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

Curso: **2020-2021. 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: Alejandro Hernández Martín

DNI: 09074363N

ALUMNO 2:

Nombre y Apellidos: Noelia Marca Retamozo

DNI: 09048809B

Fecha: 09/05/2021

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 Suspenso – Cero.

Plazos

Tarea en Laboratorio:	Semana 22 de Marzo, Semana 6 de Abril, Semana 12 de Abril, semana 19 de Abril y semana 26 de Abril.
Entrega de práctica:	Semana 4 de Mayo (Martes). Aula Virtual
Documento a entregar:	Este mismo fichero con las respuestas a las cuestiones planteadas y el programa que genera los datos de carga de la base de datos. No se pide el script de carga 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 (13.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 **NETFLIX**. Básicamente, la base de datos guarda información sobre las cuentas de los usuarios que se suscriben a la plataforma NETFLIX para poder acceder a los contenidos que ofrece, en este caso, series y películas. Hay una serie de suscripciones definidas en el sistema y el usuario en todo momento esta suscrito a una de ellas. Cada mes que el usuario está dado de alta en la plataforma, se produce un pago de la cantidad por la

que está suscrito. Esa suscripción del usuario le permite acceder solamente a los contenidos de películas y series que proporciona esa suscripción. Por último, cada uno de los contenidos de la plataforma tiene asociado una serie de géneros de tal manera que es posible clasificar los contenidos por géneros.

Los datos referidos al año 2020 que hay que generar deben de ser los siguientes:

- Hay 2.000.000 de usuarios dados de alta, donde cada uno tiene una cuenta.
- Existe un total de 100 tipos de suscripción diferente.
- Por cada usuario se almacenan los pagos realizados mensualmente durante el año 2020. Por lo tanto, hay que generar un total de 12 pagos por cada usuario.
- Existen un total de 20.000.000 de contenidos. El decidir si un contenido es película o serie se debe de hacer de manera aleatoria por medio de la función random.
- Cada contenido se asocia aleatoriamente a un tipo de suscripción.
- Existen 20 géneros de contenidos diferentes, y el número de géneros que tiene asociado un contenido es un valor aleatorio que va entre 1 y 6.
- El campo duración de la tabla películas debe tener valores aleatorios entre 80 y 120 minutos.
- El campo capítulos de la tabla series debe tener valores aleatorios entre 10 y 20.
- El campo temporadas de la tabla series debe tener valores aleatorios entre 1 y 15.
- El campo valoración de los contenidos debe tener valores aleatorios entre 1 y 10.
- El campo fecha de la tabla pago debe tener los días generados de forma aleatoria para cada mes de pago de la suscripción de cada usuario.

Actividades y Cuestiones

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?**

Sí, el recopilador de estadísticas de PostgreSQL es un subsistema que admite la recopilación y genera informes de información sobre la actividad del servidor.

Puede contar los accesos a tablas e índices tanto en términos de bloque de disco como de fila individual. Además, realiza un seguimiento del número total de filas en cada tabla e información sobre las acciones de vacío y análisis por tabla. Por último, puede contar las llamadas a funciones definidas por el usuario y el tiempo total invertido en cada una.

Estos archivos se almacenan en el directorio nombrado por el parámetro stats_temp_directory , pg_stat_tmp de forma predeterminada. Cuando el servidor se apaga limpiamente, se almacena una copia permanente de los datos estadísticos en el

pg_stat subdirectorio, de modo que las estadísticas se pueden retener durante los reinicios del servidor.

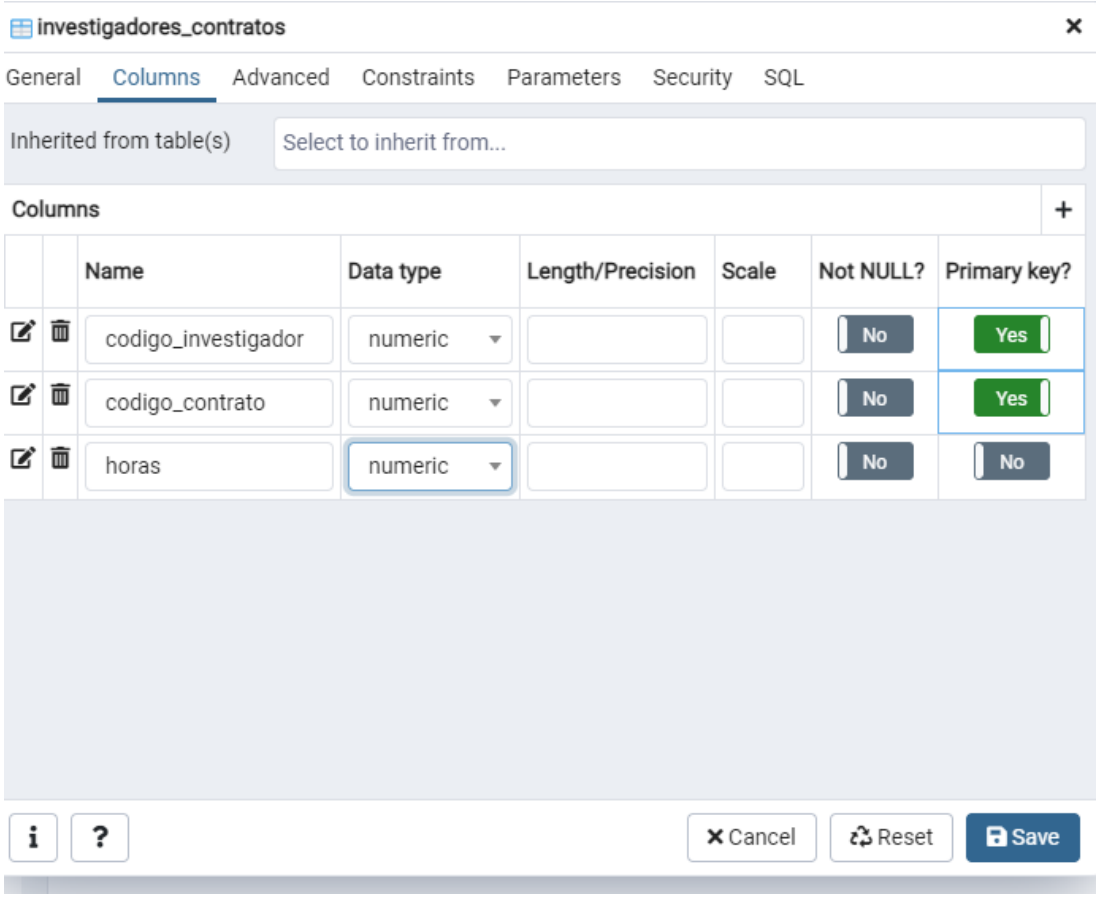
Cuando la recuperación se realiza al iniciar el servidor (por ejemplo, después de un apagado inmediato, un bloqueo del servidor y una recuperación en un momento determinado), se restablecen todos los contadores de estadísticas.

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

- **investigadores(codigo_investigador tipo numeric PRIMARY KEY, nombre tipo text, apellidos tipo text, salario tipo numeric)**
- **contratos(codigo_contrato tipo numeric PRIMARY KEY, nombre tipo text, entidad tipo text, coste tipo numeric)**
- **investigadores_contratos(codigo_investigador tipo numeric que sea FOREIGN KEY del campo codigo_investigador de la tabla investigadores con restricciones de tipo RESTRICT en sus operaciones, codigo_contrato tipo numeric que sea FOREIGN KEY del campo codigo_contrato de la tabla contratos con restricciones de tipo RESTRICT en sus operaciones, horas de tipo numeric. La PRIMARY KEY debe ser compuesta de codigo_investigador y codigo_contrato.**

Se pide:

- Indicar el proceso seguido para generar esta base de datos.



Lo primero que hemos hecho es crear la tabla e introducir los atributos que nos proporciona el enunciado en las columnas.

```
alter table investigadores_contratos
add constraint FK_codigo_investigador
foreign key (codigo_investigador)
references investigadores(codigo_investigador)
on update restrict
on delete restrict
```

Después mediante esta consulta hemos definido al atributo codigo_investigador la restricción FK.

