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

Curso: **2021-2022. Convocatoria Ordinaria de Junio**

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

Practica 2: Carga Masiva de Datos, Procesamiento y Optimización de Consultas

ALUMNO 1:

Nombre y Apellidos: Andrea Villa Puyot

DNI: 06614234D

ALUMNO 2:

Nombre y Apellidos: _____

DNI: _____

Fecha: 25/04/2022

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 calificará la asignatura como Suspenso – Cero.

Plazos

Tarea en Laboratorio:	Semana 14 de Marzo, Semana 21 de Marzo, Semana 28 de Marzo y semana 4 de Abril.
Entrega de práctica:	Semana 19 de Abril (Martes). Aula Virtual
Documento a entregar:	Este mismo fichero con las respuestas a las cuestiones planteadas, el programa que genera los datos de carga de la base de datos en Python, el fichero history del pgcli y los ficheros de log de PostgreSQL de esta práctica. No se piden los ficheros de los datos en bruto de la base de datos. Se entregará en un ZIP comprimido llamado: `DNI'sdelosAlumnos_PECL2.zip`

AMBOS ALUMNOS DEBEN ENTREGAR EL FICHERO EN LA PLATAFORMA.

Introducción

El contenido de esta práctica versa sobre la monitorización de la base de datos, manipulación de datos, técnicas para una correcta gestión de los mismos, así como tareas de mantenimiento relacionadas con el acceso y gestión de los datos. También se trata el tema de procesamiento y optimización de consultas realizadas por PostgreSQL (14.x). Se analizará PostgreSQL en el proceso de carga masiva y optimización de consultas.

En general, la monitorización de la base de datos es de vital importancia para la correcta implantación de una base de datos, y se suele utilizar en distintos entornos:

- Depuración de aplicaciones: Cuando se desarrollan aplicaciones empresariales no se suele acceder a la base de datos a bajo nivel, sino que se utilizan librerías de alto nivel y mapeadores ORM (Hibernate, Spring Data, MyBatis…) que se encargan de crear y ejecutar consultas para que el programador pueda realizar su trabajo más rápido. El problema en estos entornos está en que se pierde el control de qué están haciendo las librerías en la base de datos, cuántas consultas ejecutan, y con qué parámetros, por lo que la monitorización en estos entornos es vital para saber qué consultas se están realizando y poder optimizar la base de datos y los programas en función de los resultados obtenidos.
- Entornos de prueba y test de rendimiento: Cuando una base de datos ha sido diseñada y se le cargan datos de prueba, una de las primeras tareas a realizar es probar que todos los datos que almacenan son consistentes y que las estructuras de datos dan un rendimiento adecuado a la carga esperada. Para ello se desarrollan programas que simulen la ejecución de aquellas consultas que se consideren de interés para evaluar el tiempo que le lleva a la base de datos devolver los resultados, de cara a buscar optimizaciones, tanto en la estructura de la base de datos como en las propias consultas a realizar.
- Monitorización pasiva/activa en producción: Una vez la base de datos ha superado las pruebas y entra en producción, el principal trabajo del administrador de base de datos es mantener la monitorización pasiva de la base de datos. Mediante esta monitorización el administrador verifica que los parámetros de operación de la base de datos se mantienen dentro de lo esperado (pasivo), y en caso de que algún parámetro salga de estos parámetros ejecuta acciones correctoras (reactivo). Así mismo, el administrador puede evaluar nuevas maneras de acceso para mejorar aquellos procesos y tiempos de ejecución que, pese a estar dentro de los parámetros, muestren una desviación tal que puedan suponer un problema en el futuro (activo).

