TEMPLATE

AFTER INSERT or update ON etl\_zone."CRM\_PRICING"

FOR EACH ROW

EXECUTE FUNCTION etl\_zone."UPDATE\_OR\_INSERT\_AI\_TEMPLATE"();

68-> YTD LOGIC

**select** t.plant,

t.d\_date, t.transporter,

(**select** **sum**(t1.qty) **from** test\_data.vendor\_performance t1

**where** t1.d\_date::**date** **between** **date\_trunc**('year',t.d\_date::**date**)::**date** **and** t.d\_date::**date**

**and** t1.plant = t.plant

**and** t1.transporter = t.transporter)qty\_sum

**from** test\_data.vendor\_performance t

**order** **by** plant ,d\_date;

69-> To get object\_type and object name in postgresql

**SELECT**

**CASE**

**WHEN** relkind = 'r' **THEN** 'Table'

**WHEN** relkind = 'v' **THEN** 'View'

**WHEN** relkind = 'm' **THEN** 'Materialized View'

**WHEN** relkind = 'f' **THEN** 'Foreign Table'

-- WHEN relkind = 'i' THEN 'Index'

**WHEN** relkind = 'S' **THEN** 'Sequence'

**ELSE** 'Other'

**END** **AS** object\_type,

relname **AS** object\_name,

relkind

**FROM**

pg\_catalog.pg\_class

**WHERE**

relnamespace = (

**SELECT** **oid**

**FROM** pg\_catalog.pg\_namespace

**WHERE** nspname = 'etl\_zone' -- Replace with your schema name

)

**and** relkind = 'r'

**ORDER** **BY**

object\_type, object\_name;

70-> converting date to text format for concat in excel  
="Date: " & TEXT(A1, "mm/dd/yyyy")

71-> Query and structure to get oracle tables in Postgresql dev:

**SELECT** table\_schema,

table\_name,

column\_name,

data\_type,

ordinal\_position,

is\_nullable,

column\_default ,

character\_maximum\_length **AS** max\_length,

character\_octet\_length **AS** **octet\_length**,

substr(table\_name,8) **as** ORACLE\_SOURCE\_TABLE

**FROM**

information\_schema.**columns**

**WHERE**

table\_schema = 'etl\_zone'

**and** table\_name **like** 'T\_OEBS%'

**order** **by** table\_name,ordinal\_position

72->TO GET MIN RANK from table where consisting of other columns with no pattern

**select** p."PRIMARY\_SECONDARY\_ROUTE",

p."PLANT\_ID",

p."WAREHOUSE",

p."PRIMARY\_FRT",

p."SECONDARY\_FRT",

p."HANDLING\_CHARGES",

p."DEMURRAGE",

p."NOTIONAL\_FREIGHT",

p."HA\_COMMISSION",

p."RAKE\_CHARGES",

p."GRADE",

p."MODE",

p."BRAND",

p."DESTINATION\_CITY",

p."DESTINATION\_TALUKA",

p."DESTINATION\_STATE",

p."CUST\_CATEGORY",

p."PLANT\_ID",

p."FREIGHT\_TYPE",

--p."SLA",

t."RANK"

**from** etl\_zone."LP\_MODEL\_DF\_RANK" p,

(**select** "GRADE", "MODE" ,"BRAND" ,

"DESTINATION\_CITY" ,"DESTINATION\_TALUKA" ,

"DESTINATION\_STATE" ,"CUST\_CATEGORY" ,"PLANT\_ID" ,"FREIGHT\_TYPE" , **min**("RANK")"RANK"

**from** etl\_zone."LP\_MODEL\_DF\_RANK"

**group** **by** "GRADE", "MODE" ,"BRAND" ,

"DESTINATION\_CITY" ,"DESTINATION\_TALUKA" ,

"DESTINATION\_STATE" ,"CUST\_CATEGORY" ,"PLANT\_ID" ,"FREIGHT\_TYPE")t

**where** p."GRADE" = t."GRADE"

**and** p."MODE" = t."MODE"

**and** p."BRAND" = t."BRAND"

**and** p."DESTINATION\_CITY" = t."DESTINATION\_CITY"

**and** p."DESTINATION\_TALUKA" = t."DESTINATION\_TALUKA"

**and** p."DESTINATION\_STATE" = t."DESTINATION\_STATE"

**and** p."CUST\_CATEGORY" = t."CUST\_CATEGORY"

**and** p."PLANT\_ID" = t."PLANT\_ID"

**and** p."FREIGHT\_TYPE" = t."FREIGHT\_TYPE"

**and** p."RANK" = t."RANK"

**and** p."GRADE" = 'PPC'

**and** p."MODE" = 'ROAD'

**AND** p."BRAND" = 103

**And** p."DESTINATION\_CITY" = 'DELHI'

**And** p."DESTINATION\_TALUKA" = 'DELHI'

**And** p."DESTINATION\_STATE" = 'Delhi'

**And** p."CUST\_CATEGORY" = 'TR'

**And** p."PLANT\_ID" = 'FGK'

**And** p."FREIGHT\_TYPE" = 'TP'

**order** **by** "RANK"

73-> TO GET MATERIALIZED VIEWS LIST

**SELECT** \*

**FROM** pg\_catalog.pg\_matviews

74-> To convert interval seconds to interval of format(hh:mm:ss)

**select** **make\_interval**(secs => (**extract**(epoch **from** '15 days 12:15:53'::**interval**))::**int**)

75-> with cte as(

select distinct

"PRODUCT" , date\_part('quarter',"TAX\_INVOICE\_DATE") as "Quarter" , sum(dispatch\_qty)::int as "Quantity"

from semantic2."MV\_TGT\_NLH\_DISPATCH"

where "TAX\_INVOICE\_DATE"::date between '2022-01-01' and '2022-12-31'

group by

"PRODUCT" , date\_part('quarter',"TAX\_INVOICE\_DATE"))

select distinct c1."PRODUCT", T."Quantity" as Q1,T1."Quantity" as Q2, T2."Quantity" as Q3,T2."Quantity" as Q4

from cte c1

left join (select C2."PRODUCT", C2."Quantity" FROM cte c2

where c2."Quarter" = 1)T

on

c1."PRODUCT" = T."PRODUCT"

left join (select C3."PRODUCT", C3."Quantity" FROM cte C3

where C3."Quarter" = 2)T1

on

c1."PRODUCT" = T1."PRODUCT"  
left join (select C4."PRODUCT", C4."Quantity" FROM cte C4

where C4."Quarter" = 3)T2

on

c1."PRODUCT" = T2."PRODUCT"

left join (select C5."PRODUCT", C5."Quantity" FROM cte C5

where C5."Quarter" = 4)T3

on

c1."PRODUCT" = T3."PRODUCT"

76-> **convert JUL-2023 into 2023-07-01**

SELECT TO\_DATE('JUL - 2023', 'MON-YYYY');

77-> Query used to delete duplicate data from accounts\_userrole table

**delete** **from** etl\_Zone.accounts\_userrole t

**where** t.id::**bigint** **in** (**select** id::**bigint** **from** (

(

**select** au1.id,A.user\_id,A.role\_name **from** (

**select** au.user\_id ,au.role\_name,**count**(\*) **from** etl\_Zone.accounts\_userrole au

**group** **by** au.user\_id ,au.role\_name **having** **count**(\*)> 1)A,

etl\_Zone.accounts\_userrole au1

**WHERE** A.user\_id = au1.user\_id

**and** A.role\_name = au1.role\_name

**and** au1.id **not** **in** (**select** **min**(id) **from** etl\_Zone.accounts\_userrole au2

**where** au2.user\_id = A.user\_id

**and** au2.role\_name = A.role\_name)

))b)

**76->Update a table using another table**

**update etl\_zone."DEALER\_LEDGER\_TEST" as b**

**set "Active" =0**

**from etl\_zone.temp\_dl c**

**where b."TRX\_LINE\_GL\_RC\_ID" = c."TRX\_LINE\_GL\_RC\_ID"**

**and b."TRX\_LINE\_LAST\_UPDATE\_DATE" < c.max**

**and b."TRX\_LINE\_GL\_RC\_ID" = c."TRX\_LINE\_GL\_RC\_ID"**

**77->** To get result of a function in proper tabular form using a function.

**CREATE** **OR** **REPLACE** **FUNCTION** etl\_zone.get\_user\_list\_for\_alerts(p\_alert\_id **bigint**, p\_role\_name **text**)

**RETURNS** **TABLE**(user\_id **bigint**, role\_name **character** **varying**, "ALERT\_ID" **bigint**, email **character** **varying**, "EMAIL\_CONTENT" **text**, "SMS\_CONTENT" **text**, "NOTIFICATION\_CONTENT" **text**)

