

Índices en PostgreSQL

info@todopostgresql.com




**Todo
PostgreSQL**
by Abatic

1. Introducción
2. Sintaxis
3. Consideraciones
4. Índices paralelos
(CONCURRENTLY)
 - Inconvenientes del
CONCURRENTLY
5. Inconvenientes de los
índices
6. Recuperación de un índice

7. Tipos de índices

- Índice HASH
- Índice B-Tree
- Índices GIN
- Índices GiST
- Índice BRIN
 - Ejemplo de índice BRIN

8. Índices multi-columnas

- 
9. Índices ordenados
 10. Índices únicos
 11. Índices funcionales
 12. Índices parciales
 13. Index-Only Scan
 14. Mantenimiento
 15. ¿Se usan?

Introducción

- Permite mejorar el rendimiento de la base de datos.
- Permite recuperar filas específicas mucho más rápido.
- Soporta una gran variedad de índices:
 - **B-tree** (Por defecto): se utiliza en consultas de igualdad y de rango ($<$, $<=$, $=$, $>=$, $>$).
 - **Hash**: sólo puede manejar comparaciones de igualdad simple.

Introducción

- Soporta una gran variedad de índices:
 - **GiST** (Generalized Search Tree): optimizado para búsquedas de texto completos, datos espaciales, jerárquicos, etc.
 - **GIN** (Generalized Inverted Index): especialización de Gist, más rápido para búsquedas pero más lento para actualizar.
 - **BRIN** (Block Ranger Indexes): almacenamiento de un rango de páginas.

Introducción

- Soporta una gran variedad de índices:
 - **Indexes on Expression:** útil para acceder rápidamente a consultas basadas en una computación.
 - **Partial Index:** para indexar valores comunes.
 - **Index-Only-Scans:** puede responder consultas sin acceder al área de datos principal de la tabla (montón de la tabla).

Introducción

- Los índices son herramientas que permiten acelerar determinadas operaciones de la base de datos.
 - Sin ellos, el sistema tendría que leer las tablas, fila por fila, cada vez.
 - Cada vez que se realiza una inserción, una actualización o un borrado, el sistema actualizará el índice.

Introducción

- Equivalentemente, en un libro, suele incluirse un índice de términos / glosario al final del libro.
 - El autor se anticipa a las consultas más frecuentes de los lectores.
 - De igual manera, el DBA tiene que predecir (¡o detectar!) la mayor parte de los índices necesarios.

