

**Titulación:** Grado en Ingeniería Informática y Sistemas de Información

**Curso:** 2019-2020. Convocatoria Ordinaria de Junio

**Asignatura:** Bases de Datos Avanzadas – Laboratorio

## **Practica 1: Arquitectura PostgreSQL y almacenamiento físico**

**ALUMNO 1:**

**Nombre y Apellidos:** Javier Martín Gómez\_\_\_\_\_

**DNI:** 47231977M\_\_\_\_\_

**ALUMNO 2:**

**Nombre y Apellidos:** Alberto González Martínez\_\_\_\_\_

**DNI:** 09072311F\_\_\_\_\_

**Fecha:** 29-02-2020\_\_\_\_\_

**Profesor Responsable:** \_\_

\_\_Oscar Gutiérrez\_\_\_\_\_

Mediante la entrega de este fichero los alumnos aseguran que cumplen con la normativa de autoría de trabajos de la Universidad de Alcalá, y declaran éste como un trabajo original y propio.

**En caso de ser detectada copia, se calificará la asignatura como Suspensa – Cero.**

### **Plazos**

Trabajo de Laboratorio: Semana 27 Enero, 3 Febrero, 10 Febrero, 17 Febrero y 24 de Febrero.

Entrega de práctica: Día 3 de Marzo. Aula Virtual

Documento a entregar: Este mismo fichero con las respuestas a las cuestiones planteadas. Si se entrega en formato electrónico el fichero se deberá llamar: **DNIdelosAlumnos\_PECL1.doc**

**AMBOS ALUMNOS DEBEN ENTREGAR EL FICHERO EN LA PLATAFORMA.**

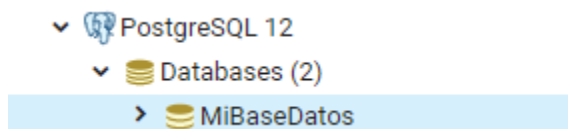
### **Introducción**

En esta primera práctica se introduce el sistema gestor de bases de datos **PostgreSQL versión 11 o 12**. Está compuesto básicamente de un motor servidor y de una serie de clientes que acceden al servidor y de otras herramientas externas. En esta primera práctica se entrará a fondo en la arquitectura de PostgreSQL, sobre todo en el almacenamiento físico de los datos y del acceso a los mismos.

## Actividades y Cuestiones

### Almacenamiento Físico en PostgreSQL

**Cuestión 1.** Crear una nueva Base de Datos que se llame **MiBaseDatos**. ¿En qué directorio se crea del disco duro, cuanto ocupa el mismo y qué ficheros se crean? ¿Por qué?



Aquí se puede ver que existe la base de datos **MiBaseDatos**

El directorio en el que se crea tiene la siguiente ruta:

**“C:\Program Files\PostgreSQL\12\data\base\16534”**

El directorio 16534 es la base de datos creada y el número 16534 corresponde al identificador (OID) de la misma.

El tamaño del directorio es 7.72 MB

Se crean los diferentes ficheros y tablas (que llevan el nombre de la tabla o el número de nodo que encontramos en pg\_class.relfilenode) y para las relaciones temporales los archivos son nombrados de la forma tBBB\_FFF donde BBB es el backend del ID del backend donde se creó el archivo. También, para el archivo principal cada tabla y cada índice tiene un “free space map” que se almacena con nombre con sufijo \_fsm. Además, las tablas tienen un “visibility map” con el sufijo \_vsm. Tienen también una tercera bifurcación que es almacenada con el sufijo \_init.

**Cuestión 2.** Crear una nueva tabla que se llame **MiTabla** que contenga un campo que se llame id\_cliente de tipo integer que sea la Primary Key, otro campo que se llame nombre de tipo text, otro que se llame apellidos de tipo text, otro dirección de tipo text y otro puntos que sea de tipo integer. ¿Qué ficheros se han creado en esta operación? ¿Qué guarda cada uno de ellos? ¿Cuánto ocupan? ¿Por qué?

Se crea un archivo con el OID de la tabla (en este caso es 16543), que contiene la tabla y ocupa 0 KB. También se crea un archivo con OID 16546 que contiene la pk de la tabla que ocupa 8 KB. También, se ha creado un archivo con OID 16548 (pg\_toast\_16543) que contiene el TOAST que ocupa 0 KB. Por último, se crea el archivo con OID 16550 (pg\_toast\_16543\_index) que contiene el índice del TOAST que ocupa 8 KB.

**Cuestión 3.** Insertar una tupla en la tabla. ¿Cuánto ocupa la tabla? ¿Se ha producido alguna actualización más? ¿Por qué?

La tabla ahora ocupa 9 KB. También se ha modificado el archivo de la PK (con OID 16546). Estas actualizaciones se han producido porque hemos insertados datos a la tabla.

**Cuestión 4.** Aplicar el módulo pg\_buffercache a la base de datos **MiBaseDatos**. ¿Es lógico lo que se muestra referido a la base de datos anterior? ¿Por qué?

Es lógico, debido a que no tienen ningún registro introducido, solo usa un cache de memoria compartida para mantenerlo accesible.

**Cuestión 5.** Borrar la tabla **MiTabla** y volverla a crear. Insertar los datos que se entregan en el fichero de texto denominado datos\_mitabla.txt. ¿Cuánto ocupa la información original a insertar? ¿Cuánto ocupa la tabla ahora? ¿Por qué? Calcular teóricamente el tamaño en bloques que ocupa la relación **MiTabla** tal y como se realiza en teoría. ¿Concuerda con el tamaño en bloques que nos proporciona PostgreSQL? ¿Por qué?

La información original ocupa 921.093 KB. Ahora la tabla ha cambiado de OID (ahora 16557) y ocupa 1.048.576 KB la tabla, pero se ha particionado en 2, y el segundo archivo ocupa 230.882 KB. Ocupa tanto, porque se han insertado los datos. Con esta consulta hemos obtenido el tamaño de cada campo:

```
1. "select
    avg(pg_column_size(id_cliente)),avg(pg_column_size(nombre)),avg(pg_column_size(apellidos)),
    avg(pg_column_size(direccion)),avg(pg_column_size(puntos))          from
    "MiTabla";"
```

Nos da el siguiente resultado:

	avg numeric	avg numeric	avg numeric	avg numeric	avg numeric
1	4.0000000000000000	14.259259333333333	17.259259333333333	17.259259333333333	4.0000000000000000

Con esto hemos calculado la media que ocupa cada unidad de las columnas, por lo tanto sabiendo que LR es la suma de la longitud de todos los campos, obtenemos que LR=56,777.

$$NR=15 \cdot 10^6$$

$$B=8192 \text{ B}$$

$$LR=56,777$$

$$FR=B/LR=8192/56.777=144 \text{ reg/bloque}$$

$$Br=NR/FR=15 \cdot 10^6/144=104167 \text{ bloques}$$

La tabla en postgres ocupa 159926 bloques.