**LANGUAGE** plpgsql

**AS** **$function$**

**BEGIN**

**RETURN** QUERY

**SELECT** au.user\_id, au.role\_name, am."ID" , u.email, am."EMAIL\_CONTENT", am."SMS\_CONTENT", am."NOTIFICATION\_CONTENT"

**FROM** etl\_Zone.accounts\_userrole au

**JOIN** etl\_zone.users u **ON** au.user\_id = u.id

**JOIN** etl\_zone."ALERT\_MASTER" am **ON** am."ID" = p\_alert\_id

**WHERE** au.role\_name = p\_role\_name;

**END**;

**$function$**

;

* Call above function using below query to get data in tabular form

**SELECT**

user\_id **AS** "User ID",

role\_name **AS** "Role Name",

"ALERT\_ID" **AS** "Alert ID",

email **AS** "Email",

"EMAIL\_CONTENT" **AS** "Email Content",

"SMS\_CONTENT" **AS** "SMS Content",

"NOTIFICATION\_CONTENT" **AS** "Notification Content"

**from** etL\_zone.get\_user\_list\_for\_alerts(10004,'RRCCEMENT');

78 -> Update one table using another

**update etl\_zone."DEALER\_LEDGER\_TEST" as b**

**set "Active" =0**

**from etl\_zone.temp\_dl c**

**where b."TRX\_LINE\_GL\_RC\_ID" = c."TRX\_LINE\_GL\_RC\_ID"**

**and b."TRX\_LINE\_LAST\_UPDATE\_DATE" < c.max**

**and b."TRX\_LINE\_GL\_RC\_ID" = c."TRX\_LINE\_GL\_RC\_ID"**

**77->** **To get result of a function in proper tabular form using a function with single parameter having multiple values.**

CREATE OR REPLACE FUNCTION etl\_zone.get\_user\_list\_for\_alerts\_test(p\_alert\_id bigint, p\_role\_names text[])

RETURNS TABLE (

user\_id bigint,

role\_name varchar(360),

"ALERT\_ID" bigint,

email varchar(360),

"EMAIL\_CONTENT" text,

"SMS\_CONTENT" text,

"NOTIFICATION\_CONTENT" text

) AS $$

BEGIN

RETURN QUERY

SELECT au.user\_id, au.role\_name, am."ID", u.email, am."EMAIL\_CONTENT", am."SMS\_CONTENT", am."NOTIFICATION\_CONTENT"

FROM etl\_zone.accounts\_userrole au

JOIN etl\_zone.users u ON au.user\_id = u.id

JOIN etl\_zone."ALERT\_MASTER" am ON am."ID" = p\_alert\_id

WHERE au.role\_name = ANY(p\_role\_names);

END;

$$ LANGUAGE plpgsql;

* Call the function

SELECT \* FROM etl\_zone.get\_user\_list\_for\_alerts\_test(10008,ARRAY['NSH', 'TSM']);

**78-> CONVERT UTC timestamp to IST**

**SELECT** '2023-03-13 06:35:49.000 +0530'::**timestamptz** **AT** **TIME** **ZONE** 'Asia/Kolkata' **AS** ist\_timestamp

**79-> give message alert in a single row in proper tabular format**

**CREATE** **OR** **REPLACE** **FUNCTION** etl\_zone.ALERT\_FOR\_MRN\_COMPLETED()

**RETURNS** **numeric**

**LANGUAGE** plpgsql

**AS** **$function$**

**DECLARE**

msg **text**;

v\_delivery\_id **text**;

v\_receipt\_number **text**;

C1 **CURSOR** **FOR**

**SELECT** **distinct**

user\_id **AS** "USER\_ID",

-- role\_name AS "ROLE\_NAME",

"ALERT\_ID" **AS** "ALERT\_ID",

email **AS** "EMAIL",

"EMAIL\_CONTENT" ,

"SMS\_CONTENT" ,

"NOTIFICATION\_CONTENT"

**from** etL\_zone.get\_user\_list\_for\_alerts\_roles(10060,**ARRAY**['SLR']);

C2 **CURSOR** **FOR**

**SELECT** **DISTINCT** ttcc."DELIVERY\_ID" ,ttcc."RECEIPT\_NUMBER"

**FROM** target."TGT\_MRN\_DATA" ttcc

**WHERE** ttcc."RECEIPT\_DATE"::**timestamp** **between** ('2023-07-24 16:27:18.000'::**timestamp** - '3 hour'::**interval**) **and** '2023-07-24 16:27:18.000'::**timestamp** ;

**BEGIN**

**FOR** I **IN** C1 **LOOP**

-- Initialize variables for each iteration of the outer loop

v\_delivery\_id := **NULL**;

v\_receipt\_number := **NULL**;

**FOR** I1 **IN** C2 **LOOP**

-- Concatenate truck numbers and token IDs with commas

v\_delivery\_id := **COALESCE**(v\_delivery\_id, '') || E'\n' || I1."DELIVERY\_ID"|| E'\t' ||I1."RECEIPT\_NUMBER";

-- v\_receipt\_number := COALESCE(v\_receipt\_number, '') || E'\n' || I1."RECEIPT\_NUMBER";

**END** **LOOP**;

-- Remove leading comma from v\_vehicle and v\_token

v\_delivery\_id := **NULLIF**(**TRIM**(**LEADING** ',' **FROM** v\_delivery\_id), '');

-- v\_receipt\_number := NULLIF(TRIM(LEADING ',' FROM v\_receipt\_number), '');

-- Construct the message based on the values

**IF** v\_delivery\_id **IS** **NOT** **NULL** **THEN**

msg := I."EMAIL\_CONTENT" || 'for: '||E'\n' || 'DELIVERY ID' || E'\t' || 'RECEIPT NUMBER' || E'\n' || v\_delivery\_id;

-- || E'\t' || v\_receipt\_number;

**ELSE**

msg := I."EMAIL\_CONTENT" || E'\n' || 'for no delivery id.';

**END** **IF**;

**if** (msg != I."EMAIL\_CONTENT") **then**

**INSERT** **INTO** etl\_zone."ALERT\_TRANSACTION"

("ALERT\_ID", "USER\_ID", "TYPE", "IS\_READ", "MOBILE\_NUMBER", "EMAIL", "EMAIL\_CONTENT", "SMS\_CONTENT", "NOTIFICATION\_CONTENT", "IS\_ACTIVE", "IS\_SMS\_SEND", "IS\_EMAIL\_SEND")

**VALUES** (I."ALERT\_ID"::**bigint**, I."USER\_ID"::**bigint**, **NULL**, 'N', **NULL**::**bigint**, I."EMAIL", msg, msg, msg, 'Y', 'Y', 'Y');

**else**

-- Log the message

**RAISE** **NOTICE** 'msg is %', msg;

**end** **if**;

**END** **LOOP**;

**RETURN** 0;

**END**;

**$function$**;

80-> To cancel the endless loop have used the condition for new last update date new zone must not be null

**CREATE** **OR** **REPLACE** **FUNCTION** etl\_zone.populate\_branding\_activity\_zone()

**RETURNS** **trigger**

**LANGUAGE** plpgsql

**AS** **$function$**

**DECLARE**

v\_zone **varchar**(360);

**BEGIN**

v\_zone := **NULL**;

**select** **distinct** shm."ZONE\_SCL" **into**

v\_zone **from** etl\_zone."BRANDING\_ACTIVITY" ba,

etl\_zone."SCL\_HIERARCHY\_MASTER" shm

**WHERE** **UPPER**(shm."STATE\_ERP") = **UPPER**(ba."STATE")

**and** **UPPER**(shm."DISTRICT\_ERP") = **UPPER**(ba."DISTRICT" )

**and** ba."LAST\_UPDATE\_DATE" = **NEW**."LAST\_UPDATE\_DATE"

**and** ba."ZONE" **is** **null** ;

**raise** **notice** 'zone is -> %', v\_zone;

**if** v\_zone **is** **not** **null** **then**

**UPDATE** etl\_zone."BRANDING\_ACTIVITY" ba

**SET** "ZONE" = v\_zone

**where** "LAST\_UPDATE\_DATE" = **NEW**."LAST\_UPDATE\_DATE";

**else**

**raise** **notice** 'zone is -> null';

**end** **if**;

**RETURN** **NEW**;

**END**;

**$function$**

;

--drop trigger populate\_branding\_activity\_zone on etl\_zone."BRANDING\_ACTIVITY"

**create** **trigger** populate\_branding\_activity\_zone **after** **insert** **or**

**update**

**on**

etl\_zone."BRANDING\_ACTIVITY" **for** **each** **row** **execute** **function** etl\_zone.populate\_branding\_activity\_zone();