Para codigo_contrato la consulta es similar.

```
alter table investigadores_contratos
add constraint FK_codigo_contrato
foreign key (codigo_contrato)
references contratos(codigo_contrato)
on update restrict
on delete restrict
```

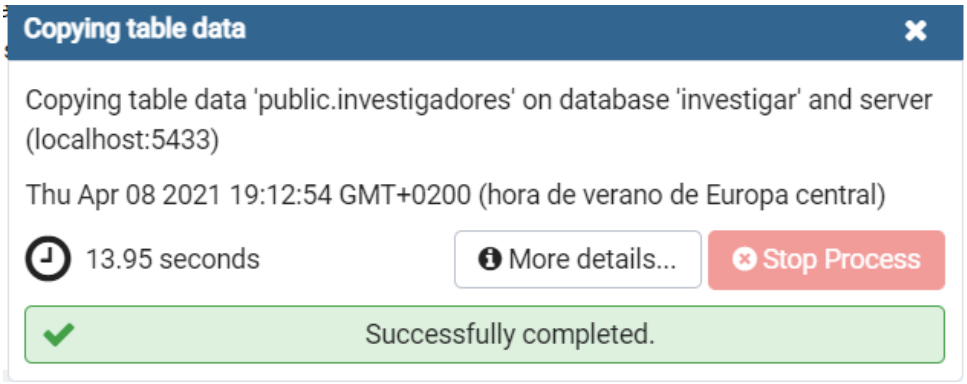
Messages Data Output

```
Query returned successfully in 63 msec.
```

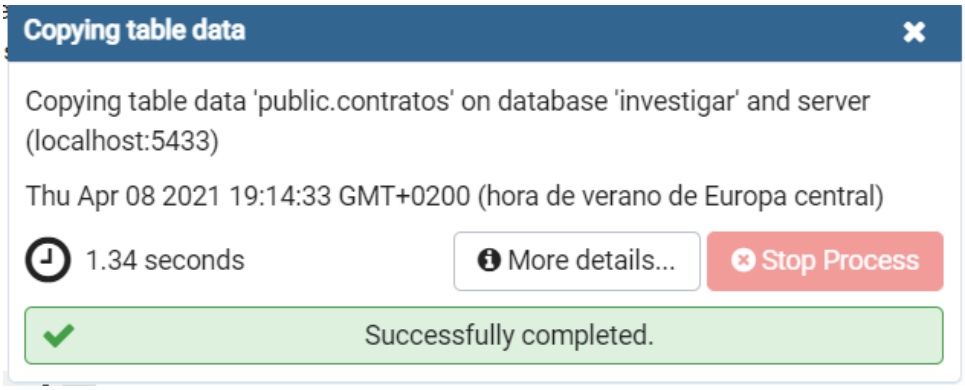
- Cargar la información del fichero datos_investigadores.csv, datos_contratos.csv y datos_investigadores_contratos.csv en dichas tablas de tal manera que sea lo más eficiente posible.

- Indicar los tiempos de carga.

Sobre la tabla investigador, el tiempo de carga ha sido de 13,95 segundos.



Sobre la tabla 'contratos' el tiempo de carga ha sido de 1.34 segundos.



Sobre la tabla 'investigadores_contratos' el tiempo de carga ha sido 469.01 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?

Las estadísticas obtenidas en este momento para cada tabla son:

- Tabla contratos:

Statistics	Value
Sequential scans	1
Sequential tuples read	0
Index scans	9000000
Index tuples fetched	9000000
Tuples inserted	200000
Tuples updated	0
Tuples deleted	0
Tuples HOT updated	0
Live tuples	200000
Dead tuples	0
Heap blocks read	3589
Heap blocks hit	27007023
Index blocks read	1102
Index blocks hit	27350209
Toast blocks read	0
Toast blocks hit	0
Toast index blocks read	0
Toast index blocks hit	0
Last vacuum	

Last autovacuum	2021-04-08 19:15:18.450606+02
Last analyze	
Last autoanalyze	2021-04-08 19:15:18.980177+02
Vacuum counter	0
Autovacuum counter	1
Analyze counter	0
Autoanalyze counter	1
Table size	14 MB
Toast table size	8192
Indexes size	4408 kB

- Tabla investigadores

Statistics	Value
Sequential scans	1
Sequential tuples read	0
Index scans	9000000
Index tuples fetched	9000000
Tuples inserted	3000000
Tuples updated	0
Tuples deleted	0
Tuples HOT updated	0
Live tuples	3000000
Dead tuples	0
Heap blocks read	6527633
Heap blocks hit	20640571
Index blocks read	2526722
Index blocks hit	27685870
Toast blocks read	0
Toast blocks hit	0
Toast index blocks read	0
Toast index blocks hit	0
Last vacuum	
Last autovacuum	2021-04-08 19:14:16.816435+02
Last analyze	
Last autoanalyze	2021-04-08 19:14:37.599747+02
Vacuum counter	0
Autovacuum counter	1
Analyze counter	0
Autoanalyze counter	1
Table size	219 MB
Toast table size	8192
Indexes size	64 MB

- Tabla investigadores_contratos:

Statistics	Value
Sequential scans	4
Sequential tuples read	0
Index scans	0
Index tuples fetched	0
Tuples inserted	9000000
Tuples updated	0
Tuples deleted	0
Tuples HOT updated	0
Live tuples	8999977
Dead tuples	0
Heap blocks read	201739
Heap blocks hit	18114908
Index blocks read	2453278
Index blocks hit	24642887
Toast blocks read	0
Toast blocks hit	0
Toast index blocks read	0
Toast index blocks hit	0

Last vacuum	
Last autovacuum	2021-04-08 19:27:00.383676+02
Last analyze	
Last autoanalyze	2021-04-08 19:27:23.795396+02
Vacuum counter	0
Autovacuum counter	1
Analyze counter	0
Autoanalyze counter	1
Table size	448 MB
Toast table size	8192
Indexes size	346 MB

Cuestión 4: Aplicar el comando EXPLAIN a una consulta que obtenga la información de los investigadores con salario de menos de 50000 euros. ¿Son correctos los resultados del comando EXPLAIN? ¿Por qué? Comparar con lo que se obtendría con el procedimiento visto en teoría.

Query Editor Query History

```
1  explain select * from investigadores where salario<50000
```

Data Output Explain Messages Notifications

	QUERY PLAN text
1	Seq Scan on investigadores (cost=0.00..65523.00 rows=742030 width=41)
2	Filter: (salario < '50000'::numeric)

Como se puede apreciar en la captura realiza una lectura secuencial ('Seq Scan'). Según el EXPLAIN el coste de puesta en marcha estimado es cero y el coste total estimado es 65523 bloques.

Query Editor Query History

```
1  select relpages, reltuples from pg_class where relname='investigadores'
```

Data Output Explain Messages Notifications

	relpages integer	reltuples real
1	28023	3e+06

Coste total estimado = (bloques en disco*1.0) + (número de tuplas escaneadas* 0.01) = (28023*1.0) + (3000000*0.01) = **58023,** se aproxima a 65523 pero no llega a ser igual, debido a que no se han leído todos los bloques, ya que hay datos que postgresql guarda en la caché y por tanto ya no accede al disco para buscarlos. Es por esto que el número de bloques totales no coincide con el que realmente ha leído.

A continuación, calcularemos de forma teórica el coste total estimado:

Lectura secuencial:

Investigadores:

Lr = 6+6+13+16 = 41 bytes (Coincide con el campo 'width' del EXPLAIN)

Nr = 3000000 tuplas (No coincide con el campo 'rows' del EXPLAIN, el comando EXPLAIN suele dar un número menor que el número escaneado como resultado del filtrado de WHERE)

fr=[B/Lr]=[8192/41]=199 reg/bloq

Br=[nr/fr]=[3000000/199]=**15076 bloques**

Como podemos comprobar da muy distinto a lo que dice EXPLAIN, la mayoría de las veces EXPLAIN obtiene un coste mayor. Esto se debe a, como explicamos en la PECL1, que Postgres no considera que todos los bloques estén al 100% ocupados y nosotros sí.

Cuestión 5: Aplicar el comando EXPLAIN a una consulta que obtenga el nombre de los investigadores en los cuales están asignados a contratos por 8 horas o tienen un salario

de 30.000 euros. ¿Son correctos los resultados del comando EXPLAIN? ¿Por qué? Comparar con lo que se obtendría con el procedimiento visto en teoría.

```
1  explain select nombre from investigadores inner join investigadores_contratos on
2  investigadores.codigo_investigador= investigadores_contratos.codigo_investigador
3  where investigadores_contratos.horas = 8 or investigadores.salario = 30000
```