No concuerda, ya que postgres además de los 15 millones de registros guarda el índice del archivo de datos con la PK.

**Cuestión 6.** Volver a aplicar el módulo pg\_buffercache a la base de datos **MiBaseDatos**. ¿Qué se puede deducir de lo que se muestra? ¿Por qué lo hará?

Hay muchas más entradas en la pg\_buffercache debido a que se han insertado nuevos registros y postgres ha asignado nuevos caches para optimizarla.

**Cuestión 7.** Aplicar el módulo pgstattuple a la tabla **MiTabla**. ¿Qué se muestra en las estadísticas? ¿Cuál es el grado de ocupación de los bloques? ¿Cuánto espacio libre queda? ¿Por qué?

	table_len bigint	tuple_count bigint	tuple_len bigint	tuple_percent double precision	dead_tuple_count bigint	dead_tuple_len bigint	dead_tuple_percent double precision	free_space bigint	free_percent double precision	
1	655360	2972	604719	92.27		14	6203	0.95	19648	3

El grado de ocupación es 0.95 y el espacio libre es 19648.

El espacio libre queda porque el registro no ocupa todo el espacio asignado a la tupla, además de la posibilidad de que no complete algún bloque.

**Cuestión 8.** ¿Cuál es el factor de bloque medio real de la tabla? Realizar una consulta SQL que obtenga ese valor y comparar con el factor de bloque teórico siguiendo el procedimiento visto en teoría.

Realizamos la siguiente consulta:

“select avg(bloque) from (select count(\*) as bloque from "MiTabla" group by ((ctid::text::point)[0]::bigint)) as factor;”

Y nos sale lo siguiente:

```

181
182 select avg(bloque) from (select count(*) as bloque from "MiTabla" group by ((ctid::text::point)[0]::bigint)) as factor;
183
184 (select count(*) as bloque from "MiTabla" group by ((ctid::text::point)[0]::bigint)) ;
185

```

	avg numeric
1	85.8359274858084600

El factor de bloque que hemos calculado en el ejercicio 5 es mayor que el que nos proporciona postgres.

**Cuestión 9** Con el módulo pageinspect, analizar la cabecera y elementos de la página del primer bloque, del bloque situado en la mitad del archivo y el último bloque de la tabla **MiTabla**. ¿Qué diferencias se aprecian entre ellos? ¿Por qué?

Para el primer bloque

```

25 select * from page_header(get_raw_page('MiTabla',0));
26

```

	lsn pg_lsn	checksum smallint	flags smallint	lower smallint	upper smallint	special smallint	pagesize smallint	version smallint	prune_xid xid
1	0/175BD70	0	0	400	448	8192	8192	4	0

Para el bloque de la mitad

```











28
29 select * from page_header(get_raw_page('MiTabla',159925));
30

```

	lsn pg_lsn	checksum smallint	flags smallint	lower smallint	upper smallint	special smallint	pagesize smallint	version smallint	prune_xid xid
1	0/3CA9C9...	0	0	128	5984	8192	8192	4	0

Para el último bloque.

```
26
27     select * from page_header(get_raw_page('MiTabla',79963));|
28
```

Data Output		Explain	Messages	Notifications					
	lsn pg_lsn 	checksum smallint 	flags smallint 	lower smallint 	upper smallint 	special smallint 	pagesize smallint 	version smallint 	prune_xid xid 
1	0/175BD70	0	0	400	448	8192	8192	4	0

Se diferencian en los campos lower y upper, que indican cuando una página está llena. Por lo que se puede comprobar que en el último bloque hay más espacio que en el primero y el de la mitad. Como los registros se colocan secuencialmente, el último bloque es el que tiene más espacio libre.

**Cuestión 10.** Crear un índice de tipo árbol para el campo puntos. ¿Dónde se almacena físicamente ese índice? ¿Qué tamaño tiene? ¿Cuántos bloques tiene? ¿Cuántos niveles tiene? ¿Cuántos bloques tiene por nivel? ¿Cuántas tuplas tiene un bloque de cada nivel?

Se ha creado con la siguiente consulta: “create index “ArbolMiTabla” on “MiTabla” using BTREE (puntos);”

Se ha creado con el OID (16609). Se almacena en la ruta de la base de datos que es “C:\Program Files\PostgreSQL\12\data\base\16534”. Ocupa 329504 KB.

Tiene 41188 bloques que obtenemos de la siguiente manera:

```
15
16
17     select relpages from pg_class where oid=16609;
18
```

	Data Output	Explain	Messages	Notifications
	relpages integer			
1	41188			

Tiene dos niveles.

El nodo hoja tiene 40984 bloques y el interno tiene 203 bloques como se muestra con la siguiente consulta

```
15
16     select * from pgstatindex('ArbolMiTabla'::regclass);
```

Data Output								Explain	Messages	Notifications
	version integer	tree_level integer	index_size bigint	root_block_no bigint	internal_pages bigint	leaf_pages bigint				
1	4	2	337412096	209	203	40984				

En cada bloque hay 367 tuplas, excepto en el último bloque (41187) que habrá 222, como se muestra a continuación.

```
18 select * from bt_page_stats('ArbolMiTabla',500);
19
```

Data Output Explain Messages Notifications

	blkno integer	type "char" (1)	live_items integer	dead_items integer	avg_item_size integer	page_size integer	free_size integer	btpo_prev integer
1	500	I	367	0	16	8192	800	499

```
18 select * from bt_page_stats('ArbolMiTabla',41187);
19
```

Data Output Explain Messages Notifications

	blkno integer	type "char" (1)	live_items integer	dead_items integer	avg_item_size integer	page_size integer	free_size integer	btpo_prev integer	btpo_next integer	btpo integer	btpo_flags integer
1	41187	I	222	0	16	8192	3708	41186	0	0	1

**Cuestión 11.** Determinar el tamaño de bloques que teóricamente tendría de acuerdo con lo visto en teoría y el número de niveles. Comparar los resultados obtenidos teóricamente con los resultados obtenidos en la cuestión 10.

