# Abbreviations

PS: PostgresSql

# Installation

**Install PostGreSql from EDB and not BigSql. The BigSql installers have problems installing the extensions**. Plv8 will not be working with BigSql installations.

On 15-07-2018 I installed plv8 version 2.3.2 (April 13 release) successfully. Previously installed version was 1.4.10 which misses many new JavaScript features.

The new version I installed from xtuple.

Function overloading is available in PS

## Schemas

PS has analogy with folder structure which is called schema. Different users can have access to different schemas. You can organize your entire database objects in different schemas which are like folders. You need to give database object names starting with schema such as schemaname.tablename. If you do not give this path and just write table name then the schema search path is used. This contains the search path to follow for all schema if schema name is not available in object name. ‘SHOW search\_path’ command shows the search path. When you create a database a default schema as public is auto created. Nesting is not allowed in schema. Different schemas can have objects with same name.

A [schema](https://en.wikipedia.org/wiki/Database_schema) holds all objects (with the exception of roles and tablespaces). Schemas effectively act like namespaces, allowing objects of the same name to co-exist in the same database. By default, newly created databases have a schema called "public", but any additional schemas can be added, and the public schema isn't mandatory.

A "search\_path" setting determines the order in which PostgreSQL checks schemas for unqualified objects (those without a prefixed schema). By default, it is set to "$user, public" ($user refers to the currently connected database user). This default can be set on a database or role level, but as it is a session parameter, it can be freely changed (even multiple times) during a client session, affecting that session only.

Non-existent schemas listed in search\_path are silently skipped during objects lookup.

New objects are created in whichever valid schema (one that presently exists) appears first in the search\_path.Schema.

### Inheritance**[**[**edit**](https://en.wikipedia.org/w/index.php?title=PostgreSQL&action=edit&section=11)**]**

Tables can be set to inherit their characteristics from a "parent" table. Data in child tables will appear to exist in the parent tables, unless data is selected from the parent table using the ONLY keyword, i.e. SELECT \* FROM ONLY parent\_table;. Adding a column in the parent table will cause that column to appear in the child table.

## Backup / Restore

PGAdmin / SqlStudio tool does a backup of your database on right click. On restore you have to manually create a database then right click and select a backup file created. This will restore the data in new database. Restore does not create a new database by default. To restore the file from other sources this should be in .tar format. Restire can also be done from sql file. You can also run pgsql from command line.

>pgsql -d dbName -f filename -U username

## Json JsonB and HStore data types

HStore is purely key store. Json is simply text type with Json validation. Its retains the beautification. JsonB allows complex search inside data. If you do index of type gin and gist indexes then the keys and values of hstore and JsonB are also indexed and they can participate in searching. Use Json data type for logging purpose with no search processing. JsonB is most flexible. Wherever possible use jsonB over HStore data type.

## Array columns like int[], json[], jsonb[]

Array columns in ps can be appended as:

update test set arrayColumn = arrayColumn || '{"b":2}'::json

where id = 9;

The || operator is append operator and :: operator is cast operator. The text needs to be casted as json. The json properties need to be within double quotes (“ “); Append to null column also works.

### Add V8/ Javascript to Postgresql

* Download PL/V8 binaries for the version of PostGreSql
* Copy the binaries of PL/V8 in corresponding folders of PostGreSql installation. Say bin->bin, lib->postgresql/lib, share/extension -> C:\Program Files\PostgreSQL\9.6\share\extension.
* Need to restart PostgreSQL service in windows.
* In psql command create extension plv8;

## Recursive query

08/10/2019

With recursive cte as (

*finite query*

union

query from cte with join to finite query

) select \* from cte;

If there is a table accM having columns as id, accCode, accName, parentId, isLedger

Then the recursive cte which gives the ledger accounts along with their all parents is:

with recursive cte as (

select "id", "accCode", "parentId", "isLedger" from "AccM"

where "isLedger" = true

union

select a.id, a."accCode", a."parentId" , a."isLedger"

from "AccM" a join cte on

cte."parentId" = a.id

) select \* from cte order by id;

Method for joining cte with table is as per following steps:

1. Create the finite query. Rows from finite query are returned as cte. Here the finite query is ‘select … from AccM where isLedger = true;’

At present the cte has only the above rows. Now join this cte with the table on cte.parentId = AccM.id. Note that cte.id cannot be joined with AccM.parentId. Only cte.parentId can be joined with AccM.Id. That is:

select a.id, a."accCode", a."parentId" , a."isLedger"

from "AccM" a join cte on

cte."parentId" = a.id

1. Union the above two queries
2. Now select the results from cte.

Complete query is as above.

04/01/2017

Step 1: create cte1

Step 2: Inside cte1 create an exit point from main table with finite number of rows. *Like select id, label, parent\_id from mainTable where parent\_id is null*.

Step 3: Create union / union all. Join cte1 and main with cte1.id = main.parent\_id, **select rows from main and not from cte1.**

Explain: The output rows will be at first the finite number of rows corresponding to exit point in cte1as in step 1. Then one after another rows from main is delivered where main.parent\_id = cte1.id.

