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

ALUMNO 1:

Nombre y Apellidos: Sergio Andrés Yaccelga Castillo

DNI: Y0668612Q

ALUMNO 2:

Nombre y Apellidos: _____

DNI: _____

Fecha: _____

Profesor Responsable: _____

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 6 de marzo, Semana 13 de marzo, Semana 20 de marzo y semana 27 de marzo.
Entrega de práctica:	Semana 11 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, 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 (15.1). 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 **Peliculas**. Básicamente la base de datos guarda personas y películas. Las personas pueden actuar en películas y además una de ellas puede dirigirla. Las películas tienen una serie de géneros y generan visualizaciones y críticas por parte de las personas dadas de alta. 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 2022 que hay que generar deben de ser los siguientes:

- Existen 20 géneros de contenidos diferentes, y el número de géneros que tiene asociada una película es un valor aleatorio que va entre 1 y 6.
- La tabla películas tiene 400.000 entradas. Las restricciones de algunos campos de la tabla son:
    - El año debe ser un valor aleatorio entre 1950 y el 2023.
    - La duración debe ser un valor aleatorio entre 90 y 240 minutos.
    - El idioma debe ser un valor aleatorio entre 1 y 30 donde dos de ellos deben ser EN (english) y ES (español).
    - La calificación será entre 0 y 10 estrellas.
- Cada película tiene un número de críticas que se eligen aleatoriamente entre 1 y 100. Todas las críticas son del año 2022, y el día y el mes se deben de generar aleatoriamente. La puntuación debe generarse aleatoriamente entre 0 y 10.
- Cada película tiene un director que se asigna aleatoriamente entre las personas que hay dadas de alta. El número de actores que tiene una película varía entre 10 y 30, debiendo elegirse aleatoriamente entre el número de personas que hay.
- Hay un total de 400.000 personas de 100 nacionalidades distintas, donde España debe ser una de ellas. Las nacionalidades se elegirán aleatoriamente entre todas las que hay. El año de nacimiento será un valor aleatorio desde 1950 hasta 2020, valores incluidos.
- Cada película tiene un número de visualizaciones comprendidas entre 1 y 100, todas realizadas en el año 2022, debiendo generarse el día y el mes de manera aleatoria.

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.

para realizar esto se tiene que ir al archivo de configuración postgres.conf que se encuentra en /var/lib/postgres/data, una vez encontrado, se editan las lineas de log_statement = 'ddl, mod', log_line_prefix = '%t DNI:[Y0668612Q] %h:%p ' y max_parallel_workers_per_gather = 0. Luego se reinicia el servidor y ya todo debería estar funcionando.

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?

- Número de filas en la tabla (reltuples) y tamaño de la tabla (relpages).
- Número de filas insertadas, actualizadas o eliminadas desde la última vez que se analizó la tabla (n_live_tup, n_dead_tup).
- Estadísticas de columnas individuales, como el número de valores distintos (n_distinct), la correlación entre el orden almacenado y el orden de columna (correlation), y los valores más comunes (most_common_vals) y sus frecuencias (most_common_freqs).
- Histogramas de valores de columna para estimar la distribución de datos en la columna.

Estas estadísticas se almacenan en las vistas del sistema pg_stat_all_tables, pg_stat_user_tables, pg_stat_xact_all_tables, pg_stat_xact_user_tables y pg_statistic.

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

- clientes(dni tipo numeric PRIMARY KEY, nombre tipo text, direccion tipo text, telefono tipo numeric)
- automoviles(bastidor tipo numeric PRIMARY KEY, proveedor tipo text, modelo tipo text, precio tipo numeric)
- compras(dni_cli tipo numeric que sea FOREIGN KEY del campo dni de la tabla clientes con restricciones de tipo RESTRICT en sus operaciones, bastidor_auto tipo numeric que sea FOREIGN KEY del campo bastidor de la tabla automóviles con restricciones de tipo RESTRICT en sus operaciones, fecha de tipo date, coste tipo numeric. La PRIMARY KEY debe ser bastidor_auto).