Para la realización de esta práctica será necesario generar una muestra de datos de cierta índole en cuanto a su volumen de datos. Para ello se generarán, dependiendo del modelo de datos suministrado, para una base de datos denominada **BECAS**. Básicamente la base de datos guarda las becas que se ha concedido a cada uno de los estudiantes que quieren realizar una estancia en una Universidad de destino diferente a donde están matriculado. La tabla estudiantes guarda la información de cada estudiante, la tabla universidades guarda la información de cada Universidad que participa en el programa, la tabla asignaturas guarda las asignaturas ofertadas a los estudiantes y que pertenecen a cada Universidad, la tabla estancias guarda la información sobre la beca concedida a cada estudiantes; y la tabla convalidaciones

guarda la relación de las asignaturas que se estudian en destino junto con la asignatura que se convalida en origen, para cada estancia. Se suministra el modelo relacional construido en **pgmodeler** donde dentro de cada tabla se comenta lo que se guarda, así como la descripción de cada campo.

Los datos referidos al año 2021 que hay que generar deben de ser los siguientes:

- Hay un total de 100.000 universidades dadas de alta. Hay 50 países almacenados y uno de ellos debe ser "Spain". Los países se asignan a la Universidad de manera aleatoria entre los 50.
- Hay 5.000.000 de estudiantes dados de alta. A cada estudiante se le asigna una universidad de origen de manera aleatoria. Los créditos aprobados se definen entre 60 y 200 (enteros e incluidos) ECTS y deben de estar generados de manera aleatoria.
- Cada universidad oferta entre 15 y 20 asignaturas que pueden cursar los estudiantes en su estancia. Esas asignaturas pueden ser de tipo "básica", "obligatoria", "transversal" y "optativa". Se asigna el tipo de manera aleatoria a cada asignatura. El número de créditos ECTS varía entre 4 y 12 de manera aleatoria.
- Cada estudiante ha realizado una estancia, y la universidad de destino se elige de manera aleatoria sin que sea la propia universidad donde estudia el alumno. El tipo de beca puede ser: "KA103", "KA107", "MIT", "Franklin", "SICUE" y "Global". La duración puede ser primer cuatrimestre "1", segundo cuatrimestre "2" o anual "A". La cuantía de las becas asciende a valores entre 500 y 2000 euros. Todas las asignaciones se hacen de manera aleatoria.
- Cada estancia tiene asignadas entre 3 y 5 asignaturas si la estancia se realiza el primer (1) o el segundo cuatrimestre (2); y entre 6 y 10 asignaturas si la estancia es anual (A) de la universidad de destino elegida. La nota de origen puede ser: A+, A, A-, B+, B, B-, C+, C, C-, D+, D, D-, E o F. La nota en origen realizada una vez la convalidación tendrá que ver con la siguiente tabla de equivalencia. Todas las asignaciones de hacen de manera aleatoria

Nota Destino	Nota Origen
A+	10
A	9.8
A-	9.5
B+	9.0
B	8.5
B-	8.0
C+	7.5
C	7.3
C-	7.0
D+	6.5
D	6.0
D-	5.5
E	5.0
F	4.0

Actividades y Cuestiones

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") aparezca el DNI de los alumnos que realizan la práctica (ambos), el nombre del host con su puerto, y la fecha y hora de la operación que se ha realizado. Se ha de configurar también el servidor para que no use el procesamiento paralelo de consultas.

Hemos accedido a la ruta C:\Program Files\PostgreSQL\14\data y hemos abierto el archivo postgresql.conf. En el archivo he buscado donde ponía log_line_prefix el cual aparecia comentado, hemos eliminado el # y cambiado el texto que aparecia entre comillas simples por uno de nuestros DNI y también hemos puesto %r para que aparezca el nombre del host con su puerto, es decir, hemos escrito: log_line_prefix = '09071886L %r'.

*Apesar de no haber podido tener todos los datos al completo he realizado una comprobación de todos los comandos de la práctica con una carga menor de datos para poder estar segura de funcionan de manera correcta.

Cuestión 1: ¿Tiene el servidor postgres un recolector de estadísticas sobre el contenido de las tablas de datos? Si es así, ¿Qué tipos de estadísticas se recolectan y donde se guardan?