**81-> delete duplicate**

**delete** **from** etl\_Zone."LP\_SCHEDULING\_CRM\_CHECKS1" t

**where** t."ID" ::**bigint** **in** (**select** "ID"::**bigint** **from** (

(

**select** au1."ID",A."ERP\_LINE\_NUMBER" **from** (

**select** "ERP\_LINE\_NUMBER",**count**(\*) **from** etl\_Zone."LP\_SCHEDULING\_CRM\_CHECKS1" au

**group** **by** "ERP\_LINE\_NUMBER" **having** **count**(\*)> 1)A,

etl\_Zone."LP\_SCHEDULING\_CRM\_CHECKS1" au1

**WHERE** A."ERP\_LINE\_NUMBER" = au1."ERP\_LINE\_NUMBER"

**and** au1."ID" **not** **in** (**select** **max**("ID") **from** etl\_Zone."LP\_SCHEDULING\_CRM\_CHECKS1" au2

**where** au2."ERP\_LINE\_NUMBER" = A."ERP\_LINE\_NUMBER"

)

))b)

**82-> Description cust categ and sub category wise**

**SELECT** "PARENT\_FLEX\_VALUE\_LOW" **as** "CUST\_CATEG","FLEX\_VALUE" **as** "CUST\_SUB\_CATEG","DESCRIPTION"

**FROM** etl\_zone."T\_OEBS\_FND\_FLEX\_VALUES\_TL" T,

etl\_zone."T\_OEBS\_FND\_FLEX\_VALUES" B

**WHERE** B."FLEX\_VALUE\_ID" = T."FLEX\_VALUE\_ID"

**AND** B."FLEX\_VALUE\_SET\_ID" =1011993

**AND** T."ZD\_EDITION\_NAME"='SET1'

**AND** B."ZD\_EDITION\_NAME"='SET1'

82-> /\*\* POPULATE GODOWN MASTER \*\*/

**CREATE** **OR** **REPLACE** **FUNCTION** semantic."POPULATE\_GODOWN\_MASTER"()

**RETURNS** **numeric**

**LANGUAGE** plpgsql

**AS** **$function$**

**begin**

**INSERT** **INTO** etl\_zone."GODOWN\_MASTER"("NAME","STATE","CITY","DISTRICT","CREATED\_AT","UPDATED\_AT")

**with** cte **as**

(**select** **distinct** "PARTY\_NAME" **as** "NAME","STATE" ,"CITY" ,"DISTRICT",

(**now**() **AT** **TIME** **ZONE** 'Asia/Kolkata'::**text**) **as** "CREATED\_AT", (**now**() **AT** **TIME** **ZONE** 'Asia/Kolkata'::**text**) **as** "UPDATED\_AT"

**from** target."TGT\_PLANT\_DEPO\_MASTER" tpdm

**where** "TYPE" = 'DEPO'

**and** "CATEGORY\_CODE" **in** ('BANGUR DEPOT','SHREE DEPOT', 'CEMENTO DEPOT')

**and** "Active" = 1)

**select** \* **from** cte

-- where substr("NAME",1,3) = 'BQW'

**ON** **conflict** ("NAME")

**do** **update** **set**

"STATE" = excluded."STATE",

"CITY" = excluded."CITY",

"DISTRICT" = excluded."DISTRICT",

"UPDATED\_AT" = (**now**() **AT** **TIME** **ZONE** 'Asia/Kolkata'::**text**);

**delete** **from** etl\_zone."GODOWN\_MASTER" gm

**where** substr("NAME",1,3) **in** (**select**

substr("PARTY\_NAME",1,3)

**from** target."TGT\_PLANT\_DEPO\_MASTER"

**where** "ORG\_ACTIVE" = 0

);

**return** 0;

**END**;

**$function$**

;

**82-> /\*\*\* CUSTOMER BILL TO SHIP TO DYNAMIC DATA \*\*\*/**

**CREATE** **OR** **REPLACE** **FUNCTION** etl\_zone.get\_customer\_data(p\_account\_number **text**[], p\_date **date**)

**RETURNS** **TABLE**("ACCOUNT\_NUMBER" **character** **varying**, "CUST\_ACCOUNT\_ID" **bigint**, "CUSTOMER\_NAME" **character** **varying**, "CUST\_CAT" **character** **varying**, "CUST\_SUB\_CAT" **character** **varying**, "CITY" **character** **varying**, "TALUKA" **character** **varying**, "DISTRICT" **character** **varying**, "STATE" **character** **varying**, "MOBILE\_NUMBER" **bigint**, "EMAIL" **character** **varying**, "DATE\_OF\_JOINING" **timestamp** **without** **time** **zone**, "ORG\_ID" **bigint**, "BRAND" **character** **varying**, "CUSTOMER\_TYPE" **character** **varying**, "BANK\_ACCOUNT\_NO" **character** **varying**, "BANK\_NAME" **character** **varying**, "IFSC\_CODE" **character** **varying**, "GSTIN" **text**, "PanNo" **text**)

**LANGUAGE** plpgsql

**AS** **$function$**

**BEGIN**

**RETURN** QUERY

**SELECT** hzca."ACCOUNT\_NUMBER",

hzca."CUST\_ACCOUNT\_ID",

hzp."PARTY\_NAME" **AS** "CUSTOMER\_NAME",

hzca."ATTRIBUTE1" **AS** "CUST\_CAT",

hzca."ATTRIBUTE2" **AS** "CUST\_SUB\_CAT",

hl."CITY",

hl."ADDRESS\_LINES\_PHONETIC" **AS** "TALUKA",

hl."COUNTY" **AS** "DISTRICT",

hl."STATE",

hl."ATTRIBUTE20"::**bigint** **as** "MOBILE\_NUMBER",

hl."ATTRIBUTE17" **as** "EMAIL",

hzca."CREATION\_DATE" **AS** "DATE\_OF\_JOINING",

hcasa."ORG\_ID"::**bigint** **as** "ORG\_ID",

(**CASE**

**WHEN** hcasa."ORG\_ID" = 102::**numeric** **THEN** 'SHREE'::**text**

**WHEN** hcasa."ORG\_ID" = 103::**numeric** **THEN** 'BANGUR'::**text**

**WHEN** hcasa."ORG\_ID" = 104::**numeric** **THEN** 'ROCKSTRONG'::**text**

**ELSE** 'OTHERS'::**text**

**END**)::**varchar**(360) **AS** "BRAND",

(**CASE**

**WHEN** hzca."ATTRIBUTE1"::**text** = 'TR'::**text** **AND** hzca."ATTRIBUTE2"::**text** = 'TR'::**text** **THEN** 'DEALER'::**text**

**WHEN** hzca."ATTRIBUTE1"::**text** = 'TR'::**text** **AND** hzca."ATTRIBUTE2"::**text** = 'RT'::**text** **THEN** 'RETAILER'::**text**

**ELSE** 'Others'::**text**

**END**)::**varchar**(360) **AS** "CUSTOMER\_TYPE",

hzca."ATTRIBUTE4"::**varchar**(360) **as** "BANK\_ACCOUNT\_NO",

hzca."ATTRIBUTE6"::**varchar**(360) **as** "BANK\_NAME",

hzca."ATTRIBUTE5"::**varchar**(360) **AS** "IFSC\_CODE",

(**SELECT** **MIN**(jprl."SECONDARY\_REGISTRATION\_NUMBER")

**FROM** etl\_zone."T\_OEBS\_JAI\_PARTY\_REG\_LINES" JPRL,etl\_zone."T\_OEBS\_JAI\_PARTY\_REGS" JPR

**WHERE** JPRL."PARTY\_REG\_ID" = JPR."PARTY\_REG\_ID"

**AND** JPR."PARTY\_SITE\_ID" = hcasa."CUST\_ACCT\_SITE\_ID"

**AND** JPR."PARTY\_ID" = hzca."CUST\_ACCOUNT\_ID"

**AND** JPRL."SEC\_REGISTRATION\_TYPE\_CODE" = 'GSTNO'

**AND** JPR."PARTY\_TYPE\_CODE" **IN** ('THIRD\_PARTY', 'THIRD\_PARTY\_SITE')

**AND** JPR."Active" = 1

**AND** JPRL."Active" =1

) "GSTIN",

(**SELECT** **MIN**(JPRL."REGISTRATION\_NUMBER")

**FROM** etl\_zone."T\_OEBS\_JAI\_PARTY\_REG\_LINES" JPRL, etl\_zone."T\_OEBS\_JAI\_PARTY\_REGS" JPR

**WHERE** JPRL."PARTY\_REG\_ID" = JPR."PARTY\_REG\_ID"

**AND** JPR."PARTY\_SITE\_ID" = hcasa."CUST\_ACCT\_SITE\_ID"

**AND** JPR."PARTY\_ID" = hzca."CUST\_ACCOUNT\_ID"

**AND** JPRL."REGISTRATION\_TYPE\_CODE" = 'PAN'

**AND** JPR."PARTY\_TYPE\_CODE" **IN** ('THIRD\_PARTY', 'THIRD\_PARTY\_SITE')

**AND** JPR."Active" = 1

**AND** JPRL."Active" =1

) "PanNo"

