PRÁCTICA 1 – BBDD

Marcos Barranquero Fernández – 51129104N

Eduardo Graván Serrano – 03212337L

PREGUNTAS DE LA PRÁCTICA

CUESTIÓN 1: CREAR UNA NUEVA BASE DE DATOS QUE SE LLAME **MIBASEDATOS**. ¿EN QUÉ DIRECTORIO SE CREA DEL DISCO DURO Y CUANTO OCUPA EL MISMO? ¿POR QUÉ?

Se crea en "C:\Program Files\PostgreSQL\10\data\base\24610" y ocupa 7,38 MB.

A pesar de no haber insertado ningún registro, no está vacía. Esto es debido a que se crean todos los archivos iniciales tales como el esquema público, estadísticas iniciales, etc.

CUESTIÓN 2: CREAR UNA NUEVA TABLA QUE SE LLAME **MITABLA** QUE CONTENGA UN CAMPO QUE SE LLAME CÓDIGO DE TIPO INTEGER QUE SEA LA PRIMARY KEY, OTRO CAMPO QUE SE LLAME NOMBRE DE TIPO TEXT, OTRO QUE SE LLAME DESCRIPCIÓN DE TIPO TEXT Y OTRA REFERENCIA QUE SEA DE TIPO INTEGER. ¿QUÉ FICHEROS SE HAN CREADO EN ESTA OPERACIÓN? ¿QUÉ GUARDAN CADA UNO DE ELLOS? ¿CUÁNTO OCUPAN? ¿POR QUÉ?

Al ejecutar:

```
CREATE TABLE public.mitabla
(
    codigo integer,
    nombre text,
    descripcion text,
    referencia integer,
    PRIMARY KEY (codigo)
)
WITH (
    OIDS = FALSE
);

ALTER TABLE public.mitabla
    OWNER to postgres;
```

Se han creado los ficheros 24611,24614, 24616, 24617. Usando el módulo oid2name con el siguiente comando obtenemos los siguientes resultados:

```
oid2name.exe -U postgres -d MiBaseDatos -f 2461X
```

FICHERO	ASOCIADO A	TAMAÑO	POR QUÉ
24611	mitabla	0 KB	Porque aún no contiene ningún dato.
24614	pg_toast 24611	0 KB	Porque no hay ninguna tupla en la tabla que exceda el tamaño de su bloque. De hecho, no hay ninguna tupla en la tabla.
24616	pg_toast_index	8 KB	Archivo de índices asociado al toast.
24617	mitabla_pk	8 KB	Archivo de claves primarias.

CUESTIÓN 3. INSERTAR UNA TUPLA EN LA TABLA. ¿CUÁNTO OCUPA AHORA LA TABLA? ¿SE HA PRODUCIDO ALGUNA ACTUALIZACIÓN MÁS? ¿POR QUÉ?

El archivo asociado a la tabla (24611) ocupa ahora 8 KB, debido a que se ha insertado una nueva tupla.
El archivo asociado a las primary keys (24617) ocupa ahora 16 KB, debido a la nueva tupla insertada.
Los otros dos siguen ocupando lo mismo, debido a que no se ha almacenado nada que exceda el límite de tamaño de bloque.

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? ¿POR QUÉ?

Tras crear el módulo pg_buffercache, ejecutamos lo siguiente:

```
SELECT c.relname, count(*) AS buffers
             FROM pg_buffercache b INNER JOIN pg_class c
             ON b.relfilenode = pg_relation_filenode(c.oid) AND
                b.reldatabase IN (0, (SELECT oid FROM pg_database
                                      WHERE datname = current_database()))
             GROUP BY c.relname
             ORDER BY 2 DESC
```

Observamos que tenemos 102 tablas con distinta cantidad de buffers dedicados. Destacan, por ejemplo, la tabla pg_proc o la tabla pg_attribute por su gran cantidad de buffers dedicados. Sin embargo, nuestra tabla, "MiTabla", solo posee un buffer dedicado en estos momentos. Esto es debido a que tan sólo hemos insertado una tupla, por lo que no ha sido necesario más de un buffer para insertar dichos datos.

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 eliminamos la tabla. Después, la volvemos a crear. Para insertar los datos, elegimos insert y el archivo, marcando como delimitador el ";". Tarda 213 segundos en importar todos los datos.

La información original a insertar (el archivo datos_mitabla.txt) ocupa 574.245 KB.

Ejecutando la función para ver el tamaño de tabla:

```
select pg_relation_size('mitabla');
```

Observamos que la table ocupa 918642688 KB. (Unos 897112 MB). Vemos que hay bastante diferencia de tamaño. Esto es debido a la distinta forma de almacenar los datos y el formato de archivo. PostgreSQL almacena los datos asignando espacios fijos para los enteros y variables para los textos, frente a todo el formato texto que guarda el .txt.

