Titulación: Grado en Ingeniería Informática e Ingeniería en Sistemas de Información
Curso: 2022-2023. Convocatoria Ordinaria de Junio
Asignatura: Bases de Datos Avanzadas – Laboratorio

Practica 1: Arquitectura PostgreSQL y almacenamiento físico

ALUMNO 1:
Nombre y Apellidos: Alvaro Camacho Martin
DNI: 48083824R

ALUMNO 2:
Nombre y Apellidos: nico las
DNI:

Fecha: 2023

Profesor Responsable: Oscar

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.

Es obligatorio proporcionar una explicación a lo que está ocurriendo en PostgreSQL cuando así se indica en la cuestión. No solo vale poner un pantallazo. La ausencia de una explicación hará que sea invalidada esa cuestión.

Plazos

Trabajo de Laboratorio: semana 30 enero, 6 febrero, 13 febrero, 20 febrero y 27 de febrero.

Entrega de práctica: día 7 de marzo. Aula Virtual

Documento a entregar: un fichero con formato ZIP con las respuestas a las cuestiones planteadas, así como los ficheros de log de postgresql relacionados con la resolución de la práctica. El fichero se deberá llamar: **`DNIdelosAlumnos_PL1.zip`**

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 15.1. 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. Antes de comenzar es obligatorio configurar lo que se comenta en la cuestión 0. Hay que resolver la práctica con consultas SQL.

Cuestión 0. Configurar el fichero de Error Reporting and Logging de PostgreSQL para que aparezcan recogidas las sentencias SQL DDL (Lenguaje de Definición de Datos) + DML (Lenguaje de Manipulación de Datos) generadas en dicho fichero. No se pide activar todas las sentencias. No activar la duración de la consulta. También se debe de configurar el log para que en el comienzo de la línea de registro de la información del log ("line prefix") aparezcan vuestros DNI's y el nombre del host con su puerto. ¿Cómo se ha realizado la configuración?

Para cambiar la configuración debemos entrar en la carpeta de 'data' de postgres y en el archivo de configuración cambiaremos los campos siguientes:
Para mostrar las sentencias y consultas realizadas en el log (CREATE, ALTER y DROP), en la línea 575 debemos cambiar la variable 'log_statement' a 'ddl'. Por defecto está en 'none'.
Para que aparezca nuestro DNI y el número de host y puerto, en la línea 542 cambiamos 'log_line_prefix' a 'DNI %r'.

Organización de Archivos en PostgreSQL

Cuestión 1. Crear una nueva Base de Datos que se llame **PL1**. Después crear una tabla **sales** con los campos que vienen en la primera línea para poder cargar los datos correctamente. El campo order_ID es la Primary Key. Los tipos de datos de cada campo deben ser los más ajustados en capacidad de almacenamiento al dato que se inserta. Localizar los ficheros relacionados con la tabla. ¿cómo se localizan? ¿Cuánto ocupan y por qué?

Para crear la base de datos utilizaremos la sentencia: 'CREATE DATABASE PL1'.
Una vez dentro de la base de datos creamos la tabla sales: 'CREATE TABLE sales (Region text, Country text, Item_Type text, Sales_Channel varchar(7), Order_Priority char(1), Order_Date date, Order_ID varchar(9) PRIMARY KEY, Ship_Date date, Units_Sold integer, Unit_Price double precision, Unit_Cost double precision, Total_Revenue double precision, Total_Cost double precision, Total_Profit double precision)'
Introducimos en la tabla sales el contenido de los ficheros csv:
COPY sales FROM '/Users/alvarocamacho/Desktop/DatosPL1/sales[].csv' csv header delimiter ';' NULL 'NULL' encoding 'UTF8';
(Poniendo dentro del corchete el numero del archivo (1,2,3,4,5)).

Para encontrar los archivos de la tabla, usamos la sentencia: 'SELECT pg_relation_filepath('sales');' que nos devuelve la ruta para encontrar los archivos de la tabla que se le ha especificado. En este caso se encuentran dentro de la carpeta 'base', y es la carpeta 16386 el archivo 16402.
Para ver el espacio que ocupa la tabla en el disco usamos esta sentencia: 'SELECT pg_relation_filepath(oid), relpages FROM pg_class WHERE relname = 'sales';'
El resultado es 87129 bloques.)

Cuestión 2. Calcular teóricamente el tamaño en bloques que ocupa la relación **sales** tal y como se realiza en clase de teoría. ¿Concuerda con el tamaño en bloques que nos proporciona PostgreSQL? ¿Cuál es el factor de bloque medio real de la tabla **sales**? ¿Por qué? Realizar una consulta SQL que obtenga ese valor y comparar con el factor de bloque teórico.