$Br = Lt/Lb = 337412096/8192 = 41188$  bloques

Suponiendo que  $Lk = 4$  bytes;  $LPb$  y  $LPr = 8$  bytes;  $B = 8192$  bytes

NODO HOJA

$Nh * (Lk + LPr) + LPb \leq B$ ;  $12Nh \leq 8184$ ;  $Nh = 682$  punteros a registro (al ser nodo hoja)

NODO INTERMEDIO/RAÍZ

$N * LPb + (N - 1) * Lk \leq B$ ;  $12N \leq 8188$ ;  $N = 683$  punteros a bloque (al ser intermedio o raíz)

HOJA:  $br = nr/nh = 15 * 10^6 / 682 = 21995$  bloques

INTERMEDIO1:  $br = 21995 / 683 = 33$  bloques

RAÍZ:  $br = 1$

3 niveles

$Br_{total} = 21995 + 33 + 1 = 22029$  bloques totales

**Cuestión 12.** Crear un índice de tipo hash para el campo id\_cliente y otro para el campo puntos.

Se crean con las siguientes consultas:

```
create index "HashMiTabla" on "MiTabla" using HASH (puntos);
create index "HashMiTablaPK" on "MiTabla" using HASH (id_cliente);
```

**Cuestión 13.** A la vista de los resultados obtenidos de aplicar los módulos pgstattuple y pageinspect, ¿Qué conclusiones se puede obtener de los dos índices hash que se han creado? ¿Por qué?

Con pgstattuple hemos visto que, al hacer un índice hash, el índice hash para el campo id\_cliente (clave) ocupa menos que el índice hash que se ha hecho del campo puntos.

**Cuestión 14.** Realice las pruebas que considere de inserción, modificación y borrado para determinar el manejo que realiza PostgreSQL internamente con los registros de datos y las estructuras de los archivos que utiliza. Comentar las conclusiones obtenidas.

## MODIFICACIÓN

update "MiTabla"

set nombre='Juan', apellidos='Garcia',direccion='Barcelona',puntos=300 where id\_cliente=4456535;

```
16 select * from pg_class WHERE oid=1249 or oid=16611
17 or oid=1259 or oid=2608 or oid=2610 or oid=16557 or oid=16609 or oid=16610 or oid=16614 ;
18
```

Data Output Explain Messages Notifications

	oid oid	relname name	relnamespace oid	reltype oid	reloftype oid	relowner oid	relam oid	relfilenode oid	n c
1	16611	HashMiTablaPK	2200	0	0	10	405	16611	
2	16614	MiTabla_pkey	2200	0	0	10	403	16614	
3	16557	MiTabla	2200	16559	0	10	2	16557	
4	1249	pg_attribute	11	75	0	10	2	0	
5	1259	pg_class	11	83	0	10	2	0	
6	2610	pg_index	11	12003	0	10	2	2610	
7	2608	pg_depend	11	12025	0	10	2	2608	
8	16609	ArbolMiTabla	2200	0	0	10	403	16609	
9	16610	HashMiTabla	2200	0	0	10	405	16610	

Se modifican los índices creados y nuestra tabla y su tabla pk.

## BORRADO

“delete from "MiTabla" where id\_cliente=10;”

	objectname name	livetuples bigint	dead tuples bigint
193	pg_authid	0	0
194	pg_shadow	0	0
195	pg_buffercache	0	0
196	pg_statistic_ext_...	0	0
197	pg_roles	0	0
198	pg_toast_16557	0	0
199	pg_toast_16557_...	0	0
200	MiTabla	14999603	3

Quedan 3 tuplas muertas

El fichero no baja de tamaño, ya que postgres no elimina la tupla físicamente, sino que cambia el bit de referencia que indica si una tupla está muerta o no.

## INSERCIÓN

“insert into "MiTabla" (id\_cliente,nombre,apellidos,direccion,puntos)  
values(10,'Pepe','Martinez','Bilbao',345);”

	ctid tid
1	(138604,28)

Se ha insertado al final del archivo. Para ello hemos utilizado la consulta “select CTID from "MiTabla" where id\_cliente=10;”.

**Cuestión 15.** Borrar 2.000.000 de tuplas de la tabla **MiTabla** de manera aleatoria usando el valor del campo id\_cliente. ¿Qué es lo que ocurre físicamente en la base de datos? ¿Se observa algún cambio en el tamaño de la tabla y de los índices? ¿Por qué? Adjuntar el código de borrado.

	objectname name	livetuples bigint	dead tuples bigint
198	pg_toast_16557/	0	0
199	pg_toast_16557_...	0	0
200	MiTabla	12995944	2003894

Las tuplas no se borran físicamente, sino que quedan “muertas”, cambiando su bit de referencia a 0, por lo que el fichero sigue ocupando el mismo espacio.

Realizamos la siguiente consulta:

“delete from "MiTabla" where id\_cliente in (select id\_cliente from "MiTabla" where id\_cliente>0 limit 2000000);”

Al no estar ordenado por id\_cliente, se borrarán los primeros 2000000 que coja, los cuales no tienen por qué tener id\_cliente secuencial.



**Cuestión 16.** En la situación anterior, ¿Qué operaciones se puede aplicar a la base de datos **MiBaseDatos** para optimizar el rendimiento de esta? Aplicarla a la base de datos **MiBaseDatos** y comentar cuál es el resultado final y qué es lo que ocurre físicamente.

Habrá que utilizar el comando vacuum para quitar del espacio las tuplas muertas. Como hemos aplicado el vacuum full, se ha borrado la información de las tuplas muertas y se ha creado otro archivo con su información sin espacios vacíos. Además, se han creado nuevos archivos para los índices, la tabla y la tabla pk creadas anteriormente.

```
41 select * from pg_class where oid=16622 or oid=16623 or oid=16625
42 or oid=16624 or oid=16614 or oid=16611 or oid=16610 or oid=16609
43 or oid=16562 or oid=16557 or oid=16616;
44
```

Data Output Explain Messages Notifications

	oid oid	relname name	relnamespace oid	reltype oid	reloftype oid	relowner oid	relam oid
1	16611	HashMiTabl...	2200	0	0	10	405
2	16609	ArbolMiTabla	2200	0	0	10	403
3	16610	HashMiTabla	2200	0	0	10	405
4	16614	MiTabla_pkey	2200	0	0	10	403
5	16557	MiTabla	2200	16559	0	10	2
6	16562	pg_toast_16...	99	0	0	10	403

**Cuestión 17.** Crear una tabla denominada **MiTabla2** de tal manera que tenga un factor de llenado de tuplas que sea un 40% que el de la tabla **MiTabla** y cargar el archivo de datos anterior Explicar el proceso seguido y qué es lo que ocurre físicamente.

Hemos seguido el proceso para crear la tabla e importar los datos y en el campo fill factor hemos puesto el valor 40. Como el factor de llenado es mucho menor que en la tabla anterior, se ha tardado mucho más tiempo en importar los datos del txt y además el número de bloques necesarios para crear el archivo es mayor, debido a que el factor de bloque será menor.

 MiTabla2

✕

General Columns Constraints **Advanced** Parameters Security SQL

Of type

Fill factor

40

**Cuestión 18.** Realizar las mismas pruebas que la cuestión 14 en la tabla **MiTabla2**. Comparar los resultados obtenidos con los de la cuestión 14 y explicar las diferencias encontradas.

