Informe N°2 Base de datos avanzadas

Dante Hortuvia
Diego Lara

1 Introducción

En este informe, nos adentraremos en las labores encomendadas relacionadas con la manipulación y procesamiento de dos conjuntos de datos: "PERSONAS1" y "PERSONAS2". Ambos conjuntos albergan aproximadamente 49.875.500 registros, detallando información crucial sobre individuos, como RUT, nombres, direcciones y edades.

El objetivo principal de estas actividades es ejecutar una serie de operaciones en estas tablas mediante el uso de un archivo titulado "50mil", que contiene 50.000 RUT. Estas operaciones abarcan desde la eliminación de índices, hasta consultas sobre nombres y direcciones, pasando por la supresión de registros y la modificación de edades según criterios específicos. Además, será necesario medir los tiempos de ejecución de estos procesos y obtener los planes de ejecución correspondientes para efectuar un análisis comparativo del rendimiento entre ambas tablas. Este análisis permitirá detectar las discrepancias en los tiempos de procesamiento y comprenderlas a la luz de los planes de ejecución obtenidos.

2 Desarollo de la tarea

Se elimina el índice de personas1 con el nombre idx rut

```
tarea_1=# drop index idx_rut;
DROP INDEX
```

Figura 1: Eliminación del índice.

Se utiliza una consulta (query) para verificar si existe una llave primaria en la tabla "personas2".

```
tarea_1=# SELECT constraint_name FROM information_schema.table_constraints WHERE table_name = 'personas2' AND constrain
t_type = 'PRIMARY KEY';
 personas2_pkey

Time: 31.225 ms
```

Figura 2: comprobación de la PK.

Se verifica la cantidad de filas en ambas tablas utilizando la función COUNT.

```
tarea_1=# select count(*) from personas1;
 49875490

Time: 66265.256 ms (01:06.265)
tarea_1=# select count(*) from personas2;
 49875490

Time: 77929.520 ms (01:17.930)
```

Figura 3: Cantidad de filas en ambas tablas.

Para obtener el nombre y la dirección del primer RUT, se accede al archivo y se extrae el RUT siguiente, que es "88901377". Luego, utilizando la función COPY de PostgreSQL, se crea un archivo CSV para almacenar

los datos obtenidos.

```
tarea_1=# COPY (select nombre, direccion  from personas1 where rut = 88901377) TO '/home/dante/Desktop/Tarea_2/Datos/cs
v1' DELIMITER ',' CSV HEADER;
COPY 1
Time: 44070.281 ms (00:44.070)
```

Figura 4: Creación del CVS de el rut "88901377" de personas1 .

El comando EXPLAIN en PostgreSQL proporciona un plan de ejecución detallado para una consulta SQL sin ejecutarla realmente. Al analizar la consulta, muestra qué índices se utilizarán, el orden de unión de las tablas y los métodos de filtrado que se aplicarán. Es una herramienta para entender cómo PostgreSQL planea ejecutar una consulta.

```
                                QUERY PLAN
---------------------------------------------------------------------------
 Gather  (cost=1000.00..2694460.50 rows=1 width=152)
   Workers Planned: 2
   ->  Parallel Seq Scan on personas1  (cost=0.00..2693460.40 rows=1 width=152)
         Filter: (rut = 88901377)
 JIT:
   Functions: 4
   Options: Inlining true, Optimization true, Expressions true, Deforming true
(7 rows)
```

Figura 5: EXPLAIN de la primera consulta (query).

Para la segunda parte, donde se necesita obtener el nombre y la dirección de los 50 mil registros, se crea una tabla adicional llamada "datos" con un único campo "rut" (de tipo Bigint este se cambia a NUMERIC(10,0)) y se ingresan los 50 mil registros. Esta tabla se utiliza para comparar los RUT con la tabla original.

```
tarea_1=# create table datos(
rut bigint);
CREATE TABLE
Time: 0.774 ms
tarea_1=# \COPY datos FROM '/home/dante/Desktop/Tarea_2/Datos/50mil'
COPY 50000
Time: 12.766 ms
```

Figura 6: Creación de una tabla auxiliar.

```
tarea_1=# ALTER TABLE personas1
tarea_1-# ALTER COLUMN rut TYPE NUMERIC(10,0);
ALTER TABLE
```

Figura 7: Cambio de Big int a Numeric.