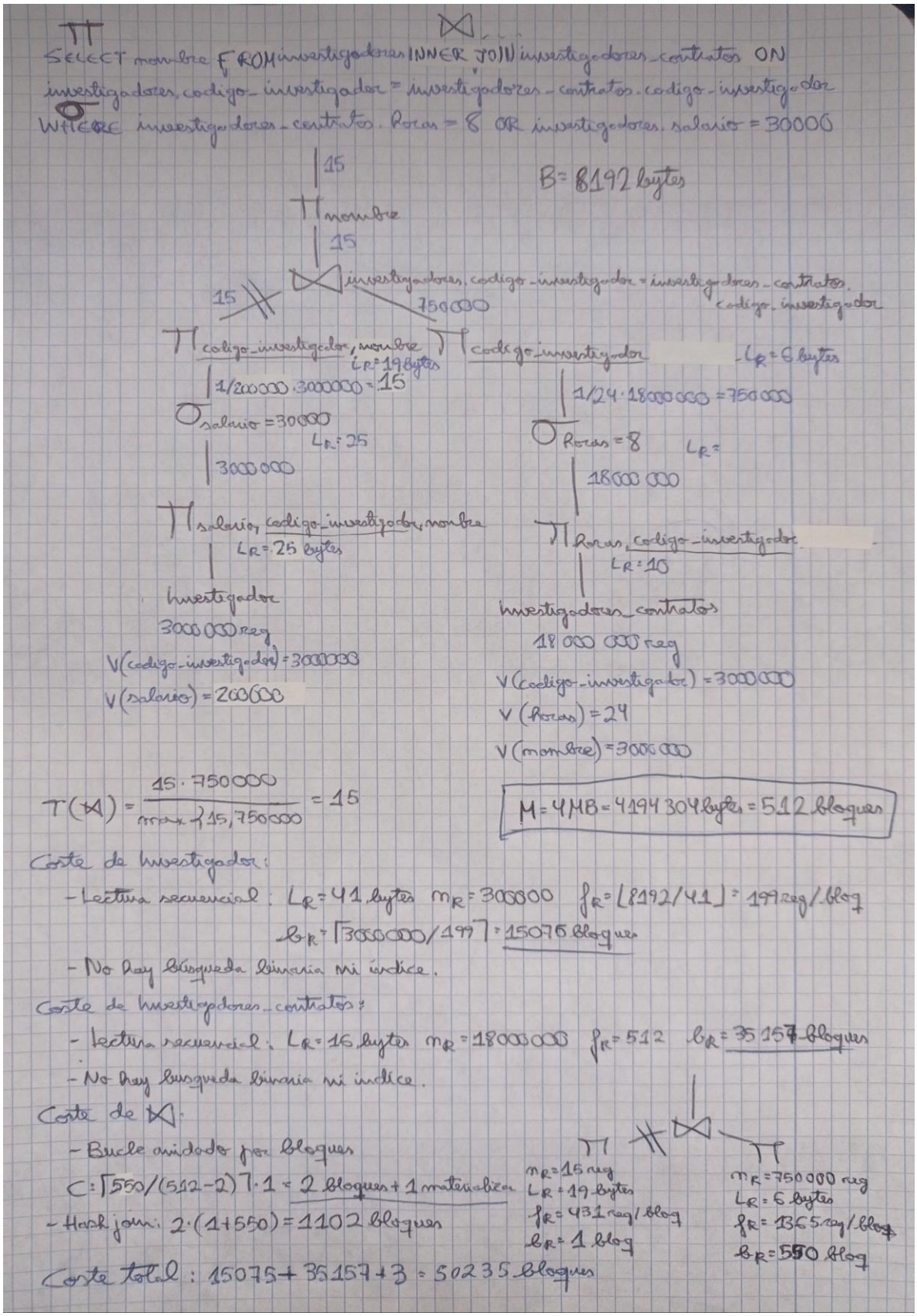
Messages | Data Output

	QUERY PLAN text
1	Gather (cost=65693.00..252583.03 rows=370544 width=13)
2	Workers Planned: 2
3	-> Parallel Hash Join (cost=64693.00..214528.63 rows=154393 width=13)
4	Hash Cond: (investigadores_contratos.codigo_investigador = investigadores.codigo_investigador)
5	Join Filter: ((investigadores_contratos.horas = '8'::numeric) OR (investigadores.salario = '30000'::numeric))
6	-> Parallel Seq Scan on investigadores_contratos (cost=0.00..94824.90 rows=3749990 width=10)
7	-> Parallel Hash (cost=40523.00..40523.00 rows=1250000 width=25)
8	-> Parallel Seq Scan on investigadores (cost=0.00..40523.00 rows=1250000 width=25)

El coste estimado según Postgres es **252583 bloques.**

```
#work_mem = 4MB
```

M =4MB= 4194304 Bytes / 8192 =512 bloques



Como se puede apreciar en la imagen, si seguimos los pasos de teoría en clase el coste total es de **50235 bloques**.

Cuestión 6: Aplicar el comando EXPLAIN a una consulta que obtenga la información del nombre de los contratos y entidad que tienen un coste mayor de 15000, y tienen empleados de salario de 24000 euros y trabajan menos de 2 horas en ellos. ¿Son correctos los resultados del comando EXPLAIN? ¿Por qué? Comparar con lo que se obtendría con lo visto en teoría.

explain select contratos.nombre, contratos.entidad from contratos inner join investigadores_contratos

on contratos.codigo_contrato=investigadores_contratos.codigo_contrato inner join investigadores

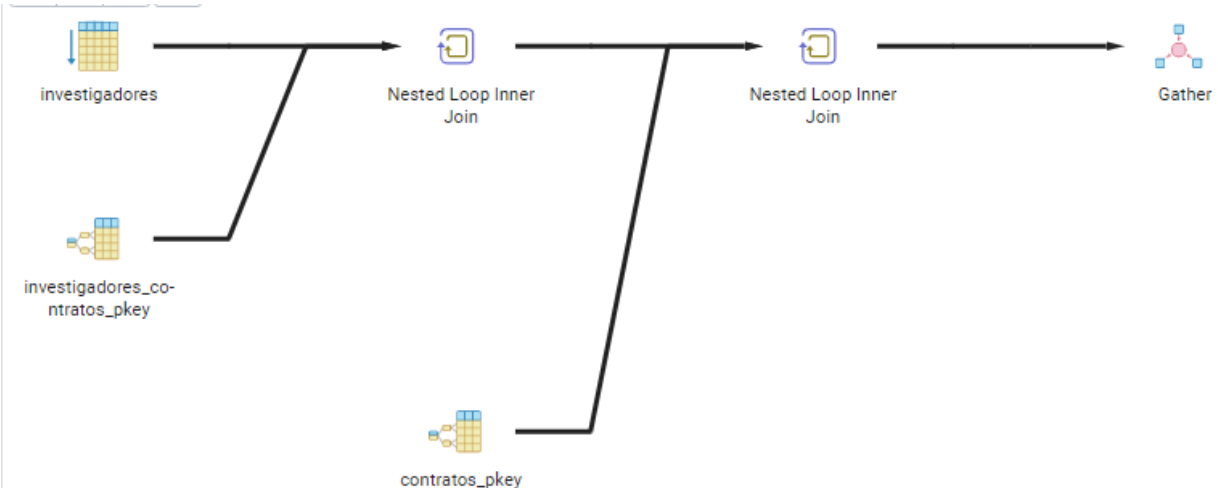
on investigadores.codigo_investigador=investigadores_contratos.codigo_investigador

where coste>15000 and salario=24000 and horas<2;

```
1  explain select contratos.nombre, contratos.entidad from contratos inner join investigadores_contratos
2  on contratos.codigo_contrato=investigadores_contratos.codigo_contrato inner join investigadores
3  on investigadores.codigo_investigador=investigadores_contratos.codigo_investigador
4  where coste>15000 and salario=24000 and horas<2;
```

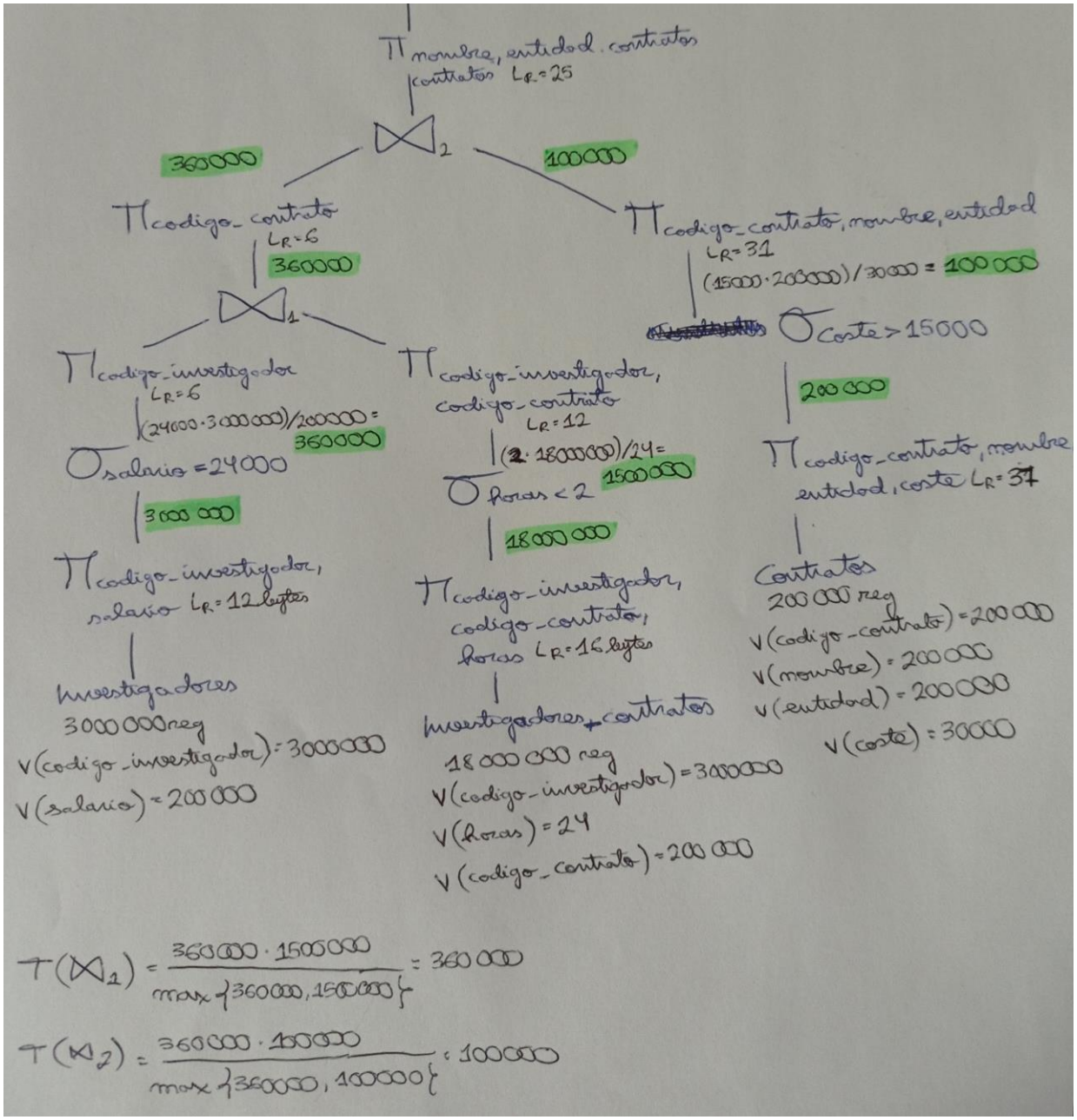
Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Gather (cost=1000.86..44792.88 rows=3 width=25)
2	Workers Planned: 2
3	-> Nested Loop (cost=0.85..43792.58 rows=1 width=25)
4	-> Nested Loop (cost=0.43..43791.67 rows=2 width=6)
5	-> Parallel Seq Scan on investigadores (cost=0.00..43648.00 rows=7 width=6)
6	Filter: (salario = '24000'::numeric)
7	-> Index Scan using investigadores_contratos_pkey on investigadores_contratos (cost=0.43..20.51 rows=1 width=12)
8	Index Cond: (codigo_investigador = investigadores.codigo_investigador)
9	Filter: (horas < '2'::numeric)
10	-> Index Scan using contratos_pkey on contratos (cost=0.42..0.45 rows=1 width=31)
11	Index Cond: (codigo_contrato = investigadores_contratos.codigo_contrato)
12	Filter: (coste > '15000'::numeric)