Para saber cuánto ocupa cada campo de texto usamos la sentencia: 'SELECT AVG(char_length(campo)) FROM sales;'
Nos dará el promedio que ocupa cada campo de caracteres, ya que al usar tipos como text o varchar el tamaño puede variar según los datos que tengan. Para las fechas y los integer serán siempre el mismo, 4 bytes y para los double será 8 bytes.
Con esto conseguimos saber cuanto ocupa cada campo:
Region(15,83 bytes), Country(8,9 bytes), Item_type(8,58), Sales_channel(6,49 bytes), Order_priority(1 byte), Order_date(4 bytes), Order_id(9 bytes), Ship_date(4 bytes), Units_sold(4 bytes), Unit_price(8 bytes), Unit_cost(8 bytes) y Total_profit(8 bytes).
La suma de todos nos da un total de 85,8 bytes.
Butil es 8kb (8.192 bytes) (documentación).
Para ver el tamaño de la tabla según postgres usamos la misma consulta que en la cuestión 1:
'SELECT pg_relation_filepath(oid), relpages FROM pg_class WHERE relname = 'sales';'
El resultado es 87129 bloques.
Con todo esto tenemos:

—TEÓRICAMENTE:
Factor de bloque = 8.192 bytes / 85,8 bytes = 95 reg/bloque
Tamaño tabla = 5.000.000 / 95 = 52.632 bloques
—POSTGRES:
Factor de bloque = 5.000.000 / 87.129 = 57 reg/bloque
Tamaño tabla = 87.129 bloques

Cuestión 3. Realizar una consulta que muestre el número de países de Europa donde haya habido una venta. ¿Cuántas tuplas se obtienen y cuántos bloques se leen por Postgres? ¿Por qué? Comparar con los resultados obtenidos al aplicar el método visto en teoría.

La consulta es: SELECT COUNT (DISTINCT country) FROM sales WHERE region='Europe';
O para ver qué países son:
SELECT DISTINCT country FROM sales WHERE region='Europe';

La primera devuelve una tupla con el número de países, y la segunda devuelve 48 tuplas con sus nombres.

En teoría:
El número de registros a recuperar según la teoría serían todos (87.129), ya que se utiliza la búsqueda secuencial y tiene que leerlos todos.

Según postgres :
Se leen 87.129 bloques, es decir, todos. Para saber cuántos bloques ha leído postgres en la consulta, ponemos: 'explain (analyze, buffers)' antes de la consulta y en el apartado de 'buffers' sumamos 'hit' y 'read'.

Los resultados coinciden, ya que siempre tiene que leer todos los bloques y no hace falta hacer cálculos.

Cuestión 4. Crear una tabla **sales2** cuyas tuplas estén ordenadas físicamente por el campo región que tenga la misma información que los 5 ficheros de ventas. Indicar el proceso de generación de dicha tabla ordenada. ¿Cuántos ocupa la tabla ahora? ¿Hay algún cambio? ¿Por qué?

Para crear la tabla partimos de sales, usaremos esta sentencia, en la que creamos una tabla nueva a partir de la tabla ya existente:CREATE TABLE sales2 AS SELECT * FROM sales ORDER BY region;
Hacemos que order_id sea PK: ALTER TABLE sales2 ADD CONSTRAINT order_id PRIMARY KEY (order_id);

Para ver cuánto ocupa la tabla usamos la misma sentencia que en la cuestión 1: 'SELECT pg_relation_filepath(oid), relpages FROM pg_class WHERE relname = 'sales2'.
Ahora 'sales2' ocupa 87278 bloques. El cambio es mínimo ya que se siguen teniendo los mismos registros y la misma cantidad de datos. Lo único que cambia es que están ordenados por un campo.

Cuestión 5. Repetir la cuestión 3 sobre la tabla **sales2** y comparar los resultados obtenidos indicando las conclusiones obtenidas.

Según la teoría, utilizando la búsqueda secuencial se leerían todos los bloques, 87.278.

Según postgres el número de bloques que se leen es 87.278, es decir, todos.

Postgres no utiliza la búsqueda binaria, ya que postgres no sabe que están ordenador por no tener índice.

Cuestión 6. Borrar 1.000.000 de tuplas de la tabla **sales** de manera aleatoria usando el valor del campo order_ID. ¿Qué es lo que ocurre físicamente en la base de datos? ¿Se observa algún cambio en el tamaño de la tabla y estructuras asociadas a ella? ¿Por qué? Adjuntar el código de borrado.