Se utiliza la función COPY para crear un archivo CSV que contiene el nombre y la dirección de los datos solicitados de personas1. El filtro aplicado es que los RUT sean iguales a los de la tabla auxiliar creada llamada "datos".

```
tarea_1=# COPY (select nombre,direccion from personas1 where personas1.rut in (select rut from datos) ) TO '/home/dante
/Desktop/Tarea_2/Datos/csv2' DELIMITER ',' CSV HEADER;
COPY 50000
Time: 59240.956 ms (00:59.241)
tarea_1=#
```

Figura 8: Creación del CSV para los 50 mil datos de personas1.

Utilizando el comando EXPLAIN, del que hemos hablado anteriormente, se obtiene el plan de ejecución.

```
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
 Gather  (cost=2347.00..2703638.15 rows=50000 width=152)
   Workers Planned: 2
   ->  Hash Semi Join  (cost=1347.00..2697638.15 rows=20833 width=152)
         Hash Cond: (personas1.rut = datos.rut)
         ->  Parallel Seq Scan on personas1  (cost=0.00..2641480.72 rows=20791872 width=160)
         ->  Hash  (cost=722.00..722.00 rows=50000 width=8)
               ->  Seq Scan on datos  (cost=0.00..722.00 rows=50000 width=8)
 JIT:
   Functions: 9
   Options: Inlining true, Optimization true, Expressions true, Deforming true
(10 rows)
Show Apps
```

Figura 9: Explain de la segunda consulta .

Para eliminar los RUT pares, se emplea una consulta que filtra los RUT a través de su módulo: si es igual a cero, se eliminan, de lo contrario, se mantienen intactos. Esta acción resulta en la eliminación de 24,967 registros de la tabla. Se verifica esta acción mediante un script de Python para obtener el recuento total de datos pares en la tabla previamente mencionada.

```
Time: 68.599 ms
tarea_1=*# delete from personas1 where personas1.rut in (select rut from datos where rut%2 = 0);
DELETE 24967
Time: 83793.690 ms (01:23.794)
tarea_1=*#
```

Figura 10: Datos eliminados de personas1.

```python
    # Variable para contar la cantidad de números divisibles por 2
cantidad_divisibles_por_dos = 0
    # Abrir el archivo CSV
with open("/home/dante/Desktop/Tarea_2/Datos/50mil", 'r') as file:
        # Crear un lector CSV
    for i, linea in enumerate(file, 1):
        campos = linea.strip().split("|")[0]
        if int(campos) % 2 == 0 :
            cantidad_divisibles_por_dos +=1


print(f"La cantidad de números divisibles por 2 en el archivo {cantidad_divisibles_por_dos}")
```

Figura 11: Script de python.

```
La cantidad de números divisibles por 2 en el archivo 24967
dante@dante-1-2:~/Desktop/Tarea_2/Codigos$
```

Figura 12: Cantidad de pares.

Al emplear el comando EXPLAIN, se obtiene el plan de ejecución de la consulta, revelando los siguientes resultados: filtra por módulo, condición, unión de hash donde y escaneo de personas 1 los cuales resultan en costo total de la consulta.

```
                                QUERY PLAN
---------------------------------------------------------------------------
Delete on personas1  (cost=1023.09..2124609.53 rows=0 width=0)
  ->  Hash Join  (cost=1023.09..2124609.53 rows=24937744 width=12)
        Hash Cond: (personas1.rut = datos.rut)
        ->  Seq Scan on personas1  (cost=0.00..1715230.88 rows=49875488 width=22)
        ->  Hash  (cost=1021.30..1021.30 rows=143 width=22)
              ->  HashAggregate  (cost=1019.88..1021.30 rows=143 width=22)
                    Group Key: datos.rut
                    ->  Seq Scan on datos  (cost=0.00..1019.25 rows=250 width=22)
                          Filter: ((rut % '2'::numeric) = '0'::numeric)
JIT:
  Functions: 14
  Options: Inlining true, Optimization true, Expressions true, Deforming true
12 rows)
```