El servidor de postgres si tiene un recolector de estadísticas sobre el contenido de las tablas que se generan ya que este es capaz de contabilizar el acceso a índices y tablas tanto en el bloque del disco como individualmente.
También se realiza un seguimiento del número de filas de cada tabla y de la información acerca del análisis de la misma. Además también es capaz de contar las llamadas a funciones que han sido creadas por un usuario y el tiempo que se ha tardado en crear dichas funciones-
Una vez que se realiza el alamcenamiento en el directorio que ha sido nombrado por el parametro stats_temp_directory, pg_stat_tmp de manera predeterminada.Una vez que se apaga el servidor ya se ha almacenado una copia de los datos estadisticos.
Cuando realizamos la recuperación al inicar el servidor se nos restablecen todos los contadores de las estadisticas creadas.

Cuestión 2: Crear una nueva base de datos llamada **matricula** y que tenga las siguientes tablas con los siguientes campos y características:

- estudiantes(carnet tipo numeric PRIMARY KEY, nombre tipo text, apellidos tipo text, creditos tipo numeric)
- asignaturas(codigo tipo numeric PRIMARY KEY, nombre tipo text, caracter tipo text, creditos tipo numeric)
- matriculas(carnet_estu tipo numeric que sea FOREIGN KEY del campo carnet de la tabla estudiantes con restricciones de tipo RESTRICT en sus operaciones, codigo_asig tipo numeric que sea FOREIGN KEY del campo codigo de la tabla asignaturas con restricciones de tipo RESTRICT en sus operaciones, nota de tipo numeric. La PRIMARY KEY debe ser compuesta de carnet_estu y codigo_asig.

Se pide:

- Indicar el proceso seguido para generar esta base de datos.
- Cargar la información del fichero datos_estudiantes.csv, datos_asignaturas.csv y datos_matriculas.csv en dichas tablas de tal manera que sea lo más eficiente posible, asegurando la integridad y consistencia de la base de datos.
- Indicar los tiempos de carga.

Para poder crear la base de datos utilizamos la ruta C:\Program Files\PostgresSQL\14\bin>, una vez que tenemos la ruta seleccionada lo que hacemos es crear la base de datos en este caso con el nombre de Matricula y quedaría así:
C:\Program Files\PostgreSQL\14\bin> createdb.exe -U postgres -p 5433 Matricula

Hemos creado la Foreign key que nos pedía el enunciado mediante los comandos:
- alter table matriculas
add constraint FK_codigo_asig
foreign key (codigo_asig)
references asignaturas (codigo)
on update restrict
on delete restrict
- alter table matriculas
add constraint FK_carnet_estu
foreign key (carnet_estu)
references estudiantes(carnet_estu)
on update restrict
on delete restrict
*Lo inplemengtamos para poder asegurar la integridad y cosnitencia de la base de datos que hemos creado.
*Para poder introducir los datos de los archivos csv hemos realizado el siguiente comando:
copy estudiante from 'C:\Pl2\'datos_estudantes.csv' DELIMITER ',' CSV HEADER;
Una vez que hemos hecho esto ya podremos acceder a nuestra base de datos y ahora procederemos a crear las tablas y a metemos los datos que nos proporcionan. La tabla estudiantes tarda en cargar 1.278 segundos, la tabla asignatura tarda en cargar 8.685 segundos y por último la tabla de matrículas tarda en cargar 3 minutos y 39 segundos

Cuestión 3: Mostrar las estadísticas obtenidas en este momento para cada tabla. ¿Qué se almacena? ¿Son correctas? Si no son correctas, ¿cómo se pueden actualizar?

Para poder las estadísticas de las tablas que hemos creado utilizamos el comando select * from pg_stats where tablename = 'nombre de la tabla'; de esta forma podremos comprobar las estadísticas. lo que se almacena es la correlación de valores entre las columnas.

