Creación y comparación de bases de datos indexadas y no indexadas respecto al tiempo de consulta

Trujillo López Angélica Sarahy

ABSTRACT

En este trabajo se creo una base de datos desde cero en lenguaje SQL bajo el sistema gestor de bases de datos PostgreSQL. Se realizaron distintos comandos necesarios para siete requerimientos. Se creo una segunda base de datos con implementación de índices y se comparo el funcionamiento de estas dos respecto a las mismas acciones con los comandos *insert, select, delete* y *update*.

1. INTRODUCCION

Una base de datos es un conjunto estructurado de datos que representa entidades y sus interrelaciones. El software de gestión de ficheros era demasiado elemental para dar satisfacción a todas estas necesidades. La utilización de estos conjuntos de ficheros por parte de los programas de aplicación era excesivamente compleja, de modo que, fue saliendo al mercado software más sofisticado: los Data Base Management Systems, sistemas de gestión de bases de datos (Camps et al., 2005).

SQL es un sublenguaje de datos para accesar a las bases de datos manejadas por un sistema gestor de bases de datos relacionales (SGDB).

PostgreSQL es un sistema de administración de bases de datos relacionales de objetos. Un desarrollo más moderno es la base de datos orientada a objetos. Cada tabla es una colección de filas con nombre. Cada fila de una tabla determinada tiene el mismo conjunto de columnas con nombre y cada columna es de un tipo de datos específico. Mientras que las columnas tienen un orden fijo en cada fila, es importante recordar que SQL no garantiza el orden de las filas dentro de la tabla de ninguna manera (Documentación PostgreSQL).

Para una búsqueda sin una preparación previa, el sistema tendría que escanear toda la tabla o tablas que se indican, fila por fila, para encontrar todas las entradas coincidentes. Pero si el sistema

ha recibido instrucciones de mantener un índice, puede utilizar un método más eficaz para localizar filas coincidentes. Una vez que se crea un índice, no se requiere ninguna otra intervención. Los índices también pueden beneficiarse UPDATEy los DELETEcomandos con condiciones de búsqueda. Además, los índices se pueden utilizar en búsquedas de combinaciones. Por lo tanto, un índice definido en una columna que forma parte de una condición de combinación también puede acelerar significativamente las consultas con combinaciones (Documentación PostgreSQL).

2. MATERIALES Y METODOS

Se construyó una base de datos relacional desde cero con base en el lenguaje SQL. Esta base de datos fue creada a partir del conjunto de datos denominado “The Movies Dataset” utilizando el gestor de bases de datos PostgreSQL.

Se crearon dos tipos de bases de datos una basada solo en las tablas que contienen la información y otra basada en las tablas e índices. Las bases se crearon de la siguiente forma:

2.1. Base de datos sin índice

CREATE DATABASE moviedatasetfbd;

\c moviedatasetfbd;

CREATE TABLE pelis (id_peli INT PRIMARY KEY, name_peli CHAR(200), prom REAL, countv INT);

CREATE TABLE generos (id_gen INT PRIMARY KEY, genero CHAR(40));

CREATE TABLE peli_genero (id_new BIGSERIAL PRIMARY KEY, id_gen INT, id_peli INT, FOREIGN KEY (id_gen) REFERENCES generos (id_gen), FOREIGN KEY (id_peli) REFERENCES pelis (id_peli));

CREATE TABLE individuos (id_ind INT PRIMARY KEY, name_ind CHAR(60));

CREATE TABLE produccion (id_cred CHAR(40) PRIMARY KEY, id_ind INT, puesto_pr CHAR(60), id_peli INT, FOREIGN KEY (id_ind) REFERENCES individuos (id_ind), FOREIGN KEY (id_peli) REFERENCES pelis (id_peli));

CREATE TABLE acteres (id_cred CHAR(40) PRIMARY KEY, id_ind INT, id_peli INT, FOREIGN KEY (id_ind) REFERENCES individuos (id_ind), FOREIGN KEY (id_peli) REFERENCES pelis (id_peli));

2.2. Base de datos con índice

Para este caso necesitamos exactamente lo mismo que la base de datos anterior, pero agregaremos los índices al final de la tabla como se muestra a continuación.