CÁLCULO TEÓRICO DEL NÚMERO DE BLOQUES

Tenemos:

2 enteros en el registro * tamaño integer = 2 x 4 bytes/int = 8 bytes.

Tamaño medio del texto descripción * tamaño char = 18 chars de media * 1 byte/carácter = 18 bytes.

Tamaño medio del texto nombre * tamaño char = 18 chars de media * 1 byte/carácter = 15 bytes.

Tamaño medio del registro = (8 + 18 + 15) = 41 bytes/registro.

Los bloques de PostgreSQL son de 8192 bytes.

En un bloque caben, por tanto, (8192 bytes/bloque / 41 bytes/registro) = **200 registros/bloque.**

Si tenemos 12 millones de registros, deberían haber (12000000 registros / 200 registros/bloque) = **60000 bloques de registros.**

VISTA REAL DEL NÚMERO DE BLOQUES

Vemos que **tenemos 112138 páginas (bloques)** almacenando nuestra tabla. Si tenemos 12 millones de registros, habrá unos 107 registros por bloque. Esto es debido a la cabecera de página, a ciertos elementos al final de página que también ocupan bytes, y a que los bloques no se rellenan completamente.

Por lo tanto, **no concuerda el cálculo teórico con el cálculo real** por las razones explicadas arriba.

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?

Observamos que ahora las relaciones “mitabla_pkey” y “mitabla” se llevan la mayoría de buffers:

	relname name	buffers bigint
1	mitabla_pkey	15929
2	mitabla	224
3	pg_proc	27
4	pg_proc_proname_args_nsp_index	13
5	pg_class	11
6	pg_attribute	8

Esto es debido a que hemos metido muchísimos datos en esas relaciones, y por tanto se han asignado más buffers a esas tablas. Deducimos que existe tal cantidad de buffers para el archivo asociado a las PK debido a que primero realizará las operaciones de inserción de datos en la tabla, y tras esto la asociación con el archivo de las PK. Por ello, primero, existirá gran cantidad de buffers a "mitabla", y tras esto, se liberarán esos buffers para trabajar con "mitabla_pkey".

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? ¿CUÁL ES EL FACTOR DE BLOQUE REAL DE LA TABLA? COMPARAR CON EL FACTOR DE BLOQUE TEÓRICO OBTENIDO.

Ejecutando:

```
select * from pgstattuple('mitabla')
```

Se muestran las estadísticas asociadas a las tuplas: el tamaño de tabla, la cantidad de tuplas que hay, la longitud de las tuplas, el porcentaje de tuplas vivas y muertas, etc.

Vemos que el porcentaje de espacio libre es un 0,47%. El porcentaje de ocupación es, por tanto, un 99,53%.

Quedan 4307312 bytes libres. Esto es debido a que los bloques no se completan con los datos.

Como hemos visto en la cuestión 5, teóricamente el factor de bloque debería ser 107 reg/bloque.

En la práctica, veíamos antes que en realidad hay una media de 199 registros/bloque.

Esto indica que los bloques no se encuentran ocupados al 100%, sino que hay algo de espacio libre por bloque.

CUESTIÓN 8 CON EL MÓDULO PAGEINSPECT, ANALIZAR LA CABECERA 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?

Ejecutando:

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

Vemos lo siguiente:

Primera página

Data Output | Explain | Messages | Notifications | Query History

	lsn pg_lsn	checksum smallint	flags smallint	lower smallint	upper smallint	special smallint	pagesize smallint	version smallint	prune_xid xid
1	0/9CB0...	0	0	452	504	8192	8192	4	0

Página intermedia

	lsn pg_lsn	checksum smallint	flags smallint	lower smallint	upper smallint	special smallint	pagesize smallint	version smallint	prune_xid xid
1	0/D631...	0	0	452	496	8192	8192	4	0

Página final

	lsn pg_lsn	checksum smallint	flags smallint	lower smallint	upper smallint	special smallint	pagesize smallint	version smallint	prune_xid xid
1	1/291E...	0	0	340	2504	8192	8192	4	0

Vemos ciertas similitudes, como el tamaño de bloque y la versión que utilizan, coinciden siempre. También, en la mayoría de páginas, lower y upper, parámetros que marcan el comienzo y final de espacio libre, son bastante similares.

Como hemos visto en la pregunta anterior, las páginas no se ocupan completamente. Por ello, la mayoría de páginas tienen algo de espacio libre.