Los valores no son correctos por lo que procedemos a actualizar la base de datos de tal forma que podremos mejorar los tiempos de ejecucion para todas aaquellas tablas que tienen muchas modificaciones de datos. Con un update.

Datos de las estadísticas:
Tabla asignaturas:

Sequential scans 4
Sequential tuples read 8000000
Index scans 10
Index tuples fetched 5
Tuples inserted 0
Tuples updated 0
Tuples deleted 0
Tuples HOT updated 0
Live tuples 0
Dead tuples 0
Heap blocks read 65938
Heap blocks hit 77
Index blocks read 12
Index blocks hit 23
Toast blocks read 0

Cuestión 4: Aplicar el comando EXPLAIN a una consulta que obtenga el número de alumnos con créditos de más de 200 créditos ECTS. ¿Son correctos los resultados del comando EXPLAIN? ¿Por qué? Comparar con lo que se obtendría con el procedimiento visto en teoría obteniendo las estadísticas de postgres.

Utilizamos el comando:explain select * from estudiantes where creditos>200 según los datos obtenidos podemos ver con el explain que el coste estimado es cero y el coste total es 6453 bloques.
El coste total estimado = (bloques en disco *1.0)+(número de tuplas escaneadas*0.01) = (2703*1.0) + (300000.0*0.01)= 5703
Podemos apreciar como son números próximos, pero no iguales y esto es debido a que se han quedado bloques por el camino sin leer, y es por lo que no coinciden los bloques totales con los leídos.
Realización de los cálculos de manera teórica:
Se realiza una lectura secuencial:
Estudiantes:
Lr = 37 byts
Nr = 300000 (no coincide con el número de rows del explain)
fr=[B/Lr]=[6453/37] = 174 reg/bloq
Br=[nr/fr]=[300000/174]= 1724 bloques
Se puede apreciar que nos dan números distintos a los que nos han salido en el explain ya que no considera que todos los bloques están ocupados y nosotros si lo damos por hecho.

Cuestión 5: Aplicar el comando EXPLAIN a una consulta que obtenga el nombre de los alumnos que tienen 180 créditos y además tienen por lo menos una asignatura suspensa. ¿Son correctos los resultados del comando EXPLAIN? ¿Por qué? Comparar con lo que se obtendría con el procedimiento visto en teoría obteniendo las estadísticas de postgres.

Utilizamos el comando explain:
explain select nombre from estudiantes inner join asignaturas on estudiantes.creditos = asignatura.creditos where estudiantes.creditos = 180 or asiganturas.creditos <= 0
-Esto nos da que 10105609 bloques
work_mem = 4MB
M = 4MB = 4194304/6453 = 649 bloques

En el procedimiento obtenido en teoría nos salen unos 11983 bloques.

Cuestión 6: Aplicar el comando EXPLAIN a una consulta que obtenga el nombre de las asignaturas de 6 créditos en los que los alumnos hayan obtenido un 8 de nota y los alumnos tengan un número de 60 créditos. ¿Son correctos los resultados del comando EXPLAIN? ¿Por qué? Comparar con lo que se obtendría con lo visto en teoría obteniendo las estadísticas de postgres.

explain select asignaturas.nombre from asignaturas inner join estudiantes on asignaturas.cre ditos=estudiantes.creditos where asignaturas.creditos = 6 and estudiantes.creditos >= 60 and estudiantes.creditos > 0

Los resultados obtenidos en el EXPLAIN son los correctos ya que nos dicen con certeza como optimizar los resultados que hemos obtenido al realizar la consulta.
Comparándolos con los resultados que nos proporcionan las estadísticas de postgresSQL. Se puede apreciar que hay una diferencia, ya que con el EXPLAIN lo que queremos es que nos muestre el plan de ejecución de la consulta y con las estadísticas lo que nos da son los valores predertiminados de las tablas con consultas poco óptimas y de ejecución larga (antes de su actualización).