CREATE INDEX ixpelisnp ON pelis(name_peli);

CREATE INDEX ixpelispr ON pelis(prom);

CREATE INDEX ixpelisct ON pelis(countv);

CREATE INDEX ixgenerosgen ON generos(genero);

CREATE INDEX ixpeligenig ON peli_genero(id_gen);

CREATE INDEX ixpeligenip ON peli_genero(id_peli);

```
CREATE INDEX ixindni ON individuos(name_ind);
CREATE INDEX ixprodii ON produccion(id_ind);
CREATE INDEX ixprodpp ON produccion(puesto_pr);
CREATE INDEX ixprodip ON produccion(id_peli);
CREATE INDEX ixactii ON acteres(id_ind);
CREATE INDEX ixactip ON acteres(id_peli);
```

Los datos requerían ser procesados antes de utilizarse para crear la base de datos debido a que estos no solo no eran atómicos, sino que también tenían errores que causaban problemas al momento de intentar ser cargados en nuestra base de datos.

Una vez creáramos la base de datos y los cargáramos debemos realizar 7 consultas. Se utilizaron los archivos “credits.csv” y “movies_metadata.csv” ya que eran las únicas que se necesitaban para satisfacer estas consultas. De los cuales solo nos quedamos las columnas y tablas que se muestran en la estructura de la base de datos documentado anteriormente. Además de esto, también se realizaron correcciones manuales sobre los archivos generados debido a que existían cosas que nos estorbaban tal como comas, apostrofes etc.

Para la parte de comparación compararemos los tiempos de los distintos comandos que son INSERT, DELETE y UPDATE.

Se inserto de dos maneras como fue requerido, las cuales son el *bulk load* y el *insert* simple. Se presentan a continuación.

2.3. Bulk load

```
\COPY pelis(id_peli,name_peli,prom,countv) FROM 'pelis.csv' DELIMITER ',' CSV HEADER;
\COPY generos(id_gen,genero) FROM 'generos.csv' DELIMITER ',' CSV HEADER;
\COPY peli_genero(id_gen ,id_peli) FROM 'peli_genero.csv' DELIMITER ',' CSV HEADER;
\COPY individuos(id_ind,name_ind) FROM 'individuos.csv' DELIMITER ',' CSV HEADER;
\COPY acteres(id_cred ,id_ind,id_peli) FROM 'acteres.csv' DELIMITER ',' CSV HEADER;
\COPY produccion(id_cred,id_ind,puesto_pr,id_peli) FROM 'produccion.csv' DELIMITER ',' CSV HEADER;
```

2.4. Insert

Para los *insert* además de utilizar el comando correcto también después de este comando debemos poner exactamente la información que se quiere ingresar, estos datos no serán mostrados por razones de espacio.

```
INSERT INTO pelis (id_peli,name_peli,prom,countv) VALUES
INSERT INTO peli_genero (id_gen,id_peli) VALUES
INSERT INTO individuos (id_ind,name_ind) VALUES
INSERT INTO acteres (id_cred,id_ind,id_peli) VALUES
INSERT INTO produccion (id_cred,id_ind,puesto_pr,id_peli) VALUES
```

INSERT INTO generos (id_gen,genero) VALUES

Una vez concluimos con la inserción de loa datos esta parte del proceso fue repetida en 10 ocasiones en modo *cold start* del sistema eliminando la memoria cache para las comparaciones que se quería hacer.

Como última parte del proceso pasamos a ejecutar las 7 consultas creadas para los requerimientos, las cuales quedaron de la siguiente manera.

1. (peliculas por genero) SELECT id_gen, COUNT(DISTINCT id_peli) as np FROM peli_genero GROUP BY id_gen;

   (titulo,genero ordenado por genero) SELECT name_peli,genero FROM pelis,generos GROUP BY genero,name_peli;

2. SELECT name_ind FROM individuos WHERE id_ind=(SELECT id_ind FROM (SELECT id_ind, COUNT(DISTINCT id_cred) as np FROM acteres GROUP BY id_ind ORDER BY np DESC LIMIT 1) T);
3. SELECT name_peli FROM pelis WHERE id_peli=(SELECT id_peli FROM (SELECT id_peli, COUNT(DISTINCT id_ind) as np FROM acteres GROUP BY id_peli ORDER BY np DESC LIMIT 1) T);
4. SELECT name_ind FROM individuos WHERE id_ind=(SELECT id_ind FROM (SELECT id_ind, COUNT(DISTINCT id_peli) as np FROM produccion GROUP BY id_ind ORDER BY np DESC LIMIT 1) T);
5. SELECT DISTINCT name_peli FROM pelis,produccion,acteres WHERE pelis.id_peli=produccion.id_peli AND produccion.puesto_pr='Director' AND produccion.id_ind=acteres.id_ind;
6. SELECT name_peli FROM pelis WHERE countv=(SELECT MAX(countv) FROM pelis);
7. SELECT name_peli FROM pelis WHERE prom=(SELECT MAX(prom) FROM pelis);