Se pide:

- Indicar el proceso seguido para generar esta base de datos.
- Cargar la información del fichero datos_clientes.csv, datos_autmoviles.csv y datos_compras.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.

primero se crea la base de datos autos: CREATE DATABASE autos;
Luego entramos a la base de datos: \c autos
Luego se crean las 3 tablas:

```
CREATE TABLE clientes (
  dni numeric PRIMARY KEY,
  nombre text,
  direccion text,
  telefono numeric
);
CREATE TABLE automoviles (
  bastidor numeric PRIMARY KEY,
  proveedor text,
  modelo text,
  precio numeric
);
CREATE TABLE compras (
  dni_cli numeric REFERENCES clientes(dni) ON DELETE RESTRICT ON UPDATE RESTRICT,
  bastidor_auto numeric REFERENCES automoviles(bastidor) ON DELETE RESTRICT ON UPDATE
RESTRICT,
  fecha date,
  coste numeric,
  PRIMARY KEY (bastidor_auto)
);
```

ahora se prueba los tiempos de carga:
tabla clientes: 00:00:03.772667
tabla automoviles: 00:00:50.935199
tabla compras: 00:10:50.87026

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?

no eran correctas y se tuvo que actualizar los registros con:

```
ANALYZE clientes;
ANALYZE automoviles;
ANALYZE compras;
```

una vez actualizados, ya se pueden ver las estádisticas con el comando:

```
SELECT * FROM pg_stats WHERE tablename = 'nombreTabla';
```

se almacena la correlación de las columnas y otros valores, ejemplo con la tabla automoviles:

```
schemaname                | public
tablename                 | automoviles
attname                   | bastidor
inherited                 | f
null_frac                       | 0
avg_width                 | 6
n_distinct                | -1
most_common_vals          |
most_common_freqs         |
```

Cuestión 4: Aplicar el comando EXPLAIN a una consulta que obtenga el número de automóviles del proveedor500 que no se han vendido. ¿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.

```
EXPLAIN
SELECT COUNT(*)
FROM automoviles a
LEFT JOIN compras c ON a.bastidor = c.bastidor_auto
WHERE a.proveedor = 'proveedor500' AND c.bastidor_auto IS NULL;
```

y el costo que sale es de cost=134495.08

Cuestión 5: Aplicar el comando EXPLAIN a una consulta que obtenga el modelo de los automóviles con un precio mayor de 50000 que se vendieron en el mes de mayo. ¿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.

al aplicar el comando de:

```
EXPLAIN
SELECT a.modelo
FROM automoviles a
JOIN compras c ON a.bastidor = c.bastidor_auto
WHERE a.precio > 50000 AND EXTRACT(MONTH FROM c.fecha) = 5;
```

vemos que el coste total es de: 112077.36

Cuestión 6: Aplicar el comando EXPLAIN a una consulta que obtenga el coste de las ventas de los automóviles vendidos entre el mes de febrero y agosto (incluidos), que sean del proveedor300 o del provedor500, y además el teléfono de los clientes esté incluido entre los números 7000000 y 9000000 (incluidos). ¿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.

al usar el comando:

```
EXPLAIN
SELECT c.coste
FROM automoviles a
JOIN compras c ON a.bastidor = c.bastidor_auto
JOIN clientes cl ON c.dni_cli = cl.dni
WHERE (a.proveedor = 'proveedor300' OR a.proveedor = 'proveedor500')
AND EXTRACT(MONTH FROM c.fecha) BETWEEN 2 AND 8
AND cl.telefono BETWEEN 7000000 AND 9000000;
```

vemos que nos da un coste total de: 149267.

Cuestión 7: Generar los datos solicitados al comienzo de la práctica para la base de datos **Películas** creando un programa para tal fin 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.

```
#!/usr/bin/python3

# hecho por serch :)
# github.com/serch24/
import csv
import random
from datetime import date, timedelta

# Archivos CSV para cada tabla
with open('personas.csv', 'w', newline='') as personas_file, \
      open('peliculas.csv', 'w', newline='') as peliculas_file, \
      open('generos.csv', 'w', newline='') as generos_file, \
      open('criticas.csv', 'w', newline='') as criticas_file, \
      open('actua.csv', 'w', newline='') as actua_file, \
      open('visualizaciones.csv', 'w', newline='') as visualizaciones_file:

   # Crear los escritores de CSV
   personas_writer = csv.writer(personas_file)
   peliculas_writer = csv.writer(peliculas_file)
   generos_writer = csv.writer(generos_file)
   criticas_writer = csv.writer(criticas_file)
   actua_writer = csv.writer(actua_file)
   visualizaciones_writer = csv.writer(visualizaciones_file)

   # Escribir encabezados de las tablas
   personas_writer.writerow(['nombre', 'nacionalidad', 'año_nacimiento', 'codigo_personas'])
   peliculas_writer.writerow(['año', 'titulo', 'duracion', 'idioma', 'calificacion', 'codigo_pelicula',
'codigo_director'])
   generos_writer.writerow(['genero', 'codigo_pelicula_peliculas'])
```