Sintaxis

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ]  
  [ nombre_indice ] ON tabla  
  [ USING tipo_indice ]  
  ( { columna | ( expresión ) }  
  [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...] )  
  [ TABLESPACE nombre_tablespace ]  
  [ WHERE predicado ]
```

Sintaxis

- **UNIQUE:** El sistema chequea los valores duplicados en la tabla en el momento de creación del índice y cada vez que se añaden datos a la tabla.
- **CONCURRENTLY:** Crea un índice sin realizar ningún bloqueo que prevenga inserciones, actualizaciones y borrados sobre la tabla.

Sintaxis

- Se puede crear un índice utilizando el comando SQL:
`CREATE INDEX nombre_indice ON
tabla(expr);`
- Para eliminar un índice, se emplea:
`DROP INDEX nombre_indice;`

Consideraciones

- Los índices pueden añadirse y eliminarse libremente en cualquier momento.
- Una vez se añade, no es necesaria ninguna intervención posterior. El sistema se encarga de su mantenimiento.
- Conlleva una pequeña sobrecarga.

Consideraciones

- Una vez definido el índice, el sistema lo utilizará en las consultas cuando considere que probablemente sea más eficaz que una lectura secuencial de las filas.
- Puede ser necesario actualizar las estadísticas (Comando ANALYZE) para que sea realmente útil.

Consideraciones

- Los índices también se pueden usar cuando se incluyan condiciones de filtrado en:
 - UPDATE
 - DELETE
- Y para construir los JOIN, ya que puede suponer una gran mejora de rendimiento.

Consideraciones

- Crear un índice sobre una tabla con un gran número de filas puede llevar mucho tiempo.
- Consumo de recursos (I/O), especialmente si tarda varias horas (tablas grandes).

Consideraciones

- Por defecto, se permiten las lecturas mientras se crea un índice sobre una tabla.
- Permite construir el índice con un solo tablescan.
- Pero se bloquean las escrituras hasta que se termine, esto puede resultar inaceptable.

Índices paralelos (CONCURRENTLY)

- Es posible crear un índice sin bloquear las escrituras a la tabla.

CREATE INDEX CONCURRENTLY

- Se soporta la creación paralela de:
 - Índices sobre expresiones.
 - Índices parciales.

Inconvenientes del CONCURRENTLY

- Los inconvenientes que tiene PostgreSQL en el modo “*concurrente*” son los siguientes:
 - Tiene que hacer al menos dos tablescan.
 - Esperar a que todas las transacciones que puedan usar el índice se completen.
 - Mucho más lento, pero puede ser necesario.
 - Sólo un CREATE INDEX CONCURRENTLY a la vez por la tabla.

Inconvenientes del CONCURRENTLY

- Durante la creación de un índice en forma paralela:
 - El índice se inserta en el catálogo en una transacción.
 - Después, se llevan a cabo dos tablescans (cada uno en una transacción).
 - Cualquier transacción activa al inicio del segundo “barrido” puede bloquear la creación del índice.

Inconvenientes de los índices

- Durante la creación de un índice:
 - No se pueden llevar a cabo ALTERs de la tabla.
- Un CREATE INDEX normal se puede llevar a cabo en una transacción (atómico), pero la versión CONCURRENTLY no lo soporta.

Inconvenientes de los índices

- Si surgiera algún problema durante el escaneo de la tabla.
 - Por ejemplo, una violación de unicidad al añadir un índice UNIQUE.

Inconvenientes de los índices

- El comando CREATE INDEX fallará, dejando un índice inválido.
 - Éste índice invalido, no se podrá usar en consultas.
 - Se seguirá actualizando (penalización I/O y uso de disco).

Recuperación de un índice

- Recuperación de un índice fallido:
 - DROP del index.
 - Reintentar la creación concurrente.
 - Comando REINDEX (pero no es concurrente).

Tipos de índices

- Índice HASH:
 - Usa una “Función resumen”.
 - Sólo se puede utilizar para comparaciones de igualdad.
 - No transaccionales (“crash-safe”). En PG 10, Sí.
 - Muy rápidos para consultas de igualdad.

Tipos de índices

- Índice B-Tree:

- El habitual en todas las bases de datos.
- Soporta todas las operaciones $<, >, <=, >=, =, <>$
- Creación atómica (transaccional).
- Muy eficiente.
- El índice por defecto en PostgreSQL.

Tipos de índices

- Índices GIN:
 - Generalized INverted index.
 - Adecuado para búsquedas de texto.
 - Mucha sobrecarga de espacio, pero útil.
 - Los índices GIN pueden manejar valores que contengan más de una clave, por ejemplo arrays.

Tipos de índices

- Índices GiST:
 - Generalized Search Tree.
 - Más bien una infraestructura para construir índices.
 - Por ejemplo: SP-GiST (PostGIS).

Tipos de índices

- Índices BRIN:
 - Block Ranger Indexes.
 - Almacenamiento de metadatos.
 - Índices de pequeño tamaño.
 - Útiles en tablas grandes y ordenadas de manera natural (por ejemplo, facturas por fechas).

Tipos de índices

- Índices BRIN:
 - Búsquedas más lentas que índices B-TREE.
 - Nuevo parámetro: **pages_per_range**.
 - Bloque de 128 páginas (1 Mb).

Ejemplo de índice BRIN

```
CREATE TABLE pedidos (  
    id serial primary key,  
    fecha_pedido timestamptz,  
    producto text);
```

Ejemplo de índice BRIN

```
INSERT INTO pedidos (fecha_pedido, producto)
SELECT x, 'PostgreSQL'
FROM generate_series('2018-01-01
00:00:00'::timestamptz, '2020-01-01
00:00:00'::timestamptz, '2 seconds'::interval) a(x);
```

```
INSERT O 31536001
```


Ejemplo de índice BRIN

```
postgres=# \dt+ pedidos
```

Listado de relaciones

Esquema	Nombre	Tipo	Dueño	Tamaño	Descripción
public	pedidos	tabla	postgres	1812 MB	

(1 fila)

```
EXPLAIN ANALYSE SELECT count(*) FROM  
pedidos WHERE fecha_pedido BETWEEN  
'2018-04-22 06:00:00' AND '2019-07-06 12:30:00';
```

Ejemplo de índice BRIN

