Titulación: Grado en Ingeniería Informática y Sistemas de Información
Curso: 2018-2019. Convocatoria Ordinaria de Enero
Asignatura: Bases de Datos Avanzadas – Laboratorio

Practica 1: Arquitectura PostgreSQL y almacenamiento físico

ALUMNO 1:

Nombre y Apellidos:

DNI:

ALUMNO 2:

Nombre y Apellidos:

DNI:

Fecha: 24/9/2018

Profesor Responsable: Iván Gonzalez

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 puntuará **TODA** la práctica como Suspenso – Cero.

Plazos

Trabajo de Laboratorio: Semana 17 Septiembre, 24 Septiembre, 1 de Octubre, 8 de Octubre y 15 de Octubre.

Entrega de práctica: Día 22 de Octubre. 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 10.4. 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 y cuanto ocupa el mismo? ¿Por qué?

(Dentro de arquitectura Ubuntu Server 16.04 – Esta versión no soporta completamente la versión 10 y genera un directorio 9.5, que igualmente accesible por psql 10)

La base de datos se ha creado en el directorio /var/lib/postgresql/9.5/main/base/16386 con un tamaño de 6,84 MB.

Se ha generado en ese directorio pues el directorio de datos predefinido de postgresql10.

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?

Se ha creado el archivo 16404 que recoge la pkey de MiTabla, el archivo 16398 que recoge la estructura de la tabla y otros dos archivos que son el 16401 y 16403 encargado de mantener el toast y su index de la tabla para evitar excesos de uso de memoria de las row.

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

El archivo de la tabla ocupar ahora 8KB más y se ha producido una actualización en el archivo de pkeys por lo que ahora ocupa 8KB más. Esto se debe a la inserción de datos en la tabla y un nuevo row en la tabla de pkeys con la correspondiente clave.

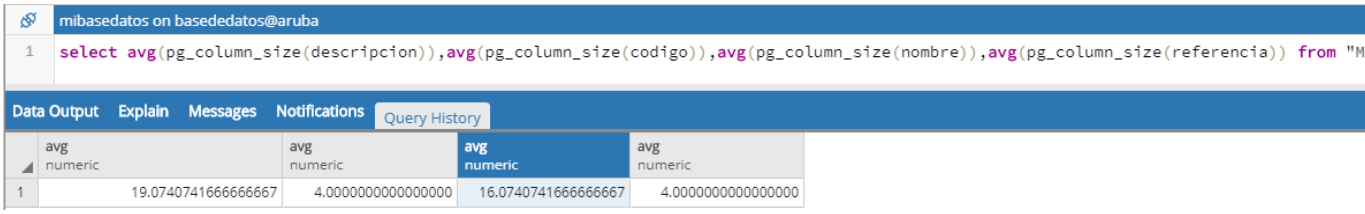
Cuestión 4. Aplicar el módulo pg_buffercache a la base de datos **MiBaseDatos.** ¿Es lógico lo que se muestra referido a la base de datos? ¿Por qué?

Si, debido a que como 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 son 574245KB mientras que los datos insertados en la tabla ocupan 897112 KB en la tabla MiTabla y 342016KB en la tabla MiTabla_pkey.

Sabiendo que la longitud media de las row es la incluida en la siguiente imagen:



B = 8192 B

Lr= 45 B

Nr = 12 * 10^6 registros

Fr= ⌊B/Lr⌋ =182 reg/bloq

br= ⌈Nr/Fr⌉ =65935 Bloques

El tamaño de los bloques no concuerda ya que el postgres no solo se ha guardado los 12M de registros, sino que también guardan el índice del archivo de datos con la primary key. Ya que el número de bloques reales es 112139.

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?

Ahora las tablas de MiTabla y MiTabla_pkey muestran 476 y 15351 caches. Esto indica que, al introducir nuevos registros, postgres automáticamente asigna nuevos caches al mantenimiento y acceso de esa tabla. Esto lo hace para optimizar la base de datos. Qué al fin y al cabo son el número de bloques en la ram.