## MODIFICACIÓN

update "MiTabla2"

set nombre='Juan', apellidos='Garcia',direccion='Barcelona',puntos=300 where id\_cliente=4456535;

## BORRADO

	oid oid	relname name	relnamespace oid	reltype oid	relotype oid	relowner oid	relam oid	relfilenode oid	reltablespace oid	relpages integer
1	16611	HashMiTabl...	2200	0	0	10	405	16624	0	53882
2	2619	pg_statistic	11	12016	0	10	2	2619	0	19
3	1247	pg_type	11	71	0	10	2	0	0	10
4	16626	MiTabla2	2200	16628	0	10	2	16626	0	405225
5	16632	MiTabla2_p...	2200	0	0	10	403	16632	0	54615
6	16609	ArbolMiTabla	2200	0	0	10	403	16622	0	35697
7	16610	HashMiTabla	2200	0	0	10	405	16623	0	80701
8	16614	MiTabla_pkey	2200	0	0	10	403	16625	0	35648

Se vuelven a modificar los índices, la tabla y la tabla pk.

“delete from "MiTabla" where id\_cliente=10;”

	objectname name	livetuples bigint	deadtuples bigint
1	HashMiTablaPK	0	0
2	pg_statistic	10	5
3	pg_type	10	9
4	MiTabla2	15000537	2

En este caso quedan 2 tuplas muertas

## INSERCIÓN

“insert into "MiTabla2" (id\_cliente,nombre,apellidos,direccion,puntos)  
values(10,'Pepe','Martinez','Bilbao',345);”

	ctid tid
1	(405224,21)

Se inserta en este bloque. Es mucho mayor que el anterior, ya que en esta tabla hay muchos más bloques.

**Cuestión 19.** Las versiones 11 y 12 de PostgreSQL permite trabajar con particionamiento de tablas. ¿Para qué sirve? ¿Qué tipos de particionamientos se pueden utilizar? ¿Cuándo será útil el particionamiento?

Para optimizar tablas muy grandes con millones de tuplas, es decir, subdividir una tabla padre en varias tablas pequeñas hijas y vaciar la tabla padre utilizando la cláusula ONLY. Los tipos son: de lista, de rango y de hash. Será útil cuando se trabaje con grandes volúmenes de datos y para asignar permisos a un grupo de datos específico de una tabla.

**Cuestión 20.** Crear una nueva tabla denominada **MiTabla3** con los mismos campos que la cuestión 2, pero sin PRIMARY KEY, que esté particionada por medio de una función HASH que devuelva 10 valores sobre el campo puntos. Explicar el proceso seguido y comentar qué es lo que ha ocurrido físicamente en la base de datos.

Hemos creado la tabla con la partición de hash de la siguiente manera:

```
53 create table "MiTabla3" (  
54     "id_cliente"        int ,  
55     "nombre"           text ,  
56     "apellidos"        text,  
57     "direccion"        text,  
58     "puntos"           int  
59 ) PARTITION BY HASH (puntos);  
70
```

Una vez creada la tabla, se crean las 10 particiones:

```
1 create table "particion0" partition of "MiTabla3" for values with (modulus 10, remainder 0);  
2 create table "particion1" partition of "MiTabla3" for values with (modulus 10, remainder 1);  
3 create table "particion2" partition of "MiTabla3" for values with (modulus 10, remainder 2);  
4 create table "particion3" partition of "MiTabla3" for values with (modulus 10, remainder 3);  
5 create table "particion4" partition of "MiTabla3" for values with (modulus 10, remainder 4);  
6 create table "particion5" partition of "MiTabla3" for values with (modulus 10, remainder 5);  
7 create table "particion6" partition of "MiTabla3" for values with (modulus 10, remainder 6);  
8 create table "particion7" partition of "MiTabla3" for values with (modulus 10, remainder 7);  
9 create table "particion8" partition of "MiTabla3" for values with (modulus 10, remainder 8);  
0 create table "particion9" partition of "MiTabla3" for values with (modulus 10, remainder 9);
```

Aquí se ve que se han creado

- ▼ Partitions (10)
  - > particion0
  - > particion1
  - > particion2
  - > particion3
  - > particion4
  - > particion5
  - > particion6
  - > particion7
  - > particion8
  - > particion9

**Cuestión 21.** ¿Cuántos bloques ocupa cada una de las particiones? ¿Por qué? Comparar con el número bloques que se obtendría teóricamente utilizando el procedimiento visto en teoría.

	oid oid	relname name	relpages integer
1	16643	particion0	16220
2	16673	particion5	15094
3	16649	particion1	15531
4	16655	particion2	16682
5	16661	particion3	13715
6	16667	particion4	19200
7	16679	particion6	17594
8	16685	particion7	14626
9	16691	particion8	15520
10	16697	particion9	15747

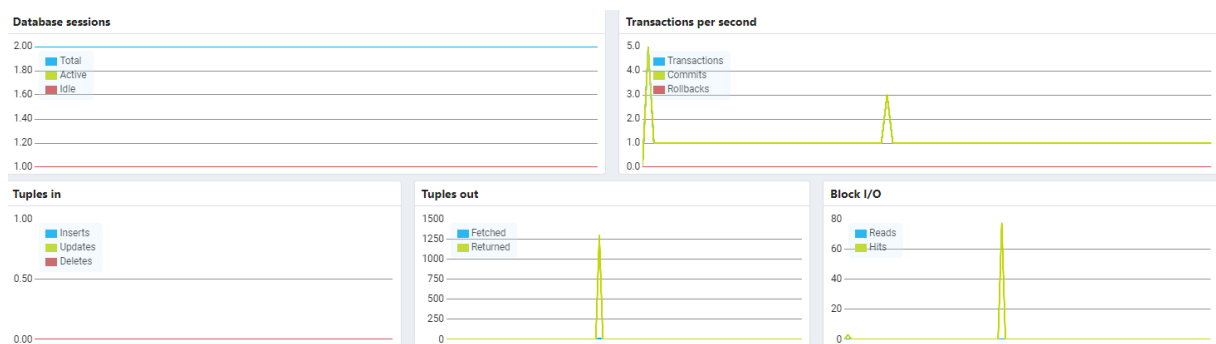
No disponemos de los medios para realizar el final de la cuestión, ya que no lo hemos visto en teoría.

## Monitorización de la actividad de la base de datos

En este último apartado se mostrará el acceso a los datos con una serie de consultas sobre la tabla original. Para ello, borrar todas las tablas creadas y volver a crear la tabla MiTabla como en la cuestión 2. Cargar los datos que se encuentran originalmente en el fichero datos\_mitabla.txt

**Cuestión 22.** ¿Qué herramientas tiene PostgreSQL para monitorizar la actividad de la base de datos sobre el disco? ¿Qué información se puede mostrar con esas herramientas? ¿Sobre qué tipo de estructuras se puede recopilar información de la actividad? Describirlo brevemente.