Nos hemos dado cuenta de que el join entre investigadores e investigadores_contratos (360000 tuplas) que hace Postgres es más caro que si hiciese el primer join entre investigadores_contratos y contratos (100000 tuplas). Entendemos que lo ha hecho de esa forma por el orden en que hemos realizado nosotros la consulta en el pgAdmin.

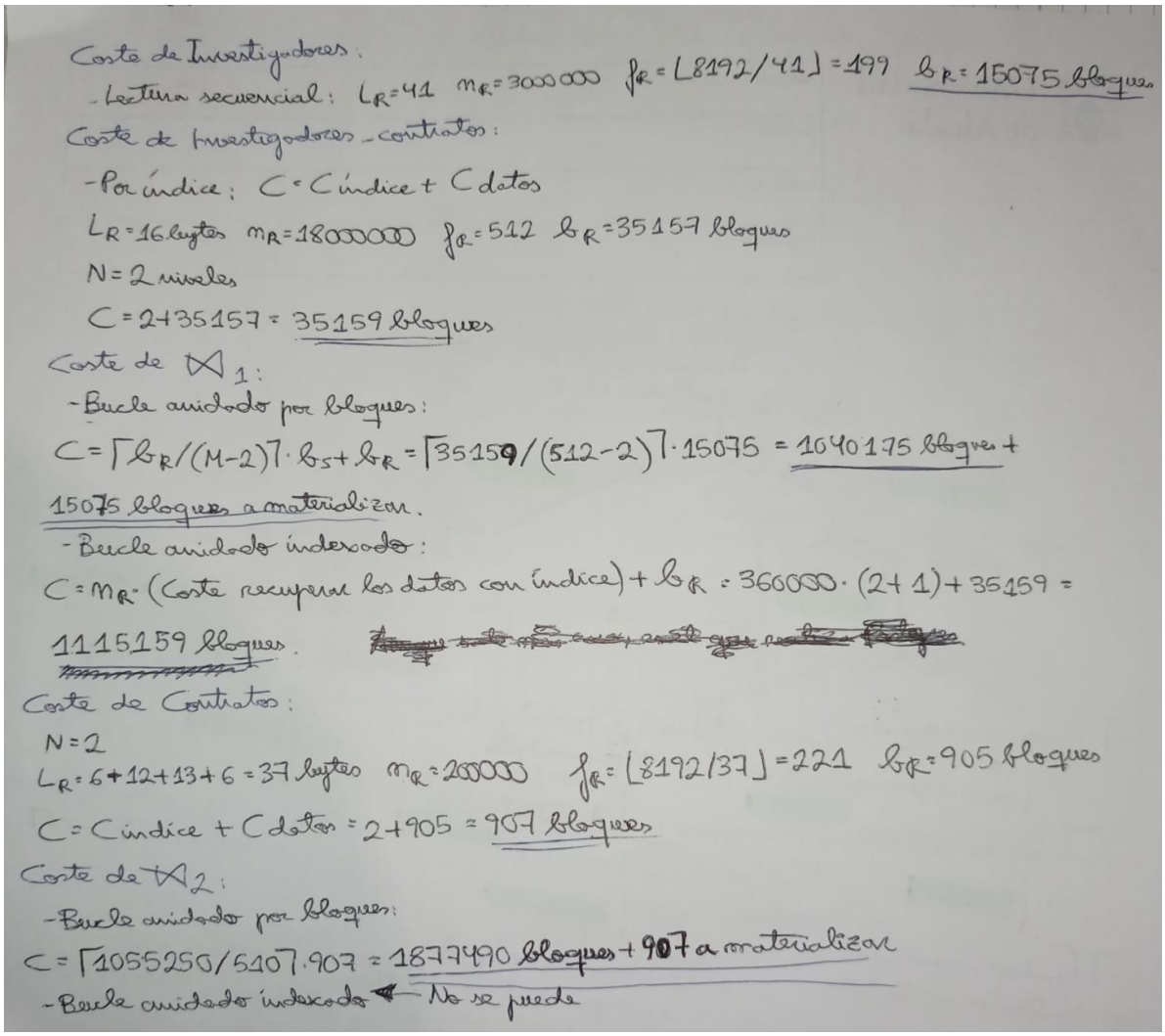
A continuación, unas capturas de como lo hemos calculado teóricamente (el árbol se ha dibujado como lo hace Postgres):



Para sacar el número de niveles del índice hemos realizado la consulta “select * from pgstatindex(OID de la PK)”.

Ninvestigadores_contrato = 2.

Ncontratos = 2.



Cuestión 7: Generar los datos solicitados al comienzo de la práctica para la base de datos **NETFLIX**.

Cuestión 8: Realizar la carga masiva de los datos generados en la cuestión 7 en la base de datos **NETFLIX**. Indicar el proceso seguido y el orden de carga de las tablas, explicando el porqué de dicho orden. 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)
Suscripcion	0.17
Cuenta_usuario	56.32
Pago	1088,76
Contenidos	105.05
Series	145.06
Peliculas	142.82

Suscripcion_contenidos	84.98
Genero	0.39
Genero_contenidos	1880.59

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.

Cuestión 9: Realizar una consulta SQL que muestre "el nombre y el email de los usuarios que pueden acceder a contenidos con una valoración de más de 5 y tengan películas de una duración entre 90 y 100 minutos o series con menos de 6 temporadas o más de 15 capítulos; y además que esos usuarios puedan ver contenidos que tienen más de 3 géneros asociados habiendo realizado por lo menos algún pago en los 7 primeros días del mes de mayo.

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.