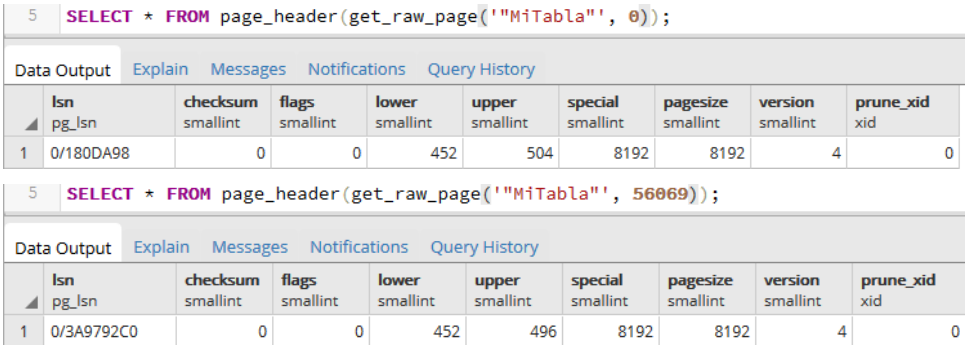
Cuestión 7. Aplicar el módulo pgstattuple a la tabla **MiTabla**. ¿Qué se muestra en las estadísticas? ¿Cuál es el grado de ocupación de los bloques? ¿Cuánto espacio libre queda? ¿Por qué? ¿Cuál es el factor de bloque real de la tabla? Comparar con el factor de bloque teórico obtenido.

Muestra información sobre el espacio utilizado por las tuplas y el espacio libre, el número de tuplas de la tabla, el número de tuplas muertas en la tabla, así como el porcentaje de estas últimas.

El grado de espacio libre es de 0.47 luego el grado de ocupación es de 0.53

Quedan 4307312 bytes de espacio libre, 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 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?



```
5  SELECT * FROM page_header(get_raw_page('"MiTabla"', 112138));
```

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	0/8D10A528	0	0	340	2504	8192	8192	4	0

Se diferencian en los campos lower y upper, que son indicadores que avisan cuando una página está llena, esto es así cuando no se puede agregar nada entre pd_lower y pd_upper. Por lo que se puede comprobar que en el último bloque hay mucho más espacio libre que en el primero y el que se encuentra a la mitad. Esto se debe ya que los registros se van colocando secuencialmente por lo que el último bloque es el que más espacio libre tiene.

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?

En la página del primer bloque y en el de la mitad, encontramos un mayor número de elementos. Esto se debe a que estos bloques están completos, mientras que el ultimo no está lleno y por tanto tiene menos elementos. Concordando así con los resultados obtenidos en el anterior apartado.

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?

Se almacena en el directorio /var/lib/postgresql/9.5/main/base/16386/ junto con los demás archivos de la base de datos. Tiene un total de 263240 KB.

Lt / Lb = 269557760 / 8192 = 32905 Bloques.

Tiene 2 niveles.

Nivel Hoja: 32787 Bloques

Nivel Intermedio: 117 Bloques

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.

Suponiendo que L_K=4 bytes; LP_B y LP_R= 8 bytes; B=8192 bytes...

$Nh*(L_K+LP_R)+LP_B<=B$; $12Nh <=8184$; Nh=682 punteros a registro

$N*LP_B+(N-1)*L_k<=B$; $12N<=8188$; N=683 punteros a bloque

$\lceil 12*10^6/682 \rceil$ =17596 bloques, $\lceil 17596/683 \rceil$ =26 bloques; $\lceil 26/683 \rceil$ = 1 bloque.

Total de niveles= 3, total de bloques=17623 bloques.

Cuestión 12. Crear un índice de tipo hash para el campo código y otro para el campo 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?

Con pgstattuple podemos comprobar que el hash de el atributo codigo ocupa menos que el hash de referencia debido a que es clave y no necesita cajones de punteros.

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?

Al cambiar la tupla, han tenido que ser actualizada la tabla MiTabla y MiTabla_pkey así como todos los índices que se han creado anteriormente

Cuestión 15. Borrar las tuplas de la tabla **MiTabla** con código entre 7.000.000 y 8.000.000 ¿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?

Realmente lo que postgres realiza al borrar, es cambiar un bit referencia que indica si la tupla está muerta o no. Por lo que la información persiste y el fichero no baja de tamaño.

Cuestión 16. Insertar una nueva tupla que contenga la información de (7.500.010,producto7500010,descripcion7500010,187). ¿En qué bloque y posición de bloque se inserta esa tupla? ¿Por qué?

Haciendo la consulta:

```
Select CTID from "MiTabla" where codigo = 7500010;
```

Postgres nos indica que se encuentra en el bloque 112138 en la posición 81.

Se ha insertado al final ya que, aunque hemos “borrado” las tuplas anteriores, como continúan estando almacenadas no hay espacio en esas posiciones y por tanto lo introduce en el último.

Cuestión 17. En la situación anterior, ¿Qué operaciones se puede aplicar a la base de datos 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. ¿Dónde se encuentra ahora la tupla de la cuestión 14 y 16? ¿Por qué?

Podemos utilizar la función vacuum para reclamar el espacio de las tuplas muertas, generando así una copia de la tabla sin estos rows, que pasa a ser la utilizada.

