

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: Adina Murg

DNI:

ALUMNO 2:

Nombre y Apellidos: Victoria Lorena Ordenes Orbegozo

DNI:

Fecha:29/02/2020

Profesor Responsable: Iván González

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

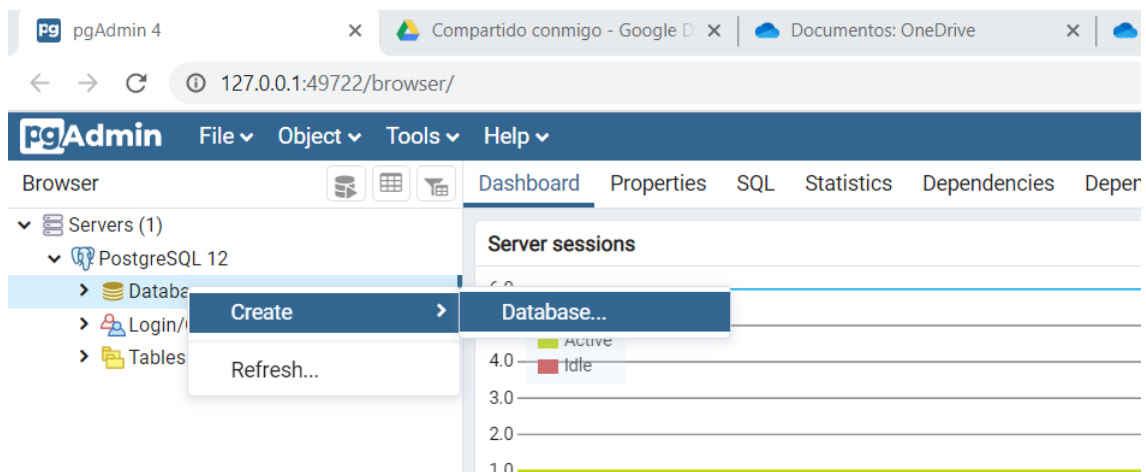
todo en el almacenamiento físico de los datos y del acceso a los mismos.

Actividades y Cuestiones

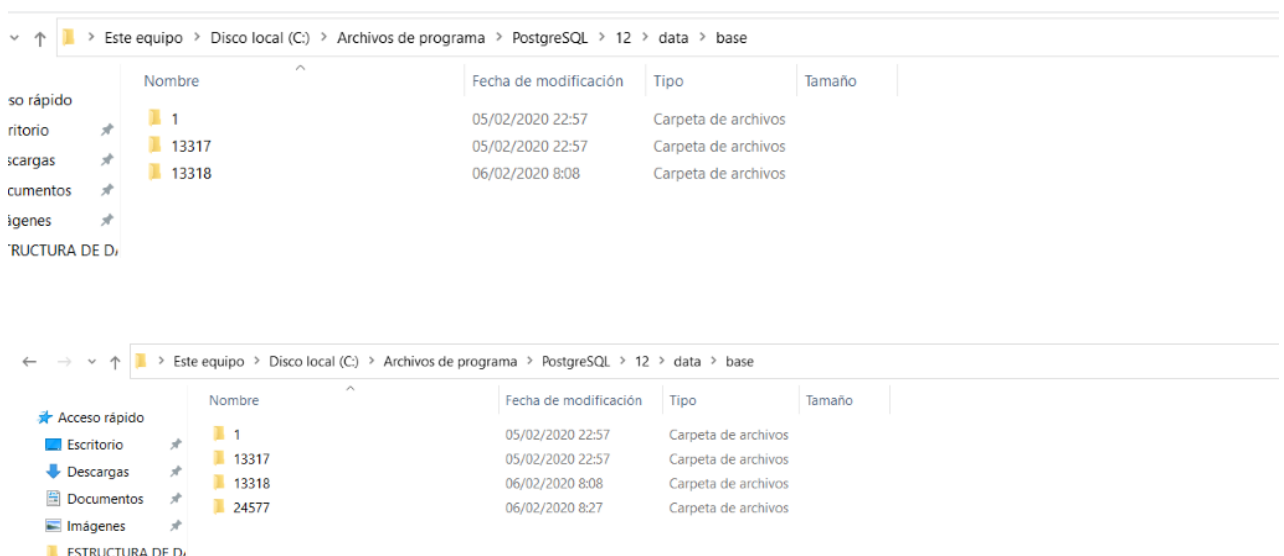
Almacenamiento Físico en PostgreSQL

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?

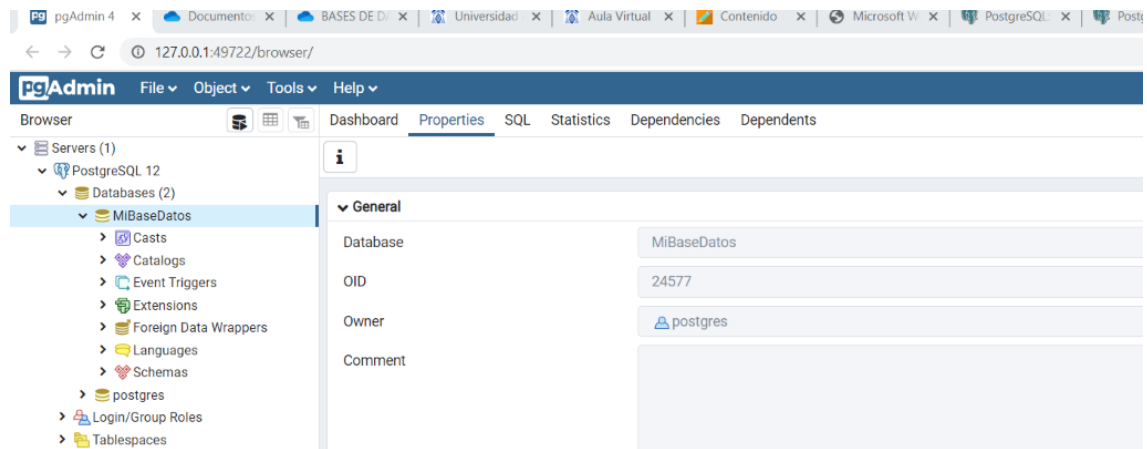
Empezamos creando la base de datos:



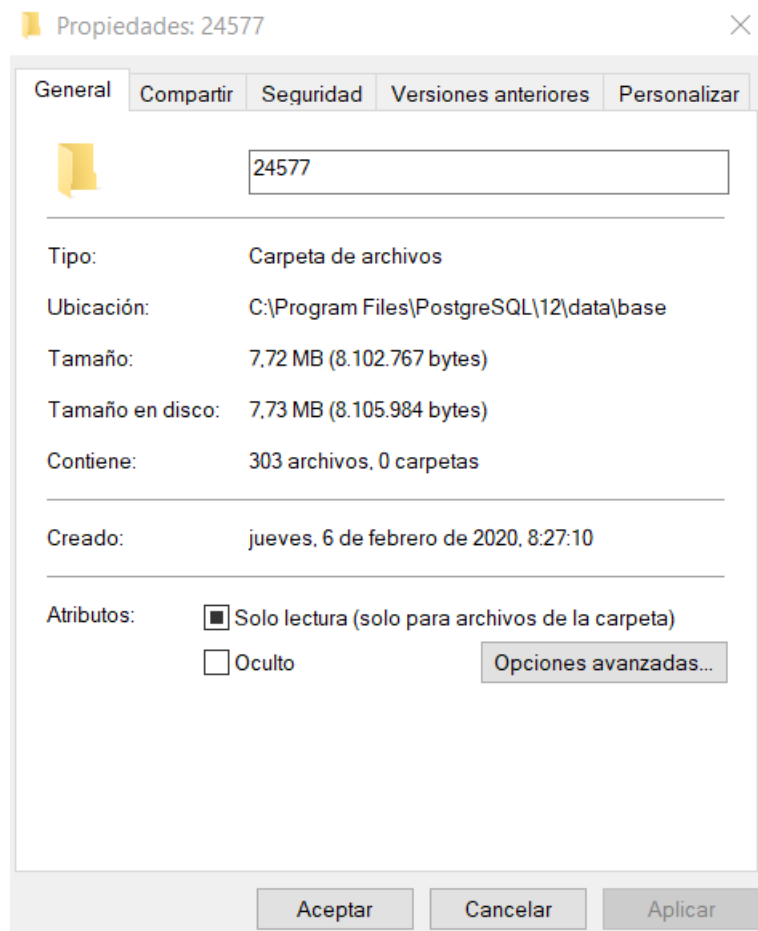
A continuación, tenemos una comparación entre los archivos generados nada más empezar a usar PostgreSQL desde cero y cuando ya hemos creado la base de datos MiBaseDatos, por lo que se puede apreciar la aparición de la carpeta 24577 que será el OID y que contendrá los ficheros físicos de la base de datos y que se crea en el directorio **PostgreSQL>12>data>base** del disco duro.

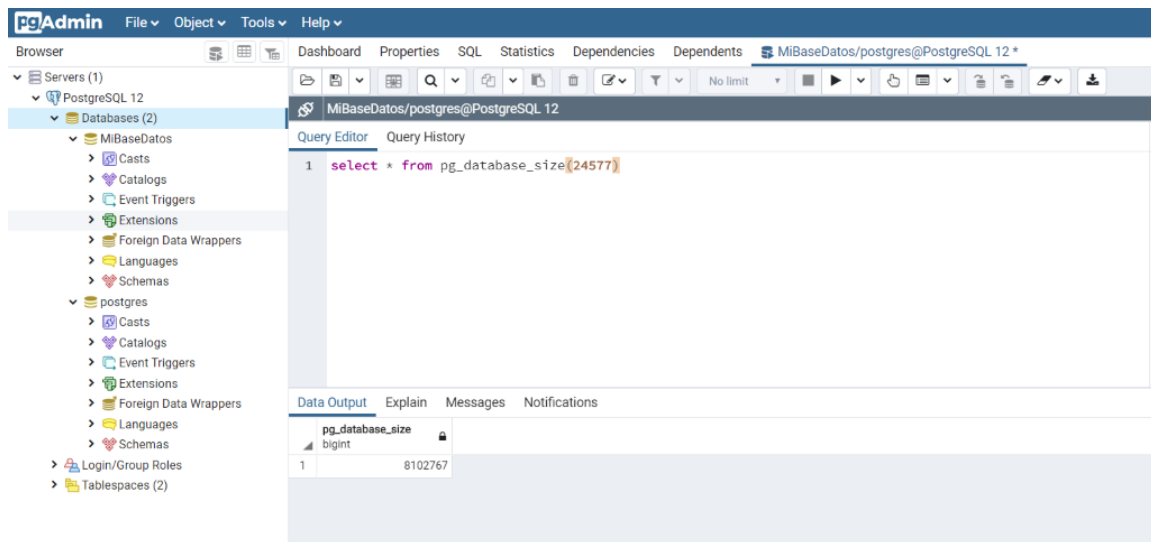


Podemos comprobar que el OID coincide correctamente con el de la carpeta.



Dentro de la carpeta tenemos 303 archivos, que en total ocupan **8.102.767 bytes** (7,72MB), que podemos comprobar mediante una consulta en la propia base de datos junto al OID que coincide correctamente. Los ficheros que se crean son los básicos y necesarios para el arranque y correcta gestión de la base de datos pese a estar vacía.





Usando el exe oid2name (que mapea el disco y tiene números físicos para saber dónde está cada elemento) encontramos todas las carpetas anteriores.

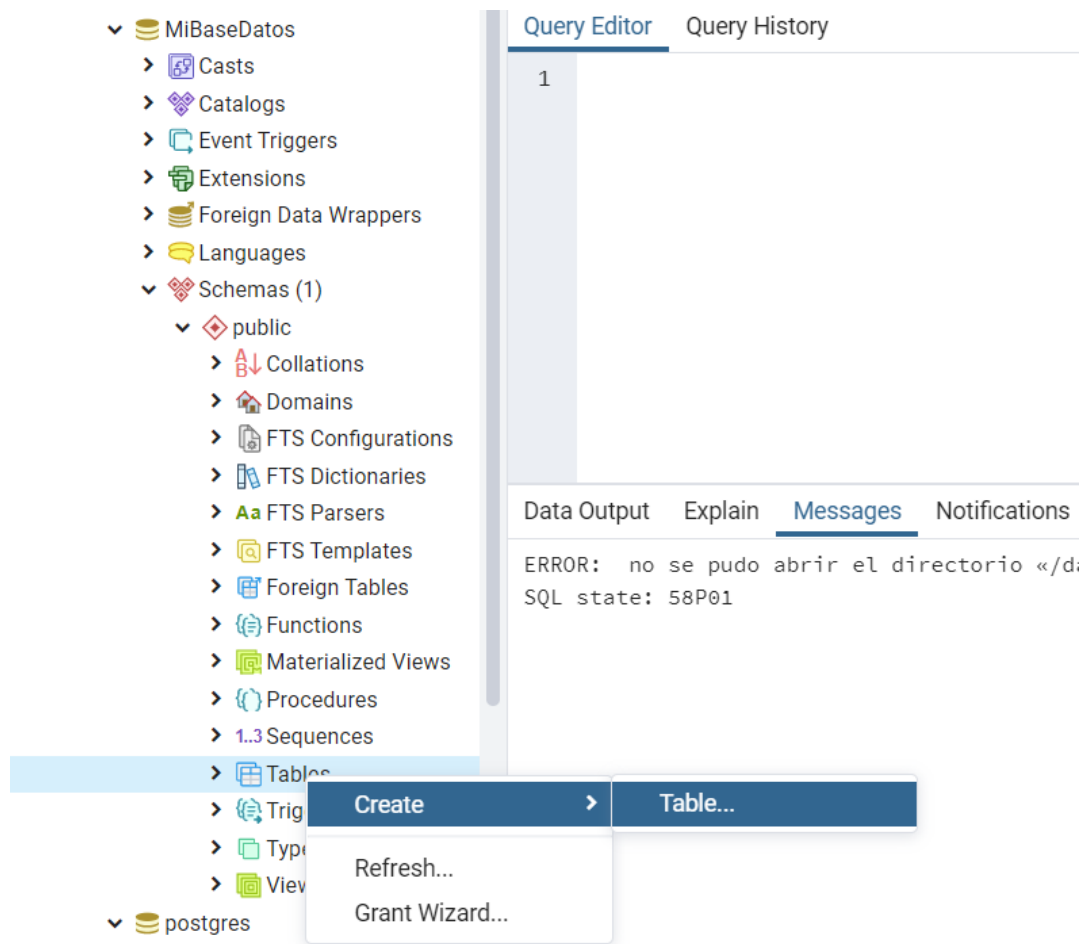
```

PS C:\Program Files\PostgreSQL\12\bin> .\oid2name.exe -U postgres
Password:
All databases:
  Oid Database Name Tablespace
-----
 24577 MiBaseDatos pg_default
 13318 postgres pg_default
 13317 template0 pg_default
 1 template1 pg_default
PS C:\Program Files\PostgreSQL\12\bin>

```

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?

