Optimización de BD

Índices

Índices

- Los índices son una forma común de mejorar el rendimiento de la base de datos.
- Un índice permite que el servidor de la base de datos encuentre y recupere filas específicas mucho más rápido de lo que podría hacerlo sin un índice.
- Pero los índices también agregan sobrecarga al sistema de base de datos en su conjunto, por lo que deben usarse con sensatez.
- Invisible para la aplicación.

Índices en Postgres

- Un índice es un objeto más de la base de datos.
- Vinculado a una tabla, está formado por una o más columnas de la tabla y por un puntero a la fila de la tabla que representa, con la interesante característica de que las filas dentro del índice mantienen siempre un orden especificado por algún algoritmo.
- Cuando insertamos una fila nueva en la tabla, ésta se graba en un lugar indeterminado del espacio libre del fichero físico sin embargo, en el índice, el valor de la columna indexada (o columnas) se insertarán en su lugar correspondiente según dicho algoritmo. El índice nos garantiza que se mantendrá el orden también en operaciones de eliminación y de actualización de la columna o columnas indexadas.
- Sin índice, cualquier búsqueda en la tabla obligará al planificador de consultas a hacer un escaneo completo secuencial de la tabla que, para tablas grandes, supone una importante inversión de tiempo.

Metodología

\timing on (*)

create table test (id numeric, name varchar, fecha date);

insert into test (select generate_series(1,10000000), 'name-'||generate_series(1,10000000), now() + interval '1' minute);

explain select * from test where id > 100 and id <= 150;

explain analyze select * from test where id > 100 and id <= 150;

Evalué los resultados de QUERY PLAN

Comando EXPLAIN

- Costo inicio estimado (tiempo inicial que toma devolverse la primer tupla)
- Costo total estimado (tiempo total para devolver todas las tuplas.
- Número estimado de filas escaneadas (Se cumple solamente si la ejecución de la consulta es completa.).
- Cantidad estimado de filas de salida.

Comando EXPLAIN ANALIZE

- Permite visualizar el costo estimado de ejecución de la sentencia, que es la suposición del planificador en el tiempo que se necesita para ejecutar la sentencia (medido en unidades de página de disco obtiene). Nos muestra:
- Tiempo de puesta en marcha antes de la primera fila se puede devolver
- Tiempo total para devolver todas las filas.

Metodología

- create index test_idx on test (id);
- **explain** select * from test where id > 100 and id <= 150;
- **explain analyze** select * from test where id > 100 and id <= 150;
- Evalué los resultados de QUERY PLAN
- ¿Alguna mejora ?
- Las operaciones de inserción y eliminación de filas en la tabla, y las operaciones de actualización de los campos indexados se verán aumentados puesto que ahora, además de realizar la operación correspondiente sobre la tabla y el índice. Por otra parte, el índice también ocupa espacio físico.

Metodología

\dti+

```
                          List of relations

Schema |   Name    | Type  |  Owner   | Table |  Size   | Description
public | test      | table | postgres |       | 498 MB |
public | test_idx | index | postgres | test  | 214 MB |
```

Índices estándares de PostgreSQL

- B-Tree siempre que una columna indexada esté involucrada en una comparación, usando alguno de estos operadores:

- <, <=, =, >=, >, BETWEEN, IN, IS/IS NOT NULL

- También LIKE y ~ si el patrón de búsqueda es una constante anclada al comiendo de la cadena.

Índices estándares de PostgreSQL

- Los **índices Hash** almacenan un código hash de 32 bits a partir del valor de la columna indexada.
- Estos índices, por lo tanto, solo pueden manejar comparaciones de igualdad simple.
- Por ejemplo…

```
explain analyze select * from test where name = ‘name-123’;
create index test_name_idx_hash on test using hash (name);
explain analyze select * from test where name = ‘name-123’;
 \dti+
```

Índices estándares de PostgreSQL

- Los índices GiST no son un tipo de índice al uso sino una infraestructura dentro de la cual se pueden implementar muchas estrategias de indexación diferentes según la clase de operador.
- La distribución estándar de PostgreSQL incluye clases de operadores GiST para varios tipos de datos geométricos bidimensionales, que admiten consultas indexadas utilizando los operadores:

  <<, &<, &>, >>, <<|, &<|, |&>, |>>, @>, <@, -=, &&
- Los índices GiST son capaces de optimizar, por ejemplo, las consultas que buscan los lugares más cercanos a un punto geográfico dado.

Índices estándares de PostgreSQL

- Los **índices SP-GiST** ofrecen una infraestructura que admite varios tipos de búsquedas.
- Los operadores SP-GiST que incluye la versión estándar de PostgreSQL son:

<<, >>, ~=, <@, <<|, |>>

Índices estándares de PostgreSQL

- Los índices GIN se conocen como «índices invertidos» y son apropiados para datos que contienen múltiples componentes con sus correspondientes valores como por ejemplo las matrices.
- Un índice invertido contiene una entrada separada para cada valor de componente y puede manejar de manera eficiente las consultas que prueban la presencia de valores de componentes específicos.
- Al igual que los dos anteriores, los índices GIN puede admitir muchas estrategias de indexación diferentes. Los operadores con los que se pueden usar un índice GIN son:

<@, @>, = &&

Índices estándares de PostgreSQL

- Índices BRIN es la abreviatura de «Block Range INdexes». Almacenan resúmenes sobre los valores almacenados en bloques físicos consecutivos de una tabla.
- Los hace óptimos para columnas cuyos valores están ordenador linealmente con el orden físico de las filas de la tabla.
- Para los tipos de datos que tienen un orden de clasificación lineal, los datos indexados corresponde a los valores mínimo y máximo en la columna para cada rango de bloque. El índice en lugar de almacenar un valor por cada fila de la tabla, almacena un par de valores por cada conjunto de n filas.
- El tamaño del índice es mucho menor que en los casos anteriores, y aún más eficiente.
- Los operadores con los que el planificador puede utilizar un índice BRIN son:

<, <=, =, >=, >

Índices estándares de PostgreSQL

explain analyze select * from test where id > 300 and id <= 350;

create index test_name_idx_brin on test **using brin** (id);

explain analyze select * from test where id > 300 and id <= 350;

\dti+

Debe de apreciarse una mejora en el tiempo de ejecución de la consulta. El tamaño del índice debe ser menor que en los casos anteriores.

