TRABAJO PRÁCTICO N°3 – Sistema de Soporte para la Toma de Decisiones

Acosta, Luis Matías.
Mazzaglia, Ian.
Murua, Federico.

1. Diseño Lógico e Implementación de Data Warehouse

Vinos

dimensiones:

- Cliente
- Tiempo
- Vino
- Clase -> Vino

medidas:

- Cliente (dirección, fecha cumpleaños, genero)
- Vino (tipo, temporada, precio botella, precio caja, clase)
- Clase (región)
- Tiempo (fecha, año)
- Orden (cliente, vino, fecha)

hechos:

- Orden

Inmobiliaria

dimensiones:

- Inmueble -> Dueño
- Agente -> Agenda
- Cliente
- Visita
- Venta
- Alquiler

medidas:

- Inmueble: categoría, área, ciudad, provincia, habitaciones, metros
- Dueño: -
- Agente: -
- Agenda: hora
- Cliente: nombre, apellido
- Visita: duración
- Venta: precio acordado, estado
- Alquiler: precio, estado, tiempo

hechos:

- fecha

Dentro del repositorio en la carpeta Punto1-Vineria se encuentran las querys solicitadas y el backup de la base de datos, de igual forma en la carpeta Punto1-Inmobiliaria.

2. Configurar PostgreSQL para funcionar como DW:

a) Configurar *Write-ahead logging* en un disco distinto.

Para tener los WAL logs en un disco distinto se deben seguir los siguientes pasos:

1. Parar el servicio de postgres.
   “systemctl stop postgres-<versión>”
2. Montar el nuevo disco con los permisos adecuados.
   a. Creamos la carpeta donde montar el disco.
      “mkdir /media/temp”
   b. Montamos el disco
      “mount /dev/sdb1 /media/temp/”
3. Copiamos los archivos del directorio $PGDATA/pg_xlog al nuevo disco
   “cp -rf /directorio/de/datos/postgres/pg_xlog/* /media/temp/pg_xlog/”
4. Creamos un enlace simbólico desde la ubicación original al nuevo directorio.
   “ln -s /media/temp/pg_xlog/ /directorio/de/datos/postgres/pg_xlog”
5. Levantamos nuevamente el servicio de postgres.
   “systemctl start postgres-<versión>”

b) Aumentar el límite de conexiones.

Datos reflejados en el archivo .conf adjunto.

c) Aumentar límites de memória. (*shared_buffers, work_mem, temp_buffers,etc.*)

Datos reflejados en el archivo .conf adjunto.

d) Qué es AUTOVACUUM? Es recomendable tener esta característica en nuestro DW? Cómo se configura en PostgreSQL?

Vacuum es el proceso en el cual se eliminan definitivamente tuplas marcadas para borrar y hay un reorganización de datos a nivel físico. La opción AUTOVACUUM permite la realización de manera automática del proceso de VACUUM. No es recomendable tener activada la función de AUTOVACUUM en nuestro DW ya que pocas veces se realizaran eliminaciones en el mismo y este proceso degrada la performance del servidor.
Dentro del archivo postgresql.conf existe una apartado para configurar AUTOVACUUM, los parámetros a configurar son los siguientes:

- **autovacuum:** Es un booleano que determina si correra el proceso demonio de autovacuum, nosotros lo pondremos en “off”.
- **log_autovacuum_min_duration:** Un entero que especifica la cantidad de milisegundos que debe alcanzar una acción del AUTOVACUUM para que

quede logeada, si el valor es 0 todas las acciones del demonio se escribirán en un log, si el valor es -1 ninguna acción se escribirá.