Cuestión 8: Realizar la carga masiva de los datos generados en la cuestión 7 en la base de datos **Películas**. 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)
generos	10 segundos
visualizaciones	1200 segundos
criticas	600 segundos
actua	180 segundos
personas	180 segundos

Como estoy trabajando en un entorno linux los logs no me aparecen en /var/lib/postgres/data, he descomentado la linea de log_directory, y cuando se ejecuta una sentencia en psql no aparece los logs

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 "porcentaje de las películas que tienen más de 2 géneros diferentes cuyo idioma no sea inglés (EN) o español (ES) con una duración de más de 100 minutos, teniendo por lo menos 10 críticas con una puntuación de al menos 5 puntos, y habiendo tenido visualizaciones entre el mes de mayo y agosto por personas que no sean españolas, no siendo estas películas dirigidas por directores españoles".

Obtener el plan de ejecución con el resultado del comando EXPLAIN. 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, tiempo de ejecución y resultado obtenido:

```
SELECT COUNT(*) * 100.0 / (SELECT COUNT(*) FROM peliculas.peliculas) AS porcentaje
FROM (
  SELECT p.codigo_pelicula, p.titulo
  FROM peliculas.peliculas AS p
  JOIN peliculas.generos AS g ON p.codigo_pelicula = g.codigo_pelicula_peliculas
  JOIN peliculas.criticas AS c ON p.codigo_pelicula = c.codigo_pelicula_peliculas
  JOIN peliculas.visualizaciones AS v ON p.codigo_pelicula = v.codigo_pelicula_peliculas
  JOIN peliculas.personas AS pers ON v.codigo_personas_personas = pers.codigo_personas
  JOIN peliculas.personas AS dir ON p.codigo_director = dir.codigo_personas
  WHERE p.idioma NOT IN ('EN', 'ES')
  AND p.duracion > 100
  AND c.puntuacion >= 5
  AND v.fecha BETWEEN '2022-05-01' AND '2022-08-31'
  AND pers.nacionalidad != 'ES'
  AND dir.nacionalidad != 'ES'
  GROUP BY p.codigo_pelicula, p.titulo
  HAVING COUNT(DISTINCT g.genero) > 2 AND COUNT(c.codigo_pelicula_peliculas) >= 10
```

Resultado comando EXPLAIN

```
Aggregate  (cost=502129.13..502129.15 rows=1 width=32) (actual time=886.352..886.360 rows=1
loops=1)
  Buffers: shared hit=931 read=3615
  InitPlan 1 (returns $0)
    ->  Aggregate  (cost=9544.00..9544.01 rows=1 width=8) (actual time=90.535..90.537 rows=1
loops=1)
          Buffers: shared hit=929 read=3615
          ->  Seq Scan on peliculas  (cost=0.00..8544.00 rows=400000 width=0) (actual
time=0.028..56.087 rows=400000 loops=1)
                Buffers: shared hit=929 read=3615
  ->  GroupAggregate  (cost=492285.90..492569.37 rows=1260 width=21) (actual
time=795.795..795.802 rows=0 loops=1)
        Group Key: p.codigo_pelicula
        Filter: ((count(DISTINCT g.genero) > 2) AND (count(c.codigo_pelicula_peliculas) >= 10))
        Buffers: shared hit=2
        ->  Sort  (cost=492285.90..492314.25 rows=11339 width=62) (actual time=795.794..795.800
rows=0 loops=1)
              Sort Key: p.codigo_pelicula
              Sort Method: quicksort  Memory: 25kB
              Buffers: shared hit=2
              ->  Nested Loop  (cost=29518.84..491522.27 rows=11339 width=62) (actual
time=795.788..795.793 rows=0 loops=1)
                    Buffers: shared hit=2
                    ->  Hash Join  (cost=29518.42..486470.65 rows=11356 width=68) (actual
time=795.787..795.791 rows=0 loops=1)
                          Hash Cond: (v.codigo_pelicula_peliculas = p.codigo_pelicula)
                          Buffers: shared hit=2
                          ->  Seq Scan on visualizaciones v  (cost=0.00..430952.70 rows=6902928 width=12)
(actual time=795.744..795.744 rows=1 loops=1)
```