Para la parte del *delete* se utilizó en siguiente comando, el cual eliminaba toda la tabla.

DELETE FROM [Nombre Tabla]

En el caso de *update* se utilizó el siguiente, el cual como dice reemplaza las veces que se repite un valor con uno nuevo.

UPDATE [tabla] SET [columna=nuevo valor] WHERE [columna=viejo valor]

3. RESULTADOS

Inserción bulk load sin índice.

Repetición	Tiempo(ms)
1	34611.415
2	33704.126
3	33751.408
4	33096.058
5	32660.85

6	32323.131
7	32362.759
8	34545.636
9	33299.185
10	33618.193
Promedio	33397.277

Inserción bulk load con índice.

Repetición	Tiempo(ms)
1	48720.758
2	50522.209
3	49356.475
4	50202.876
5	52147.338
6	50444.948
7	52506.056
8	51960.426
9	50305.272
10	51428.551
Promedio	50759.4909

Inserción mediante *Inser*t sin índice.

Repetición	Tiempo(ms)
1	41587.768
2	41335.602
3	41564.766
4	42579.431
5	41186.881
6	40866.418
7	40858.751
8	41333.035
9	39974.738
10	40830.906
Promedio	41287.5158

Inserción mediante *Insert* con índice.

10	Tiempo(ms)
1	41587.768
2	41335.602
3	41564.766
4	42579.431
5	41186.881
6	40866.418
7	41308.751
8	41333.035
9	39974.738
10	40830.906
Promedio	41256.8296

Ejecución de los *selects* sin índice.

1.

Repetición	Tiempo(ms)
1	12107.815
2	11873.164
3	11861.931
4	12324.192
5	11883.087
6	12482.059
7	12335.257
8	11702.216
9	12150.153
10	12015.377
Promedio	12073.5251

2.

Repetición	Tiempo(ms)
1	991.542
2	892.312
3	855.267
4	857.897
5	878.655
6	838.839
7	860.733
8	829.708
9	845.365
10	884.561
Promedio	873.4879

3.

Repetición	Tiempo(ms)
1	343.632
2	333.917
3	349.866
4	349.34
5	339.916
6	339.512
7	337.411
8	348.897
9	353.878
10	370.411
Promedio	346.678

4.

Repetición	Tiempo(ms)
1	520.375
2	483.042
3	450.055
4	477.885
5	453.747
6	440.942
7	439.08
8	446.647
9	479.934
10	453.145
Promedio	464.4852

5.

Repetición	Tiempo(ms)
1	1074.659
2	1053.985
3	1110.727
4	1128.156
5	1078.131
6	1082.907
7	1070.605
8	1073.819
9	1058.046
10	1045.488
Promedio	1077.6523

6.

Repetición	Tiempo(ms)
1	11.822
2	16.113
3	13.717
4	12.032
5	13.014
6	14.347
7	15.836
8	15.092
9	12.687
10	12.016
Promedio	13.6676

7.

Repetición	Tiempo(ms)
1	15.219
2	14.759
3	15.256
4	14.239
5	22.959
6	15.373
7	18.695
8	17.248
9	16.694
10	19.257
Promedio	16.9699

Suma del tiempo promedio de todo el proceso.

Tiempo total promedio	**14866.466**

Ejecución de los *selects* con índice.

1.

Repetición	Tiempo(ms)
1	12555.268
2	12288.296
3	12722.109
4	12657.956
5	12829.076
6	12816.023
7	13744.685
8	12877.521
9	13413.885
10	12991.845
Promedio	12889.6664

2.

Repetición	Tiempo(ms)
1	944.52
2	988.542
3	896.511
4	844.704
5	890.042
6	888.648
7	1018.662
8	916.463
9	945.145
10	888.984
Promedio	922.2221

3.