- **autovacuum_max_workers:** Un entero que especifica la cantidad de workers que pueden estar corriendo como parte del proceso de autovacuum.
- **autovacuum_naptime:** Un entero que especifica en segundos el retraso mínimo entre ejecuciones del proceso autovacuum en cualquier base de datos.
- **autovacuum_vacuum_threshold:** Un entero que especifica el mínimo número de tuplas actualizadas o borradas que se necesitan para activar el proceso de AUTOVACUUM en una tabla.
- **autovacuum_analyze_threshold:** Un entero que especifica el mínimo número de tuplas actualizada o borradas que se necesita para activar el proceso de ANALYZE en una tabla.
- **autovacuum_vacuum_sacle_factor:** Un número de punto flotante que especifica una fracción del tamaño de la tabla para agregar al *"autovacuum_vacuum_threshold"* al momento de decidir si lanzar o no el proceso de autovacuum.
- **autovacuum_analyze_scale_factor:** Un número de punto flotante que especifica una fracción del tamaño de la tabla para agregar al *"autovacuum_analyze_threshold"* al momento de decidir si lanzar o no el proceso de analyze.
- **autovacuum_freeze_max_age:** Un entero que especifica la antigüedad máxima que el campo "pg_class.relfrozenxid" de una tabla puede alcanzar antes de que se fuerce a una operación de VACUUM para evitar que el valor de los ID de transacciones llegue a su límite.
- **autovacuum_vacuum_cost_limit:** Un entero que especifica el valor máximo de costo que puede ser usado en operaciones de AUTOVACUUM, si se supera este valor el proceso pasa a un estado de "dormido".
- **autovacuum_vacuum_cost_delay:** Un entero que especifica el valor de retraso en milisegundos que el proceso de AUTOVACUUM se suspenderá cunado haya excedido el valor de "*autovacuum_vacuum_cost_limit",*

e) Qué es un *tablespace*? Cuáles son los beneficios de utilizarlos? Crear dos dbs y colocarlas en discos diferentes.

Los tablespaces son referencias a ubicaciones físicas del almacenamiento de bases de datos y/o de los objetos que éste contiene.

Es recomendable utilizar tablespaces cuando se quiere especificar ubicaciones alternativas para determinadas bases de datos o tablas, como cuando queremos que ciertas tablas estén en otros discos distintos a los que se encuentran, por ejemplo, para ubicar los tablespaces con datos a los que rara vez se acceden en dispositivos de almacenamiento más lentos y a los tablespaces con datos a los que se accede constantemente, en medios de almacenamiento mucho más rápidos.

En cuanto a DW, es recomendable tener tablespaces para paralelizar el acceso a las tablas, teniendo la tabla de hechos en un tablespace diferentes que las tablas de dimensiones.

Para demostrar el funcionamiento de tablespaces y al no tener la disponibilidad de dos discos se verá el funcionamiento con dos directorios distintos dentro del mismo disco:

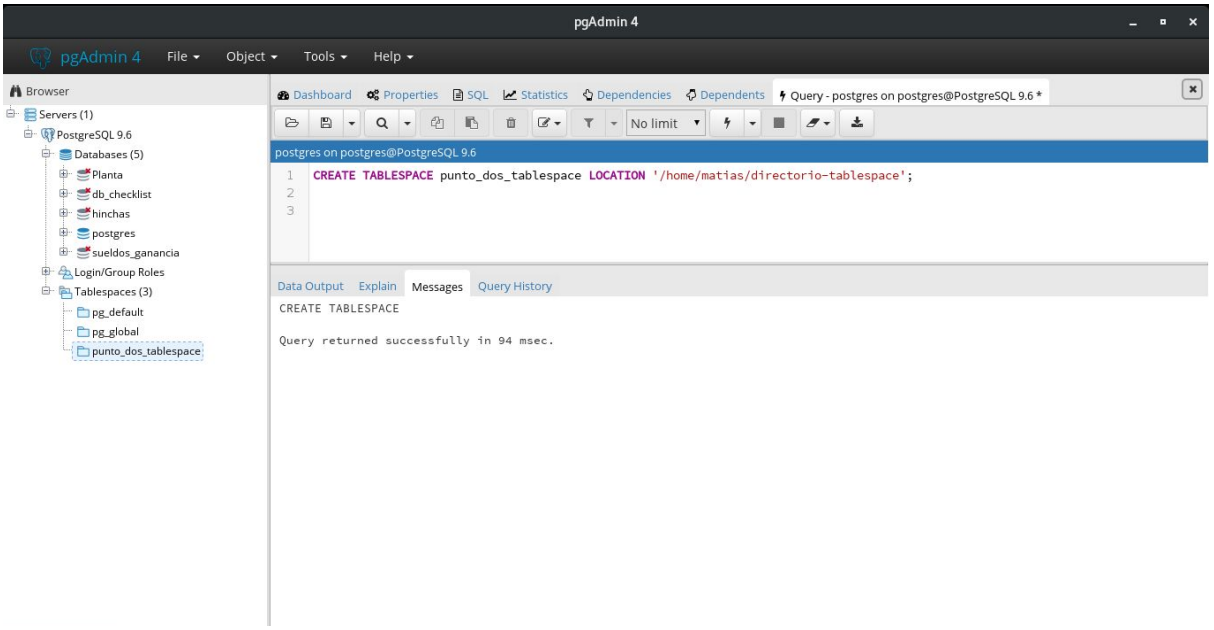
Primero creamos un directorio donde crear nuestro nuevo **tablespace**:

mkdir ~/directorio-tablespace

Luego cambiamos el propietario de la carpeta creada:

chown -R postgres:postgres ~/directorio-tablespace

Luego desde PgAdmin creamos el nuevo tablespace en la carpeta recién creada:

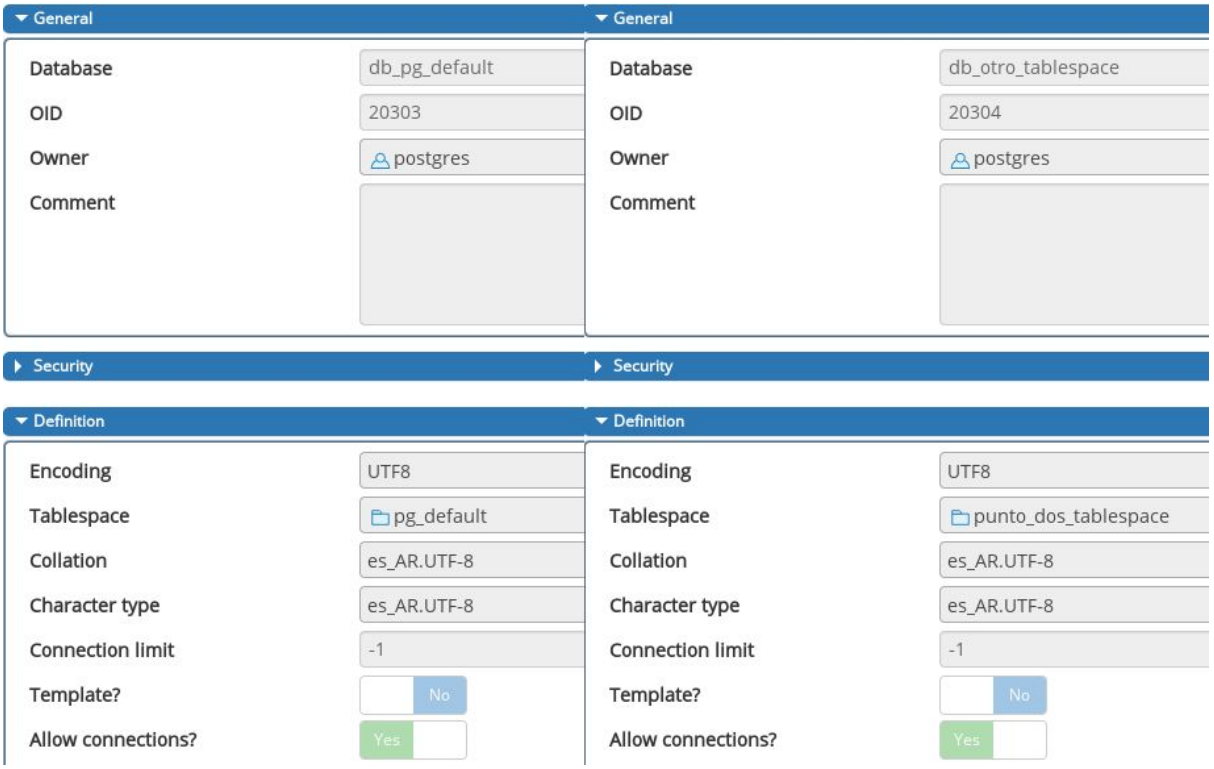


Luego creamos una base de datos llamada db_pg_default en el tablespace por default y otra base de datos llamada db_otro_tablespace en el tablesapace creado recientemente:

```
CREATE DATABASE db_pg_default
    WITH
    OWNER = postgres
    ENCODING = 'UTF8'
    LC_COLLATE = 'es_AR.UTF-8'
    LC_CTYPE = 'es_AR.UTF-8'
    TABLESPACE = pg_default
    CONNECTION LIMIT = -1;

CREATE DATABASE db_otro_tablespace
    WITH
    OWNER = postgres
    ENCODING = 'UTF8'
    LC_COLLATE = 'es_AR.UTF-8'
    LC_CTYPE = 'es_AR.UTF-8'
    TABLESPACE = punto_dos_tablespace
    CONNECTION LIMIT = -1;
```