Dentro de la aplicación pgadmin, postgres tiene el apartado tablero que muestra, por ejemplo, gráficas en tiempo real con el número de sesiones de base de datos o transacciones por segundo. En la siguiente captura se muestra todo:



**Cuestión 23.** Crear un índice primario btree sobre el campo puntos. ¿Cuál ha sido el proceso seguido?

Se ha realizado la siguiente orden para crear el índice:

```
“create index "MiTablaBTree" on "MiTabla" using btree(puntos);”
```

**Cuestión 24.** Crear un índice hash sobre el campo puntos y otro sobre id\_cliente

El de puntos se crea así:

```
“create index "HashPuntosMiTabla" on "MiTabla" using hash(puntos);”
```

El de id\_cliente se crea así:

```
“create index "HashIdMiTabla" on "MiTabla" using hash(id_cliente);”
```

**Cuestión 25.** Analizar el tamaño de todos los índices creados y compararlos entre sí. ¿Qué conclusiones se pueden extraer de dicho análisis?

	Data Output	Explain	Messages	Notifica
	oid oid	relname name	relpages integer	
1	16717	MiTablaBTree	41188	
2	16718	HashPuntosMiTabla	85625	
3	16719	HashIdMiTabla	65391	

Para este caso es más recomendable usar el índice BTree ya que al tener menos bloques, el número de accesos será menor.

**Cuestión 26.** Para cada una de las consultas que se muestran a continuación, ¿Qué información se puede obtener de los datos monitorizados por la base de datos al realizar la consulta? ¿Comentar cómo se ha realizado la resolución de la consulta? ¿Cuántos bloques se han leído? ¿Por qué? Importante, reinicializar los datos recolectados de la actividad de la base de datos antes de lanzar cada consulta:

Cada vez que realizamos una consulta ejecutamos “select pg\_stat\_reset();” para reinicializar los datos recolectados de la anterior consulta.

Para saber los índices que se utilizan en cada consulta ejecutamos la consulta “select \* from pg\_statio\_all\_indexes where schemaname = 'public' ;”

Se obtiene información de los bloques leídos tanto en pila (heap) como los bloques normales. Además, encontramos información sobre los bloques encontrados en caché “blocks hit”. En la segunda imagen se ven los índices que se utilizan en la consulta. Dependiendo del tipo de consulta utiliza un índice u otro, ya que postgres usa el índice óptimo dependiendo de la consulta.

1. Mostrar la información de las tuplas con id\_cliente=8.101.000.

```

110 select * from "MiTabla" where "id_cliente"=8101000;
111 select * from pg_statio_all_tables where schemaname = 'public' ;
112

```

	relid oid	schemaname name	relname name	heap_blks_read bigint	heap_blks_hit bigint	idx_blks_read bigint	idx_blks_hit bigint
1	16649	public	particion1	0	0	[null]	[null]
2	16707	public	MiTabla	0	1	0	2

```

116 select * from "MiTabla" where "id_cliente">8000 and "id_cliente"<10000;

```

	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_blks_read bigint	idx_blks_hit bigint
1	16707	16715	public	MiTabla	MiTabla_pkey	0	0
2	16707	16721	public	MiTabla	MiTablaBTree	1	0
3	16707	16722	public	MiTabla	HashPuntosMiTabla	0	0
4	16707	16723	public	MiTabla	HashIdMiTabla	2	1

2. Mostrar la información de las tuplas con id\_cliente <30000.

```

111
112 select * from "MiTabla" where "id_cliente"<30000;
113 select * from pg_statio_all_tables where schemaname = 'public' ;
114
115

```

	relid oid	schemaname name	relname name	heap_blks_read bigint	heap_blks_hit bigint	idx_blks_read bigint	idx_blks_hit bigint
1	16649	public	particion1	0	0	[null]	[null]
2	16707	public	MiTabla	27533	2	100	3

	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_blks_read bigint	idx_blks_hit bigint
1	16707	16715	public	MiTabla	MiTabla_pkey	100	3
2	16707	16721	public	MiTabla	MiTablaBTree	0	0
3	16707	16722	public	MiTabla	HashPuntosMiTabla	0	0
4	16707	16723	public	MiTabla	HashIdMiTabla	0	0

3. Mostrar el número de tuplas cuyo id\_cliente >8000 y id\_cliente <100000.

```
115 select * from "MiTabla" where "id_cliente">8000 and "id_cliente"<10000;
116 select * from pg_statio_all_tables where schemaname = 'public' ;
117
```

Data Output Explain Messages Notifications

	relid oid	schemaname name	relname name	heap_blks_read bigint	heap_blks_hit bigint	idx_blks_read bigint	idx_blks_hit bigint	tc bi
1	16649	public	particion1	0	0	[null]	[null]	
2	16707	public	MiTabla	905	1085	10	7	

Data Output Explain Messages Notifications

	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_blks_read bigint	idx_blks_hit bigint
1	16707	16715	public	MiTabla	MiTabla_pkey	10	5
2	16707	16721	public	MiTabla	MiTablaBTree	0	0
3	16707	16722	public	MiTabla	HashPuntosMiTabla	0	0
4	16707	16723	public	MiTabla	HashIdMiTabla	0	0

4. Mostrar la información de las tuplas con id\_cliente=34500 o id\_cliente=30.204.000.

```
117
118 select * from "MiTabla" where "id_cliente">34500 or "id_cliente"<30204000;
119 select * from pg_statio_all_tables where schemaname = 'public' ;
120
121
```

Data Output Explain Messages Notifications

	relid oid	schemaname name	relname name	heap_blks_read bigint	heap_blks_hit bigint	idx_blks_read bigint	idx_blks_hit bigint	t t
1	16649	public	particion1	0	0	[null]	[null]	
2	16707	public	MiTabla	158798	15956	2	4	

Data Output Explain Messages Notifications

	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_blks_read bigint	idx_blks_hit bigint
1	16707	16715	public	MiTabla	MiTabla_pkey	2	4
2	16707	16721	public	MiTabla	MiTablaBTree	0	0
3	16707	16722	public	MiTabla	HashPuntosMiTabla	0	0
4	16707	16723	public	MiTabla	HashIdMiTabla	0	0

5. Mostrar las tuplas cuyo id\_cliente es distinto de 3450000.

```
120
121 select * from "MiTabla" where not("id_cliente"=3450000);
122 select * from pg_statio_all_tables where schemaname = 'public' ;
123
```

	relid oid	schemaname name	relname name	heap_blks_read bigint	heap_blks_hit bigint	idx_blks_read bigint	idx_blks_hit bigint
1	16649	public	particion1	0	0	[null]	[null]
2	16707	public	MiTabla	158775	15977	0	0

	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_blks_read bigint	idx_blks_hit bigint
1	16707	16715	public	MiTabla	MiTabla_pkey	0	0
2	16707	16721	public	MiTabla	MiTablaBTree	0	0
3	16707	16722	public	MiTabla	HashPuntosMiTabla	0	0
4	16707	16723	public	MiTabla	HashIdMiTabla	0	0