En la última página, debido a que no está tan completa como el resto de páginas, existe más espacio libre. Por tanto, lower empieza antes y upper señala que hay más desplazamiento (espacio libre) hasta alcanzar el final de página o bloque.

CUESTIÓN 9. ANALIZAR LOS ELEMENTOS QUE SE ENCUENTRAN EN LA PÁGINA DEL PRIMER BLOQUE, DEL BLOQUE SITUADO EN LA MITAD DEL ARCHIVO Y DEL ÚLTIMO BLOQUE DE LA TABLA **MITABLA**. ¿QUÉ DIFERENCIAS SE APRECIAN ENTRE ESOS BLOQUES? ¿POR QUÉ?

Para ver los elementos de páginas utilizamos:

```
SELECT * FROM heap_page_items(get_raw_page('mitabla', numero_pagina));
```

Al ejecutarlo nos salen 107 registros.

El primer archivo muestra que está lleno desde la posición 504 hasta el final. Vemos que la longitud de los elementos es de 72 bytes, aunque algunos de ellos ocupan 68 bytes y dejan libres los 4 restantes.

En la página del medio, vemos que el desplazamiento a final de espacio libre comienza un poco antes, y por lo demás es similar a la página 0.

En la página final, sobresale que el desplazamiento a final de espacio libre (desde donde empieza a estar ocupada) es en la posición 2504, frente a las posiciones 496-504 de las páginas anteriores. Esto demuestra una vez más, que por ser la última aún no está tan llena como las demás. Al ejecutar el comando, nos salen tan solo 79 registros.

CUESTIÓN 10. CREAR UN ÍNDICE DE TIPO ÁRBOL PARA EL CAMPO CODIGO. ¿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?

Para crear un índice de árbol sobre el campo código ejecutamos:

```
CREATE INDEX indice_codigo_arbol ON mitabla(codigo);
```

Tras unos segundos, tenemos nuestro índice creado. Se almacena físicamente en “C:\Program Files\PostgreSQL\10\data\base\24610” como 24670. Ocupa 263240 KB.

Para ver el número de bloques, sabemos que ocupa 263240 KB / 8 KB/bloque = **32905 bloques**.

Viendo las estadísticas, vemos que el árbol **tiene 2 niveles + raiz.**

La raíz se encuentra en la página 290.

En el nivel 1 tiene 117 páginas.

En el nivel 2 tiene 32787 páginas.

Para ver la información de las tuplas ejecutamos:

```
SELECT * FROM pgstatindex('indice_codigo_arbol');
```

Vemos que la media de tuplas de hojas son 90’09 tuplas/hoja.

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

CÁLCULO TEÓRICO

Ni puta idea de qué responder pt2.

CUESTIÓN 12. CREAR UN ÍNDICE DE TIPO HASH PARA EL CAMPO CÓDIGO Y OTRO PARA EL CAMPO REFERENCIA.

Introduzco en consola:

```
CREATE INDEX índice_codigo ON mitabla USING hash (codigo);
```

```
CREATE INDEX indice_referencia ON mitabla USING hash (referencia);
```

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?

Resultado de PGSTATTUPLE con el índice hash sobre código:

	version integer	bucket_pages bigint	overflow_pages bigint	bitmap_pages bigint	unused_pages bigint	live_items bigint	dead_items bigint	free_percent double precision
1	4	40960	385	1	0	12000000	0	28.7927728462158

Resultado de PGSTATTUPLE con el índice hash sobre referencia:

	version integer	bucket_pages bigint	overflow_pages bigint	bitmap_pages bigint	unused_pages bigint	live_items bigint	dead_items bigint	free_percent double precision
1	4	40960	29217	1	0	12000000	0	58.0480384360516

Última página índice hash código(41346):

	live_items integer	dead_items integer	page_size integer	free_size integer	hasho_prevblkno bigint	hasho_nextblkno bigint	hasho_bucket bigint	hasho_flag integer	hasho_page_id integer
1	4	0	8192	8068	32701	4294967295	32700	1	65408

Última página índice hash referencia(70178):

Data Output | Explain | Messages | Notifications | Query History

	live_items integer	dead_items integer	page_size integer	free_size integer	hasho_prevblkno bigint	hasho_nextblkno bigint	hasho_bucket bigint	hasho_flag integer	hasho_page_id integer
1	392	0	8192	308	70177	4294967295	40687	1	65408

Observando la ruta de la base de datos, vemos que el índice sobre referencia ocupa 561432 KB y el índice hash sobre código ocupa 330776 KB.

Esto se refleja también en el número de páginas que encontramos en los índices hash de referencia y código con el módulo pageinspect. (41346 vs 70178).

Esto es congruente con el campo que denota el espacio libre, donde vemos que en el hash de código queda menos espacio libre que en el del índice hash de referencia.