Cuestión 7: Generar los datos solicitados al comienzo de la práctica para la base de datos **BECAS** usando un programa en Python que deberá de estar en un único fichero y comentado. Pegar el código del fichero en el cuadro de texto que se adjunta a continuación.

```
"""
Implementacion de datos aleatorios para la base de datos becas.
"""

#  -*- coding: utf-8 -*-

import random
import datetime
import psycopg2

from faker.providers import internet
from faker import Faker

fake = Faker()
fake.add_provider(internet)


def generar_ciudades(cantidad):
```

Cuestión 8: Realizar la carga masiva de los datos generados en la cuestión 7 en la base de datos **BECAS**. Indicar el proceso seguido y el orden de carga de las tablas, explicando el porqué de dicho orden; y asegurando la consistencia e integridad de los datos cargados. Comparar los tiempos en las tablas implicadas y explicar a qué es debida la diferencia. ¿Existe diferencia entre los tiempos que ha obtenido y los que aparecen en el LOG de operaciones de postgreSQL? ¿Por qué?

Tabla	Tiempo (seg)
universidades	3600 seg
asignaturas	0.236 seg
estudiantes	180000 seg
estancias	180000 seg
convalidaciones	7 seg

He cargado las tablas en el siguiente orden univerisidad ya que es la única que no depende de ninguna otra tabla, luego asignaturas que depende de universidad pero debe estar creada antes que convaldiaciones, luego estudiantes que depende de universidad y crea dependencia en estancias la cual es la siguiente en crearse y por último convalidaciones que depende como he comentando anteriormente de asignaturas y de estancias. De está forma una vez que cargamos los datos estos se introducen sin problema y en su debido orden.
La diferencia se debe a la cantidad de datos a introducir en cada una de ellas no es lo mismo el tiempo de estudiantes que el de asignaturas ya que se puede apreciar que tiene una carga de datos mucho mayor.
*Aquí en los tiempos no me ha sido posible cargar todos los datos al completo he podido realizar los tiempos de algunos datos, en lo más pesados que son los que se me han cargado al completo he cronometrado el tiempo que tardaban en cargarse 100.000 datos y de ahí he sacado el tiempo completo que tardarían

A partir de este momento en adelante, se deben de realizar las siguientes cuestiones con la base de datos que tiene la integridad referencial activada. Es obligatorio y queda prohibido cambiar la integridad referencial de la base de datos.

Cuestión 9: Realizar una consulta SQL que muestre “el nombre de las universidades españolas junto con el porcentaje de alumnos que tienen que se han ido de estancia con una beca de tipo “KA103” o “KA107” teniendo superados un número de créditos de más de 120 ECTS a Universidades de destino que no son españolas, habiéndose llevado asignaturas de tipo “obligatorio” o “básicas” y que hayan tenido una nota en destino de tipo B+,B o B- “.

Obtener el plan de ejecución con el resultado del comando EXPLAIN en forma de árbol de álgebra relacional. Explicar la información obtenida en el plan de ejecución de postgreSQL. Comparar el árbol obtenido por nosotros al traducir la consulta original al álgebra relacional y el que obtiene postgreSQL. Comentar las posibles diferencias entre ambos árboles.

Consulta SQL creada:

```
SELECT
   COUNT(universidades.nombre)
FROM
   becas.public.universidades universidades,
   becas.public.estudiantes estudiantes,
   becas.public.estancias estancias,
   becas.public.convalidaciones convalidaciones,
   becas.public.asignaturas asignaturas
WHERE
   universidades.codigo_erasmus = estudiantes.codigo_erasmus_universidades AND
   universidades.codigo_erasmus = estancias.universidad AND
   universidades.codigo_erasmus = convalidaciones.universidad_estancias AND
   universidades.codigo_erasmus = asignaturas.universidad AND
   estudiantes.codigo_estudiante = estancias.estudiante AND
```

Resultado comando EXPLAIN

Comentarios y explicaciones.