Empezamos creando la tabla MiTabla en la base de datos MiBaseDatos.



A continuación, rellenamos los datos que se nos piden:

Create - Table

General

Columns

Constraints

Advanced

Partition

Parameters

Security











SQL


Inherited from table(s)


Select to inherit from...

Columns

+

	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?
 	id_cliente	integer			<div>Yes</div>	<div>Yes</div>
 	nombre	text			<div>Yes</div>	<div>No</div>
 	apellidos	text			<div>Yes</div>	<div>No</div>
 	direccion	text			<div>Yes</div>	<div>No</div>
 	puntos	integer			<div>Yes</div>	<div>No</div>





Cancel

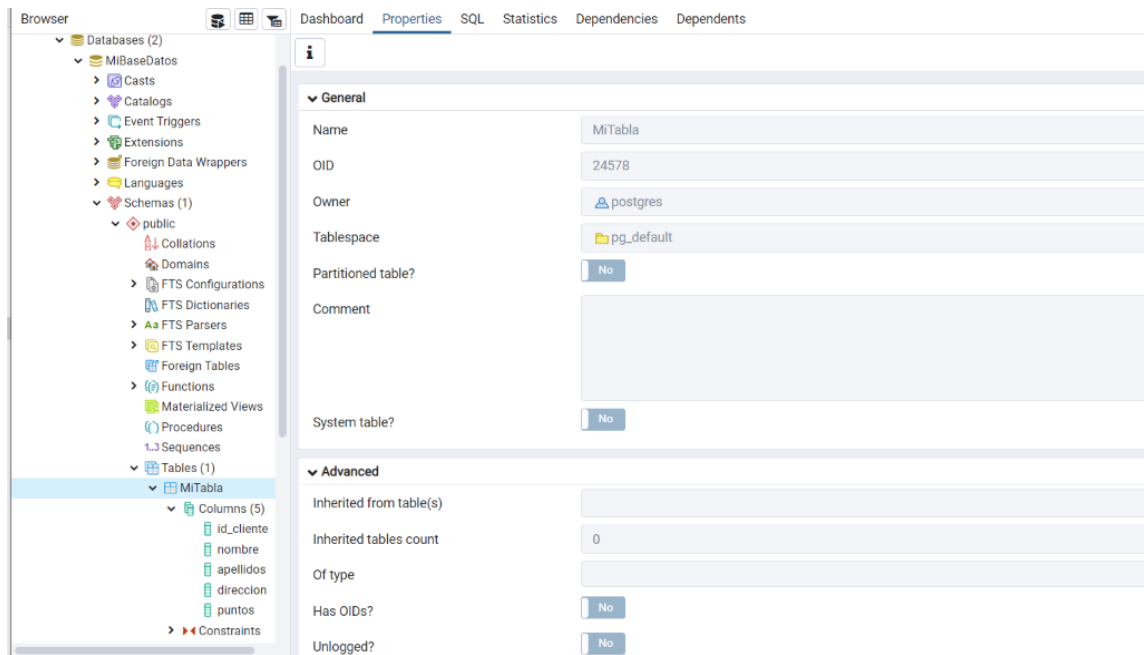
Reset

Save

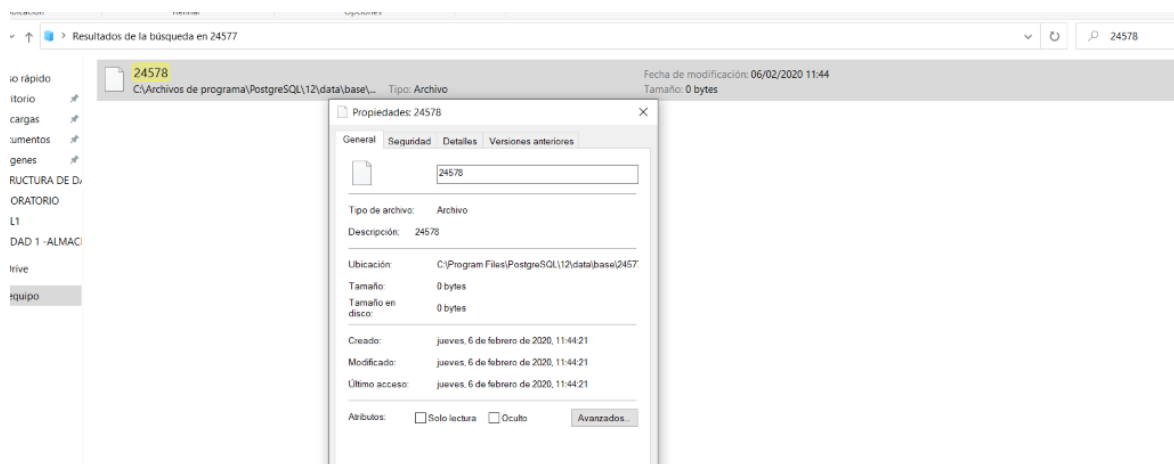
Observamos que aparecen 4 nuevos archivos, de 0, 0, 8 y 8Kb.

<div> <div>← → ↑</div> <div>Este equipo > Disco local (C:) > Archivos de programa > PostgreSQL > 12 > data > base > 24577</div> <div> <div>Buscar</div> </div> </div>			
Acceso rápido	Nombre	Fecha de modificación	Tamaño
Escritorio	24578	06/02/2020 11:44	0 KB
Descargas	24581	06/02/2020 11:44	0 KB
Documentos	24583	06/02/2020 11:44	8 KB
	24584	06/02/2020 11:44	8 KB