Al inspeccionar la última página en ambos, vemos que realmente no es la última, sino que es la última sin overflow. (Ambas comparten el mismo nº de ID en sus respectivas tablas). Sin embargo, el hash sobre referencia contiene muchísimas más paginas con overflow respecto al hash de código.

Todo ello es congruente porque en nº de referencia en la tabla escala más rápido que el nº de código. Podemos ver esto consultando las primeras 100 tuplas de la tabla.

CUESTIÓN 14. ACTUALIZAR LA TUPLA CON CÓDIGO 7000020 PARA PONER LA REFERENCIA A 240. ¿QUÉ OCURRE CON LA SITUACIÓN DE ESA TUPLA DENTRO DEL FICHERO? ¿POR QUÉ?

Ejecutamos:

```
UPDATE mitabla SET referencia = 240 WHERE codigo = 7000020;
```

Ni puta idea de qué contestar. No sé como seleccionar una página dada una tupla en concreto…

```
--select * from mitabla where(codigo=7000020)
--UPDATE mitabla SET referencia = 240 WHERE codigo = 7000020;
--SELECT * FROM pgstattuple('mitabla');
SELECT * FROM mitabla WHERE(CODIGO>7000000 and codigo<7001000) order by
(codigo)
```

ANEXO – MÓDULOS Y PROGRAMAS

OID2NAME

Oid2name permite ver el OID de tablas y archivos, y ver información de las mismas.

PG_BUFFERCACHE

Este módulo permite consultar el buffer de caché compartido en tiempo real. Para crear esta extensión debemos escribir:

```
create extension pg_buffercache
```

Una vez ejecutado, podremos realizar las consultas a los buffers.

INFORMACIÓN ÚTIL SOBRE BLOQUES

“Lo primero que tenemos que decir es que la unidad mínima de almacenamiento en PostgreSQL se denomina, indistintamente, página (page) o bloque (block). Un bloque en PostgreSQL ocupa siempre por defecto 8K si no se ha definido un valor diferente durante la compilación. Esto independientemente de si se usa en su totalidad o solo parcialmente.”

Para ver el nº de bloques de un archivo basta con hacer: Tamaño(KB) / (8KB/BLOQUE)

PGSTATTUPLE

Devuelve información sobre las tuplas, tales como los tamaños de tupla, porcentajes de uso, etc.

Para crearla, ejecutamos:

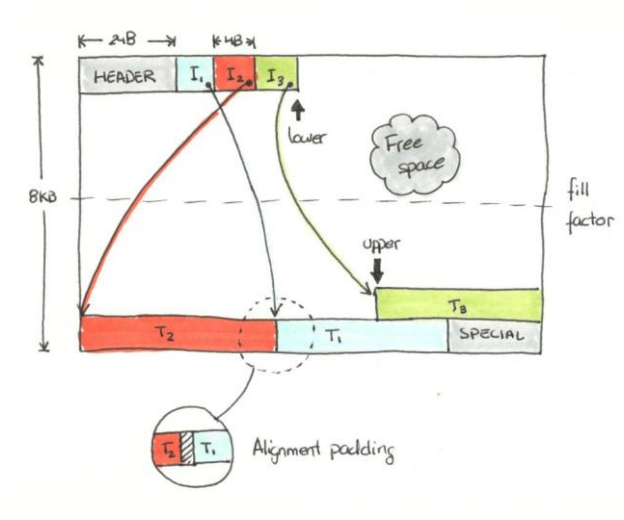
```
CREATE EXTENSION pgstattuple;
```

Para consultar una tabla, ejecutamos:

```
SELECT * FROM pgstattuple('”Nombre_de_tabla”');
```

PAGEINSPECT

Este módulo proporciona información sobre los bloques a bajo nivel. El formato de un bloque en PostgreSQL es el siguiente:



Consultar cabecera:

```
SELECT * FROM page_header(get_raw_page('pg_class', 0));
```

También se puede consultar el número de bloques de una tabla haciendo prueba y error con el comando de arriba.

FUENTES

Información sobre bloques: https://e-mc2.net/es/donde-estan-nuestros-datos-en-el-disco

Pageinspect: https://www.postgresql.org/docs/10/static/pageinspect.html

Pgstattuple: https://www.postgresql.org/docs/9.5/static/pgstattuple.html

Pgbuffercache: https://www.postgresql.org/docs/9.1/static/pgbuffercache.html

Tipos de datos: https://www.postgresql.org/docs/9.1/static/datatype-numeric.html

Tamaños de DB e índices: http://www.postgresqltutorial.com/postgresql-database-indexes-table-size/