Cuestión 10: Usando PostgreSQL, y a raíz de los resultados de la cuestión anterior, ¿qué modificaciones realizaría para mejorar el rendimiento de la misma y por qué? Obtener la información pedida de la cuestión 9 y explicar los resultados. Obtener el plan de ejecución con el resultado del comando EXPLAIN en forma de árbol de algebra relacional. Comentar los resultados obtenidos y comparar con la cuestión anterior.

Consulta SQL creada:

```
create index arbolb_nombre on universidad using btree(nombre, pais)
create index arbolb_asiganturas on asignaturas using btree(tipo, creditos)
create index arbolb_convalidaciones on convalidaciones using btree(nota_destino)
create index arbolb_estancias on estancias using btree (tipo_beca, estudiante)
*Luego realizamos un set enable_seqscan = off para poder desactivar el escaneo secuencia de la tabla y proceder a utilizar los indices que hemos creado
```

Resultado comando EXPLAIN

La salida del explain nos dice que hemos optimizado la base de datos.

Comentarios y explicaciones.

Para poder mejorar el rendimiento lo que hacemos es aumentar M, de está manera podremos permitir que más ramas se materialicen y se encaucen, para poder ayudar a la base de datos a optimizarse. Lo que hace que PostgresSQL pueda almacenar en su caché mayor cantidad de datos y realizar llamadas mucho menos costosas al disco.
Otra medida sería hacer un hash join para poder encauzar ramas y no solo una como suele ocurrir con los bucles anidados.
Otra forma podría ser realizando proyecciones que se realizan en los niveles más bajos y para ello en vez de realizar una búsqueda secuencial lo que deberíamos hacer es una búsqueda binaria o tener un índice sobre ese campo.
Podríamos realizar ramas que tengan menos tuplas o realizar un VACUUM ANALIZER las veces que sea necesario.
O podemos crear árboles y una vez que están creados desactivar el escaneo secuencial para poder utilizar los indices.

Cuestión 11: Usando PostgreSQL, borre el 40% de las universidades almacenadas de manera aleatoria y todos sus datos relacionados de la manera más eficiente posible, ¿cuál ha sido el proceso seguido? ¿Y el tiempo empleado en el borrado? Prohibido modificar la Integridad Referencial.

Lo que he hecho ha sido crear una tabla temporal para poder almacenar los que se van a borrar de manera temporal, y una vez realizado esto lo que hemos hecho a sido ordenarlo a través de un árbol llamado indice_temp, una vez que hemos hecho procedemos a realizar un borrado de lso contenidos de las tablas que en este caso universidad ordenado por el campo nombre.
Una vez que hemos hecho esto lo que hacemos es comprobar que los datos se han borrado.
Ahora que ejecutamos un expalin en la consulta que hemos realizado en la cuentión 9 podemos apreciar que el coste es mucho menor ya que hay árboles btree y muchos menos tuplas de las que teníamos antes.

Cuestión 12: Ejecute la consulta que pide los datos de la cuestión 9 de nuevo. Obtener el plan de ejecución con el resultado del comando EXPLAIN en forma de árbol de algebra relacional. Comparar con los resultados anteriores.

Consulta SQL creada:

Sería relizar un explain a la consulta nueve una vez que está actualizada con los cambios realizados en la cuestión anterior.

Resultado comando EXPLAIN

Se aprecia un menor coste con la creacion de indices y árboles y la eliminacion de tuplas innecesrias

Comentarios y explicaciones.

Simplemente lo que hemos hecho ha sido al ejecutar el comando explain con los cambios que hemos hecho en la cuestión anterior de tal forma que hemos podido apreciar como optimizar las consultas, reducir costes y eliminar toda la basura de las tuplas que no nos sirven.

Cuestión 13: ¿Qué optimización/mejoras de la BD propondría para mejorar los resultados de dicho plan sin modificar el código de la consulta? ¿Por qué?