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

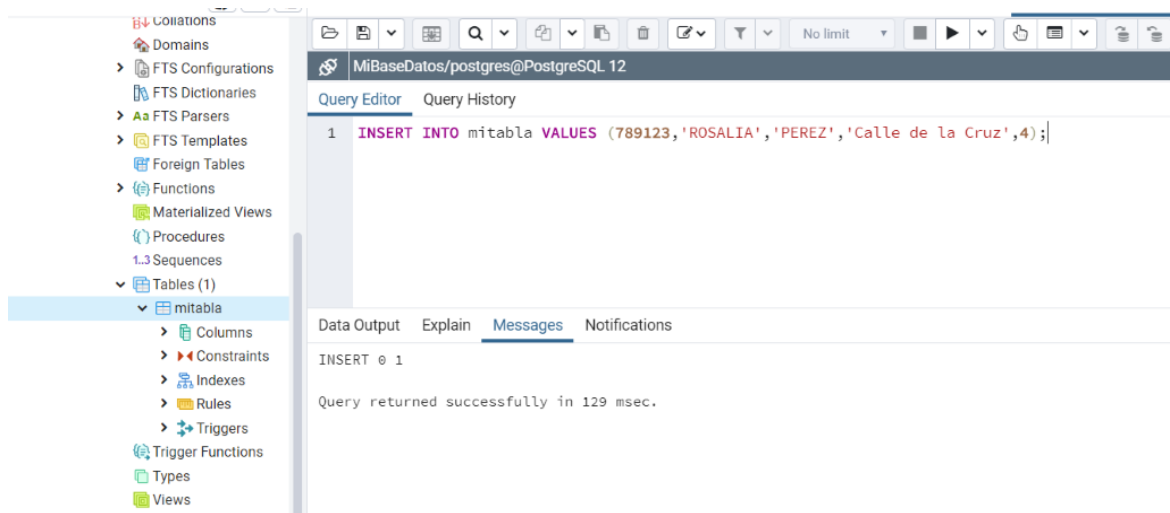


Primero buscamos el OID de nuestra tabla MiTabla con sus respectivos campos que están vacíos. A continuación, buscamos en el explorador de archivos ese OID y encontramos el archivo que corresponde con la tabla, comprobamos que tiene 0 bytes dado que está vacía.

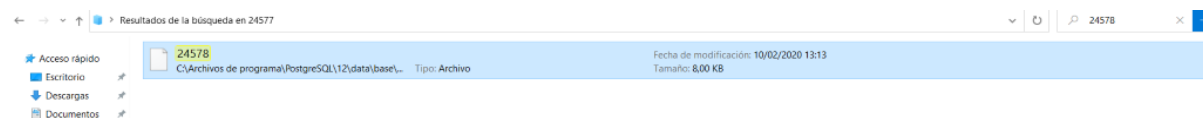


A continuación, insertamos dos tuplas en la tabla (dos insert) y podemos observar un cambio de tamaño del archivo con el OID 24578. Pasamos de 0 KB a 8 KB, dado que estamos usando/reservando un bloque entero, por eso ocupa 8kb que es el estándar en Postgres.

Nota: se ha cambiado el nombre de MiTabla a mitabla



Trigger Functions	
Types	
Views	
postgres	
Casts	
Catalogs	
Event Triggers	
Extensions	
Foreign Data Wrappers	
Languages	
Schemas	
Login/Group Roles	
Tablespaces	
	Last autovacuum
	Last analyze
	Last autoanalyze
	Vacuum counter
	Autovacuum counter
	Analyze counter
	Autoanalyze counter
	Table size
	Toast table size
	Indexes size



Además de lo comentado anteriormente, se ha producido una actualización, al principio, cuando no teníamos tuplas insertadas, el tamaño del índice era de 8KB. Al hacer los insert de las tuplas, éste aumenta a 16KB, esto es porque se ha reservado espacio en el índice a pesar de no estar lleno, por lo que nos lleva a pensar que los datos se almacenan unos detrás de otros y se necesitaban más de 8KB por lo que se ha reservado otro bloque de 8KB llegando a los 16 KB que no están llenos.

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?

El `pg_buffercache` es un módulo para poder examinar que está pasando con el buffer en tiempo real. Primero tenemos que instalar la extensión:


```
Suggestion [3,General]: No se encontró el comando psql, pero existe en la ubicación actual. Windows PowerShell no carga comandos de la ubicación actual de forma predeterminada. Si confía en este comando, escriba ".\psql". Vea "get-help about Command Precedence" para obtener información más detallada.
PS C:\Program Files\PostgreSQL\12\bin> .\psql -U postgres MiBaseDatos
Contraseña para usuario postgres:
psql (12.1)
ADVERTENCIA: El código de página de la consola (650) difiere del código
de página de Windows (1252).
Los caracteres de 8 bits pueden funcionar incorrectamente.
Vea la página de referencia de psql «Notes for Windows users»
para obtener más detalles.
Dígitelo «help» para obtener ayuda.

MiBaseDatos=# CREATE EXTENSION pg_buffercache;
MiBaseDatos=# CREATE EXTENSION pg_buffercache;
ERROR: error de sintaxis en o cerca de «CREATE»
LÍNEA 2: CREATE EXTENSION pg_buffercache;
          ^

MiBaseDatos=# CREATE EXTENSION pg_buffercache;
MiBaseDatos=# CREATE EXTENSION pg_buffercache;
ERROR: error de sintaxis en o cerca de «CREATE»
LÍNEA 2: CREATE EXTENSION pg_buffercache;
          ^

MiBaseDatos=# /dx
MiBaseDatos=# \dx
          Listado de extensiones instaladas
+-----+-----+-----+-----+
Nombre | Versión | Esquema | Descripción
+-----+-----+-----+-----+
pgsql  | 1.0     | pg_catalog | PL/pgSQL procedural language
(1 filas)

MiBaseDatos=# CREATE EXTENSION IF NOT EXISTS pg_buffercache;
ERROR: error de sintaxis en o cerca de «/»
LÍNEA 1: /dx
          ^

MiBaseDatos=# CREATE EXTENSION IF NOT EXISTS pg_buffercache;
ERROR: error de sintaxis en o cerca de «/»
LÍNEA 1: CREATE EXTENSION IF NOT EXISTS pg_buffercache;
          ^

MiBaseDatos=# CREATE EXTENSION IF NOT EXISTS pg_buffercache;
MiBaseDatos=# \dx
          Listado de extensiones instaladas
+-----+-----+-----+-----+
Nombre | Versión | Esquema | Descripción
+-----+-----+-----+-----+
pg_buffercache | 1.3 | public | examine the shared buffer cache
pgsql         | 1.0 | pg_catalog | PL/pgSQL procedural language
(2 filas)

MiBaseDatos=#
```

A continuación, realizamos una búsqueda en el manual y según lo hallado en la página 2518 aplicamos la consulta y obtenemos:

The screenshot shows a database management tool interface. On the left is a 'Browser' pane showing a tree view of the database structure, including 'Tables (1)' and 'Columns (5)'. The main area is the 'Query Editor' for 'MiBaseDatos/postgres@PostgreSQL 12'. It contains a SQL query to select the number of buffers for each relation name. Below the query editor is a 'Data Output' pane showing the results of the query.