Figura 13: Explain de la consulta

Para los casos en que el último dígito sea impar, se empleó una consulta que, en primer lugar, saca el módulo del RUT y si es distinto a 0 se confirma su naturaleza impar y se suma 10 a la edad asociada al RUT. Si esta suma resultara en una edad superior a 99 años, se ajusta a cero, al emplear la consulta cambia 25033 datos (la suma de las dos consultas da 50 mil datos).

```
tarea_1=*# UPDATE PERSONAS1
SET edad = CASE
    WHEN Rut % 2 <> 0 THEN
        CASE
            WHEN edad + 10 <= 99 THEN edad + 10
            ELSE 0
        END
    ELSE edad
END
WHERE PERSONAS1.Rut IN (SELECT rut FROM datos);
UPDATE 25033
Time: 76001.046 ms (01:16.001)
```

Figura 14: Consulta Update.

Al solicitar el tiempo de ejecución del la función update , donde vuelve a indicar los costos que se basan en Hash, Scan

```
                                QUERY PLAN
---------------------------------------------------------------------------
 Update on personas1  (cost=1347.00..3066333.96 rows=0 width=0)
   ->  Hash Semi Join  (cost=1347.00..3066333.96 rows=50000 width=14)
         Hash Cond: (personas1.rut = datos.rut)
         ->  Seq Scan on personas1  (cost=0.00..2932566.92 rows=49900492 width=16)
         ->  Hash  (cost=722.00..722.00 rows=50000 width=14)
               ->  Seq Scan on datos  (cost=0.00..722.00 rows=50000 width=14)
 JIT:
   Functions: 10
   Options: Inlining true, Optimization true, Expressions true, Deforming true
(9 rows)
```

Figura 15: Explain de la consulta Update.

Para personas2, se sigue un procedimiento similar al descrito anteriormente. para obtener el nombre y la dirección asociados al RUT solicitado, se emplea la función copy para exportar los datos a un archivo CSV.

```
tarea_1=# COPY (select nombre, direccion from personas2 where rut =88901377) TO '/home/dante/Desktop/Tarea_2/Datos/csv1
_personas2' DELIMITER ',' CSV HEADER;
COPY 1
Time: 5.218 ms
```

Figura 16: Consulta del rut solicitado.

Se solicita el plan de ejecución de la consulta realizada, donde verificamos la primera diferencia frente a personas1, esta directamenta hace un Index Scan en la llave primaria de la tabla.

```
                                  QUERY PLAN
------------------------------------------------------------------------------
 Index Scan using personas2_pkey on personas2  (cost=0.56..8.58 rows=1 width=152)
   Index Cond: (rut = 88901377)
(2 rows)
```

Figura 17: Explain de la consutla.

Para obtener todos los datos solicitados, se emplea la función copy en conjunto con la tabla previamente creada denominada "datos", exportando así los datos a un archivo CSV.

```
tarea_1=# COPY (select nombre, direccion from personas2 where personas2.rut in (select rut from datos)) TO '/home/dante
/Desktop/Tarea_2/Datos/csv2_personas2' DELIMITER ',' CSV HEADER;
COPY 50000
Time: 2320.763 ms (00:02.321)
```

Figura 18: Consulta para todos los datos.

Al obtener el plan de ejecución nos volvemos a percatar que utiliza el índice de la llave primaria y que utiliza menos funciones que en personas1.

```
                                    QUERY PLAN
-------------------------------------------------------------------------------------
 Nested Loop  (cost=847.57..400425.55 rows=50000 width=152)
   ->  HashAggregate  (cost=847.00..1347.00 rows=50000 width=8)
         Group Key: datos.rut
         ->  Seq Scan on datos  (cost=0.00..722.00 rows=50000 width=8)
   ->  Index Scan using personas2_pkey on personas2  (cost=0.56..7.99 rows=1 width=160)
         Index Cond: (rut = datos.rut)
 JIT:
   Functions: 7
   Options: Inlining false, Optimization false, Expressions true, Deforming true
(9 rows)

tarea_1=#
```

Figura 19: Explain de la consutla.