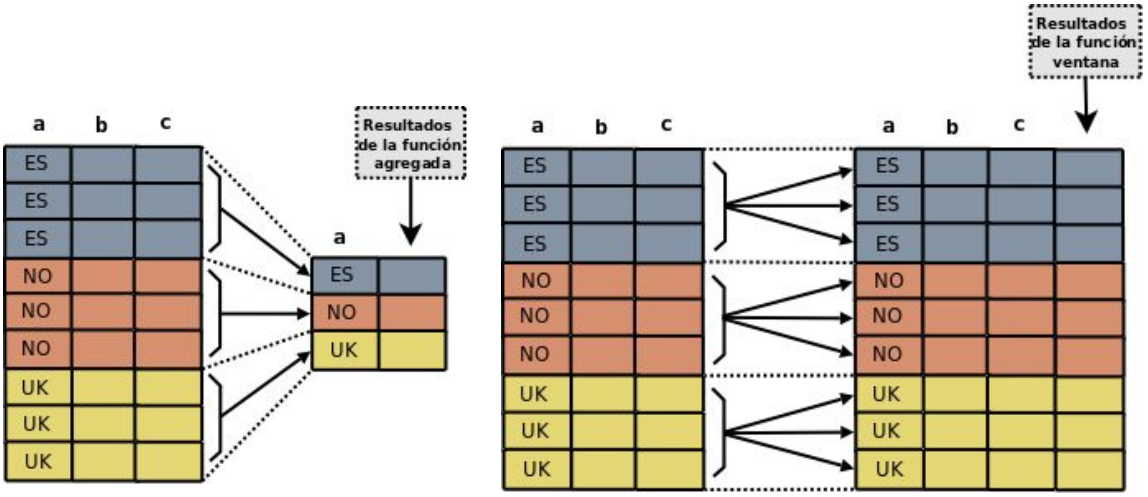
Como podemos ver en la siguiente captura el PgAdmin muestra que cada base de datos se creó efectivamente en su tablespace correspondiente.



f) Qué es una *Windowing function*? Cómo se implementan en SQL? Aplicar un ejemplo en la db creada anteriormente.

Una función ventana o Window Function es un tipo de función que realiza un calcula sobre un conjunto de filas pertenecientes a una tabla que de alguna manera están relacionadas con la final actual.

Este tipo de funciones es comparable al tipo de cálculo que se puede hacer con una función agregada, pero a diferencia de las funciones agregadas, el uso de una función ventana no hace que las filas se agrupen en una sola fila de salida: las filas conservan sus identidades separadas.



En SQL se define una función ventana utilizando la cláusula OVER después de una función de agregación. Una ventana está formada por una partición (PARTITION) y un marco (FRAME), si no utilizamos la cláusula PARTITION BY, todas las filas se consideran dentro de la misma ventana.
La ventana se puede definir luego de la cláusula OVER como en el siguiente ejemplo:

“SELECT empid, departamento, salario, edad, avg(salario) OVER **(PARTITION BY departamento)** AS salario_medio FROM empleado;”

o se puede utilizar la clausula WINDOW para definir la ventana de manera separada, como se muestra a continuación:

“SELECT empid, departamento, salario, edad, avg(salario) OVER ventana_departamento AS salario medio FROM empleado **WINDOW ventana_departamento AS (PARTITION BY departamento);**”

NOTA: Para ver más ejemplos restaure la BD que se encuentra en el repositorio con el nombre punto2.backup y abra el archivo funcion_ventanta.sql

g) Es necesaria un política de caducidad de los datos en un DW? Justifique.

Creo que es necesaria una política de caducidad de los datos en un DW ya que al ser una base de datos de soporte para la toma de decisiones llega un momento que existen datos dentro del almacén que dada su antigüedad dejan de ser relevantes para tomar una decisión.

3. ETL (extract, transform, load) en Python

El desarrollo de este punto se encuentra en el archivo **"ETL.ipynb"** en la carpeta Punto3-ETL.

Bibliografía:

https://es.wikipedia.org/wiki/Tabla_de_hechos
http://itpn.mx/recursosisc/1semestre/calculodiferencial/Unidad%20I.pdf
https://es.slideshare.net/PGExperts/really-big-elephants-postgresql-dw-15833438
https://e-mc2.net/es/funciones-ventana-window-functions
https://www.postgresql.org/docs/9.6/static/runtime-config-autovacuum.html