La tupla de la cuestión 16 se encuentra ahora en el bloque 102798 posición 29, mientras que la tupla de la cuestión 14 ya no se encuentra almacenada

Cuestión 18. Crear una tabla denominada **MiTabla2** de tal manera que tenga un factor de bloque que sea un tercio que la de la tabla **MiTabla** y cargar el archivo de datos anterior Explicar el proceso seguido y qué es lo que ocurre físicamente.

Se ha creado una tabla con el atributo fillfactor=33 (tercio del por defecto en postgres) y se han cargado los datos.

La introducción de datos tarda más debido a que, al tener un menor factor de bloque, debe acceder a un mayor número de bloques para introducir los datos, produciendo así que tarde más en realizar la operación.

Cuestión 19. Insertar una nueva tupla que contenga la información de (33.500.010,producto3500010,descripcion13500010,185) en la tabla **MiTabla2**. ¿En qué bloque y posición de bloque se inserta esa tupla? ¿Por qué?

Se encuentra en el bloque 335586 posición 14. Esto se debe a que al tener un menor fillfactor, se utilizan muchos más bloques y por tanto es lógico que al introducir datos en esta tabla, se encuentre en un bloque muy superior a cómo podría haber sido en la otra.

Cuestión 20. Actualizar la referencia con código 9.000.010 de la tabla **MiTabla2** para poner la referencia a 350 ¿Qué ocurre con la situación de esa tupla dentro del fichero? ¿Por qué?

Cambia la posición del row pero no cambia el bloque en el que se encuentra ya que esta ordenado.

Cuestión 21. A la vista de las pruebas realizadas anteriormente, ¿se puede obtener alguna conclusión sobre la estructura de los archivos que utiliza PostgreSQL y el manejo de las tuplas dentro de los archivos?

Las nuevas tuplas se insertan al final de bloque. Los datos no se borran, si no que se marcan como "Muertos" mediante a un bit indicador los cuales luego pueden ser borrados por la función vacuum para la optimización de la base de datos. Postgres tiene un factor de bloque del 100% por defecto, pero puede ser modificado para la optimización de consultas, aunque provoca un mayor uso de bloques en la introducción de datos.

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 de 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 en tiempo real graficas con el número de sesiones de bases de datos, transacciones por segundo, tuplas insertadas y leídas, así como bloqueos por segundo

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

Se ha creado una query dentro de la herramienta pgadmin y se ha introducido la orden sql :

```
CREATE INDEX ArbolMiTabla ON "MiTabla" USING BTREE (referencia);
```

Cuestión 24. Crear un índice hash sobre el campo referencia.

Se ha creado una query dentro de la herramienta pgadmin y se ha introducido la orden sql :

CREATE INDEX HashMiTabla ON "MiTabla" USING HASH (referencia);

Cuestión 25. Crear un índice sobre el campo código de tipo btree y otro de tipo hash sobre el mismo campo.

Se ha creado una query dentro de la herramienta pgadmin y se ha introducido la orden sql :

CREATE INDEX ArbolMiTabla2 ON "MiTabla" USING BTREE (codigo);

CREATE INDEX HashMiTabla2 ON "MiTabla" USING HASH (codigo);

Cuestión 26. Analizar el tamaño de cada índice creado y compararlos entre sí. ¿Qué conclusiones se pueden extraer de dicho análisis?

ArbolMiTabla = 269557760

HashMiTabla = 776232960

ArbolMiTabla2 = 269557760

HashMiTabla2 = 536887296

Siempre es más recomendable utilizar un índice BTREE a la hora de realizar consultas en esta tabla pues al ocupar menos, se debe acceder a un menor número de bloques y por tanto es más óptimo.

Cuestión 27. 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:

1. Mostar la información de las tuplas con código=9.001.000.

Se ha realizado un escaneo por index usando hashmitabla2

Se ha indexado por la condición

Leyendo 95993 bloques de pila y dos bloques de índice.

2. Mostar la información de las tuplas con código=90.001.000.

Se ha realizado un escaneo por index usando hashmitabla2

Se ha indexado por la condición

Leyendo 95897 bloques en pila y 1 bloque de índice

3. Mostrar la información de las tuplas con código <2000.

Se ha realizado un Bitmap Heap Scan en mitabla

Se ha recomprobado la condición

Se han comprobado 1979 bloques heap

Se ha realizado un escaneo bitmap por index en arbolmitabla2

Se ha indexado por la condición

Leyendo 97711 bloques de pila y 11 bloques de índice

4. Mostrar el número de tuplas cuyo código >2000 y código <5000.

Se ha realizado un escaneo de bitmap en mitabla

Se ha recomprobado la condición

Se han usado 2951 bloques Heap

Se ha realizado un escaneo de bitmap en arbolmitabla2

Indexando por la condición.