La consulta a realizar es la siguiente:

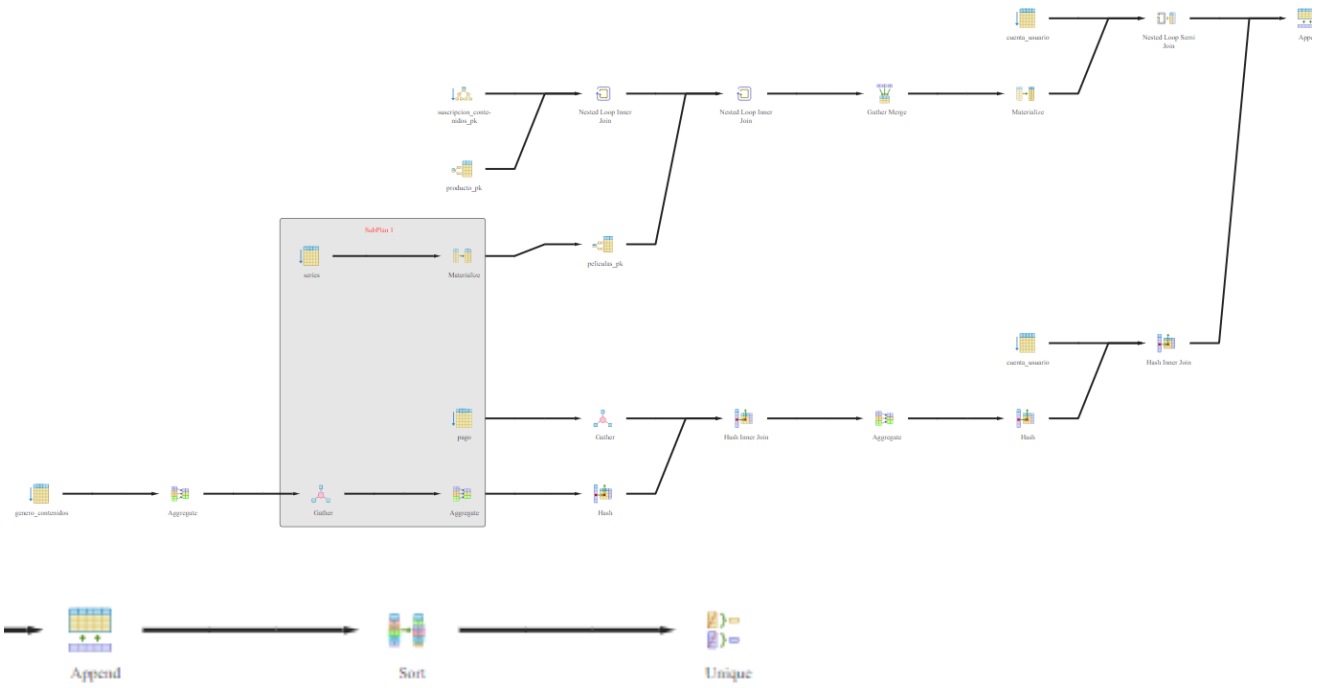
select nombre,e_mail from cuenta_usuario where suscripcion_id_suscripcion in (select suscripcion_id_suscripcion from contenidos inner join suscripcion_contenidos on contenidos.producto_id = suscripcion_contenidos.producto_id_contenidos where contenidos.valoracion >5 and contenidos.producto_id in (select producto_id_contenidos from peliculas where duracion between 90 and 100 or producto_id_contenidos in (select producto_id_contenidos from series where series.temporadas < 6 or series.capitulos>15))) union select nombre,e_mail from cuenta_usuario where suscripcion_id_suscripcion in (select suscripcion_id_suscripcion from pago where fecha between '2020-05-01' and '2020-05-07' and suscripcion_id_suscripcion in (select producto_id_contenidos from genero_contenidos group by producto_id_contenidos having count (producto_id_contenidos)>3))

```
select nombre,e_mail from cuenta_usuario where suscripcion_id_suscripcion in (
select suscripcion_id_suscripcion
from contenidos inner join suscripcion_contenidos on contenidos.producto_id = suscripcion_contenidos.producto_id_contenidos
    where contenidos.valoracion >5 and contenidos.producto_id in (
select producto_id_contenidos from peliculas where duracion between 90 and 100
        or producto_id_contenidos in (
select producto_id_contenidos from series where series.temporadas < 6
    or series.capitulos>15)))

union

select nombre,e_mail from cuenta_usuario where suscripcion_id_suscripcion in
(select suscripcion_id_suscripcion from pago where fecha
between '2020-05-01' and '2020-05-07' and suscripcion_id_suscripcion in
(select producto_id_contenidos from genero_contenidos group by producto_id_contenidos
having count (producto_id_contenidos)>3))
```

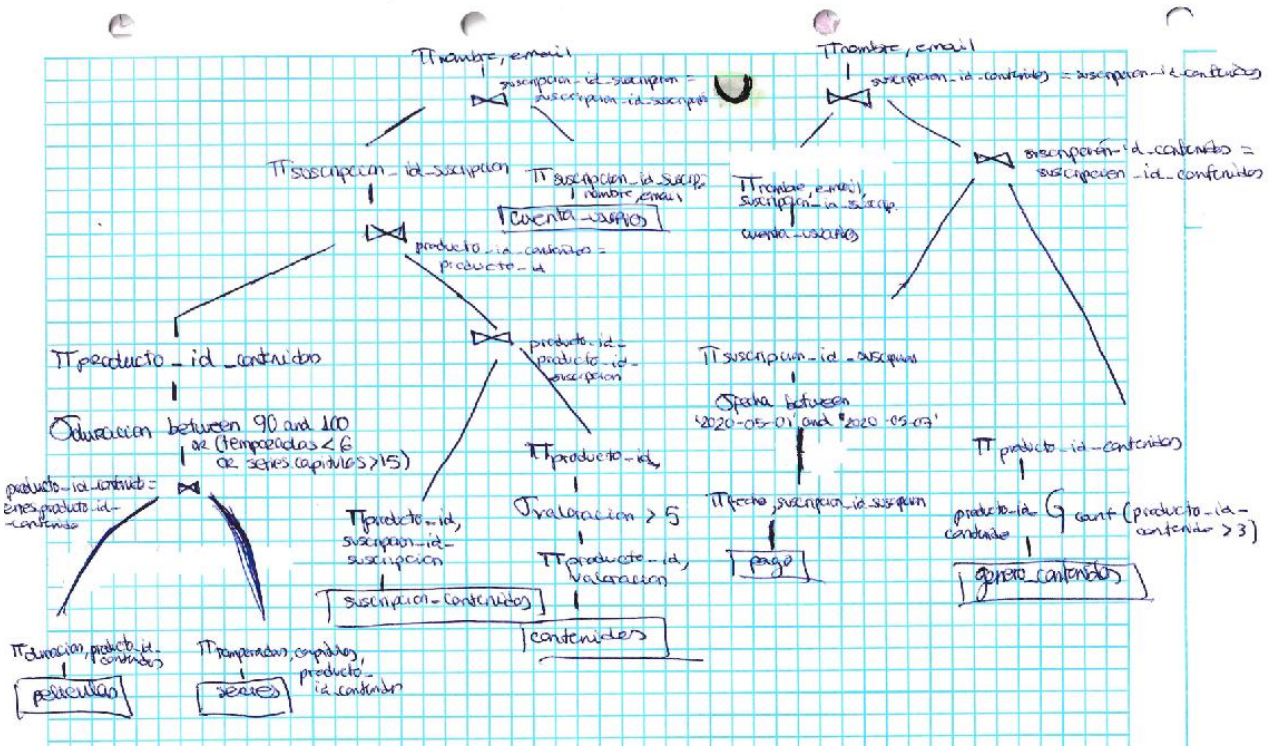
El plan de ejecución con el resultado del comando EXPLAIN es el siguiente:

	QUERY PLAN text
1	Unique (cost=54300568968.51..54300598968.51 rows=4000000 width=64)
2	-> Sort (cost=54300568968.51..54300578968.51 rows=4000000 width=64)
3	Sort Key: cuenta_usuario.nombre, cuenta_usuario.e_mail
4	-> Append (cost=1001.32..54299829554.13 rows=4000000 width=64)
5	-> Nested Loop Semi Join (cost=1001.32..54292815643.23 rows=2000000 width=36)
6	Join Filter: (cuenta_usuario.suscripcion_id_suscripcion = suscripcion_contenidos.suscripcion_id_suscripcion)
7	-> Seq Scan on cuenta_usuario (cost=0.00..46567.00 rows=2000000 width=41)
8	-> Materialize (cost=1001.32..54285207175.35 rows=345605 width=5)
9	-> Gather Merge (cost=1001.32..54285204096.32 rows=345605 width=5)
10	Workers Planned: 2
11	-> Nested Loop (cost=1.30..54285163204.91 rows=144002 width=5)
12	-> Nested Loop (cost=0.86..798318.79 rows=417195 width=17)
13	-> Parallel Index Only Scan using suscripcion_contenidos_pk on suscripcion_contenidos (cost=0.43..49525.76 rows=833333 width=11)
14	-> Index Scan using producto_pk on contenidos (cost=0.44..0.90 rows=1 width=6)
15	Index Cond: (producto_id = suscripcion_contenidos.producto_id_contenidos)
16	Filter: (valoracion > '5'::numeric)
17	-> Index Scan using peliculas_pk on peliculas (cost=0.43..130117.49 rows=1 width=6)
18	Index Cond: (producto_id_contenidos = contenidos.producto_id)
19	Filter: (((duracion >= '90'::numeric) AND (duracion <= '100'::numeric)) OR (SubPlan 1))
20	SubPlan 1
21	-> Materialize (cost=0.00..245779.57 rows=5781765 width=6)
22	-> Seq Scan on series (cost=0.00..194284.74 rows=5781765 width=6)
23	Filter: ((temporadas < '6'::numeric) OR (capitulos > '15'::numeric))
24	-> Hash Join (cost=6879843.90..6953910.90 rows=2000000 width=36)
25	Hash Cond: (cuenta_usuario_1.suscripcion_id_suscripcion = pago.suscripcion_id_suscripcion)
26	-> Seq Scan on cuenta_usuario cuenta_usuario_1 (cost=0.00..46567.00 rows=2000000 width=41)
27	-> Hash (cost=6879842.65..6879842.65 rows=100 width=11)
28	-> HashAggregate (cost=6879841.65..6879842.65 rows=100 width=11)
29	Group Key: pago.suscripcion_id_suscripcion
30	-> Hash Join (cost=3158450.40..3590515.30 rows=1315730541 width=11)
31	Hash Cond: (pago.suscripcion_id_suscripcion = genero_contenidos.producto_id_contenidos)
32	-> Gather (cost=1000.00..425731.93 rows=504349 width=5)
33	Workers Planned: 2
34	-> Parallel Seq Scan on pago (cost=0.00..374297.03 rows=210145 width=5)
35	Filter: ((fecha >= '2020-05-01'::date) AND (fecha <= '2020-05-07'::date))
36	-> Hash (cost=3148889.48..3148889.48 rows=521754 width=6)
37	-> Finalize GroupAggregate (cost=2743200.50..3143671.94 rows=521754 width=6)
38	Group Key: genero_contenidos.producto_id_contenidos
39	Filter: (count(genero_contenidos.producto_id_contenidos) > 3)
40	-> Gather Merge (cost=2743200.50..3108453.56 rows=3130522 width=14)
41	Workers Planned: 2
42	-> Sort (cost=2742200.47..2746113.62 rows=1565261 width=14)
43	Sort Key: genero_contenidos.producto_id_contenidos
44	-> Partial HashAggregate (cost=2310863.71..2554400.49 rows=1565261 width=14)
45	Group Key: genero_contenidos.producto_id_contenidos
46	Planned Partitions: 64
47	-> Parallel Seq Scan on genero_contenidos (cost=0.00..670097.73 rows=29169173 width=6)