¿Cuándo se deben evitar los índices?

- Los índices están destinados a mejorar el rendimiento de una base de datos, hay momentos en los que deben evitarse:
- Los índices no deben usarse en tablas pequeñas.

- Tablas que tienen operaciones frecuentes de actualización o inserción de lotes grandes.

- Los índices no deben usarse en columnas que contienen una gran cantidad de valores NULL.

- Las columnas que se manipulan con frecuencia no deben indexarse.

Práctica 6.

Base de Datos aérea de Postgres

- La base de datos es de una compañía aérea virtual llamada Postgres Air.
- Esta empresa conecta más de 600 destinos virtuales en todo el mundo, ofrece alrededor de 32.000 vuelos virtuales directos semanalmente y cuenta con más de 100.000 miembros virtuales en su programa de viajero frecuente y muchos más pasajeros cada semana.
- Los datos son ficticios y se proporcionan solo con fines ilustrativos.

Base de Datos aérea de Postgres

- Aparece en dos formatos en el Drive de Google.
- SQL y Backup
- En caso de que emplee Backup
- pg_restore **-x --no-owner** -U postgres -d hettie /home/usuario/postgres_air.backup

- Identificar objetivos de optimización en sistemas OLTP (Online Transaction Processing) y OLAP (Online Analytical Processing).
- Leer y comprender los planes de ejecución de PostgreSQL.
- Identificar índices que mejorarán el rendimiento de las consultas.
- Optimizar exploraciones de tablas completas.
- Distinguir entre consultas largas y consultas cortas.
- Elija la técnica de optimización adecuada para cada tipo de consulta.
- Evite las trampas de los marcos ORM.
- Algoritmo de optimización definitivo, que guía a un desarrollador de bases de datos a través del proceso de producir la consulta de mayor rendimiento.

Resumen

- Escribir una consulta de base de datos es diferente de escribir código de aplicación usando lenguajes imperativos. SQL es un lenguaje declarativo, lo que significa que especificamos el resultado deseado, pero no especificamos una ruta de ejecución. Dado que dos consultas que arrojan el mismo resultado pueden ejecutarse de manera diferente, utilizando diferentes recursos y tomando una cantidad diferente de tiempo, la optimización y "pensar como una base de datos" son partes fundamentales del desarrollo de SQL.

- En lugar de optimizar consultas que ya están escritas, nuestro objetivo es escribir consultas correctamente desde el principio. Idealmente, la optimización comienza en el momento de recopilar los requisitos y diseñar la base de datos. Luego, podemos proceder a optimizar tanto las consultas individuales como la forma en que se estructuran las llamadas a la base de datos desde la aplicación.

- Pero la optimización no termina ahí; Para mantener el rendimiento del sistema, necesitamos monitorear el rendimiento a lo largo del ciclo de vida del sistema.

Esquema de la BD

- List available schema
- \dn
- set search_path to postgres_air, public;
- \timing on
- EXPLAIN ANALYZE.

Otras optimizaciones

Consideraciones

- Cuando se ejecuta un UPDATE o un DELETE, marca la fila como borrada.
- Esto produce internamente “filas muertas”, que son aquellas filas cuyos valores han sido eliminados o los valores anteriores a una actualización.
- Las “filas vivas” son los nuevos valores insertados o los nuevos valores de una fila actualizada.

Comandos básicos

- ANALYZE. Permite analizar cada una de las tablas o la base de datos para informar al planificador de consultas del estado de las mismas, de esta forma obtenemos mejor rendimiento cuando se ejecutan las consultas.

- VACUUM. Permite realizar limpieza en cada tabla o en la base de datos, así evitamos que el sistema se sobrecargue de filas muertas o que las tablas ocupen demasiado espacio físico en el disco duro. Esto podría hacer que el sistema se vea mermado en su rendimiento con el paso del tiempo.

- REINDEX. Este comando es similar al anterior, con la diferencia que actúa sobre los índices. Reconstruye los índices eliminando aquellas páginas que no contienen filas. De esta forma se disminuye el tamaño y se incrementa la velocidad de nuestras consultas.

Referencias

- PostgreSQL: Documentation: 15: ANALYZE. https://www.postgresql.org/docs/current/sql-analyze.html

- PostgreSQL: Documentation: 15: VACUUM. https://www.postgresql.org/docs/current/sql-vacuum.html

- PostgreSQL: Documentation: 15: REINDEX . https://www.postgresql.org/docs/current/sql-reindex.html

```
drop table test;
create table test (id numeric, name varchar, fecha date);
insert into test (select generate_series(1,1000), 'name-
'||generate_series(1,1000), now() + interval '1' minute);
select pg_relation_size ('test’); # SET search_path TO postgres_air;
DELETE FROM test WHERE id > 500;
select pg_relation_size ('test');
VACUUM FULL;
VACUUM (VERBOSE, ANALYZE) test;
```

- ANALYZE test;
- ANALYZE test(id);

Reconstruir un índice

- REINDEX INDEX my_index;
- # Reconstruir todos los índices de una tabla
- REINDEX TABLE my_table;

Cliente principal Postgres