Comentarios y explicaciones.

Al observar el plan de ejecución, se puede ver que la consulta utiliza varios pasos, incluidos Nested Loop, Hash Join y Seq Scan, para recuperar y filtrar los datos necesarios. Se observa que la consulta no devuelve ninguna fila que cumpla con todos los criterios especificados en las condiciones WHERE y HAVING. Por eso, el resultado del porcentaje es 0.

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. Comentar los resultados obtenidos y comparar con la cuestión anterior.

Consulta SQL creada, tiempo de ejecución y resultado obtenido:

```
CREATE INDEX idx_peliculas_idioma_duracion ON peliculas.peliculas (idioma, duracion);
CREATE INDEX idx_generos_codigo_pelicula_peliculas ON peliculas.generos
(codigo_pelicula_peliculas);
CREATE INDEX idx_criticas_codigo_pelicula_peliculas_puntuacion ON peliculas.criticas
(codigo_pelicula_peliculas, puntuacion);
CREATE INDEX idx_visualizaciones_codigo_pelicula_peliculas_fecha ON peliculas.visualizaciones
(codigo_pelicula_peliculas, fecha);
CREATE INDEX idx_personas_nacionalidad ON peliculas.personas (nacionalidad);
```

crearía estos índices en las columnas donde se hacen las condiciones de join.

Resultado comando EXPLAIN

```
 Aggregate  (cost=15840.19..15840.21 rows=1 width=32) (actual time=84.444..84.448 rows=1 loops=1)
   Buffers: shared hit=1023 read=3522
   InitPlan 1 (returns $0)
     ->  Aggregate  (cost=9544.00..9544.01 rows=1 width=8) (actual time=84.421..84.422 rows=1
loops=1)
           Buffers: shared hit=1022 read=3522
           ->  Seq Scan on peliculas  (cost=0.00..8544.00 rows=400000 width=0) (actual
time=0.028..53.464 rows=400000 loops=1)
                 Buffers: shared hit=1022 read=3522
   ->  GroupAggregate  (cost=2.41..6280.43 rows=1260 width=21) (actual time=0.011..0.014 rows=0
loops=1)
         Group Key: p.codigo_pelicula
         Filter: ((count(DISTINCT g.genero) > 2) AND (count(c.codigo_pelicula_peliculas) >= 10))
         Buffers: shared hit=1
         ->  Nested Loop  (cost=2.41..6025.30 rows=11339 width=62) (actual time=0.010..0.013 rows=0
loops=1)
               Join Filter: (p.codigo_pelicula = g.codigo_pelicula_peliculas)
```

Comentarios y explicaciones.

La creación de índices ha mejorado el rendimiento de la consulta significativamente. El tiempo de ejecución ha disminuido de 890.571 ms a 84.597 ms, lo que representa una mejora sustancial en la eficiencia de la consulta.

Al analizar el plan de ejecución, se puede ver que la consulta ahora utiliza índices en varias operaciones, lo que ha reducido el costo de las operaciones individuales y ha disminuido el tiempo de ejecución. Por ejemplo, se han utilizado Index Only Scan, Index Scan e Index Cond en lugar de Seq Scan para algunas operaciones, lo que indica que la consulta es capaz de acceder a los datos más rápidamente utilizando índices.