Para poder mejorar los resultados tras modificar las consulta podemos utilizar las siguientes formas:
El REINDEX INDEX o REINDEX TABLE se utiliza para poder reconstruir el indice o todos los índices de la tabla, cuando se sospecha que ese índice está dañado en una tabla, por lo que esto hace que los datos almacenados en la tabla del indice sean reemplazados por la copia que teníamos hecha previamente. Por lo que se recomienda realizar este comando antes de hacer el borrado .
Utilizar el VACUUM para cuando las tuplas que hemos eliminado o están obsoletas no se eliminan físicamente de su tabla, y permanezcan presentes hasta que se ejecute el VACUUM, por lo que es necesario realizar este comando para tener las tablas actualizadas con frecuencia.
El AUTOVACUUM también se utiliza para reclamar el almacenamiento al eliminar tuplas o datos que estén obsoletos. Esto se encarga de comprobar que las tablas que tengan números significativos de registros que hayan sido insertados, actualizados o eliminados.
Quitar el log_rotation_age dandole valor de cero , cuando loggign_colector se encuentre habilitado, este parámetro se encarga de determinar la vida útil máxima del archivo de registro individual. Una vez transcurridos unos minutos se vuelve a crear un nuevo archivo de registro, de esta forma

Cuestión 14: Usando PostgreSQL, lleve a cabo las operaciones propuestas en la cuestión anterior y ejecute el plan de ejecución de la consulta que pide los datos de la cuestión 9. Obtener el plan de ejecución con el resultado del comando EXPLAIN en forma de árbol de algebra relacional. Compare los resultados del plan de ejecución con los de los apartados anteriores. Coméntelos.

Consulta SQL creada:

Primero activamos:
Para deshabilitar la creación de nuevos archivos de registro ejecutamos log_rotation_age =0;
vacuum verbose univerisdades;
vacuum verbose asignaturas;
vacuum verbose convalidaciones;
vacuum verbose estudiantes;
vacuum verbose estancias;
reindex database becas

Resultado comando EXPLAIN

Podemos apreciar como el coste es menor al que teniamos inicalmente.

Comentarios y explicaciones.

Se puede apreciar como gracias la eliminación de tuplas basura hemos podido reducir el coste, ya que de esta manera hemos optimizado la consulta de nuestra base de datos. Se puede ver de forma notable como el coste es mucho menor una vez que hemos aplicado todas las técnicas de mantenimiento.

Cuestión 15: A partir de lo visto y recopilado en toda la práctica. Describir y comentar cómo es el proceso de procesamiento y optimización que realiza PostgreSQL en las consultas del usuario.

Con PostgreSQL he podido ver cómo se podía realizar la optimización de las consultas con una gran cantidad de datos que en ocasiones suele ser muy pesado y puede tardar bastante tiempo. Y para ello , se han utilizado los conocimientos vistos en clase para poder acelerar ese proceso con índices o con otros medios. Por lo que para poder conocer el coste que implica un consulta tan lenta se utiliza el comando ‘ explain ’ que lo que hace es ayudarnos y darnos la información sobre los pasos que se van realizando en postgreSQL cuando vayamos a realizar dichas consultas. Con esta información lo que podemos hacer es conseguir que sea más óptimo añadiendo todo lo aprendido en clase. Una vez realizados todos los ajustes necesarios ya podríamos ver que se materializa y que se encauza , y de esta forma conseguir que materialice menos y se encauce más, sabiendo cual es su memoria caché y todo lo que se encuentra almacenado en ella y lo que nos ocupa.

Bibliografía

PostgreSQL (14.x)

- Capítulo 14: Performance Tips.
- Capítulo 20: Server Configuration.
- Capítulo 15: Parallel Query.
- Capítulo 25: Routine Database Maintenance Tasks.
- Capítulo 51: Overview of PostgreSQL Internals.
- Capítulo 72: How the Planner Uses Statistics.
- https://pgtune.leopard.in.ua/
- https://explain.dalibo.com/