6. Mostrar las tuplas que tiene un nombre igual a 'nombre3456789'.

```
126 select * from "MiTabla" where "nombre"='nombre3456789';
127 select * from pg_statio_all_tables where schemaname = 'public' ;
128
129
```

	relid oid	schemaname name	relname name	heap_blks_read bigint	heap_blks_hit bigint	idx_blks_read bigint	idx_blks_hit bigint
1	16649	public	particion1	0	0	[null]	[null]
2	16707	public	MiTabla	158636	16116	0	0

	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_blks_read bigint	idx_blks_hit bigint
1	16707	16715	public	MiTabla	MiTabla_pkey	0	0
2	16707	16721	public	MiTabla	MiTablaBTree	0	0
3	16707	16722	public	MiTabla	HashPuntosMiTabla	0	0
4	16707	16723	public	MiTabla	HashIdMiTabla	0	0



## 7. Mostrar la información de las tuplas con puntos=650.

```

129
130 select * from "MiTabla" where "puntos"=650;
131 select * from pg_statio_all_tables where schemaname = 'public' ;
132
133 select * from "MiTabla" where "puntos"<200;

```

Data Output Explain Messages Notifications

	relid oid	schemaname name	relname name	heap_blks_read bigint	heap_blks_hit bigint	idx_blks_read bigint	idx_blks_hit bigint	t t
1	16649	public	particion1	0	0	[null]	[null]	
2	16707	public	MiTabla	19687	459	60	1	

Data Output Explain Messages Notifications

	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_blks_read bigint	idx_blks_hit bigint	
1	16707	16715	public	MiTabla	MiTabla_pkey	0	0	
2	16707	16721	public	MiTabla	MiTablaBTree	61	0	
3	16707	16722	public	MiTabla	HashPuntosMiTabla	0	0	
4	16707	16723	public	MiTabla	HashIdMiTabla	0	0	

## 8. Mostrar la información de las tuplas con puntos<200.

```

133 select * from "MiTabla" where "puntos"<200;
134 select * from pg_statio_all_tables where schemaname = 'public' ;
135
136
137

```

Data Output Explain Messages Notifications

	relid oid	schemaname name	relname name	heap_blks_read bigint	heap_blks_hit bigint	idx_blks_read bigint	idx_blks_hit bigint	t b
1	16649	public	particion1	0	0	[null]	[null]	
2	16707	public	MiTabla	158547	16205	0	0	

Data Output Explain Messages Notifications

	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_blks_read bigint	idx_blks_hit bigint	
1	16707	16715	public	MiTabla	MiTabla_pkey	0	0	
2	16707	16721	public	MiTabla	MiTablaBTree	0	0	
3	16707	16722	public	MiTabla	HashPuntosMiTabla	0	0	
4	16707	16723	public	MiTabla	HashIdMiTabla	0	0	

9. Mostrar la información de las tuplas con puntos>30000.

```

137
138 select * from "MiTabla" where "puntos">30000;
139 select * from pg_statio_all_tables where schemaname = 'public' ;
140
141

```

Data Output Explain Messages Notifications

	relid oid	schemaname name	relname name	heap_blks_read bigint	heap_blks_hit bigint	idx_blks_read bigint	idx_blks_hit bigint
1	16649	public	particion1	0	0	[null]	[null]
2	16707	public	MiTabla	0	1	3	3

Data Output Explain Messages Notifications

	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_blks_read bigint	idx_blks_hit bigint
1	16707	16715	public	MiTabla	MiTabla_pkey	0	0
2	16707	16721	public	MiTabla	MiTablaBTree	3	3
3	16707	16722	public	MiTabla	HashPuntosMiTabla	0	0
4	16707	16723	public	MiTabla	HashIdMiTabla	0	0

10. Mostrar la información de las tuplas con id\_cliente=90000 o puntos=230

```

145
146 select * from "MiTabla" where "puntos"=230 or "id_cliente"=90000;
147 select * from pg_statio_all_tables where schemaname = 'public' ;
148
149

```

Data Output Explain Messages Notifications

	relid oid	schemaname name	relname name	heap_blks_read bigint	heap_blks_hit bigint	idx_blks_read bigint	idx_blks_hit bigint
1	16649	public	particion1	0	0	[null]	[null]
2	16707	public	MiTabla	20158	1	61	2

Data Output Explain Messages Notifications








	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_blks_read bigint	idx_blks_hit bigint
1	16707	16715	public	MiTabla	MiTabla_pkey	0	0
2	16707	16721	public	MiTabla	MiTablaBTree	61	1
3	16707	16722	public	MiTabla	HashPuntosMiTabla	0	0
4	16707	16723	public	MiTabla	HashIdMiTabla	0	1







# 11. Mostrar la información de las tuplas con id\_cliente=90000 y puntos=230

```

141
142 select * from "MiTabla" where "puntos"=230 and "id_cliente"=90000;
143 select * from pg_statio_all_tables where schemaname = 'public' ;
144
145
146
147
148

```

Data Output		Explain	Messages	Notifications			
	relid oid	 schemaname name	 relname name	 heap_blks_read bigint	 heap_blks_hit bigint	 idx_blks_read bigint	 idx_blks_hit bigint
1	16649	public	particion1	0	0	[null]	[null]
2	16707	public	MiTabla	1	0	1	0

Data Output		Explain	Messages	Notifications				
	relid oid	 indexrelid oid	 schemaname name	 relname name	 indexrelname name	 idx_blks_read bigint	 idx_blks_hit bigint	
1	16707	16715	public	MiTabla	MiTabla_pkey	0	0	
2	16707	16721	public	MiTabla	MiTablaBTree	0	0	
3	16707	16722	public	MiTabla	HashPuntosMiTabla	0	0	
4	16707	16723	public	MiTabla	HashIdMiTabla	1	0	

**Cuestión 27.** Borrar los índices creados y crear un índice multiclave btree sobre los campos puntos y nombre.

Realizamos lo siguiente: “create index "IndiceMulti" on "MiTabla" (puntos,nombre);”

**Cuestión 28.** Para cada una de las consultas que se muestran a continuación, ¿Qué información se puede obtener de los datos monitorizados por la base de datos al realizar la consulta? ¿Comentar cómo se ha realizado la resolución de la consulta? ¿Cuántos bloques se han leído? ¿Por qué? Importante, reinicializar los datos recolectados de la actividad de la base de datos antes de lanzar cada consulta:

Cada vez que realizamos una consulta ejecutamos “select pg\_stat\_reset();” para reinicializar los datos recolectados de la anterior consulta.