**FROM** etl\_zone."T\_OEBS\_HZ\_PARTIES" hzp,

etl\_zone."T\_OEBS\_HZ\_PARTY\_SITES" hps,

etl\_zone."T\_OEBS\_HZ\_LOCATIONS" hl,

etl\_zone."T\_OEBS\_HZ\_CUST\_ACCOUNTS" hzca,

etl\_zone."T\_OEBS\_HZ\_CUST\_ACCT\_SITES\_ALL" hcasa,

etl\_zone."T\_OEBS\_HZ\_CUST\_SITE\_USES\_ALL" hcsua

**WHERE** hzca."PARTY\_ID" = hzp."PARTY\_ID"::**numeric** **AND** hps."PARTY\_ID" = hzp."PARTY\_ID"::**numeric**

**AND** hcasa."PARTY\_SITE\_ID" = hps."PARTY\_SITE\_ID"::**numeric**

**AND** hps."LOCATION\_ID" = hl."LOCATION\_ID"::**numeric**

**AND** hzca."CUST\_ACCOUNT\_ID"::**numeric** = hcasa."CUST\_ACCOUNT\_ID"

**AND** hcsua."CUST\_ACCT\_SITE\_ID" = hcasa."CUST\_ACCT\_SITE\_ID"::**numeric**

**AND** hcsua."SITE\_USE\_CODE"::**text** = 'BILL\_TO'::**text** **AND** hcasa."BILL\_TO\_FLAG"::**text** = 'P'::**text**

**AND** hzca."ATTRIBUTE1" = 'TR' **AND** hzca."ATTRIBUTE2" **IN** ('TR','RT')

**AND** hcasa."ORG\_ID" **in** (102, 103, 104)

**AND** (hzca."ATTRIBUTE7" != 'Y' **or** hzca."ATTRIBUTE7" **is** **null**)

**AND** hzp."STATUS" = 'A'

**AND** (hzp."LAST\_UPDATE\_DATE" **between** **date\_trunc**('day',p\_date::**date**) **and** **date\_trunc**('day',p\_date::**date**) + '1 day - 1 min'::**interval**

**OR** hzca."LAST\_UPDATE\_DATE" **between** **date\_trunc**('day',p\_date::**date**) **and** **date\_trunc**('day',p\_date::**date**) + '1 day - 1 min'::**interval**

**OR** hps."LAST\_UPDATE\_DATE" **between** **date\_trunc**('day',p\_date::**date**) **and** **date\_trunc**('day',p\_date::**date**) + '1 day - 1 min'::**interval**

**OR** hl."LAST\_UPDATE\_DATE" **between** **date\_trunc**('day',p\_date::**date**) **and** **date\_trunc**('day',p\_date::**date**) + '1 day - 1 min'::**interval**

**OR** hcasa."LAST\_UPDATE\_DATE" **between** **date\_trunc**('day',p\_date::**date**) **and** **date\_trunc**('day',p\_date::**date**) + '1 day - 1 min'::**interval**

**OR** hcsua."LAST\_UPDATE\_DATE" **between** **date\_trunc**('day',p\_date::**date**) **and** **date\_trunc**('day',p\_date::**date**) + '1 day - 1 min'::**interval**

**or** p\_date **is** **null**)

**and** hzp."Active" = 1

**and** hps."Active" = 1

**and** hl."Active" = 1

**and** hzca."Active" = 1

**and** hcasa."Active" = 1

**and** hcsua."Active" = 1

**and** (hzca."ACCOUNT\_NUMBER" = **ANY**(p\_account\_number) **or** p\_account\_number **is** **null**);

**END**;

**$function$**

;

----------------------------------------------------------------------------------

**CREATE** **OR** **REPLACE** **FUNCTION** etl\_zone.get\_customer\_billto\_data(p\_account\_number **text**[], p\_date **date**)

**RETURNS** **TABLE**("CUSTOMER NUMBER" **character** **varying**, "SITE\_USE\_CODE" **character** **varying**, "CUST\_ACCT\_SITE\_ID" **bigint**, "SITE\_USE\_ID" **bigint**, "CUST\_ACCOUNT\_ID" **bigint**, "Account name" **character** **varying**, "ADDRESS1" **character** **varying**, "ADDRESS2" **character** **varying**, "State" **character** **varying**, "District" **character** **varying**, "Taluka" **character** **varying**, "erpCity" **character** **varying**, "PinCode" **character** **varying**, "Mobile Number" **bigint**, "Email" **character** **varying**, "COUNTRY" **character** **varying**, latitude **text**, longitude **text**, "BILL\_TO\_FLAG" **character** **varying**)

**LANGUAGE** plpgsql

**AS** **$function$**

**BEGIN**

**RETURN** QUERY

**SELECT**

hzca."ACCOUNT\_NUMBER" **as** "CUSTOMER NUMBER",

hcsua."SITE\_USE\_CODE" ,

hcsua."CUST\_ACCT\_SITE\_ID"::**bigint** ,

hcsua."SITE\_USE\_ID" ,

hzca."CUST\_ACCOUNT\_ID" ,

hzp."PARTY\_NAME" **as** "Account name" ,

hl."ADDRESS1" **as** "ADDRESS1",

hl."ADDRESS2" **as** "ADDRESS2",

hl."STATE" **as** "State",

hl."COUNTY" **as** "District",

hl."ADDRESS\_LINES\_PHONETIC" **as** "Taluka",

hl."CITY" **as** "erpCity",

hl."POSTAL\_CODE" **as** "PinCode",

hl."ATTRIBUTE20"::**bigint** **as** "Mobile Number",

hl."ATTRIBUTE17" **as** "Email",

hl."COUNTRY" ,

**NULL** latitude,

**NULL** longitude,

hcasa."BILL\_TO\_FLAG"

**FROM** etl\_zone."T\_OEBS\_HZ\_PARTIES" hzp,

etl\_zone."T\_OEBS\_HZ\_CUST\_ACCOUNTS" hzca,

etl\_zone."T\_OEBS\_HZ\_PARTY\_SITES" hps,

etl\_zone."T\_OEBS\_HZ\_LOCATIONS" hl,

etl\_zone."T\_OEBS\_HZ\_CUST\_ACCT\_SITES\_ALL" hcasa,

etl\_zone."T\_OEBS\_HZ\_CUST\_SITE\_USES\_ALL" hcsua

**WHERE** hzp."PARTY\_ID" = hzca."PARTY\_ID"

**AND** hzp."PARTY\_ID" = hps."PARTY\_ID"

**AND** hps."PARTY\_SITE\_ID" = hcasa."PARTY\_SITE\_ID"

**AND** hps."LOCATION\_ID" = hl."LOCATION\_ID" ----

**AND** hzca."CUST\_ACCOUNT\_ID" = hcasa."CUST\_ACCOUNT\_ID"

**AND** hcsua."CUST\_ACCT\_SITE\_ID" = hcasa."CUST\_ACCT\_SITE\_ID"

**AND** hcsua."SITE\_USE\_CODE" = 'BILL\_TO'

**AND** hcasa."BILL\_TO\_FLAG" = 'P'

**AND** hcsua."STATUS" = 'A'

**AND** (hzca."ATTRIBUTE7" != 'Y' **or** hzca."ATTRIBUTE7" **is** **null**)

**AND** hzp."STATUS" = 'A'

**AND** hcsua."ORG\_ID" **in** (102, 103, 104)

**AND** (hzp."LAST\_UPDATE\_DATE" **between** **date\_trunc**('day',p\_date::**date**) **and** **date\_trunc**('day',p\_date::**date**) + '1 day - 1 min'::**interval**

**OR** hzca."LAST\_UPDATE\_DATE" **between** **date\_trunc**('day',p\_date::**date**) **and** **date\_trunc**('day',p\_date::**date**) + '1 day - 1 min'::**interval**

**OR** hps."LAST\_UPDATE\_DATE" **between** **date\_trunc**('day',p\_date::**date**) **and** **date\_trunc**('day',p\_date::**date**) + '1 day - 1 min'::**interval**

**OR** hl."LAST\_UPDATE\_DATE" **between** **date\_trunc**('day',p\_date::**date**) **and** **date\_trunc**('day',p\_date::**date**) + '1 day - 1 min'::**interval**

**OR** hcasa."LAST\_UPDATE\_DATE" **between** **date\_trunc**('day',p\_date::**date**) **and** **date\_trunc**('day',p\_date::**date**) + '1 day - 1 min'::**interval**

**OR** hcsua."LAST\_UPDATE\_DATE" **between** **date\_trunc**('day',p\_date::**date**) **and** **date\_trunc**('day',p\_date::**date**) + '1 day - 1 min'::**interval**