Para borrar las 1M de tuplas las ordenamos de forma aleatoria y eliminamos las millón primeras. Usamos esta consulta:

```
DELETE FROM sales3 WHERE order_id IN (
SELECT order_id FROM sales3 ORDER BY random() LIMIT 1000000)
```

Al comprobar el tamaño de la tabla con la consulta de la pregunta 1 el resultado es 87.129 bloques, cuando antes era 87.128. Es prácticamente el mismo ya que se eliminan los registros pero los bloques no se eliminan.

Cuestión 7. En la situación anterior, ¿Qué operaciones se pueden aplicar a la base de datos **PL1** para optimizar el rendimiento de esta? Aplicarla a la base de datos **sales** de tal manera que se recupere el mayor espacio posible. Comentar cuál es el resultado final y qué es lo que ocurre físicamente.

Para optimizar el espacio, debemos eliminar el espacio que ocupan los registros borrados anteriormente. Postgres tiene una función llamada VACUUM, que hace eso mismo. Al poner 'FULL', postgres reescribe todos los contenidos de la tabla en un nuevo archivo de disco, es más lento pero libera más espacio.

```
VACUUM (FULL) sales;
```

Ahora el tamaño ha pasado de 87.128 bloques a 69.789 bloques.

Cuestión 8. Crear una nueva tabla denominada **sales3** con los mismos campos que la cuestión 1 y que esté particionada por el campo región de acuerdo a sus valores diferentes. Insertar los datos de los ficheros **sales#.csv** Explicar el proceso seguido y comentar qué es lo que ha ocurrido físicamente en la base de datos. ¿Cuándo será útil el particionamiento? ¿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.

Para crear la tabla particionada usamos esta sentencia:

CREATE TABLE sales3 (Region text, Country text,
Item_Type text, Sales_Channel varchar(7),
Order_Priority char(1), Order_Date date, Order_ID varchar(9),
Ship_Date date, Units_Sold integer, Unit_Price double precision, Unit_Cost double precision,
Total_Revenue double precision, Total_Cost double precision, Total_Profit double precision)
PARTITION BY LIST (region);

Comprobamos qué regiones hay: SELECT region FROM sales GROUP BY region;

Y creamos una tabla particionada de cada región:
CREATE TABLE [region] PARTITION OF sales3 FOR VALUES IN ([‘region’]);

Metemos los datos en la tabla sales3:
COPY sales3 FROM [ruta] csv header delimiter E';' NULL 'NULL' encoding 'UTF8';

Automáticamente se llenan las tablas de cada región.

Cuestión 9. Repetir la cuestión 3 sobre la tabla **sales3** y comparar los resultados obtenidos con lo visto anteriormente en las tablas **sales** y **sales2** obteniendo conclusiones sobre el método de partición.

Indexación de PostgreSQL

PostgreSQL soporta indexación definida por el usuario para ayudar a acelerar ciertas consultas. Entre otros tipos de índices soporta árboles y hash. En este apartado se va a trabajar sobre ambos tipos de índices, pudiendo observar cómo se organizan internamente y su funcionamiento.

Cuestión 10. Borrar todas las tablas **sales, sales2** y **sales3**. Crear una nueva tabla que se llama **sales** como en la cuestión 1 y que tenga cargados todos los datos de los ficheros **sales#.csv**

Cuestión 11. Crear un índice de tipo árbol para el campo order_id. ¿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? Indicar el procedimiento seguido e incluir el código SQL utilizado.

Cuestión 12. 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 11.

Cuestión 13. Crear un índice de tipo árbol para el campo units_sold. ¿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? Indicar el procedimiento seguido e incluir el código SQL utilizado.

Cuestión 14. 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 13.

Cuestión 15. Crear un índice de tipo hash para el campo order_id. ¿Dónde se almacena físicamente ese índice? ¿Qué tamaño tiene? ¿Cuántos bloques tiene? ¿Cuántos cajones tiene? ¿Cuántas tuplas tiene de media un cajón? Indicar el procedimiento seguido e incluir el código SQL utilizado.

Cuestión 16. Determinar el tamaño de bloques que teóricamente tendría de acuerdo con lo visto en teoría. Comparar los resultados obtenidos teóricamente con los resultados obtenidos en la cuestión 15.