Realizando un f7 con el teclado, podemos obtener el árbol de álgebra relacional hecho por PostgreSQL



El dibujo del álgebra relacional realizado de manera teórica es el siguiente:



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.

Medidas para mejorar el rendimiento:

1. Aumentar la M (Aumentar el valor de work_mem de 4MB a 8MB, maintenance_work_mem de 64MB a 128MB y el valor de checkpoint_segments en el archivo postgresql.conf) . Esta medida permitirá que las ramas que se materialicen se puedan encauzar. Puede ayudar a la base de datos a optimizar su configuración para su configuración. Esto permite que PostgreSQL almacene en caché más datos en la memoria mientras realiza su clasificación, en lugar de realizar llamadas costosas al disco. La maintenance_work_mem nos puede ayudar a optimizar operaciones como realizar vacuum, crear índices y modificar tablas.
2. Otra medida para aumentar el rendimiento sería hacer un Hash Join en vez de un bucle anidado por bloques entre Cuenta_usuario y la rama que sale del anterior join. Esto es porque con Hash Join se pueden encauzar ambas ramas y no solo una como pasa con los bucles anidados.
3. Reducir las proyecciones que se llevan a cabo en los niveles más bajos. Para que ocurra esto en las tablas en las que se realiza una lectura secuencial deberíamos hacer una búsqueda binaria (habría que hacer la tabla ordenada por el campo A) o disponer de algún índice sobre el campo A.

Creación de árboles:

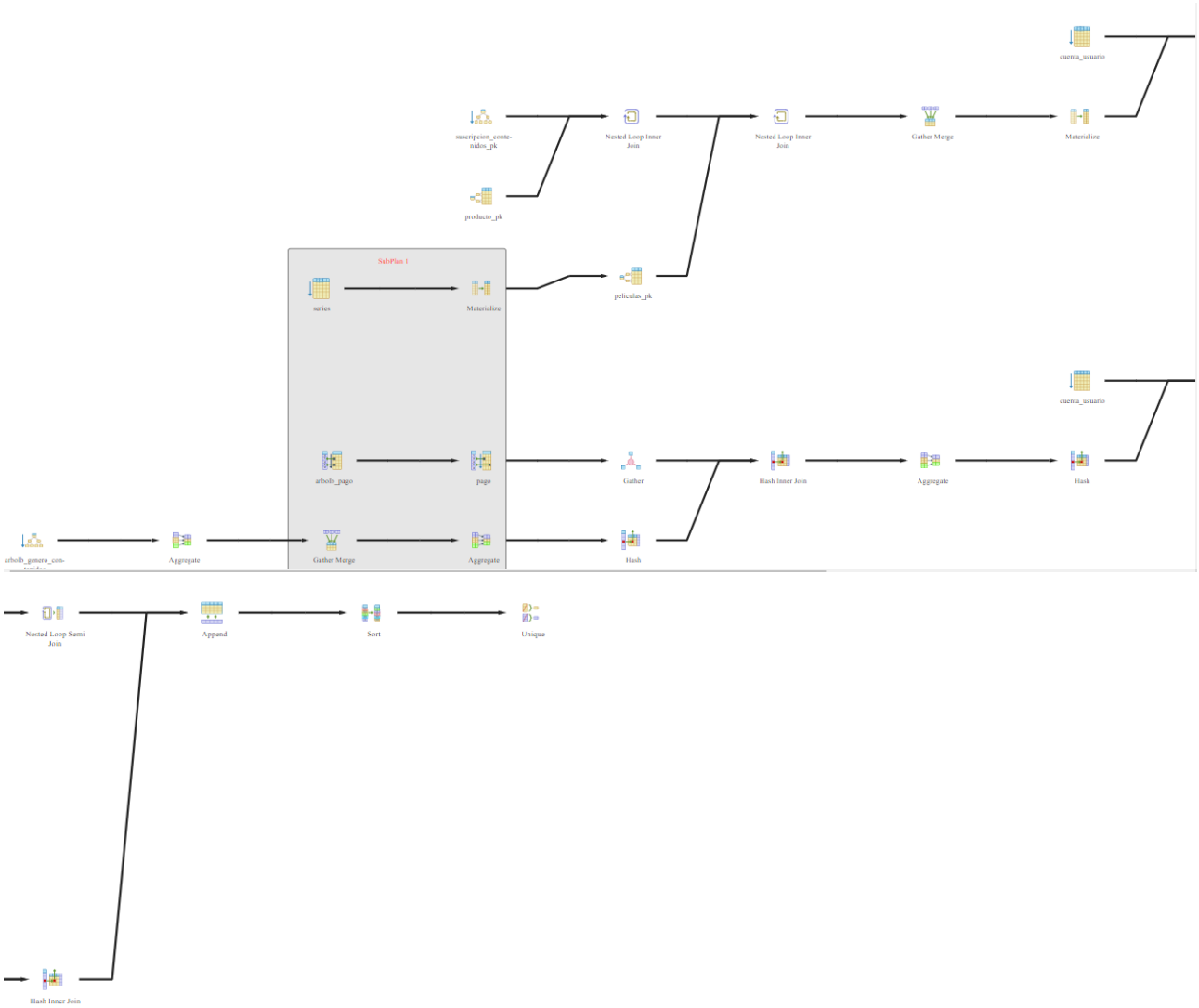
- *create index arbolb_cuenta_usuario on cuenta_usuario using btree(suscripcion_id_suscripcion)*
- *create index arbolb_series on series using btree(capitulos,temporadas)*
- *create index arbolb_pago on pago using btree(fecha)*
- *create index arbolb_genero_contenidos on genero_contenidos using btree(producto_id_contenidos)*

Después de esto realizaríamos un *set enable_seqscan = off* para desactivar el escaneo secuencial de la tabla y utilizar los índices.

4. Realizar las ramas que produzcan menos tuplas antes que las que más producen.
5. Ejecutar VACUUM ANALYZE tan a menudo como sea necesario (con autovacuum).

Salida del explain ejecutando la consulta del ejercicio anterior una vez realizados los cambios:

	QUERY PLAN text
1	Unique (cost=54195042552.41..54195072552.41 rows=4000000 width=64)
2	-> Sort (cost=54195042552.41..54195052552.41 rows=4000000 width=64)
3	Sort Key: cuenta_usuario.nombre, cuenta_usuario.e_mail
4	-> Append (cost=1001.32..54194303138.03 rows=4000000 width=64)
5	-> Nested Loop Semi Join (cost=1001.32..54188161415.16 rows=2000000 width=36)
6	Join Filter: (cuenta_usuario.suscripcion_id_suscripcion = suscripcion_contenidos.suscripcion_id_suscripcion)
7	-> Seq Scan on cuenta_usuario (cost=0.00..46567.00 rows=2000000 width=41)
8	-> Materialize (cost=1001.32..54180555587.90 rows=346459 width=5)
9	-> Gather Merge (cost=1001.32..54180552501.61 rows=346459 width=5)
10	Workers Planned: 2
11	-> Nested Loop (cost=1.30..54180511511.62 rows=144358 width=5)
12	-> Nested Loop (cost=0.86..798318.79 rows=416222 width=17)
13	-> Parallel Index Only Scan using suscripcion_contenidos_pk on suscripcion_contenidos (cost=0.43..49525.76 rows=833333 width=11)
14	-> Index Scan using producto_pk on contenidos (cost=0.44..0.90 rows=1 width=6)
15	Index Cond: (producto_id = suscripcion_contenidos.producto_id_contenidos)
16	Filter: (valoracion > '5'::numeric)
17	-> Index Scan using peliculas_pk on peliculas (cost=0.43..130170.23 rows=1 width=6)
18	Index Cond: (producto_id_contenidos = contenidos.producto_id)
19	Filter: (((duracion >= '90'::numeric) AND (duracion <= '100'::numeric)) OR (SubPlan 1))
20	SubPlan 1
21	-> Materialize (cost=0.00..245860.74 rows=5791491 width=6)
22	-> Seq Scan on series (cost=0.00..194279.29 rows=5791491 width=6)
23	Filter: ((temporadas < '6'::numeric) OR (capitulos > '15'::numeric))
24	-> Hash Join (cost=6007655.87..6081722.87 rows=2000000 width=36)
25	Hash Cond: (cuenta_usuario_1.suscripcion_id_suscripcion = pago.suscripcion_id_suscripcion)
26	-> Seq Scan on cuenta_usuario cuenta_usuario_1 (cost=0.00..46567.00 rows=2000000 width=41)
27	-> Hash (cost=6007654.62..6007654.62 rows=100 width=11)
28	-> HashAggregate (cost=6007653.62..6007654.62 rows=100 width=11)
29	Group Key: pago.suscripcion_id_suscripcion
30	-> Hash Join (cost=2049932.34..2457870.57 rows=1419913220 width=11)
31	Hash Cond: (pago.suscripcion_id_suscripcion = genero_contenidos.producto_id_contenidos)
32	-> Gather (cost=7518.08..408119.46 rows=477819 width=5)
33	Workers Planned: 2
34	-> Parallel Bitmap Heap Scan on pago (cost=6518.08..359337.56 rows=199091 width=5)
35	Recheck Cond: ((fecha >= '2020-05-01'::date) AND (fecha <= '2020-05-07'::date))
36	-> Bitmap Index Scan on arbolb_pago (cost=0.00..6398.63 rows=477819 width=0)
37	Index Cond: ((fecha >= '2020-05-01'::date) AND (fecha <= '2020-05-07'::date))
38	-> Hash (cost=2032663.12..2032663.12 rows=594331 width=6)
39	-> Finalize GroupAggregate (cost=1000.59..2026719.81 rows=594331 width=6)
40	Group Key: genero_contenidos.producto_id_contenidos
41	Filter: (count(genero_contenidos.producto_id_contenidos) > 3)
42	-> Gather Merge (cost=1000.59..1986602.47 rows=3565986 width=14)
43	Workers Planned: 2
44	-> Partial GroupAggregate (cost=0.57..1573999.18 rows=1782993 width=14)
45	Group Key: genero_contenidos.producto_id_contenidos
46	-> Parallel Index Only Scan using arbolb_genero_contenidos on genero_contenidos (cost=0.57..1410321.71 rows=29169507 width=6)



Cuestión 11: Usando PostgreSQL, borre el 50% de los contenidos almacenados de manera aleatoria y todos sus datos relacionados ¿Cuál ha sido el proceso seguido? ¿Y el tiempo empleado en el borrado? Ejecute la consulta 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.

Haciendo uso de los btrees realizados en la cuestión 10, procedemos a hacer el borrado.

Primero hemos cambiado la configuración del fichero postgresql.config con los parámetros siguientes:

- Archive_mode=off

```
# - Archiving -

archive_mode = off
```

- Wal_level=minimal

```
# - Settings -

wal_level = minimal
```

- Max_wal_senders=0;

```
# Set these on the r

max_wal_senders =0
```

- Max_wal_size=1gb

```
max_wal_size = 1GB
```

Se ha reiniciado el servidor para que se efectúen los cambios.

Posterior a esta configuración, hemos borrado todas las claves foráneas de todas las tablas para agilizar la lectura de datos, es decir, se ha desactivado la integridad referencial puesto que al ser las tablas con una gran cantidad de datos, el tiempo estimado de borrado sería demasiado largo.

```
2  ALTER TABLE contenidos DISABLE TRIGGER ALL
3
```

Después hemos creado una tabla temporal que almacene temporalmente los datos que se van a borrar, en este caso, el 50% de los datos que son 10000000 contenidos seleccionados aleatoriamente de la tabla contenidos por el campo producto_id.

Select producto_id into temp table tmp_contenidos from contenidos order by random() limit 10000000

Posteriormente, hemos creado un árbol btree sobre la tabla temporal:

create index indice_temporal on tmp_contenidos using btree(producto_id)

```
4  create index indice_temporal on tmp_contenidos using btree(producto_id)
```

Data Output | Messages

```
Query returned successfully in 12 secs 20 msec.
```

Y lo ordenamos a través de ese árbol llamado “indice_temporal”

Finalmente, borramos los contenidos de otras tablas que estén asociados a la tabla contenidos: contenidos, peliculas y series.

Delete from suscripcion_contenidos using tmp_contenidos where suscripcion_contenidos.producto_id_contenidos = tmp_contenidos.producto_id

```
8
9  Delete from suscripcion_contenidos
10 using tmp_contenidos
11 where suscripcion_contenidos.producto_id_contenidos = tmp_contenidos.producto_id
12
13
14
15
16
17
```

Data Output | Messages

```
Query returned successfully in 18 secs 123 msec.
```

Delete from series using tmp_contenidos where series.producto_id_contenidos = tmp_contenidos.producto_id

```
8
9  Delete from series
10 using tmp_contenidos
11 where series.producto_id_contenidos = tmp_contenidos.producto_id
```

Data Output | Messages

```
Query returned successfully in 34 secs 225 msec.
```

Delete from peliculas using tmp_contenidos where peliculas.producto_id_contenidos = tmp_contenidos.producto_id

```
8
9  Delete from peliculas
10 using tmp_contenidos
11 where peliculas.producto_id_contenidos = tmp_contenidos.producto_id
12
```

Data Output | Messages

```
Query returned successfully in 42 secs 511 msec.
```

Delete from contenidos using tmp_contenidos where contenidos.producto_id=tmp_contenidos.producto_id

```
9  Delete from contenidos
10 using tmp_contenidos
11 where contenidos.producto_id = tmp_contenidos.producto_id
12
13
14
15
16
17
```

Data Output | Messages

```
Query returned successfully in 1 min 6 secs.
```

Delete from genero_contenidos using tmp_contenidos where genero_contenidos.producto_id_contenidos = tmp_contenidos.producto_id

```
Delete from genero_contenidos
using tmp_contenidos
where genero_contenidos.producto_id_contenidos = tmp_contenidos.producto_id
```

Para comprobar que se han borrado los datos, se ha realizado la siguiente consulta:

Select count (producto_id) from contenidos

Ahora procedemos a volver a activar la integridad referencial y creación de fk.

```
10
11  ALTER TABLE contenidos ENABLE TRIGGER ALL
12
```

Data Output | Messages

Query returned successfully in 1 secs 764 msec.

Realizando un explain de la consulta ahora, el coste es menor puesto que hay menos tuplas que antes y hay árboles btree:

Query Editor | Query History | Notifications | Explain

```
40  select producto_id_contenidos from peliculas where duracion between 90 and 100
41         or producto_id_contenidos in (
42  select producto_id_contenidos from series where series.temporadas < 6
43      or series.capitulos>15)))
44  union
45  select nombre,e_mail from cuenta_usuario where suscripcion_id_suscripcion in
46  (select suscripcion_id_suscripcion from pago where fecha
47  between '2020-05-01' and '2020-05-07' and suscripcion_id_suscripcion in
48  (select producto_id_contenidos from genero_contenidos group by producto_id_contenidos
49  having count (producto_id_contenidos)>3))
```

Data Output | Messages

	QUERY PLAN text
1	Unique (cost=5218555158.94..5218555158.94 rows=2000000 width=64)
2	-> Sort (cost=5218555158.94..5218555158.94 rows=2000000 width=64)
3	Sort Key: cuenta_usuario.nombre, cuenta_usuario.e_mail
4	-> Append (cost=1000.00..5218469218.94 rows=2000000 width=64)
5	-> Merge Semi Join (cost=1000.00..5218251662.94 rows=1000000 width=64)
6	Merge Cond: (cuenta_usuario.suscripcion_id_suscripcion = suscripcion_contenidos.suscripcion_id_suscripcion)
7	-> Index Scan using arbol_cuenta_usuario on cuenta_usuario (cost=0.00..32054.00 rows=2000000 width=96)
8	-> Gather Merge (cost=1000.00..5218219608.94 rows=91343 width=32)

Cuestión 12: ¿Qué técnicas de optimización de la BD propondría para mejorar los resultados de dicho plan sin modificar el código de la consulta? ¿Por qué?

Para mejorar los resultados tras el borrado proponemos el uso de las siguientes técnicas:

- El uso del VACUUM, ya que según hemos comprobado, en la operación normal de PostgreSQL , las tuplas eliminadas u obsoletas por una actualización no se eliminan físicamente de su tabla; permanecen presentes hasta que se ejecuta el

26

VACUUM . Por lo tanto, es necesario hacer un VACUUM periódicamente, especialmente en tablas actualizadas con frecuencia.