**or** p\_date **is** **null**)

**and** hzp."Active" = 1

**and** hps."Active" = 1

**and** hl."Active" = 1

**and** hzca."Active" = 1

**and** hcasa."Active" = 1

**and** hcsua."Active" = 1

**and** (hzca."ACCOUNT\_NUMBER" = **ANY**(p\_account\_number) **or** p\_account\_number **is** **null**);

**END**;

**$function$**

;

----------------------------------------------------------------------------

**CREATE** **OR** **REPLACE** **FUNCTION** etl\_zone.get\_customer\_shipto\_data(p\_account\_number **text**[], p\_date **date**)

**RETURNS** **TABLE**("CUSTOMER NUMBER" **character** **varying**, "SITE\_USE\_CODE" **character** **varying**, "CUST\_ACCT\_SITE\_ID" **bigint**, "SITE\_USE\_ID" **bigint**, "CUST\_ACCOUNT\_ID" **bigint**, "Account name" **character** **varying**, "ADDRESS1" **character** **varying**, "ADDRESS2" **character** **varying**, "State" **character** **varying**, "District" **character** **varying**, "Taluka" **character** **varying**, "erpCity" **character** **varying**, "PinCode" **character** **varying**, "Mobile Number" **bigint**, "Email" **character** **varying**, "COUNTRY" **character** **varying**, latitude **text**, longitude **text**, "SHIP\_TO\_FLAG" **character** **varying**)

**LANGUAGE** plpgsql

**AS** **$function$**

**BEGIN**

**RETURN** QUERY

**SELECT** t."CUSTOMER NUMBER",

t."SITE\_USE\_CODE",

t."CUST\_ACCT\_SITE\_ID",

t."SITE\_USE\_ID",

t."CUST\_ACCOUNT\_ID",

t."Account name",

t."ADDRESS1",

t."ADDRESS2",

t."State",

t."District",

t."Taluka",

t."erpCity",

t."PinCode",

t."Mobile Number",

t."Email",

t."COUNTRY",

t."latitude",

t."longitude",

t."SHIP\_TO\_FLAG"

**FROM** (

**SELECT** hzca."ACCOUNT\_NUMBER" **as** "CUSTOMER NUMBER",

hcsua."SITE\_USE\_CODE" ,

hcsua."CUST\_ACCT\_SITE\_ID"::**bigint**,

hcsua."SITE\_USE\_ID" ,

hzca."CUST\_ACCOUNT\_ID" ,

hzp."PARTY\_NAME" **as** "Account name" ,

hl."ADDRESS1" **as** "ADDRESS1",

hl."ADDRESS2" **as** "ADDRESS2",

hl."STATE" **as** "State",

hl."COUNTY" **as** "District",

hl."ADDRESS\_LINES\_PHONETIC" **as** "Taluka",

hl."CITY" **as** "erpCity",

hl."POSTAL\_CODE" **as** "PinCode",

hl."ATTRIBUTE20"::**bigint** **as** "Mobile Number",

hl."ATTRIBUTE17" **as** "Email",

hl."COUNTRY",

**NULL** latitude,

**NULL** longitude,

hcasa."SHIP\_TO\_FLAG",

**RANK**() **OVER** (**PARTITION** **BY** hzca."ACCOUNT\_NUMBER" **ORDER** **BY** hcsua."SITE\_USE\_ID") **as** "ROW\_RANK"

**FROM** etl\_zone."T\_OEBS\_HZ\_PARTIES" hzp,

etl\_zone."T\_OEBS\_HZ\_CUST\_ACCOUNTS" hzca,

etl\_zone."T\_OEBS\_HZ\_PARTY\_SITES" hps,

etl\_zone."T\_OEBS\_HZ\_LOCATIONS" hl,

etl\_zone."T\_OEBS\_HZ\_CUST\_ACCT\_SITES\_ALL" hcasa,

etl\_zone."T\_OEBS\_HZ\_CUST\_SITE\_USES\_ALL" hcsua

**WHERE** hzp."PARTY\_ID" = hzca."PARTY\_ID"

**AND** hzp."PARTY\_ID" = hps."PARTY\_ID"

**AND** hps."PARTY\_SITE\_ID" = hcasa."PARTY\_SITE\_ID"

**AND** hps."LOCATION\_ID" = hl."LOCATION\_ID"

**AND** hzca."CUST\_ACCOUNT\_ID" = hcasa."CUST\_ACCOUNT\_ID"

**AND** hcsua."CUST\_ACCT\_SITE\_ID" = hcasa."CUST\_ACCT\_SITE\_ID"

**AND** hcsua."SITE\_USE\_CODE" = 'SHIP\_TO'

**AND** hcasa."SHIP\_TO\_FLAG" = 'P'

**AND** (hzca."ATTRIBUTE7" != 'Y' **or** hzca."ATTRIBUTE7" **is** **null**)

**AND** hcsua."STATUS" = 'A'

**AND** hzp."STATUS" = 'A'

**AND** hcsua."ORG\_ID" **IN** (102, 103, 104)

**AND** (

hzp."LAST\_UPDATE\_DATE" **BETWEEN** **date\_trunc**('day', p\_date::**date**) **AND** **date\_trunc**('day', p\_date::**date**) + '1 day - 1 min'::**interval**

**OR** hzca."LAST\_UPDATE\_DATE" **BETWEEN** **date\_trunc**('day', p\_date::**date**) **AND** **date\_trunc**('day', p\_date::**date**) + '1 day - 1 min'::**interval**

**OR** hps."LAST\_UPDATE\_DATE" **BETWEEN** **date\_trunc**('day', p\_date::**date**) **AND** **date\_trunc**('day', p\_date::**date**) + '1 day - 1 min'::**interval**

**OR** hl."LAST\_UPDATE\_DATE" **BETWEEN** **date\_trunc**('day', p\_date::**date**) **AND** **date\_trunc**('day', p\_date::**date**) + '1 day - 1 min'::**interval**

**OR** hcasa."LAST\_UPDATE\_DATE" **BETWEEN** **date\_trunc**('day', p\_date::**date**) **AND** **date\_trunc**('day', p\_date::**date**) + '1 day - 1 min'::**interval**

**OR** hcsua."LAST\_UPDATE\_DATE" **BETWEEN** **date\_trunc**('day', p\_date::**date**) **AND** **date\_trunc**('day', p\_date::**date**) + '1 day - 1 min'::**interval**

**OR** p\_date **IS** **NULL**

)

**AND** hzp."Active" = 1

**AND** hps."Active" = 1

**AND** hl."Active" = 1

**AND** hzca."Active" = 1

**AND** hcasa."Active" = 1

**AND** hcsua."Active" = 1

**AND** (hzca."ACCOUNT\_NUMBER" = **ANY**(p\_account\_number) **OR** p\_account\_number **IS** **NULL**)

) t

**WHERE** "ROW\_RANK" = 1;

**END**;

**$function$**;

-------------------------------------------------------------------------

**CREATE** **OR** **REPLACE** **FUNCTION** etl\_zone.get\_customer\_billto\_shipto\_data(p\_account\_number **text**[], p\_date **date**)

**RETURNS** **TABLE**("ACCOUNT\_NUMBER" **character** **varying**, "CUST\_ACCOUNT\_ID" **bigint**, "CUSTOMER\_NAME" **character** **varying**, "CUST\_CAT" **character** **varying**, "CUST\_SUB\_CAT" **character** **varying**, "CITY" **character** **varying**, "TALUKA" **character** **varying**, "DISTRICT" **character** **varying**, "STATE" **character** **varying**, "MOBILE\_NUMBER" **bigint**, "EMAIL" **character** **varying**, "DATE\_OF\_JOINING" **timestamp** **without** **time** **zone**, "ORG\_ID" **bigint**, "BRAND" **character** **varying**, "CUSTOMER\_TYPE" **character** **varying**, "BANK\_ACCOUNT\_NO" **character** **varying**, "BANK\_NAME" **character** **varying**, "IFSC\_CODE" **character** **varying**, "GSTIN" **text**, "PAN\_NUMBER" **text**, "BILL\_TO\_ERP\_ADDRESS\_ID" **bigint**, "BILL\_TO\_NAME" **character** **varying**, "BILL\_TO\_STREET\_NAME" **character** **varying**, "BILL\_TO\_STREET\_NAME\_AND\_NO" **character** **varying**, "BILL\_TO\_STATE" **character** **varying**, "BILL\_TO\_DISTRICT" **character** **varying**, "BILL\_TO\_TALUKA" **character** **varying**, "BILL\_TO\_ERP\_CITY" **character** **varying**, "BILL\_TO\_POSTAL\_CODE" **character** **varying**, "BILL\_TO\_CELLPHONE" **bigint**, "BILL\_TO\_EMAIL" **character** **varying**, "BILL\_TO\_FLAG" **character** **varying**, "BILL\_TO\_COUNTRY" **character** **varying**, "SHIP\_TO\_ERP\_ADDRESS\_ID" **bigint**, "SHIP\_TO\_NAME" **character** **varying**, "SHIP\_TO\_STREET\_NAME" **character** **varying**, "SHIP\_TO\_STREET\_NAME\_AND\_NO" **character** **varying**, "SHIP\_TO\_STATE" **character** **varying**, "SHIP\_TO\_DISTRICT" **character** **varying**, "SHIP\_TO\_TALUKA" **character** **varying**, "SHIP\_TO\_ERP\_CITY" **character** **varying**, "SHIP\_TO\_POSTAL\_CODE" **character** **varying**, "SHIP\_TO\_CELLPHONE" **bigint**, "SHIP\_TO\_EMAIL" **character** **varying**, "SHIP\_TO\_FLAG" **character** **varying**, "SHIP\_TO\_COUNTRY" **character** **varying**)