Cuestión 17. Crear un índice de tipo hash para el campo units_sold. ¿Dónde se almacena físicamente ese índice? ¿Qué tamaño tiene? ¿Cuántos bloques tiene? ¿Cuántos cajones tiene? ¿Cuántas tuplas tiene de media un cajón? Indicar el procedimiento seguido e incluir el código SQL utilizado.

Cuestión 18. Determinar el tamaño de bloques que teóricamente tendría de acuerdo con lo visto en teoría. Comparar los resultados obtenidos teóricamente con los resultados obtenidos en la cuestión 17.

Cuestión 19. ¿Qué conclusiones se puede obtener de la gestión y organización de PostgreSQL sobre los dos índices árbol y hash que se han creado y han sido analizados? ¿Por qué? Comparar con lo 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. En este apartado se pretende mostrar cómo es el acceso a los datos para diferentes tipos de consultas.

PostgreSQL suministra varias vistas estadísticas que se puede usar para monitorizar los bloques leídos (tipo statio) de cada una de las estructuras creadas en la base de datos. En este apartado se deben usar esas vistas y está prohibido el uso de otro comando para este fin (Table 28.2).

Para ello, borrar todas las tablas creadas y volver a crear la tabla **sales** como en la cuestión 1. Cargar los datos que se encuentran originalmente en los ficheros **sales#.csv**

Cuestión 20. Crear un índice hash sobre el campo order_id y otro sobre total_cost. También crear un índice primario tipo árbol sobre el campo total_cost. ¿Cuál ha sido el proceso seguido para crear cada tipo de índice? Incluir el código SQL utilizado para ello.

Cuestión 21. 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 de cada estructura? ¿Por qué? Importante, reinicializar los datos recolectados de la actividad de la base de datos antes de lanzar cada consulta. Se recuerda que solo se pueden usar vistas sobre las estadísticas de la base de datos.

1. Mostrar la información de las tuplas con order_id 100565711.

2. Mostrar la información de las ventas con un coste total de 387.072,00 €.

3. Contar el número de ventas que tienen coste total mayor de 32.000,00 € y menor de 100.000,00 €

4. Mostrar las ventas que son del país de Tanzania.

5. Actualizar la prioridad del pedido para que sea ‘L’ para todas las ventas que tienen un coste de 346.692,00 €

6. Mostrar la información de los alumnos que tienen un order_id igual a 103903664 o 10390366

7. Insertar una nueva venta en la tabla sales.

Cuestión 22. Crear un índice multiclave tipo árbol y otro hash sobre los campos región y country. Incluir el código SQL utilizado para ello.

Cuestión 23. 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 de cada estructura? ¿Por qué? Importante, reinicializar los datos recolectados de la actividad de la base de datos antes de lanzar cada consulta:

1. Mostrar el número de ventas totales que se han producido en Europa.

2. Mostrar el número de ventas que se han producido en la región de Asia y el país de Laos.

3. Mostrar el número de tuplas del país México o de la región de “Australia and Oceania”.

4. Mostrar el número de tuplas de la región de “Sub-Saharan Africa” y que además el número de unidades vendidas es de 2843.

Cuestión 24. Crear la tabla **sales3** particionada por rango de años completos. 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 de cada estructura? ¿Por qué? Importante, reinicializar los datos recolectados de la actividad de la base de datos antes de lanzar cada consulta.

1. Mostrar las ventas que cuyo orden de pedido (order_date) han sido en el año 2022.

2. Mostrar las ventas que cuyo orden de pedido (order_date) han sido en el año 2018 o en el año 2001.

3. Mostrar las ventas que cuyo orden de pedido (order_date) han sido entre el 05-10-2015 y el 11-8-2018

Cuestión 25. 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.

Bibliografía (PostgreSQL 15)

- Capítulo 1: Getting Started.
- Capítulo 5: 5.5 System Columns.
- Capítulo 5: 5.11 Table Partitioning.
- Capítulo 11: Indexes.
- Capítulo 20: Server Configuration.
- Capítulo 25: Routine Database Maintenance Tasks.
- Capítulo 28: Monitoring Database Activity. The statistics Collector
- Capítulo 29: Monitoring Disk Usage. Determining Disk Usage
- Capítulo VI.II: PostgresSQL Client Applications.
- Capítulo VI.III: PostgresSQL Server Applications.
- Capítulo 53: System Catalogs.
- Capítulo 67: B-Tree Indexes.
- Capítulo 73: Database Physical Storage.
- Apéndice F: Additional Supplied Modules. Pageinspect, pgstatutuple
- Apéndice G: Additional Supplied Programs. Oid2name