En el caso de la eliminación de los RUT pares en personas2, se utiliza inicialmente una consulta que calcula el módulo del RUT. Si este módulo es igual a cero, se procede con la eliminación, resultando en la eliminación de 24,967 registros de la tabla. Para verificar la correcta aplicación de la consulta, se emplea el mismo método de comprobación utilizado en personas1.

```
tarea_1=# begin;
BEGIN
Time: 0.084 ms
tarea_1=*# delete from personas2 where personas2.rut in (select rut from datos where rut%2= 0);
DELETE 24967
Time: 27095.904 ms (00:27.096)
```

Figura 20: Consulta para eliminar.

se obtiene el plan de ejecución y obtenemos resultados muy pareciedos a personas1, pero sigue usando la llave primaria de personas2 lo que podría índicar porque se demora menos.

```
                                    QUERY PLAN
-------------------------------------------------------------------------------------------
 Delete on personas2  (cost=1020.44..2248.60 rows=0 width=0)
   ->  Nested Loop  (cost=1020.44..2248.60 rows=250 width=12)
         ->  HashAggregate  (cost=1019.88..1021.30 rows=143 width=22)
               Group Key: datos.rut
               ->  Seq Scan on datos  (cost=0.00..1019.25 rows=250 width=22)
                     Filter: ((rut % '2'::numeric) = '0'::numeric)
         ->  Index Scan using personas2_pkey on personas2  (cost=0.56..8.58 rows=1 width=22)
               Index Cond: (rut = datos.rut)
(8 rows)

tarea_1=#
```

Figura 21: Explain de la consulta.

Para abordar el segundo caso, también se empleó una consulta que utiliza el módulo del RUT. Sin embargo, en este caso, si el módulo es distinto de cero, se considera como un número impar. A la edad asociada al RUT, se le suma 10, con la precaución de no exceder los 99 años; en caso de superarse, se establece la edad como cero. Mediante este método, se alteraron 25,033 datos.

```
tarea_1=*# UPDATE PERSONAS2
SET edad = CASE
    WHEN Rut % 2 <> 0 THEN
        CASE
            WHEN edad + 10 <= 99 THEN edad + 10
            ELSE 0
        END
    ELSE edad
END
WHERE PERSONAS2.Rut IN (SELECT rut FROM datos);
UPDATE 25033
Time: 23483.116 ms (00:23.483)
tarea_1=*#
```

Figura 22: Consulta cambiar los datos.

Se solicita el plan de ejecución y se nota nuevamente que la principal diferencia entre personas1 y personas2 radica en el método de búsqueda de datos en sus respectivas tablas.

```
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
Update on personas2  (cost=847.57..401300.55 rows=0 width=0)
  ->  Nested Loop  (cost=847.57..401300.55 rows=50000 width=14)
        ->  HashAggregate  (cost=847.00..1347.00 rows=50000 width=14)
              Group Key: datos.rut
              ->  Seq Scan on datos  (cost=0.00..722.00 rows=50000 width=14)
        ->  Index Scan using personas2_pkey on personas2  (cost=0.56..7.99 rows=1 width=16)
              Index Cond: (rut = datos.rut)
JIT:
  Functions: 9
  Options: Inlining false, Optimization false, Expressions true, Deforming true
10 rows)
```

Figura 23: Explain de la consulta.

Después de un minucioso análisis de los datos y los planes de ejecución, es evidente que personas2 supera a personas1 en términos de velocidad. Esto se atribuye principalmente a la implementación de un índice en la tabla personas2. Este índice optimiza la búsqueda de datos, lo que se traduce en tiempos de consulta considerablemente más rápidos. La diferencia se hace evidente al observar los planes de ejecución, donde la búsqueda en personas2 aprovecha eficientemente el índice para acceder de manera rápida a los datos deseados, a diferencia de personas1, donde la búsqueda se realiza de manera secuencial, lo que conlleva a una mayor latencia. Por ende, la inclusión del índice en personas2 marca una mejora significativa en el rendimiento de las consultas en comparación con personas1, destacando la importancia de la optimización de bases de datos para obtener un mejor rendimiento en las operaciones de consulta.

Referencias

"SQL Cookbook" por Anthony Molinaro.

"PostgreSQL: Up and Running: A Practical Introduction to the Advanced Open Source Database" por Regina O. Obe y Leo S. Hsu.