Leyendo 98472 bloques de pila y 12 bloques de índice

5. Mostrar las tuplas cuyo código es distinto de 25000.

Se realiza un escaneo secuencial en la tabla.

Se aplica el filtro código <> 25000 –Eliminando 1 row

Leyendo 191735 bloques de pila y 2 de índice

6. Mostrar las tuplas que tiene un nombre igual a 'producto234567'.

Se ha realizado un escaneo secuencial en mitabla

Se ha aplicado el filtro nombre = 'producto234567'

Se han eliminado 11999999 rows por el filtro

Leyendo 191814 bloques de pila

7. Mostar la información de las tuplas con referencia=350

Se ha realizado un escaneo bitmap heap en mitabla

Se ha recomprobado la condición

Se han usado 21552 bloques heap

Se ha utilizado un escaneo bitmap por index en arbolmitabla

Se ha indexado por la condición

Leyendo 116854 bloques de pila y 68 bloques de índice

8. Mostrar la información de las tuplas con referencia<20.

Se ha realizado un escaneo bitmap heap en mitabla

Se ha recomprobado la condición

Se han borrado 8134179 rows por la comprobación de la condición

Se han usado31415 bloques heap

Se ha realizado un escaneo bitmap por index en arbolmitabla

Se ha indexado por la condición

Leyendo 206601 Bloques de pila y 14 bloques de índice

9. Mostrar la información de las tuplas con referencia>300.

Se ha realizado una busqueda secuencial en mitabla

Se ha filtrado por la condición

Se han eliminado 7223960 rows

Leyendo 191736 bloques de pila

10. Mostrar la información de las tuplas con codigo=70000 y referencia=200

Se ha realizado un escaneo por index usando hashmitabla2

Indexando por la condición codigo = 70000

Filtrando por referencia = 200

Borrando 1 row por el filtro

Leyendo 95870 bloques de pila y 2 bloques de índice

11. Mostrar la información de las tuplas con codigo=70000 o referencia=200

Se ha realizado un escaneo bitmap heap en mitabla

Se ha recomprobado la condición

Se han usado 21501 bloques heap

Se ha usado un bitmapOr en el cual:

Se ha realizado un escaneo bitmap por index en hashmitabla2

Indexando por la condición codigo = 70000

Se ha realizado un escaneo bitmap por index en árbol mitabla

Indexando por la condición referencia = 200

Leyendo 117379 bloques de pila y 68 bloques de índice

Cuestión 28. Borrar los 4 índices creados y crear un índice multiclave btree sobre los campos referencia y producto.

Cuestión 29. 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:

1. Mostrar las tuplas cuya referencia vale 200 y su nombre es producto6300031.

Se ha realizado un escaneo por index usando multikayarbol en mitabla

Se ha indexado por las condiciones

Leyendo 95892 bloques de pila y 14 bloques de índice

2. Mostrar las tuplas cuya referencia vale 200 o su nombre es producto6300031.

Se ha realizado un escaneo secuencial en mitabla

Filtrando por las condiciones

Se borran 11976144 rows

Leyendo 191842 bloques de pila

3. Mostrar las tuplas cuyo código vale 6000 y su nombre es producto6300031.

Se ha realizado un escaneo por index usando MiTabla_pkey en mitabla

Se ha indexado por la condición código = 6000

Se ha filtrado por la condición nombre = 'producto6300031'

Se han eliminado 1 row por los filtros

Leyendo 95872 Bloques de pila y 3 bloques de índice

4. Mostrar las tuplas cuyo código vale 6000 o su nombre es producto6300031.

Se ha realizado un escaneo secuencial en mitabla

Filtrando por las condiciones

Se han borrado 11999998 rows por los filtros

Leyendo 191734 bloques de pila.

Cuestión 30. A la vista de los resultados obtenidos de este apartado, comentar las conclusiones 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 que tipo de búsqueda es la más adecuada para acceder a los datos. En el caso de que no haya ningún índice creado en la base de datos, este comprobará todos los posibles, en cambio, de haber un conjunto de índices creados, comprobara las posibilidades dentro de este conjunto, eligiendo siempre la más eficiente.

Bibliografía

- Capítulo 1: Getting Started.
- Capítulo 5: 5.4 System Columns.
- 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: PostgresSQL Client Applications.
- Capítulo VI.III: PostgresSQL Server Applications.
- Capítulo 50: System Catalogs.
- Capítulo 65: Database Physical Storage.
- Apéndice F: Additional Supplied Modules.
- Apéndice G: Additional Supplied Programs.