Para saber los índices que se utilizan en cada consulta ejecutamos la consulta “select \* from pg\_statio\_all\_indexes where schemaname = 'public' ;”

Se obtiene información de los bloques leídos tanto en pila (heap) como los bloques normales. Además, encontramos información sobre los bloques encontrados en caché “blocks hit”. En la segunda imagen se ven los índices que se utilizan en la consulta. Dependiendo del tipo de consulta utiliza un índice u otro, ya que postgres usa el índice óptimo dependiendo de la consulta.

- Mostrar las tuplas cuyos puntos valen 200 y su nombre es nombre3456789.

```

160 select * from "MiTabla" where "puntos"=200 and "nombre"='nombre3456789';
161 select * from pg_statio_all_tables where schemaname = 'public' ;
162
163
164

```

Data Output Explain Messages Notifications

	relid oid	schemaname name	relname name	heap_blks_read bigint	heap_blks_hit bigint	idx_blks_read bigint	idx_blks_hit bigint	to bi
1	16649	public	particion1	0	0	[null]	[null]	
2	16707	public	MiTabla	0	0	4	0	

Data Output Explain Messages Notifications

	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_blks_read bigint	idx_blks_hit bigint	
1	16707	16715	public	MiTabla	MiTabla_pkey	0	0	
2	16707	16720	public	MiTabla	IndiceMulti	4	0	

- Mostrar las tuplas cuyos puntos valen 200 o su nombre es nombre3456789.

```

154 select * from "MiTabla" where "puntos"=200 or "nombre"='nombre3456789';
155 select * from pg_statio_all_tables where schemaname = 'public' ;
156
157 select pg_stat_reset();
...

```

Data Output Explain Messages Notifications

	relid oid	schemaname name	relname name	heap_blks_read bigint	heap_blks_hit bigint	idx_blks_read bigint	idx_blks_hit bigint	to bi
1	16649	public	particion1	0	0	[null]	[null]	
2	16707	public	MiTabla	158745	16007	1	0	

Data Output Explain Messages Notifications

	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_blks_read bigint	idx_blks_hit bigint	
1	16707	16715	public	MiTabla	MiTabla_pkey	0	0	
2	16707	16720	public	MiTabla	IndiceMulti	0	0	

3. Mostrar las tuplas cuyo id\_cliente vale 6000 o su nombre es nombre3456789.

```
165 select * from "MiTabla" where "id_cliente"=6000 or "nombre"='nombre3456789';
166 select * from pg_statio_all_tables where schemaname = 'public' ;
167
168
169
```

Data Output Explain Messages Notifications

	relid oid	schemaname name	relname name	heap_blks_read bigint	heap_blks_hit bigint	idx_blks_read bigint	idx_blks_hit bigint	
1	16649	public	particion1	0	0	[null]	[null]	1
2	16707	public	MiTabla	158736	16016	0	0	1

```
172 SELECT * FROM "MiTabla" WHERE "id_cliente"=6000 and "nombre"='nombre3456789' ;
```

Data Output Explain Messages Notifications

	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_blks_read bigint	idx_blks_hit bigint	
1	16707	16715	public	MiTabla	MiTabla_pkey	0	0	
2	16707	16720	public	MiTabla	IndiceMulti	0	0	

4. Mostrar las tuplas cuyo id\_cliente vale 6000 y su nombre es nombre3456789.

```
170 select * from "MiTabla" where "id_cliente"=6000 and "nombre"='nombre3456789';
171 select * from pg_statio_all_tables where schemaname = 'public' ;
172
```

Data Output Explain Messages Notifications

	relid oid	schemaname name	relname name	heap_blks_read bigint	heap_blks_hit bigint	idx_blks_read bigint	idx_blks_hit bigint	tr b
1	16649	public	particion1	0	0	[null]	[null]	
2	16707	public	MiTabla	0	1	0	3	

Data Output Explain Messages Notifications

	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_blks_read bigint	idx_blks_hit bigint	
1	16707	16715	public	MiTabla	MiTabla_pkey	1	2	
2	16707	16720	public	MiTabla	IndiceMulti	0	0	

**Cuestión 29.** Crear la tabla **MiTabla3** como en la cuestión 20. Para cada una de las consultas que se muestran a continuación, ¿Qué información se puede obtener de los datos monitorizados por la base de datos al realizar la consulta? ¿Comentar cómo se ha realizado la resolución de la consulta? ¿Cuántos bloques se han leído? ¿Por qué? Importante, reinicializar los datos recolectados de la actividad de la base de datos antes de lanzar cada consulta:

Cada vez que realizamos una consulta ejecutamos “select pg\_stat\_reset();” para reinicializar los datos recolectados de la anterior consulta.

Para saber los índices que se utilizan en cada consulta ejecutamos la consulta “select \* from pg\_statio\_all\_indexes where schemaname = 'public' ;”

Se obtiene información de los bloques leídos tanto en pila (heap) como los bloques normales. Además, encontramos información sobre los bloques encontrados en caché “blocks hit”. En la segunda imagen se ven los índices que se utilizan en la consulta. Dependiendo del tipo de consulta utiliza un índice u otro, ya que postgres usa el índice óptimo dependiendo de la consulta.

### 1. Mostrar las tuplas cuyos puntos valen 200.

```

7 select pg_stat_reset();
8 select * from "MiTabla3" where "puntos"=200;
9 select * from pg_statio_all_tables where schemaname = 'public' ;
0

```

relid oid	schemaname name	relname name	heap_blks_read bigint	heap_blks_hit bigint	idx_blks_read bigint
16649	public	partition1	15339	192	[null]
16707	public	MiTabla	0	0	0
16655	public	partition2	0	0	[null]
16697	public	partition9	0	0	[null]
16643	public	partition0	0	0	[null]
16661	public	partition3	0	0	[null]
16685	public	partition7	0	0	[null]
16679	public	partition6	0	0	[null]
16667	public	partition4	0	0	[null]
16691	public	partition8	0	0	[null]
16673	public	partition5	0	0	[null]

relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_blks_read bigint	idx_blks_hit bigint
1	16715	public	MiTabla	MiTabla_pkey	0	0
2	16720	public	MiTabla	IndiceMulti	0	0

2. Mostrar las tuplas cuyos puntos valen 200 y 300.

```
177 select pg_stat_reset();
178 select * from "MiTabla3" where "puntos"=200 and "puntos"=300;
179 select * from pg_statio_all_tables where schemaname = 'public' ;
180
```

Data Output Explain Messages Notifications

	relid oid	schemaname name	relname name	heap_blks_read bigint	heap_blks_hit bigint	idx_blks_read bigint
1	16649	public	particion1	0	0	[null]
2	16707	public	MiTabla	0	0	0
3	16655	public	particion2	0	0	[null]
4	16697	public	particion9	0	0	[null]
5	16643	public	particion0	0	0	[null]
6	16661	public	particion3	0	0	[null]
7	16685	public	particion7	0	0	[null]
8	16679	public	particion6	0	0	[null]
9	16667	public	particion4	0	0	[null]
10	16691	public	particion8	0	0	[null]
11	16673	public	particion5	0	0	[null]