**LANGUAGE** plpgsql

**AS** **$function$**

--DECLARE

-- C1 CURSOR FOR

-- SELECT

-- "ACCOUNT\_NUMBER",

-- "CUST\_ACCOUNT\_ID",

-- "CUSTOMER\_NAME",

-- "CUST\_CAT",

-- "CUST\_SUB\_CAT",

-- "CITY",

-- "TALUKA",

-- "DISTRICT",

-- "STATE",

-- "MOBILE\_NUMBER",

-- "EMAIL",

-- "DATE\_OF\_JOINING",

-- "ORG\_ID",

-- "BRAND",

-- "CUSTOMER\_TYPE",

-- "BANK\_ACCOUNT\_NO",

-- "BANK\_NAME",

-- "IFSC\_CODE",

-- "LAST\_UPDATE\_DATE",

-- "GSTIN",

-- "PanNo"

-- FROM etl\_zone.get\_customer\_data(ARRAY['1132','1238','1326','1357'], NULL);

**BEGIN**

-- FOR I IN C1

-- LOOP

**RETURN** QUERY

**select** cm."ACCOUNT\_NUMBER",

cm."CUST\_ACCOUNT\_ID",

cm."CUSTOMER\_NAME",

cm."CUST\_CAT",

cm."CUST\_SUB\_CAT",

cm."CITY",

cm."TALUKA",

cm."DISTRICT",

cm."STATE",

cm."MOBILE\_NUMBER",

cm."EMAIL",

cm."DATE\_OF\_JOINING",

cm."ORG\_ID",

cm."BRAND",

cm."CUSTOMER\_TYPE",

cm."BANK\_ACCOUNT\_NO",

cm."BANK\_NAME",

cm."IFSC\_CODE",

cm."GSTIN",

cm."PanNo" **as** "PAN\_NUMBER",

bt."SITE\_USE\_ID" **as** "BILL\_TO\_ERP\_ADDRESS\_ID",

bt."Account name" **as** "BILL\_TO\_NAME",

bt."ADDRESS1" **as** "BILL\_TO\_STREET\_NAME",

bt."ADDRESS2" **as** "BILL\_TO\_STREET\_NAME\_AND\_NO",

bt."State" **as** "BILL\_TO\_STATE",

bt."District" **as** "BILL\_TO\_DISTRICT",

bt."Taluka" **as** "BILL\_TO\_TALUKA",

bt."erpCity" **as** "BILL\_TO\_ERP\_CITY",

bt."PinCode" **as** "BILL\_TO\_POSTAL\_CODE",

bt."Mobile Number" **as** "BILL\_TO\_CELLPHONE",

bt."Email" **as** "BILL\_TO\_EMAIL",

bt."BILL\_TO\_FLAG",

bt."COUNTRY" **as** "BILL\_TO\_COUNTRY",

st."SITE\_USE\_ID" **as** "SHIP\_TO\_ERP\_ADDRESS\_ID",

st."Account name" **as** "SHIP\_TO\_NAME",

st."ADDRESS1" **as** "SHIP\_TO\_STREET\_NAME",

st."ADDRESS2" **as** "SHIP\_TO\_STREET\_NAME\_AND\_NO",

st."State" **as** "SHIP\_TO\_STATE",

st."District" **as** "SHIP\_TO\_DISTRICT",

st."Taluka" **as** "SHIP\_TO\_TALUKA",

st."erpCity" **as** "SHIP\_TO\_ERP\_CITY",

st."PinCode" **as** "SHIP\_TO\_POSTAL\_CODE",

st."Mobile Number" **as** "SHIP\_TO\_CELLPHONE",

st."Email" **as** "SHIP\_TO\_EMAIL",

st."SHIP\_TO\_FLAG",

st."COUNTRY" **as** "SHIP\_TO\_COUNTRY"

**FROM** etl\_zone.get\_customer\_data(p\_account\_number, p\_date) cm

-- etl\_zone.get\_customer\_data(ARRAY['1132','1238','1326','1357'], null) cm

**left** **join** etl\_zone.get\_customer\_billto\_data(p\_account\_number, p\_date) bt

-- etl\_zone.get\_customer\_billto\_data(ARRAY['1132','1238','1326','1357'],null) bt

**on** cm."CUST\_ACCOUNT\_ID" = bt."CUST\_ACCOUNT\_ID"

**left** **join** etl\_zone.get\_customer\_shipto\_data(p\_account\_number, p\_date) st

-- etl\_zone.get\_customer\_shipto\_data(ARRAY['1132','1238','1326','1357'],null) st

**on** cm."CUST\_ACCOUNT\_ID" = st."CUST\_ACCOUNT\_ID";

-- RAISE NOTICE 'ACCOUNT\_NUMBER is %', cm."ACCOUNT\_NUMBER";

-- END LOOP;

**END**;

**$function$**

;

**/\*\*\*\* TO CALL THE ABOVE FUNCTIONS: \*\*\*\*/**

**SELECT** \* **from** etl\_zone.get\_customer\_billto\_shipto\_data(**ARRAY**['1132',

'1238',

'1326',

'1357',

'1370',

'1410'

],**null**);

**SELECT** \* **from** etl\_zone.get\_customer\_billto\_shipto\_data(**null**,'2022-05-05')

* **QUERY TO GET FIRST NAME FROM FULL NAME**

**SELECT** **SPLIT\_PART**('SANJAY TEWARI JI', ' ', 1) **AS** first\_name

* **FUNCTION TO TIMEOUT AFTER GIVEN SECONDS**  
    
  **CREATE** **OR** **REPLACE** **FUNCTION** etl\_zone.my\_function\_with\_timeout(timeout\_seconds **INT**)

**RETURNS** **VOID**

**LANGUAGE** plpgsql

**AS** **$$**

**BEGIN**

**PERFORM** pg\_sleep(timeout\_seconds); -- Sleep for the specified timeout

-- Your function logic goes here

-- Use other\_parameters in your queries or operations

-- Your function result or further processing goes here

-- ...

**END**;

**$$**;

* **Get max(pid) comntaining of given table and given schema**

**CREATE** **OR** **REPLACE** **FUNCTION** semantic.get\_pid\_from\_query(p\_schema **text**, p\_table\_name **text**)

**RETURNS** **INT**

**LANGUAGE** plpgsql

**AS** **$$**

**DECLARE**

v\_pid **INT**;

**BEGIN**

v\_pid := **null**;

**SELECT** **max**(pg\_stat\_activity.pid) **INTO** **STRICT** v\_pid

**FROM** pg\_stat\_activity

**WHERE** state <> 'idle'

**AND** query **NOT** **LIKE** '% FROM pg\_stat\_activity %'

**AND** **POSITION**(p\_schema **IN** query) > 0

**AND** **POSITION**(p\_table\_name **IN** query) > 0

**AND** query **LIKE** '%select%';

**RETURN** v\_pid;

**END**;

**$$**;

**select** semantic.get\_pid\_from\_query('etl\_zone'::**text**, 'CUST\_GRP\_OUTSTANDING'::**text**);

* **FUNCTION AND TRIGGER TO AITO POPULATE ZONE IN TABLE**

**CREATE** **OR** **REPLACE** **FUNCTION** etl\_zone.populate\_NTH\_COMPLAINTS\_LIST\_zone()

**RETURNS** **trigger**

**LANGUAGE** plpgsql

**AS** **$function$**

**DECLARE**

v\_zone **varchar**(360);

**BEGIN**

-- v\_zone := NULL;

--

-- select distinct shm."ZONE" into

-- v\_zone from etl\_zone."SITE\_CONVERSION\_PREMIUM" ba,

-- (select distinct "ZONE","STATE","DISTRICT" from etl\_zone."ZONE\_MAPPING\_NEW") shm