Query Editor:

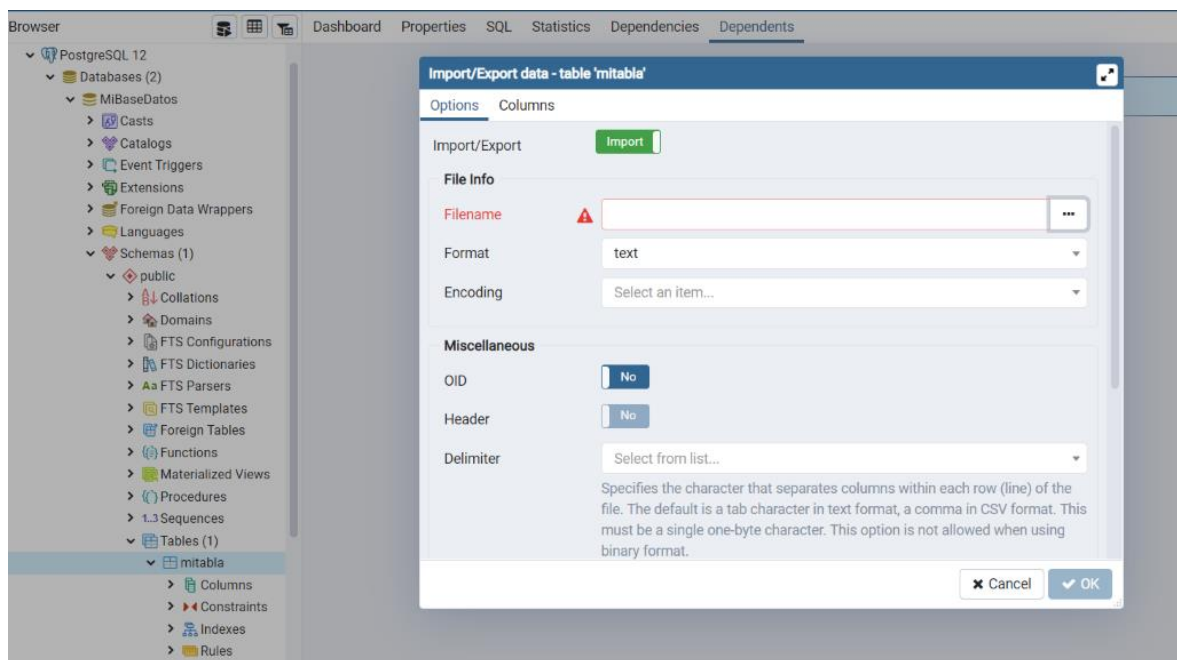
```
1 SELECT c.relname, count(*) AS buffers
2 FROM pg_buffercache b INNER JOIN pg_class c
3 ON b.relfilenode = pg_relation_filenode(c.oid) AND
4     b.reldatabase IN (0, (SELECT oid FROM pg_database
5                           WHERE datname =
6                             current_database()))
7 GROUP BY c.relname
8 ORDER BY 2 DESC
9 LIMIT 10;
```

Data Output:

relname	buffers
pg_attribute	42
pg_collation	22
pg_proc_pro...	18
pg_class	17
pg_statistic	16
pg_depend...	15
pg_toast_26...	14
pg_operator	14

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?

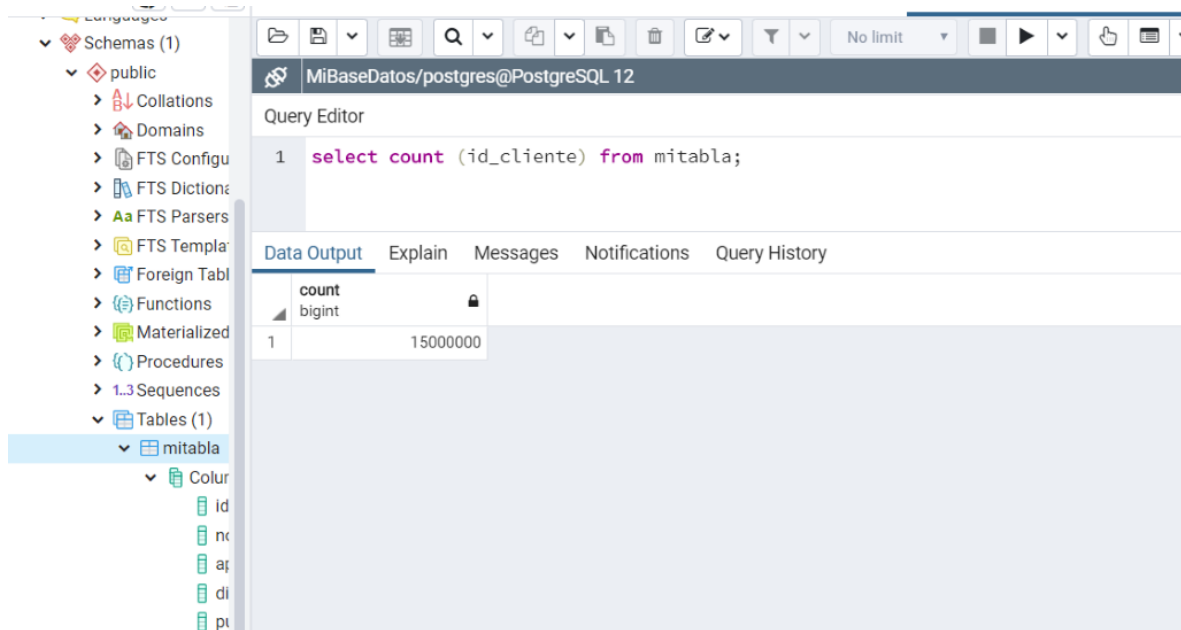
Primero pasamos a crear de nuevo la tabla y las columnas:



Resultados que se muestran tras cargar los datos:

	id_cliente [PK] integer	nombre text	apellido text	direccion text	puntos integer
1	337	nombre337	apellidos337	apellidos337	59
2	9020289	nombre902...	apellidos90...	apellidos902...	624
3	14519336	nombre145...	apellidos14...	apellidos145...	132
4	7724637	nombre772...	apellidos77...	apellidos772...	278
5	3943592	nombre394...	apellidos39...	apellidos394...	520
6	1343216	nombre134...	apellidos13...	apellidos134...	392
7	8733445	nombre873...	apellidos87...	apellidos873...	566
8	8878781	nombre887...	apellidos88...	apellidos887...	358
9	13149508	nombre131...	apellidos13...	apellidos131...	696
10	10893176	nombre108...	apellidos10...	apellidos108...	676
11	4456535	nombre445...	apellidos44...	apellidos445...	298
12	13492465	nombre134...	apellidos13...	apellidos134...	457
13	13523014	nombre135...	apellidos13...	apellidos135...	673
14	2470694	nombre247...	apellidos24...	apellidos247...	600
15	13602668	nombre136...	apellidos13...	apellidos136...	205
16	14043654	nombre140...	apellidos14...	apellidos140...	290
17	4626851	nombre462...	apellidos46...	apellidos462...	360
18	5931446	nombre593...	apellidos59...	apellidos593...	552

Mediante una consulta comprobamos que están todos los 15.000.000 de datos:

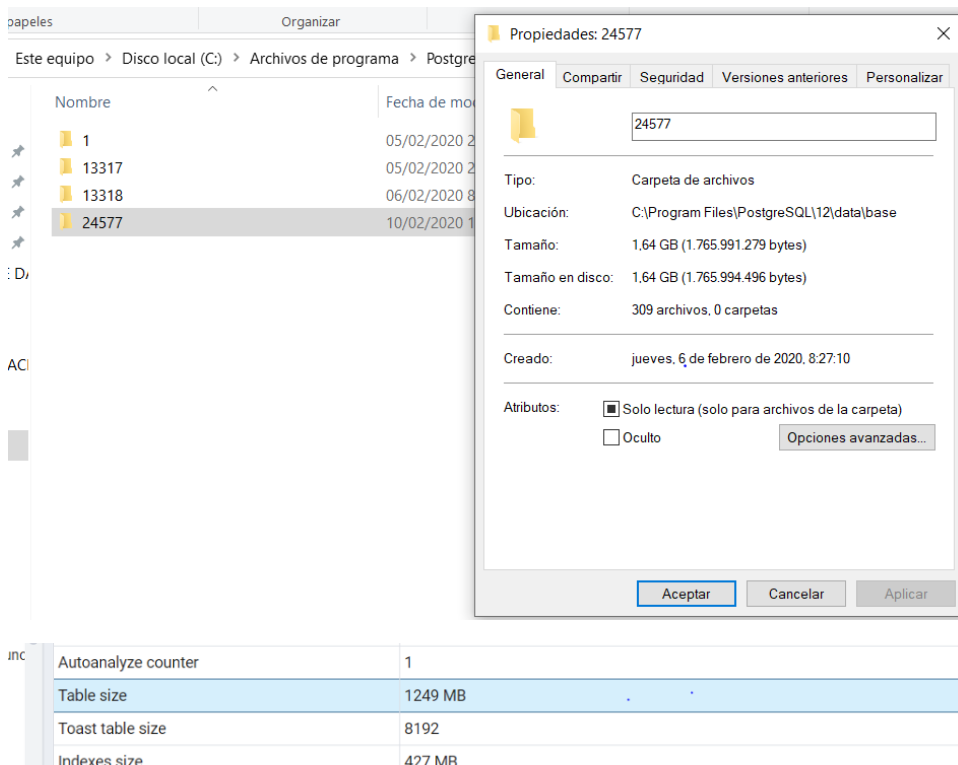


- ¿Cuánto ocupa la información original a insertar?

El archivo original ocupa 899MB

- ¿Cuánto ocupa la tabla ahora?

El tamaño de la base de datos es de 1.64GB y de la tabla 1.249GB



- ¿Por qué?

El tamaño del índice cambia porque al añadir o eliminar tuplas se necesita más bloques de índice lo que supone necesitar más memoria dado que la organización de registros es mediante montículo, por lo que se van rellenando a medida que encuentran un hueco libre.

- Calcular teóricamente el tamaño en bloques que ocupa la relación MiTabla tal y como se realiza en teoría.

Id_cliente:

<ul style="list-style-type: none"> FTS Templates Foreign Tables Functions Materialized Views Procedures 1.3 Sequences Tables (1) <ul style="list-style-type: none"> mitabla <ul style="list-style-type: none"> Columns (5) <ul style="list-style-type: none"> id_cliente nombre apellido direccion puntos 	<table> <tr> <th>Statistics</th><th>Value</th></tr> <tr> <td>Null fraction</td><td>0</td></tr> <tr> <td>Average width</td><td>4</td></tr> <tr> <td>Distinct values</td><td>-1</td></tr> <tr> <td>Most common values</td><td></td></tr> <tr> <td>Most common frequencies</td><td></td></tr> <tr> <td>Histogram bounds</td><td>{140,156113,308379,464623,630314,773830,916}</td></tr> <tr> <td>Correlation</td><td>0.0069718175</td></tr> </table>	Statistics	Value	Null fraction	0	Average width	4	Distinct values	-1	Most common values		Most common frequencies		Histogram bounds	{140,156113,308379,464623,630314,773830,916}	Correlation	0.0069718175
Statistics	Value																
Null fraction	0																
Average width	4																
Distinct values	-1																
Most common values																	
Most common frequencies																	
Histogram bounds	{140,156113,308379,464623,630314,773830,916}																
Correlation	0.0069718175																

Nombre:

- > FTS Templates
- > Foreign Tables
- > Functions
- > Materialized Views
- > Procedures
- > 1..3 Sequences
- > Tables (1)
 - mitabla
 - Columns (5)
 - id_cliente
 - nombre**
 - apellido
 - direccion
 - puntos

Statistics	Value
Null fraction	0
Average width	14
Distinct values	-1
Most common values	
Most common frequencies	
Histogram bounds	{nombre10000071,nombre10128719,nombre
Correlation	-0.007378914

- ¿Concuerda con el tamaño en bloques que nos proporciona PostgreSQL?

Lrc = longitud del registro = id_cliente + nombre+ apellido + dirección + puntos = 4 + 14 + 17 + 17 + 4 = **56 bytes**.

B = 8 KB por defecto en Postgres = **8192 bytes**

Nr = 15.000.000 registros

Factor de bloque $Fr = \text{tamaño bloque} / \text{longitud registro} = B / Lrc = 8192 / 56 = \mathbf{146 \text{ registros}}$
enteros caben en un bloque

$$Br = Nr / Fr = 15000000 / 146 = \mathbf{102740 \text{ bloques/archivo}}$$

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?

Las estadísticas nos muestran:

table_len	Tamaño de relación física en bytes
tuple_count	Nº Tuplas vivas
tuple_len	Tamaño total de tuplas vivas en bytes
tuple_percent	% Tuplas vivas
dead_tuple_count	Nº Tuplas muertas
dead_tuple_len	Tamaño total de las tuplas muertas
dead_tuple_percent	% Tuplas muertas
free_space	Total de espacio libre en bytes
free_percent	% Espacio libre

Tupla muerta: es una tupla/registro que ha sido eliminada pero aun así sigue consumiendo alguna parte de espacio, dado que aún tiene dependencias con tuplas vivas.

¿Cuál es el grado de ocupación de los bloques?

Si tenemos un 0.44% de espacio libre, eso implica que tenemos un 99,66% de ocupación de los bloques

¿Cuánto espacio libre queda? ¿Por qué?

El espacio libre es de 5761792 bytes. Porque no se han llenado, por la organización de registros en montículo que rellena como puede los huecos libres como puede.

```
MiBaseDatos=# CREATE EXTENSION IF NOT EXISTS pgstattuple;
CREATE EXTENSION
MiBaseDatos=# \dx
          Listado de extensiones instaladas
  Nombre | Versión | Esquema | Descripción
-----+-----+-----+-----
pg_buffercache | 1.3 | public | examine the shared buffer cache
pgstattuple | 1.5 | public | show tuple-level statistics
plpgsql | 1.0 | pg_catalog | PL/pgSQL procedural language
(3 filas)

MiBaseDatos=#
```

MiBaseDatos/postgres@PostgreSQL 12										
Query Editor										
1 Select * from pgstattuple('mitabla');										
Data Output										
	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	1310113792	15000000	1219555960	93.09	0	0	0	5761792	0.44	

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?

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?

Instalamos el módulo pageinspect, que proporciona funciones que nos permitirán inspeccionar los contenidos de la base de datos a bajo nivel, muy indicado sobre todo para corregir errores. Sólo puede ser usado por superusuarios.

The image shows two screenshots. The top one is a terminal window with the command `psql -d MiBaseDatos -c '\dx'` and its output, which lists installed extensions. The bottom screenshot is a PostgreSQL GUI (PgAdmin) showing a query in the Query Editor and its results in the Data Output pane.

