Chapter 13 - Performance Tuning

Introducción

Cualquier sistema de gestión de bases de datos relacionales (RDBMS) tiene como objetivo resolver las consultas de los usuarios lo más rápido posible, y PostgreSQL no es una excepción. Por lo tanto, mantener un buen rendimiento en PostgreSQL es uno de los principales objetivos para los administradores de bases de datos (DBAs).

En capítulos anteriores, hemos estudiado muchas de las capacidades de PostgreSQL y cómo configurarlas y utilizarlas. Ahora, nos adentraremos en los temas de ajuste de rendimiento. Aprenderemos sobre los conceptos básicos y los componentes involucrados en la optimización de bases de datos, además de algunas buenas prácticas que podemos usar como guía inicial.

Estructura

Durante este capítulo, estudiaremos las siguientes secciones:

- Índices
- Estadísticas
- Plan de ejecución (Explain Plan)
- Mejores prácticas para los parámetros del archivo `postgresql.conf`

Objetivos

Al completar este capítulo, estarás familiarizado con los componentes básicos para la optimización de PostgreSQL y comprenderás su relación e impacto. También aprenderás a aplicar las mejores prácticas estándar al instalar o ajustar un sistema PostgreSQL para obtener el mejor rendimiento.

Índices

¿Alguna vez has notado que los índices en un libro ayudan a ubicar los capítulos mencionando el número de página? Esto permite saltar directamente a la página en lugar de revisar cada capítulo de forma secuencial, lo que ahorra tiempo. De la misma manera, los índices en PostgreSQL crean un tipo de metadatos similares, como los números de página, que permiten acceder directamente a los datos solicitados sin tener que escanear toda la tabla de manera secuencial.

Sin embargo, este beneficio conlleva un costo, ya que agregar un índice implica que debe crearse o modificarse a medida que los datos se insertan, actualizan o eliminan de la tabla. Por ello, se deben crear índices de manera estratégica después de un análisis adecuado. Si la tabla se utiliza más para operaciones de lectura que de escritura, entonces los índices pueden ser beneficiosos. De lo contrario, pueden convertirse en una sobrecarga adicional al tener que actualizarse con cada operación de escritura.

```
CREATE [UNIQUE] INDEX [CONCURRENTLY] [IF NOT EXISTS] nombre ON
[ONLY] nombre_tabla [USING método]
  (columna | (expresión) [COLLATE collation] [opclass ...])
  [INCLUDE (columna, ...)]
  [NULLS [NOT] DISTINCT]
  [WITH (parámetro_de_almacenamiento = valor, ...)]
  [TABLESPACE nombre_tablespace]
  [WHERE predicado]
```

Sintaxis según la documentación de PostgreSQL: La creación y eliminación de un índice (`CREATE INDEX` y `DROP INDEX`) no son actividades en línea de forma predeterminada. La tabla se bloquea para cualquier modificación durante la operación. Sin embargo, las operaciones de lectura aún pueden realizarse mientras se crea o modifica un índice si se usa la palabra clave `CONCURRENTLY`. Esto permite modificar los índices de manera concurrente sin bloquear la tabla.

También se pueden crear índices en múltiples columnas, conocidos como índices multicolumna. Al hacerlo, es importante asegurarse de que el orden de las columnas en el índice coincida con el orden utilizado en la cláusula `WHERE` de las consultas.

Reindexación (REINDEX)

Reindexar un índice es una de las actividades de mantenimiento en PostgreSQL que ayuda a reconstruir índices corruptos, inflados o inválidos de manera concurrente. Un índice también puede reconstruirse cuando se cambia el almacenamiento de ese índice en particular.

```
REINDEX [ (opción, ...) ] { INDEX | TABLE | SCHEMA | DATABASE | SYSTEM } [CONCURRENTLY] nombre
```

Sintaxis según la documentación de PostgreSQL: Donde las opciones pueden ser:

- `CONCURRENTLY [boolean]`: Permite reindexar de manera concurrente.
- `TABLESPACE new_tablespace`: Permite reindexar en un nuevo tablespace.
- `VERBOSE [boolean]`: Muestra registros detallados durante la reindexación.

Cuando se crean claves primarias (`PRIMARY KEY`) o claves únicas (`UNIQUE`), el sistema genera índices por defecto en esas columnas, ya que suelen utilizarse para buscar datos en las consultas. En general, los índices deben crearse en columnas que se usan en la cláusula `WHERE`. No es una regla estricta, pero es una buena práctica crear índices en columnas utilizadas como claves foráneas (`FOREIGN KEY`), ya que a menudo se utilizan en condiciones de `JOIN` para recuperar datos.

En bases de datos pequeñas, los índices predeterminados en las claves primarias o únicas pueden ser suficientes al principio. Sin embargo, a medida que los datos crecen, las consultas pueden volverse más lentas. Se puede usar el comando `EXPLAIN PLAN` para verificar si las consultas están utilizando los índices. Este comando se explicará más adelante en este capítulo.

Los índices son un componente clave para mejorar el rendimiento de las consultas, especialmente aquellas que involucran grandes volúmenes de datos o `SELECT` que requieren tiempos de ejecución optimizados.

También pueden mejorar el rendimiento de las consultas `INSERT`, `UPDATE` y `DELETE` cuando se usa una condición `WHERE`. Se recomienda ejecutar el comando `ANALYZE` para mantener actualizadas las estadísticas de los índices y asegurar que contengan metadatos recientes, lo que ayuda a mejorar el rendimiento de las consultas.

Tipos de Índices

Cada consulta es diferente y puede que el índice `BTREE` predeterminado no sea la mejor opción en todos los casos. Para abordar esto, PostgreSQL ofrece varios tipos de índices que pueden mejorar el rendimiento de diferentes tipos de consultas. Para especificar un tipo de índice distinto a `BTREE`, se usa la palabra clave `USING` en la sentencia `CREATE INDEX`.

Índice B-Tree (BTREE) Por defecto, PostgreSQL crea un índice `BTREE` si no se especifica otro tipo. Este índice ordena los datos de manera secuencial y funciona bien cuando se usa el operador de igualdad en la cláusula `WHERE`. También es eficiente con los siguientes operadores:

- `<`
- `>`
- `<=`
- `>=`
- `IS NULL`
- `IS NOT NULL`
- `BETWEEN`
- `IN`

Los índices B-Tree también pueden utilizarse para recuperar datos en ordenado. Aunque no siempre es más rápido que un escaneo y ordenamiento normal, suele ser útil en muchas situaciones.

Índice Hash Este tipo de índice genera un código hash de 32 bits en la columna donde se crea el índice. Es más eficiente cuando se usa el operador de igualdad en la cláusula `WHERE`.

Índice GiST y SP-GiST Estos índices son útiles cuando se trabaja con datos geométricos bidimensionales.

Índice GIN Este índice se usa cuando los datos contienen tipos de datos unidimensionales como `ARRAYS`.

Índice BRIN Los índices `BRIN` (Block Range INdex) almacenan resúmenes sobre los valores en bloques físicos consecutivos de una tabla. Son más efectivos cuando los valores de las columnas están bien correlacionados con el orden físico de las filas de la tabla.