Data Output Explain Messages Notifications

	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_blks_read bigint	idx_blks_hit bigint
1	16707	16715	public	MiTabla	MiTabla_pkey	0	0
2	16707	16720	public	MiTabla	IndiceMulti	0	0

3. Mostrar las tuplas cuyos puntos valen 200 o 202

```
18 select * from "MiTabla3" where "puntos" between 200 and 202;
19 select * from pg_statio_all_tables where schemaname = 'public' ;
20
```

Data Output Explain Messages Notifications

	relid oid	schemaname name	relname name	heap_blks_read bigint	heap_blks_hit bigint	idx_blks_read bigint
1	16649	public	particion1	15243	288	[null]
2	16707	public	MiTabla	0	0	0
3	16655	public	particion2	16682	0	[null]
4	16697	public	particion9	15747	0	[null]
5	16643	public	particion0	16220	0	[null]
5	16661	public	particion3	13715	0	[null]
7	16685	public	particion7	14626	0	[null]
3	16679	public	particion6	17594	0	[null]
3	16667	public	particion4	19200	0	[null]
0	16691	public	particion8	15520	0	[null]
1	16673	public	particion5	15094	0	[null]

Data Output Explain Messages Notifications

	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_blks_read bigint	idx_blks_hit bigint
1	16707	16715	public	MiTabla	MiTabla_pkey	0	0
2	16707	16720	public	MiTabla	IndiceMulti	0	0



4. Mostrar las tuplas cuyos puntos son > 500.

```
8 select * from "MiTabla3" where "puntos">500;
9 select * from pg_statio_all_tables where schemaname = 'public' ;
0
```

Data Output		Explain	Messages	Notifications			
	relid oid	schemaname name	relname name	heap_blks_read bigint	heap_blks_hit bigint	idx_blks_read bigint	idx_blks_hit bigint
1	16649	public	particion1	15147	384	[null]	[null]
2	16707	public	MiTabla	0	0	0	0
3	16655	public	particion2	16586	96	[null]	[null]
4	16697	public	particion9	15651	96	[null]	[null]
5	16643	public	particion0	16124	96	[null]	[null]
6	16661	public	particion3	13619	96	[null]	[null]
7	16685	public	particion7	14530	96	[null]	[null]
8	16679	public	particion6	17498	96	[null]	[null]
9	16667	public	particion4	19104	96	[null]	[null]
0	16691	public	particion8	15360	160	[null]	[null]
1	16673	public	particion5	14998	96	[null]	[null]

167 select \* from "MiTabla" where "id\_cliente">6000 or "nombre">1000000;

Data Output

Explain

Messages

Notifications

	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_blks_read bigint	idx_blks_hit bigint
1	16707	16715	public	MiTabla	MiTabla_pkey	0	0
2	16707	16720	public	MiTabla	IndiceMulti	0	0

5. Mostrar las tuplas cuyos puntos son  $> 500$  y  $< 550$ .

```
78 select * from "MiTabla3" where "puntos">500 and "puntos"<550;
79 select * from pg_statio_all_tables where schemaname = 'public' ;
80
```

Data Output Explain Messages Notifications

	relid oid	schemaname name	relname name	heap_blks_read bigint	heap_blks_hit bigint	idx_blks_read bigint	idx_blks_hit bigint
1	16649	public	particion1	15115	416	[null]	[null]
2	16707	public	MiTabla	0	0	0	0
3	16655	public	particion2	16554	128	[null]	[null]
4	16697	public	particion9	15619	128	[null]	[null]
5	16643	public	particion0	16092	128	[null]	[null]
6	16661	public	particion3	13587	128	[null]	[null]
7	16685	public	particion7	14498	128	[null]	[null]
8	16679	public	particion6	17466	128	[null]	[null]
9	16667	public	particion4	19072	128	[null]	[null]
10	16691	public	particion8	15328	192	[null]	[null]
11	16673	public	particion5	14966	128	[null]	[null]

Data Output Explain Messages Notifications

	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_blks_read bigint	idx_blks_hit bigint
1	16707	16715	public	MiTabla	MiTabla_pkey	0	0
2	16707	16720	public	MiTabla	IndiceMulti	0	0

## 6. Mostrar las tuplas cuyos puntos son 800

```
select * from "MiTabla3" where "puntos"=800;  
select * from pg_statio_all_tables where schemaname = 'public' ;
```

ta Output Explain Messages Notifications

relid oid	schemaname name	relname name	heap_blks_read bigint	heap_blks_hit bigint	idx_blks_read bigint	idx_blks_hit bigint
16649	public	particion1	0	0	[null]	[null]
16707	public	MiTabla	0	0	0	0
16655	public	particion2	0	0	[null]	[null]
16697	public	particion9	0	0	[null]	[null]
16643	public	particion0	0	0	[null]	[null]
16661	public	particion3	0	0	[null]	[null]
16685	public	particion7	0	0	[null]	[null]
16679	public	particion6	17434	160	[null]	[null]
16667	public	particion4	0	0	[null]	[null]
16691	public	particion8	0	0	[null]	[null]
16673	public	particion5	0	0	[null]	[null]

Data Output Explain Messages Notifications

	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_blks_read bigint	idx_blks_hit bigint
1	16707	16715	public	MiTabla	MiTabla_pkey	0	0
2	16707	16720	public	MiTabla	IndiceMulti	0	0

**Cuestión 30.** A la vista de los resultados obtenidos de este apartado, comentar las conclusiones que se pueden obtener del acceso de PostgreSQL a los datos almacenados en disco.

Postgresql realiza un conjunto de operaciones para calcular la eficiencia de una consulta y su optimización, comprobando qué tipo de búsqueda es la más adecuada para acceder a los datos. En el caso de que n haya ningún índice creado en la base de datos, este comprobará todos los posibles, en cambio, de haber un conjunto de índices creados, comprobará las posibilidades dentro de este conjunto, eligiendo siempre la más eficiente.

## **Bibliografía (PostgreSQL 12)**

- Capítulo 1: Getting Started.
- Capítulo 5: 5.5 System Columns.
- Capítulo 5: 5.11 Table Partitioning.
- Capítulo 11: Indexes.
- Capítulo 19: Server Configuration.
- Capítulo 24: Routine Database Maintenance Tasks.
- Capítulo 28: Monitoring Database Activity.
- Capítulo 29: Monitoring Disk Usage.
- Capítulo VI.II: PostgreSQL Client Applications.
- Capítulo VI.III: PostgreSQL Server Applications.
- Capítulo 50: System Catalogs.
- Capítulo 68: Database Physical Storage.
- Apéndice F: Additional Supplied Modules.
- Apéndice G: Additional Supplied Programs.