Recursive query is done through cte. The cte has following artifacts

1. A non-recursive query: This gives finite set of rows.
2. Union all
3. A recursive query: This is inner joins with cte itself

Example:

Create an emp table with parent\_id and data

CREATE TABLE emp (

emp\_id serial PRIMARY KEY,

name text NOT NULL,

parent\_id INT

);

insert into emp(name,parent\_id) values('name1',null);

insert into emp(name,parent\_id) values('name2',1);

insert into emp(name,parent\_id) values('name3',2);

insert into emp(name,parent\_id) values('name4',3);

insert into emp(name,parent\_id) values('name5',4);

insert into emp(name,parent\_id) values('name6',5);

insert into emp(name,parent\_id) values('name7',6);

insert into emp(name,parent\_id) values('name8',1);

insert into emp(name,parent\_id) values('name9',1);

insert into emp(name,parent\_id) values('name10',1);

Now get all subordinates of emp with name1.

1. Non-recursive query

select emp\_id, name, parent\_id from emp where emp\_id = 1

1. Recursive query

Select emp\_id, name, parent\_id

from emp e inner join cte1 c

on e.parent\_id = c.emp\_id

final query

with recursive cte1 as (

select emp\_id, name, parent\_id from emp where emp\_id = 1

union all

select e.emp\_id, e.name, e.parent\_id from emp e

inner join cte1 c

on e.parent\_id = c.emp\_id

) select \* from cte1 order by emp\_id;

## Autoincrement

You use serial/ small serial/ big serial as data type of a table column.

## Table Inheritance

PS allows multiple inheritance in tables. On PS table can inherit from one or many tables. In that case columns of parent table is implied in child table. Generally one schema contains the master tables and functions. Tables in other schemas inherit from the first schema. This way if you change tables in first schema you need not make changes in other schemas because the tables are inherited.

## Features

CTE, Arrays, Json, JsonB, Lateral Join, Pivot, Limit and Offset, Full text search, Where clause in indexes,

## Stored procedure or functions

Functions can be programmed in many languages. In the heading of function the programming language is specified and body of function is provided as string. The body string of function is between $function$ and $function$ which is string delimiter. This is dollar quoting. Anything can be there between $ and $. You can write pure SQl in functions. If you require if…else, do while loop variables etc then you need PL/ pgSQL.

## Anonymous code block / functions

$$ BEGIN RAISE NOTICE 'current time: %', now(); END; $$ **LANGUAGE 'plpgsql';**

The functions can be written in many languages and the language is mentioned in body of function. Anonymous code blocks can be used to create a block of code in form of function which executes immediately. We cannot use if – else directly with sql but inside code block it can be used.

do $$

begin

if exists(select 0 from shopping\_cart where user\_id = 1 and product\_id = 6003) then

update shopping\_cart set qty = qty + 2 where user\_id = 1 and product\_id = 6003;

end if;

end $$

You can even write anonymous functions in PL/V8

do $$

var o = {};

var x = 0;

for(var i=0; i<100; i++){

x=x+1;

}

$$ LANGUAGE plv8

# Temp tables in functions

In PostGresql there are only functions and not true stored procedures. I found that it is good to program in Pl/pgsql for stored procedures instead of pure sql. In pure sql you may not be able to use temp tables. Even if you create temp table in pure sql language SP, at run time it will not find the temp table. The reason is in sql stored procedure all the commands are executed together and not one after another. So if you create temp table in first statement and then reference it in second statement the second statement will not find the temp table because second statement is executed together with the first statement. On contrary in PL/PGsql stored procedure all statements are executed serially hence you will get the temp table. Working example:

create or replace function getcats()

returns TABLE(id int, label text, parent\_id int)AS

$$

BEGIN

create temporary table temp1 (id int not null, label text, parent\_id int) on commit drop;

insert into temp1

with recursive

cte1 as (

SELECT c.id, c.label, c.parent\_id

FROM category c

WHERE to\_tsvector('english', c.label) @@ to\_tsquery('english', 'lamb')

),

cte2 as(

select c.id, c.label,

CASE

WHEN c.parent\_id in(select c.id from cte1)

then c.parent\_id

else null::int

END as parent\_id

from cte1 c

) ,

cte3 as(

select c.id, c.label, c.parent\_id from cte2 c

union all

select c1.id, c1.label, c1.parent\_id

from category c1 inner join cte3 c3

on c1.parent\_id = c3.id

) select \* from cte3 c order by c.id;

update temp1 t set parent\_id = -1 where t.parent\_id is null;

insert into temp1(id,label,parent\_id) values (-1,'Categories',null);

return Query (

select \* from temp1 t order by t.id

);

END

$$

language 'plpgsql';

select \* from getcats()

# Useful commands

* Delete all tables

DROP SCHEMA public CASCADE;

CREATE SCHEMA public;

* PGAdmin is very slow. To import sql dump in database I find simple command line fastest.
  + Change to bin folder of postgresql
  + Create a database chinook
  + psql -d Chinook -u postgres -f J:\Learning\PostGreSql\Chinook\chinook.sql

-d for database, -f for file, -u for user

Change port of PostGresql

In C:\Program Files\PostgreSQL\10\data postgresql.conf change port = 5432. Restart the PostGresql service

# Full text search

## Document

For searches in Postgres a document is a text field in a row or it can be a combination of several fields in multiple tables. Example of document is:

SELECT title || ' ' || author || ' ' || abstract || ' ' || body AS document

FROM messages

WHERE mid = 12;

SELECT m.title || ' ' || m.author || ' ' || m.abstract || ' ' || d.body AS document

FROM messages m, docs d

WHERE mid = did AND mid = 12;

Document is preprocessed in tsvector format which has searching and ranking information. Tsvector is compact form of document.