Cuestión 11: Usando PostgreSQL, borre el 30% de las películas 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.

```
CREATE TEMPORARY TABLE peliculas_a_eliminar AS
SELECT codigo_pelicula
FROM peliculas.peliculas
ORDER BY random()
LIMIT (SELECT COUNT(*) * 0.3 FROM peliculas.peliculas);

DELETE FROM peliculas.visualizaciones
WHERE codigo_pelicula_peliculas IN (SELECT codigo_pelicula FROM peliculas_a_eliminar);
Time: 5193606,125 ms (01:26:33,606)

DELETE FROM peliculas.criticas
WHERE codigo_pelicula_peliculas IN (SELECT codigo_pelicula FROM peliculas_a_eliminar);
Time: 2527,639 ms (00:02,528)

DELETE FROM peliculas.generos
WHERE codigo_pelicula_peliculas IN (SELECT codigo_pelicula FROM peliculas_a_eliminar);
Time: 339574,112 ms (05:39,574)

DELETE FROM peliculas.actua
WHERE codigo_pelicula_peliculas IN (SELECT codigo_pelicula FROM peliculas_a_eliminar);
Time: 4323263,923 ms (01:12:03,264)

DELETE FROM peliculas.peliculas
WHERE codigo_pelicula IN (SELECT codigo_pelicula FROM peliculas_a_eliminar);
Time: 1708010,595 ms (28:28,011)
```

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. Comparar con los resultados anteriores.

Consulta SQL creada, tiempo de ejecución y resultado obtenido:

```
SELECT COUNT(*) * 100.0 / (SELECT COUNT(*) FROM peliculas.peliculas) AS porcentaje
FROM (
  SELECT p.codigo_pelicula, p.titulo
  FROM peliculas.peliculas AS p
  JOIN peliculas.generos AS g ON p.codigo_pelicula = g.codigo_pelicula_peliculas
  JOIN peliculas.criticas AS c ON p.codigo_pelicula = c.codigo_pelicula_peliculas
  JOIN peliculas.visualizaciones AS v ON p.codigo_pelicula = v.codigo_pelicula_peliculas
  JOIN peliculas.personas AS pers ON v.codigo_personas_personas = pers.codigo_personas
  JOIN peliculas.personas AS dir ON p.codigo_director = dir.codigo_personas
  WHERE p.idioma NOT IN ('EN', 'ES')
  AND p.duracion > 100
  AND c.puntuacion >= 5
  AND v.fecha BETWEEN '2022-05-01' AND '2022-08-31'
  AND pers.nacionalidad != 'ES'
  AND dir.nacionalidad != 'ES'
  GROUP BY p.codigo_pelicula, p.titulo
  HAVING COUNT(DISTINCT g.genero) > 2 AND COUNT(c.codigo_pelicula_peliculas) >= 10
```

Resultado comando EXPLAIN

```
Aggregate  (cost=13879.67..13879.69 rows=1 width=32) (actual time=62.709..62.713 rows=1 loops=1)
  Buffers: shared hit=639
  InitPlan 1 (returns $0)
    ->  Aggregate  (cost=7460.42..7460.43 rows=1 width=8) (actual time=62.645..62.646 rows=1
loops=1)
          Buffers: shared hit=638
          ->  Index Only Scan using idx_peliculas_idioma_duracion on peliculas  (cost=0.42..6760.42
rows=280000 width=0) (actual time=0.081..37.209 rows=280000 loops=1)
                Heap Fetches: 0
                Buffers: shared hit=638
  ->  GroupAggregate  (cost=2.40..6403.78 rows=1237 width=21) (actual time=0.049..0.052 rows=0
loops=1)
        Group Key: p.codigo_pelicula
        Filter: ((count(DISTINCT g.genero) > 2) AND (count(c.codigo_pelicula_peliculas) >= 10))
        Buffers: shared hit=1
        ->  Nested Loop  (cost=2.40..6153.20 rows=11137 width=62) (actual time=0.048..0.050 rows=0
loops=1)
              Join Filter: (p.codigo_pelicula = g.codigo_pelicula_peliculas)
              Buffers: shared hit=1
              ->  Nested Loop  (cost=1.98..4387.05 rows=3181 width=59) (actual time=0.047..0.049 rows=0
loops=1)
                    Buffers: shared hit=1
                    ->  Nested Loop  (cost=1.56..2966.41 rows=3186 width=65) (actual time=0.047..0.048
rows=0 loops=1)
                          Buffers: shared hit=1
```

Comentarios y explicaciones.