-- WHERE UPPER(shm."STATE") = UPPER(ba."STATE")

-- and UPPER(shm."DISTRICT") = UPPER(ba."DISTRICT" )

-- and ba."LAST\_UPDATE\_DATE" = NEW."LAST\_UPDATE\_DATE"

-- and ba."ZONE" is null ;

-- raise notice 'zone is -> %', v\_zone;

--

-- if v\_zone is not null then

**UPDATE** etl\_zone."NTH\_COMPLAINTS\_LIST" ba

**SET** "ZONE" = shm."ZONE"

**from** (**select** **distinct** "ZONE","STATE","DISTRICT" **from** etl\_zone."ZONE\_MAPPING\_NEW") shm

**WHERE** **UPPER**(shm."STATE") = **UPPER**(ba."STATE")

**and** **UPPER**(shm."DISTRICT") = **UPPER**(ba."District" )

**and** ba."LAST\_UPDATE\_DATE" = **NEW**."LAST\_UPDATE\_DATE"

**and** ba."ZONE" **is** **null** ;

-- where "LAST\_UPDATE\_DATE" = NEW."LAST\_UPDATE\_DATE"

-- ;

-- else

-- raise notice 'zone is -> null';

-- end if;

**RETURN** **NEW**;

**END**;

**$function$**

;

**create** **trigger** populate\_NTH\_COMPLAINTS\_LIST\_zone\_tr **after**

**insert**

**or**

**update**

**on**

etl\_zone."NTH\_COMPLAINTS\_LIST" **for** **each** **row** **execute** **function** etl\_zone.populate\_NTH\_COMPLAINTS\_LIST\_zone();

* **To get ddl of predefined postgresql function   
    
  SELECT** pg\_get\_functiondef(**oid**) **AS** function\_ddl

**FROM** pg\_proc

**WHERE** proname = 'jsonb\_each\_text';

* **To insert values in a table from a json in a new row for each key and value pair**

**CREATE** **OR** **REPLACE** **FUNCTION** insert\_json\_data(json\_data **jsonb**)

**RETURNS** **void** **AS** **$$**

**DECLARE**

key\_text **text**;

value\_text **text**;

**BEGIN**

-- Loop through each key-value pair in the JSON object

**FOR** key\_text, value\_text **IN** **SELECT** \* **FROM** **jsonb\_each\_text**(json\_data)

**LOOP**

-- Insert data into the table

**EXECUTE** format('INSERT INTO etl\_zone."ADOPTION\_TABLE1"(%I) VALUES (%L)', key\_text, value\_text);

**END** **LOOP**;

**END**;

**$$** **LANGUAGE** plpgsql;

**SELECT** insert\_json\_data('{"Id": "079d081e-8bb4-42bc-9995-bd5f19e4d9b4", "RecordType": 20, "CreationTime": "2024-01-17T00:04:28Z"}'::**jsonb**);

* **TO INSERT DATA IN A TABLE HAVING SAME COLUMN NAMES AS OF KEY VALUES OF JSON THAT IS TO BE PROVIDED A PARAMETER FOR FUNCTION**

**CREATE** **OR** **REPLACE** **FUNCTION** semantic.insert\_json\_data(json\_data **jsonb**)

**RETURNS** **void** **AS** **$$**

**DECLARE**

-- key\_text text;

-- value\_text text;

key\_text\_in **text**;

value\_text\_in **text**;

json\_cursor **CURSOR** **FOR** **SELECT** **key** **as** key\_text, value **as** value\_text **FROM** **jsonb\_each\_text**(json\_data);

**BEGIN**

-- key\_text := NULL;

-- value\_text := NULL;

-- key\_text\_in := null;

-- value\_text\_in := null;

**FOR** i\_json **in** json\_cursor **LOOP**

-- Insert data into the table

key\_text\_in := **coalesce**(key\_text\_in,'')||',"'||i\_json.key\_text||'"';

value\_text\_in := **coalesce**(value\_text\_in,'')||','||''''||i\_json.value\_text||'''';

key\_text\_in := **NULLIF**(**TRIM**(**LEADING** ',' **FROM** key\_text\_in),'');

value\_text\_in := **NULLIF**(**TRIM**(**LEADING** ',' **FROM** value\_text\_in),'');

**raise** **notice** 'key\_text -> %', i\_json.key\_text;

**raise** **notice** 'value\_text -> %', i\_json.value\_text;

**raise** **notice** 'key\_text\_in -> %', key\_text\_in;

**raise** **notice** 'value\_text\_in -> %', value\_text\_in;

**raise** **notice** '''';

**END** **LOOP**;

**EXECUTE** 'INSERT INTO etl\_zone."ADOPTION\_TABLE1"'||'('||key\_text\_in||')'||'VALUES'||'('|| value\_text\_in||')';

**END**;

**$$** **LANGUAGE** plpgsql;

**SELECT** semantic.insert\_json\_data('{"Id": "079d081e-8bb4-42bc-9995-bd5f19e4d9b4", "RecordType": 20, "CreationTime": "2024-01-17T00:04:28Z"}'::**jsonb**);

* **DYNAMIC UPSERT IN POSTGRESQL**

**create** **table** etl\_zone."test\_emp"

(

"ID" **bigint**,

"EMP\_NAME" **varchar**(360),

"DESIGNATION" **varchar**(360),

"SALARY" **bigint**

);

**create** **unique** index test\_emp\_u1 **on** etl\_zone."test\_emp"("ID");

**select** \* **from** etl\_zone."test\_emp"

**WITH** new\_values ("ID","EMP\_NAME","DESIGNATION","SALARY" ) **as** (

**values**

(10001, 'JJ','cd',10000),

(10002, 'JK','Ex',20000),

(10003, 'JM','ab',30000)

),

upsert **as**

(

**update** etl\_zone."test\_emp" m

**set** "EMP\_NAME" = nv."EMP\_NAME"

,"DESIGNATION" = nv."DESIGNATION"

,"SALARY" = nv."SALARY"

**FROM** new\_values nv

**WHERE** m."ID" = nv."ID"

**RETURNING** m.\*

)

**INSERT** **INTO** etl\_zone."test\_emp" ("ID","EMP\_NAME","DESIGNATION","SALARY" )

**SELECT** "ID","EMP\_NAME","DESIGNATION","SALARY"

**FROM** new\_values

**WHERE** **NOT** **EXISTS** (**SELECT** 1

**FROM** upsert up

**WHERE** up."ID" = new\_values."ID");

* **TO CHECK INDEXNAME AND COLUMN COMBINATION NOT IN SAP\_SCHEMA TABLE WITH SAME NAME**

**SELECT**

tablename,

indexname,

indexdef

**FROM**

pg\_indexes

**WHERE**

schemaname = 'etl\_zone'

**and** tablename **not** **like** 'T\_OEBS%'

**and** (**select** **substring**(indexdef **from** 'USING btree (.\*)')) **not** **in**

(**SELECT**

(**select** **substring**(indexdef **from** 'USING btree (.\*)')) **as** indexdef

**FROM**

pg\_indexes

**WHERE**

schemaname = 'etl\_zone\_sap'

**and** tablename **not** **like** 'T\_OEBS%')

**and** indexname **not** **in** (**SELECT**

indexname

**FROM**

pg\_indexes

**WHERE**

schemaname = 'etl\_zone\_sap'

**and** tablename **not** **like** 'T\_OEBS%')

**ORDER** **BY**

tablename,

indexname

* **function that return id’s that have been inserted on table of given schema.**

**create** **or** **replace** **function** semantic.return\_id(p\_schema **varchar**(360),p\_table **varchar**(360),p\_from\_date **timestamp**,p\_to\_date **timestamp**)

**returns** **table** (id **bigint**)

**language** plpgsql **as** **$$**

**begin**

-- raise notice 'select "ID" from %."%";',p\_schema,p\_table;

**return** query

**execute** 'select "ID" as id from '||p\_schema||'.'||'"'||p\_table||'" where "LAST\_UPDATE\_DATE" between '||''''||p\_from\_date||''''||' and '||''''||p\_to\_date||'''';

**end**;

**$$**;

**select** semantic.return\_id('etl\_zone','CRM\_MARKET\_MAPPING\_PRICING','2024-03-07 17:30:00','2024-03-07 17:37:00')

* **To extract json array elements from a JSON**

**create** **or** **replace** **function** semantic.extracted\_json\_data(p\_json **json**)

**returns** **table** (json\_part **json**)

**LANGUAGE** plpgsql

**AS** **$function$**

**BEGIN**

**RETURN** QUERY

**select** **json\_array\_elements**(p\_json::**json**) **as** json\_part;

**END**;

**$function$**

;

* QUERY -> **regexp\_replace**('WorkSpaceName', '([a-z])([A-Z])', '\1 \2', 'g')