If a tsvector matches tsquery then @@ operator returns true. Which comes first between tsvector and tsquery is immaterial.

SELECT 'a fat cat sat on a mat and ate a fat rat'::tsvector @@ 'cat & rat'::tsquery;

?column?

----------

true

SELECT 'fat & cow'::tsquery @@ 'a fat cat sat on a mat and ate a fat rat'::tsvector;

?column?

----------

false

normalization is process of converting text into lexemes. “eat” is lexeme for “ate”, “eats”, ”eating” etc. Normalization removes stop words like is, am, at, are, up etc. Actual process of search involves normalization of text which is done through to\_tsvector() and to\_tsquery() functions. Thus this functions 1) Converts text to lexemes, 2) removes stop words, 3) outputs lexemes with their respective positions.

SELECT to\_tsvector('fat cats ate fat rats') @@ to\_tsquery('fat & rat');

?column?

----------

true

SELECT to\_tsvector('english', 'a fat cat sat on a mat - it ate a fat rats');

to\_tsvector

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

'ate':9 'cat':3 'fat':2,11 'mat':7 'rat':12 'sat':4

To\_tsvector(null) returns null. So use coalesce function as follows. Setweight() function is also used to set weight of different columns. Weights can be used in ranking. One column of a row can have higher weight than other column.

UPDATE tt SET ti =

setweight(to\_tsvector(coalesce(title,'')), 'A') ||

setweight(to\_tsvector(coalesce(keyword,'')), 'B') ||

setweight(to\_tsvector(coalesce(abstract,'')), 'C') ||

setweight(to\_tsvector(coalesce(body,'')), 'D');

Datatype tsvector is provided to store preprocessed documents and datatype tsquery is provided to represent processed query. For text search purpose a document should be compacted to tsvector.

GIN index can be created to speed up search.

CREATE INDEX pgweb\_idx ON pgweb USING GIN (to\_tsvector('english', body));

CREATE INDEX pgweb\_idx ON pgweb USING GIN (to\_tsvector('english', title || ' ' || body));

Following sql uses a new column for storage of tsvector and makes use of coalesce function to replace null values with ‘’ so that entire expression does not become null failing search.

ALTER TABLE pgweb ADD COLUMN textsearchable\_index\_col tsvector;

UPDATE pgweb SET textsearchable\_index\_col =

to\_tsvector('english', coalesce(title,'') || ' ' || coalesce(body,''));

CREATE INDEX textsearch\_idx ON pgweb USING GIN (textsearchable\_index\_col);

Here is the search command for above

SELECT title

FROM pgweb

WHERE textsearchable\_index\_col @@ to\_tsquery('create & table')

ORDER BY last\_mod\_date DESC

LIMIT 10;

For separate column approach you must use trigger to update the column to reflect the latest changes.

To\_tsquery() does the same to query text as the to\_tsvector() does for fields. There is also plainto\_tsquery() function which separates each word with & operator.

SELECT plainto\_tsquery('english', 'The Fat Rats');

plainto\_tsquery

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

'fat' & 'rat'

Trigger functions are available to set the tsvector field.

CREATE TABLE messages (

title text,

body text,

tsv tsvector

);

CREATE TRIGGER tsvectorupdate BEFORE INSERT OR UPDATE

ON messages FOR EACH ROW EXECUTE PROCEDURE

tsvector\_update\_trigger(tsv, 'pg\_catalog.english', title, body);

INSERT INTO messages VALUES('title here', 'the body text is here');

SELECT \* FROM messages;

title | body | tsv

------------+-----------------------+----------------------------

title here | the body text is here | 'bodi':4 'text':5 'titl':1

SELECT title, body FROM messages WHERE tsv @@ to\_tsquery('title & body');

title | body

------------+-----------------------

title here | the body text is here

Having created this trigger, any change in title or body will automatically be reflected into tsv, without the application having to worry about it.

Custom trigger with setting of weights to certain columns is below.

CREATE FUNCTION messages\_trigger() RETURNS trigger AS $$

begin

new.tsv :=

setweight(to\_tsvector('pg\_catalog.english', coalesce(new.title,'')), 'A') ||

setweight(to\_tsvector('pg\_catalog.english', coalesce(new.body,'')), 'D');

return new;

end

$$ LANGUAGE plpgsql;

CREATE TRIGGER tsvectorupdate BEFORE INSERT OR UPDATE

ON messages FOR EACH ROW EXECUTE PROCEDURE messages\_trigger();