Terminal Output:

```
MiBaseDatos=# \dx
Listado de extensiones instaladas

```

Nombre	Versión	Esquema	Descripción
pageinspect	1.7	public	inspect the contents of database pages at a low level
pg_buffercache	1.3	public	examine the shared buffer cache
pgstattuple	1.5	public	show tuple-level statistics
plpgsql	1.0	pg_catalog	PL/pgSQL procedural language

(4 filas)

PostgreSQL GUI:

Query Editor:

```
1 SELECT * FROM page_header (get_raw_page ('mitabla', 0));
2
```

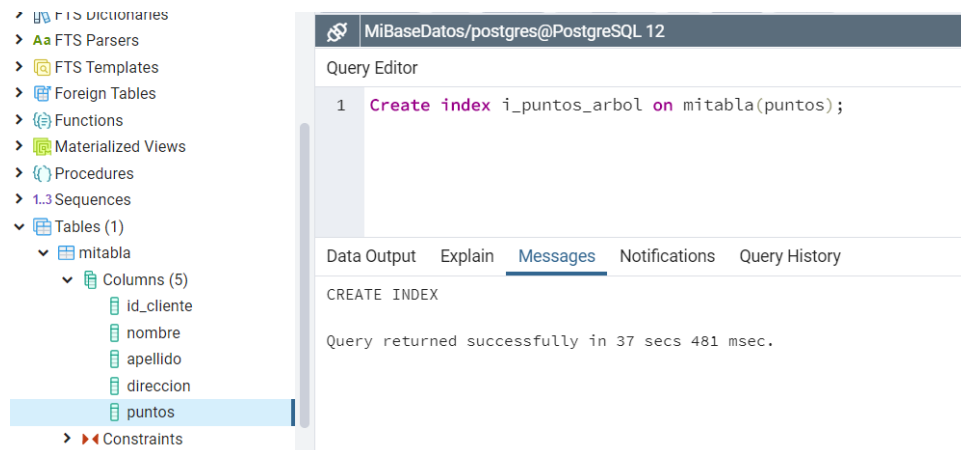
Data Output:

lsn	checksum	flags	lower	upper	special	pagesize	version	prune_xid
pg_lsn	smallint	smallint	smallint	smallint	smallint	smallint	smallint	xid
1 0/1711548	0	0	400	448	8192	8192	4	0

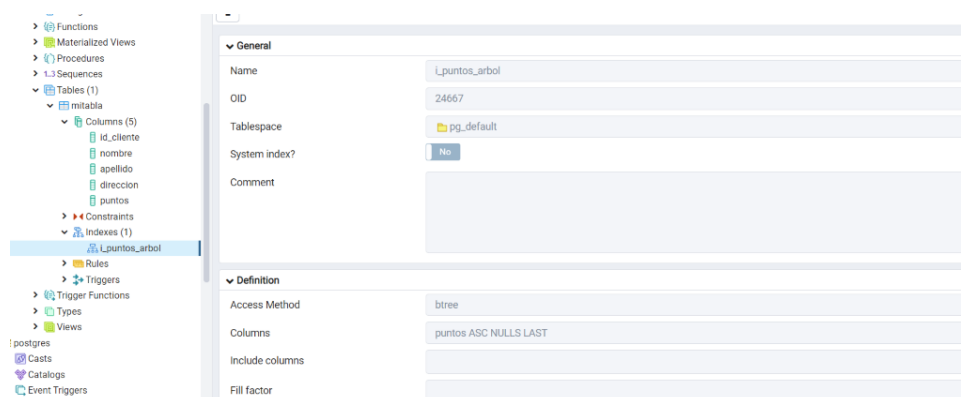
Averiguamos el número de bloques totales, y el medio (dividiendo entre 2) y el resultado lo colocamos sustituyendo el campo del 0.

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?

Creamos el índice y no especificamos nada más ya que postgres crea por defecto los índices de tipo árbol.



Buscamos el oid del índice que acabamos de crear en la pestaña de Properties. Como hemos creado el índice para el campo id_cliente, que pertenece a la tabla mitabla y ya sabemos su oid por ejercicios anteriores.



Buscaremos en el sistema dicha tabla cuyo oid es 24577 y dentro buscaremos el oid del índice que es 24667 y ya sabremos dónde está almacenado físicamente y su tamaño.

El tamaño será 239

Este equipo > Disco local (C:) > Archivos de programa > PostgreSQL > 12 > data > base > 24577				
Nombre	Fecha de modificación	Tipo	Tamaño	
1259	27/02/2020 9:22	Archivo	104 KB	
2608	27/02/2020 9:22	Archivo	456 KB	
2610	27/02/2020 9:22	Archivo	32 KB	
2690	27/02/2020 9:22	Archivo	88 KB	
2691	27/02/2020 9:22	Archivo	240 KB	
1249	27/02/2020 9:22	Archivo	432 KB	
1255	27/02/2020 9:22	Archivo	648 KB	
2674	27/02/2020 9:17	Archivo	344 KB	
2678	27/02/2020 9:17	Archivo	16 KB	
2679	27/02/2020 9:17	Archivo	16 KB	
3455	27/02/2020 9:17	Archivo	40 KB	
2658	27/02/2020 9:17	Archivo	128 KB	
2659	27/02/2020 9:17	Archivo	88 KB	
2662	27/02/2020 9:17	Archivo	32 KB	
2663	27/02/2020 9:17	Archivo	48 KB	
2673	27/02/2020 9:17	Archivo	280 KB	
24667	27/02/2020 9:17	Archivo	329.504 KB	

A continuación, tenemos las propiedades del índice que acabamos de crear, que nos mostrará la siguiente información:

Dashboard Properties SQL Statistics Dependencies Dependents MiBaseDa																																			
<ul style="list-style-type: none"> FTS Configurations FTS Dictionaries FTS Parsers FTS Templates Foreign Tables Functions Materialized Views Procedures Sequences Tables (1) <ul style="list-style-type: none"> mitabla <ul style="list-style-type: none"> Columns (5) <ul style="list-style-type: none"> id_cliente nombre apellido direccion puntos Constraints Indexes (1) <ul style="list-style-type: none"> L_puntos_arbol Rules Triggers Trigger Functions Types Views 	<table> <thead> <tr> <th>Statistics</th><th>Value</th></tr> </thead> <tbody> <tr><td>Index scans</td><td>0</td></tr> <tr><td>Index tuples read</td><td>0</td></tr> <tr><td>Index tuples fetched</td><td>0</td></tr> <tr><td>Index blocks read</td><td>41188</td></tr> <tr><td>Index blocks hit</td><td>0</td></tr> <tr><td>Index size</td><td>322 MB</td></tr> <tr><td>Version</td><td>4</td></tr> <tr><td>Tree level</td><td>2</td></tr> <tr><td>Index size-2</td><td>337412096</td></tr> <tr><td>Root block no</td><td>209</td></tr> <tr><td>Internal pages</td><td>203</td></tr> <tr><td>Leaf pages</td><td>40984</td></tr> <tr><td>Empty pages</td><td>0</td></tr> <tr><td>Deleted pages</td><td>0</td></tr> <tr><td>Average leaf density</td><td>90.19</td></tr> <tr><td>Leaf fragmentation</td><td>0</td></tr> </tbody> </table>	Statistics	Value	Index scans	0	Index tuples read	0	Index tuples fetched	0	Index blocks read	41188	Index blocks hit	0	Index size	322 MB	Version	4	Tree level	2	Index size-2	337412096	Root block no	209	Internal pages	203	Leaf pages	40984	Empty pages	0	Deleted pages	0	Average leaf density	90.19	Leaf fragmentation	0
Statistics	Value																																		
Index scans	0																																		
Index tuples read	0																																		
Index tuples fetched	0																																		
Index blocks read	41188																																		
Index blocks hit	0																																		
Index size	322 MB																																		
Version	4																																		
Tree level	2																																		
Index size-2	337412096																																		
Root block no	209																																		
Internal pages	203																																		
Leaf pages	40984																																		
Empty pages	0																																		
Deleted pages	0																																		
Average leaf density	90.19																																		
Leaf fragmentation	0																																		

Numero de Bloques (index blocks read) = 41188.