Se puede ver que el tiempo de ejecución ha disminuido de 84.444 ms a 62.709 ms. Esto indica que la consulta se ha vuelto más rápida después de eliminar el 30% de las películas y sus datos relacionados. El plan de ejecución se puede ver que las estimaciones de costo y filas también han cambiado en varias partes del plan. Por ejemplo, el costo total estimado ha disminuido, lo que sugiere una mejora en el rendimiento de la consulta.

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?

Se podría actualizar las estadisticas con analyze o vacum analyze, ya que estas estadisticas actualizadas permiten al optimizador de consultas tomar decisiones más precisas y generar planes de ejecución eficientes.
También se podría particionar tablas cuando tienen muchas consultas, por ejemplo la de visualizaciones por fecha. También se podría hacer un monitoreo constante a la base de datos para así estar más informado sobre lo que puede dar más problema, usando pg_stat_statements o EXPLAIN ANALYZE.

<u>Cuestión 14:</u> 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. Compare los resultados del plan de ejecución con los de los apartados anteriores. Coméntelos.

Consulta SQL creada, tiempo de ejecución y resultado obtenido:

```
VACUUM ANALYZE peliculas.peliculas;
VACUUM ANALYZE peliculas.visualizaciones;
VACUUM ANALYZE peliculas.generos;
VACUUM ANALYZE peliculas.criticas;
VACUUM ANALYZE peliculas.personas;
```

se ejecuta estas 5 sentencias en psql.

Resultado comando EXPLAIN

```
                                   QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Aggregate  (cost=7495.36..7495.38 rows=1 width=32) (actual time=50.249..50.253 rows=1 loops=1)
   Buffers: shared hit=641
   InitPlan 1 (returns $0)
     ->  Aggregate  (cost=7460.42..7460.43 rows=1 width=8) (actual time=50.071..50.071 rows=1 loops=1)
           Buffers: shared hit=638
           ->  Index Only Scan using idx_peliculas_idioma_duracion on peliculas  (cost=0.42..6760.42 rows=280000 width=0) (actual time=0.109..30.399 rows=280000 loops=1)
                 Heap Fetches: 0
                 Buffers: shared hit=638
   ->  GroupAggregate  (cost=33.60..34.85 rows=6 width=21) (actual time=0.160..0.163 rows=0 loops=1)
         Group Key: p.codigo_pelicula
         Filter: ((count(DISTINCT g.genero) > 2) AND (count(c.codigo_pelicula_peliculas) >= 10))
         Buffers: shared hit=3
         ->  Sort  (cost=33.60..33.73 rows=50 width=62) (actual time=0.159..0.161 rows=0 loops=1)
               Sort Key: p.codigo_pelicula
               Sort Method: quicksort  Memory: 25kB
               Buffers: shared hit=3
               ->  Nested Loop  (cost=2.25..32.19 rows=50 width=62) (actual time=0.022..0.025 rows=0 loops=1)
```

Comentarios y explicaciones.

El resultado obtenido muestra que el tiempo de ejecución de la consulta ha mejorado significativamente, pasando de 62.709 ms a 50.253 ms.

Al analizar el plan de ejecución, se pueden observar las siguientes mejoras:

El costo estimado del plan de ejecución es ahora menor.
El uso de memoria y el tiempo de ejecución de varias operaciones también se han reducido.

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.

PostgreSQL sigue un proceso riguroso de procesamiento y optimización de consultas para garantizar un rendimiento óptimo y resultados precisos. Las etapas clave incluyen el análisis y parseo de la consulta, la planificación y optimización basada en costes, la generación del plan de ejecución y la ejecución de la consulta. A lo largo del proceso, PostgreSQL utiliza estadísticas, índices y optimizaciones adicionales para mejorar aún más el rendimiento y la eficiencia de las consultas.

Bibliografía

PostgreSQL (15.1)

- Capítulo 14: Performance Tips.
- Capítulo 20: Server Configuration.
- Capítulo 15: Parallel Query.
- Capítulo 25: Routine Database Maintenance Tasks.
- Capítulo 52: Overview of PostgreSQL Internals.

- Capítulo 75: How the Planner Uses Statistics.
- https://pgtune.leopard.in.ua/
- https://explain.dalibo.com/