Repetición	Tiempo(ms)
1	264.757
2	243.632
3	258.884
4	246.995
5	254.7
6	243.823
7	267.316
8	234.086
9	241.628
10	245.183
Promedio	250.1004

4.

Repetición	Tiempo(ms)

1	529.311
2	468.603
3	453.625
4	469.912
5	454.161
6	445.841
7	447.581
8	468.643
9	481.049
10	485.625
Promedio	470.4351

5.

Repetición	Tiempo(ms)
1	595.662
2	429.651
3	514.362
4	360.052
5	584.695
6	569.38
7	342.437
8	348.243
9	469.217
10	486.169
Promedio	469.9868

6.

Repetición	Tiempo(ms)
1	9.134
2	8.159
3	9.214
4	7.154
5	8.147
6	8.321
7	8.895
8	9.236
9	9.874
10	7.459
Promedio	8.5593

7.

Repetición	Tiempo(ms)
1	8.417
2	8.147
3	7.236
4	7.852
5	8.954
6	7.152
7	6.987
8	8.196
9	9.124
10	8.526
Promedio	8.0591

Suma del tiempo promedio de todo el proceso.

Tiempo total promedio	**15019.0292**

Prueba *delete* de toda la base tabla por tabla sin índices.

Repetición	Tiempo(ms)
1	34413.883
2	3310736.175

3	1611349.482
4	86475.365
5	284598.451
6	1478536.641
7	166548.265
8	681746.951
9	2484924.28
10	79415.159
Promedio	1021874.465

Prueba *delete* de toda la base tabla por tabla con índices.

Repetición	Tiempo(ms)
1	17116.173
2	19709.477
3	17288.99
4	17310.68
5	16316.225
6	18124.259
7	17412.655
8	17547.369
9	18214.256
10	19047.128
Promedio	17808.7212

Prueba *update* de un dato en la tabla peli_genero intercambiando todas las veces que se repite un dato sin índices.

Repetición	Tiempo(ms)
1	145.485
2	130.842
3	122.543
4	91.299
5	129.654
6	142.321
7	98.547
8	116.258
9	119.378
10	134.698
Promedio	123.1025

Prueba *update* de un dato en la tabla peli_genero intercambiando todas las veces que se repite un dato con índices

Repetición	Tiempo(ms)
1	39.894
2	75.885
3	107.859
4	84.897
5	74.969
6	85.621
7	78.591
8	84.69
9	103.874
10	96.587
Promedio	83.2867

4. DISCUCIÓN Y CONCLUCIONES

Proceso/ Tiempo(ms)	Sin índice	Con índice
Inserción de los datos mediante *bulk load*	33397.277	50759.4909

Insercion de los datos mediante *insert*	41287.5158	41256.8296
Suma de todos los promedios de los *select*	14866.466	15019.0292
Promedio de borrar las tablas con *delete*	1021874.465	17808.7212
Promedio update el mismo dato con *update*	123.1025	83.2867

De acuerdo con los resultados mostrados en la tabla anterior podemos decir que la ayuda de los índices es variante dependiendo del comando que se desea utilizar. En el caso de la inserción de los datos en las tablas con el método *bulk load* podemos ver como el subirlos a la base de datos con índices es mucho mas lento, en cambio, con el método *insert* tardan prácticamente lo mismo ambos tipos de bases de datos.

De igual forma para la ejecución particularmente de los 7 *selects* que teníamos que ejecutar podemos observar como la diferencia favorece a la base de datos sin índices de manera ligera.

Sin embargo, el borrar la información de las tablas con el comando *delete* es notablemente más rápido con la implementación de los índices.

Por último, en el caso de *update* tambien podemos ver como hay una ligera diferencia favoreciendo a la que contiene índices.

BIBLIOGRAFIA

Camps, R., Casillas, L., Costal, D., Gibert, M., Escofet, Ca., & Pérez, O. (2005). Bases de datos. In *Fundació per a la Universitat Oberta de Catalunya* (first, Vol. 64, Issue 12).

Ramakrishnan, R.; Gehrke, J. (2003). Database Management Systems 3rd Edition. In *International Journal of Scientific and Technology Research* (3rd ed., Vol. 3).

Documentación Postgresql. (2021, 12 agosto). PostgreSQL Documentation. ttps://www.postgresql.org/docs/current/intro-whatis.html