Número de niveles del árbol (Tree level) es 2.

Número de bloques tipo hoja del árbol (Leaf pages) es 40984.

Número de nodos intermedios del árbol (Internal Pages) es 203.

Bloques:

- $1+203+40984=41.188$ bloques totales

Bloque por Nivel:

Raíz=1 bloque

Intermedio= 203 bloques

Hoja= 40984 bloques.

Tuplas por Bloque:

Nivel Hoja= $[15000000 \text{ registros totales} / 40984 \text{ bloques hoja}] = 365,99 \rightarrow 366$ tuplas o registros por bloque.

Nivel Intermedio= $[366 \text{ registros} / 203 \text{ bloques intermedio}] = 1,80 \rightarrow 2$ Tuplas por Bloque.

Nivel Raíz= $[1,80 / 203] = 0,008 \rightarrow 1$ tupla

Niveles:

Nivel Raíz+ Nivel Intermedio + Nivel Hoja= 3 niveles.

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.

Hacemos una división entre el tamaño del índice, 329.504 Kb, que lo hemos obtenido de la imagen del ejercicio anterior y lo dividimos entre 8 kb que es el tamaño de bloque de Postgres, para obtener el número de bloques totales que hay en el índice.

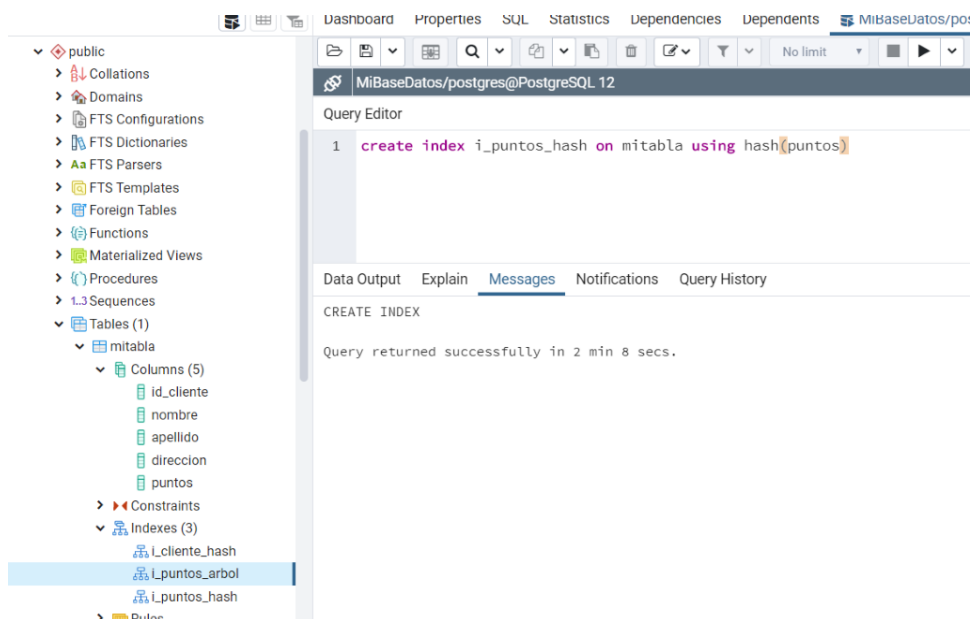
$B = 329,504$.

$[329.504 / 8] = 41.188$ bloques tendría el archivo, que coincide con los datos obtenidos en el ejercicio anterior al sumar los bloques de cada nivel.

Cuestión 12. Crear un índice de tipo hash para el campo id_cliente y otro para el campo puntos.

A diferencia de la cuestión 10, la consulta será diferente dado que esta vez especificamos que queremos un índice hash y no como antes que viene por defecto en Postgres el índice BTree.

The screenshot shows the PostgreSQL Query Editor interface. On the left, the database schema tree is visible, showing the 'public' schema and the 'mitabla' table. The 'mitabla' table has columns: id_cliente, nombre, apellido, direccion, and puntos. The 'Indexes' section shows the 'i_cliente_hash' index. The main area displays the SQL query: `create index i_cliente_hash on mitabla using hash(id_cliente);`. The 'Messages' tab shows the output: 'CREATE INDEX' and 'Query returned successfully in 1 min 7 secs.'



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?

Pgstattuple:

Usando pgstattuple hay una variación en los valores entre ambos índices, ya que cambia en % de espacio libre y la longitud de la tabla.

I_cliente_hash

Query Editor

```
1 Select * from pgstattuple ('i_cliente_hash');
2
```

Data Output Explain Messages Notifications Query History

	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	535683072	15000000	240000000	44.8	0	0	0	232789572	43.46

I_puntos_hash

Query Editor

```
1 Select * from pgstattuple ('i_puntos_hash');
2
```

Data Output Explain Messages Notifications Query History

	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	701440000	15000000	240000000	34.22	0	0	0	397648056	56.69

Pageinspect:

Podemos observar que entre ambos índices empleando el pageinspect no obtenemos ninguna

diferencia entre los valores obtenidos.

I_cliente_hash

```
1 SELECT * FROM page_header(get_raw_page('i_cliente_hash', 0));
2
```

	lsn	checksum	flags	lower	upper	special	pagesize	version	prune_xid
	pg_lsn	smallint	smallint	smallint	smallint	smallint	smallint	smallint	xid
1	0/F4FE0690		0	4568	8176	8176	8192	4	0

I_puntos_hash

```
1 SELECT * FROM page_header(get_raw_page('i_puntos_hash', 0));
2
```

	lsn	checksum	flags	lower	upper	special	pagesize	version	prune_xid
	pg_lsn	smallint	smallint	smallint	smallint	smallint	smallint	smallint	xid
1	1/36047220		0	4568	8176	8176	8192	4	0

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.

The screenshot shows a PostgreSQL client interface with a sidebar on the left listing database objects. The main window is titled "MiBaseDatos/postgres@PostgreSQL 12" and contains a "Query Editor" with the following SQL command:

```
1 DELETE FROM mitabla WHERE random() > 13000000
```

Below the query editor, the "Messages" tab is active, showing the execution result:

```
DELETE 0
```

Below the messages, a status message indicates: "Query returned successfully in 3 secs 328 msec."

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.