**RESULT** Work Space Name

* QUERY -> **select** substr('GD (godown inventory)',1,**position**(' ' **in** 'GD (godown inventory)' ) - 2)

**RESULT** -> G

* **QUERY FOR LOCATION MAPPING FROM SAP**

case

when trim("LOCATION") in ('SIDING SALE','TRANSIPMENT','DIVERSION',

'FRESH GD','FRESH','ST ON WHEEL','Fresh (RD)','GD-RK SALE') then 'ACCEPT'

when trim("LOCATION") in ('CUT & TORN','C & T(RL)') then 'CUT'

when trim("LOCATION") in ('DAMAGE') then 'DAMAGE'

when trim("LOCATION") in ('SHORTAGE') then 'SHORTAGE'

else trim("LOCATION")

**end as "LOCATION"**

* **convert timestamp value with / and AM or PM to timestamp**

TO\_TIMESTAMP('16/05/2024 11:59:00 PM', 'DD/MM/YYYY HH:MI:SS AM')

* **VACUUM**

In PostgreSQL, VACUUM is a maintenance operation used to reclaim storage occupied by dead tuples (deleted or obsolete rows). When rows are updated or deleted, the space they occupy is not immediately reclaimed for reuse; instead, it is marked as dead. Over time, these dead tuples can accumulate and waste storage, leading to bloated tables and indexes.

The VACUUM operation helps to clean up this dead space and prevent database bloat. Here are the primary purposes of VACUUM:s

1. **Reclaim Space**: Reclaims storage occupied by dead tuples, making it available for future use.
2. **Update Statistics**: Updates statistics used by the query planner to ensure efficient query execution.
3. **Prevent Transaction ID Wraparound**: In PostgreSQL, transaction IDs are finite and can wrap around. Regular VACUUM operations help prevent this by marking transactions as complete.

**Types of VACUUM:**

1. **VACUUM**: This command cleans up dead tuples but does not lock the table, allowing normal operations to continue. Example:

sql

Copy code

VACUUM;

1. **VACUUM FULL**: This command locks the table and performs a more thorough cleanup by rewriting the entire table, compacting it to remove all dead space. This can be more time-consuming and can block other operations on the table while it runs. Example:

sql

Copy code

VACUUM FULL;

1. **AUTO VACUUM**: PostgreSQL also has an autovacuum daemon that automatically performs VACUUM operations based on certain thresholds to help maintain database health without manual intervention.

**Example Usage:**

sql

Copy code

-- Basic VACUUM

VACUUM;

-- VACUUM specific table

VACUUM my\_table;

-- VACUUM FULL specific table

VACUUM FULL my\_table;

Regularly running VACUUM (or relying on autovacuum) is essential for maintaining database performance and preventing storage issues.

* **DIFFERENCE BETWEEN VARCHAR and NVARCHAR**

Varchar stores Non-unicode or English character data types, and it can contain a maximum of 8000 characters. It only supports ASCII values. Nvarchar stores Unicode or Non-English character data types, and it can contain a maximum of 4000 characters. It supports ASCII values as well as special characters.

* **TO FETCH SECOND ELEMENT OF ARRAY**   
    
  SELECT '{PHP Trainee, Node JS Trainee, PowerBi Trainee}'::TEXT[] [2] AS second\_element;
* **WHAT ARE COAGULATIVE FUNCTIONS**

**coagulative** functions are used to aggregate or combine values from multiple rows into a single result. FOR EX :- sum(), avg(), string\_agg(),array\_agg(), count(), max(), min() etc.

* **WHAT ARE NON COAGUALTIVE FUNCTIONS**

**Non-coagulative** functions are those that do not aggregate or combine multiple rows into a single result. FOR EX:- LENGTH(), UPPER(), LOWER(), SUBSTRING(),CEIL(),FLOOR(), NOW(), DATE\_TRUNC() etc.

* **SELECT UNNEST(ARRAY[1, 2, 3]); gives result in 3 rows**

ANS:- 1

2

3

* **1 D ARRAY QUERY AND RESULTS :**

select ARRAY\_POSITIONS(ARRAY[1,2,3,1,3,2,3],3) **Result** - 3,5,7

select ARRAY[1,2,3] || ARRAY[4,5,6] **RESULT** - 1,2,3,4,5,6

select (ARRAY[1,2,3,1,3,2,3]::text[])[4] **RESULT** - 1

select ARRAY\_POSITION(ARRAY[1,2,3,1,3,2,3],3) **Result** – 3

**QUES-> give rank to scores of following table**

|  |  |
| --- | --- |
| 1 | 3.5 |
| 2 | 3.65 |
| 3 | 4 |
| 4 | 3.85 |
| 5 | 4 |
| 6 | 3.65 |

**Solution**-> select score,DENSE\_RANK() over(order by score desc) as rank from Scores;

With output as

|  |  |
| --- | --- |
| 4 | 1 |
| 4 | 1 |
| 3.85 | 2 |
| 3.65 | 3 |
| 3.65 | 3 |
| 3.5 | 4 |

we can use RANK() over (order by score desc) as RANK but output will be

|  |  |
| --- | --- |
| 4 | 1 |
| 4 | 1 |
| 3.85 | 3 |
| 3.65 | 4 |
| 3.65 | 4 |
| 3.5 | 6 |

* **CREATE A TABLE WHERE LEVELS OF CATEGORY CAN INCREASE DYNAMICALLY** :

CREATE TABLE categories (

id SERIAL PRIMARY KEY,

name VARCHAR(255) NOT NULL,

parent\_id INT REFERENCES categories(id) ON DELETE CASCADE

);

* id: The unique identifier for each category.
* name: The name of the category.
* parent\_id: References another category, indicating that it is a subcategory of that category. If parent\_id is NULL, then the category is a top-level category.

**Example Data**

INSERT INTO categories (name, parent\_id) VALUES

('Electronics', NULL), -- Top-level category

('Computers', 1), -- Subcategory of Electronics

('Laptops', 2), -- Subcategory of Computers

('Smartphones', 1), -- Subcategory of Electronics

('Gaming Laptops', 3); -- Sub-subcategory of Laptops

**Querying the Hierarchy**

To query the entire hierarchy, you can use a recursive common table expression (CTE):

WITH RECURSIVE category\_hierarchy AS (

SELECT id, name, parent\_id, 1 AS level

FROM categories

WHERE parent\_id IS NULL

UNION ALL

SELECT c.id, c.name, c.parent\_id, ch.level + 1

FROM categories c

INNER JOIN category\_hierarchy ch ON c.parent\_id = ch.id

)

SELECT \* FROM category\_hierarchy ORDER BY level, id;

**Write a query to find number of A’s in SANJAY (IMP)**  
select count(\*) from (SELECT UNNEST(**regexp\_split\_to\_array**('SANJAY', '')) AS char\_array)b where char\_array = 'A';

**Write a query to convert string to array and print each element of array.** SELECT UNNEST(**string\_to\_array**('apple banana cherry', ' ')) AS fruits;

**-----usually while inserting data from one table to another, if there any foreign key present in that table, we first drop the foreign key and then the query will run for insert the data but there is a trick here we use a command, OVERRIDING SYSTEM VALUE to directly insert data without any taking care of foreign key or primary key**

insert into etl\_zone\_sap."FD\_FIXED\_COST\_COMPUTATIONS" **OVERRIDING SYSTEM VALUE**

select \* from etl\_zone."FD\_FIXED\_COST\_COMPUTATIONS"

Logs table:

+----+-----+

| id | num |

+----+-----+

| 1 | 1 |

| 2 | 1 |

| 3 | 1 |

| 4 | 2 |

| 5 | 1 |

| 6 | 2 |

| 7 | 2 |

+----+-----+

**Output:**

+-----------------+

| ConsecutiveNums |

+-----------------+

| 1 |

+-----------------+

**Explanation:** 1 is the only number that appears consecutively for at least three times.

select distinct num as ConsecutiveNums from (select num,lead(num,1) over() num1,lead(num,2) over() num2 from logs)a where num = num1 and num1 = num2

**Input:**

Seat table:

+----+---------+

| id | student |

+----+---------+

| 1 | Abbot |

| 2 | Doris |

| 3 | Emerson |

| 4 | Green |

| 5 | Jeames |

+----+---------+

**Output:**

+----+---------+

| id | student |

+----+---------+

| 1 | Doris |

| 2 | Abbot |

| 3 | Green |

| 4 | Emerson |

| 5 | Jeames |

+----+---------+

**Explanation:**

Note that if the number of students is odd, there is no need to change the last one's seat.

**with a as (select \*,lead(student) over(order by id) as next,**

**lag(student) over(order by id) as pre from Seat)**

**select id, ifnull(if(id%2=0,pre,next),student) as student**

**from a**