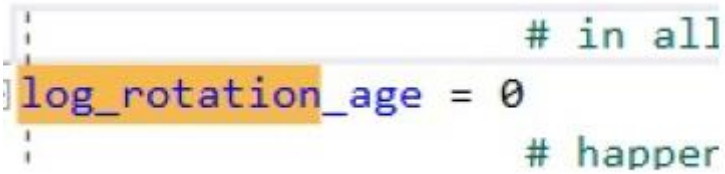
- Activar el AUTOVACUUM en el archivo postgresql.conf con el valor de on, ya que está compuesto por múltiples procesos que reclaman el almacenamiento al eliminar datos obsoletos o tuplas de la base de datos. Este omprueba las tablas que tienen un número significativo de registros insertados, actualizados o eliminados y aspira estas tablas en función de los ajustes de configuración a continuación.

- Deshabilitar el log_rotation_age en el archivo postgres.conf dándole el valor de 0 ya que, cuando logging_colector está habilitado, este parámetro determina la vida útil máxima de un archivo de registro individual. Después de que hayan transcurrido estos minutos, se creará un nuevo archivo de registro. De este modo, establecemos el valor en 0 para deshabilitar la creación basada en el tiempo de nuevos archivos de registro.

- Si se sospecha que un índice está dañado en una tabla de usuario, simplemente se puede reconstruir ese índice, o todos los índices en la tabla, utilizando REINDEX INDEX o REINDEX TABLE . REINDEX reconstruye un índice utilizando los datos almacenados en la tabla del índice, reemplazando la copia anterior del índice. Por esta razón, es recomendable ejecutar este comando tras haber hecho el borrado.

Cuestión 13: Usando PostgreSQL, lleve a cabo las operaciones propuestas en la cuestión anterior y ejecute el plan de ejecución de la misma consulta. 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.

Primero activaremos el log_rotation_age a 0.



Realizaremos un vacuum para que se eliminen las tuplas muertas después del borrado.

```
8  vacuum verbose contenidos,series, peliculas
9
10
```

Data Output Messages

```
INFO:  «peliculas»: se eliminaron 5453231 versiones de filas en 58965 páginas
DETAIL:  CPU: usuario: 0.35 s, sistema: 0.32 s, transcurrido: 3.48 s
INFO:  el índice «peliculas_pk» ahora contiene 5455150 versiones de filas en 29912 páginas
DETAIL:  5453231 versiones de filas del índice fueron eliminadas.
0 páginas de índice han sido eliminadas, 0 son reusables.
CPU: usuario: 0.00 s, sistema: 0.00 s, transcurrido: 0.00 s.
INFO:  «peliculas»: se encontraron 20 versiones de filas eliminables y 5455150 no eliminables
en 58965 de 58965 páginas
DETAIL:  0 versiones muertas de filas no pueden ser eliminadas aún, xmin máx antiguo: 1489
Hubo 0 punteros de ítem sin uso.
Omitidas 0 páginas debido a «pins» de búfers, 0 páginas marcadas «frozen».
0 páginas están completamente vacías.
CPU: usuario: 2.67 s, sistema: 0.75 s, transcurrido: 6.64 s.
INFO:  haciendo vacuum a «pg_toast.pg_toast_90694»
INFO:  el índice «pg_toast_90694_index» ahora contiene 0 versiones de filas en 1 páginas
DETAIL:  0 versiones de filas del índice fueron eliminadas.
0 páginas de índice han sido eliminadas, 0 son reusables.
CPU: usuario: 0.00 s, sistema: 0.00 s, transcurrido: 0.00 s.
INFO:  «pg_toast_90694»: se encontraron 0 versiones de filas eliminables y 0 no eliminables en
0 de 0 páginas
DETAIL:  0 versiones muertas de filas no pueden ser eliminadas aún, xmin máx antiguo: 1489
```

```
7
8  vacuum verbose genero_contenidos,suscripcion_contenidos
9
```

Data Output Messages

```
INFO:  se recorrió el índice «arbolsuscripcion_contenidos» para eliminar 1449912 versiones de
filas
DETAIL:  CPU: usuario: 2.84 s, sistema: 0.43 s, transcurrido: 3.33 s
INFO:  «genero_contenidos»: se eliminaron 1449912 versiones de filas en 15685 páginas
DETAIL:  CPU: usuario: 0.06 s, sistema: 0.18 s, transcurrido: 0.73 s
INFO:  el índice «genero_contenidos_pk» ahora contiene 35005088 versiones de filas en 514667
páginas
DETAIL:  35001808 versiones de filas del índice fueron eliminadas.
0 páginas de índice han sido eliminadas, 0 son reusables.
CPU: usuario: 0.00 s, sistema: 0.00 s, transcurrido: 0.00 s.
INFO:  el índice «arbol_genero_contenidos» ahora contiene 35005088 versiones de filas en 191955
páginas
DETAIL:  35001727 versiones de filas del índice fueron eliminadas.
0 páginas de índice han sido eliminadas, 0 son reusables.
CPU: usuario: 0.00 s, sistema: 0.00 s, transcurrido: 0.00 s.
INFO:  el índice «arbolgenero_contenidos» ahora contiene 35005088 versiones de filas en 191955
páginas
DETAIL:  35001727 versiones de filas del índice fueron eliminadas.
0 páginas de índice han sido eliminadas, 0 son reusables.
CPU: usuario: 0.00 s, sistema: 0.00 s, transcurrido: 0.00 s.
INFO:  el índice «arbolsuscripcion_contenidos» ahora contiene 35005088 versiones de filas en
```

Se puede apreciar que tras el vacuum, se han eliminado muchas tuplas 'basura'. Además se ha realizado de forma rápida gracias los árboles que previamente se habían creado.

Como último paso de mantenimiento, reindexaremos la base de datos. Examinando la consulta con explain, podemos apreciar que el coste es notablemente menor que desde un principio y se leen la mitad de 'rows' que antes.

```
reindex database netflix
```

Query Editor | Query History | Notifications | Explain

```
42  select producto_id_contenidos from series where series.temporadas < 6
43      or series.capitulos>15)))
44  union
45  select nombre,e_mail from cuenta_usuario where suscripcion_id_suscripcion in
46  (select suscripcion_id_suscripcion from pago where fecha
47  between '2020-05-01' and '2020-05-07' and suscripcion_id_suscripcion in
48  (select producto_id_contenidos from genero_contenidos group by producto_id_contenidos
49  having count (producto_id_contenidos)>3))
50
51  reindex database netflix
```

Data Output | Messages

	QUERY PLAN text
1	Unique (cost=5218532420.39..5218532420.39 rows=2000000 width=64)
2	-> Sort (cost=5218532420.39..5218532420.39 rows=2000000 width=64)
3	Sort Key: cuenta_usuario.nombre, cuenta_usuario.e_mail
4	-> Append (cost=1000.00..5218446480.39 rows=2000000 width=64)
5	-> Merge Semi Join (cost=1000.00..5218228924.39 rows=1000000 width=64)
6	Merge Cond: (cuenta_usuario.suscripcion_id_suscripcion = suscripcion_contenidos.suscripcion_id_suscripcion)
7	-> Index Scan using arbol_cuenta_usuario on cuenta_usuario (cost=0.00..32054.00 rows=2000000 width=96)
8	-> Gather Merge (cost=1000.00..5218196870.39 rows=91343 width=32)

Se puede apreciar que, finalmente, el coste es mucho menor que antes tras ejecutar todas las técnicas de mantenimiento.

Cuestión 14: 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.

Postgresql , realiza en sus consultas el camino más óptimo. Por defecto, utiliza la pk, como índice. Las consultas cuando manejan grandes cantidades de información, pueden ser muy pesadas y hasta tardar días. Por ello, se deben realizar técnicas para acelerar este proceso con índices u otros caminos que se mencionan en la cuestión 10. Postgresql, en su fichero .conf, contiene gran cantidad de opciones que se pueden activar para modificar el funcionamiento de este y /o acelerar sus procesos de consulta, aumentar memoria, etc. Para conocer el coste que va a tener una consulta que pueda ser 'lenta' , el comando explain nos ayuda y proporciona información sobre cada paso que realizará postgresql a la hora de realizar dicha consulta. Con esta información podremos actuar en función de lo que creamos que sea más óptimo, ya sea añadiendo árboles, o particiones, etc. Podemos comprobar a su vez, que es lo que se materializa y que es lo que se encauza, y ayudar a postgresql a que materialice menos y encauce más, conociendo su memoria cache y lo que almacena en ella y cuanto ocupa.

Bibliografía

PostgreSQL (13.x)

- Capítulo 14: Performance Tips.
- Capítulo 19: Server Configuration.
- Capítulo 15: Parallel Query.
- Capítulo 24: Routine Database Maintenance Tasks.
- Capítulo 50: Overview of PostgreSQL Internals.
- Capítulo 70: How the Planner Uses Statistics.