```
Finalize Aggregate (cost=449745.13..449745.14 rows=1 width=8) (actual time=116536.303..116561.916 rows=1 loops=1)
  -> Gather (cost=449744.91..449745.12 rows=2 width=8) (actual time=116535.734..116561.901 rows=3 loops=1)
    Workers Planned: 2
    Workers Launched: 2
    -> Partial Aggregate (cost=448744.91..448744.92 rows=1 width=8) (actual time=116303.789..116303.790 rows=1 loops=3)
      -> Parallel Seq Scan on pedidos (cost=0.00..428983.55 rows=7904545 width=0) (actual time=12762.255..115572.930 rows=6339900 loops=3)
        Filter: ((fecha_pedido >= '2018-04-22 06:00:00+02'::timestamp with time zone) AND (fecha_pedido <= '2019-07-06 12:30:00+02'::timestamp with time zone))
        Rows Removed by Filter: 4172100
        Planning Time: 7.033 ms
        Execution Time: 116572.252 ms
(10 filas)
```

Ejemplo de índice BRIN

```
Finalize Aggregate (cost=449745.13..449745.14 rows=1 width=8) (actual time=3858.065..3871.079 rows=1 loops=1)
  -> Gather (cost=449744.91..449745.12 rows=2 width=8) (actual time=3857.674..3871.070 rows=3 loops=1)
    Workers Planned: 2
    Workers Launched: 2
    -> Partial Aggregate (cost=448744.91..448744.92 rows=1 width=8) (actual time=3795.531..3795.532 rows=1 loops=3)
      -> Parallel Seq Scan on pedidos (cost=0.00..428983.55 rows=7904545 width=0) (actual time=620.730..3338.451 rows=6339900 loops=3)
        Filter: ((fecha_pedido >= '2018-04-22 06:00:00+02'::timestamp with time zone) AND (fecha_pedido <= '2019-07-06 12:30:00+02'::timestamp with time zone))
        Rows Removed by Filter: 4172100
        Planning Time: 0.191 ms
        Execution Time: 3871.145 ms
(10 filas)
```

Ejemplo de índice BRIN

```
CREATE INDEX idx_pedidos_fecha_brin  
ON pedidos  
USING BRIN (fecha_pedido);
```

```
postgres=# \di+ idx_pedidos_fecha_brin
```

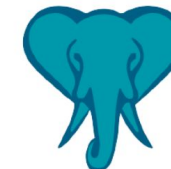
Listado de relaciones

Esquema	Nombre	Tipo	Dueño	Tabla	Tamaño
public	idx_pedidos_fecha_brin	Índice	postgres	pedidos	104 kB

(1 fila)

Ejemplo de índice BRIN

```
Finalize Aggregate (cost=443476.47..443476.48 rows=1 width=8) (actual time=3102.156..3136.838 rows=1 loops=1)
  -> Gather (cost=443476.26..443476.47 rows=2 width=8) (actual time=3101.423..3136.825 rows=3 loops=1)
    Workers Planned: 2
    Workers Launched: 2
    -> Partial Aggregate (cost=442476.26..442476.27 rows=1 width=8) (actual time=3039.737..3039.737 rows=1 loops=3)
      -> Parallel Bitmap Heap Scan on pedidos (cost=4814.63..422714.95 rows=790452 width=0) (actual time=6.018..2540.742 rows=6339900 loops=3)
        Recheck Cond: ((fecha_pedido >= '2018-04-22 06:00:00+02'::timestamp with time zone) AND (fecha_pedido <= '2019-07-06 12:30:00+02'::timestamp with time zone))
        Rows Removed by Index Recheck: 8217
        Heap Blocks: lossy=48980
        -> Bitmap Index Scan on idx_pedidos_fecha_brin (cost=0.00..71.91 rows=18987735 width=0) (actual time=10.700..10.701 rows=1400320 loops=1)
          Index Cond: ((fecha_pedido >= '2018-04-22 06:00:00+02'::timestamp with time zone) AND (fecha_pedido <= '2019-07-06 12:30:00+02'::timestamp with time zone))
        Planning Time: 16.608 ms
        Execution Time: 3136.944 ms
      (13 filas)
```



Ejemplo de índice BRIN

```
Finalize Aggregate (cost=443476.47..443476.48 rows=1 width=8) (actual time=2903.049..2922.540 rows=1 loops=1)
  -> Gather (cost=443476.26..443476.47 rows=2 width=8) (actual time=2902.596..2922.531 rows=3 loops=1)
    Workers Planned: 2
    Workers Launched: 2
    -> Partial Aggregate (cost=442476.26..442476.27 rows=1 width=8) (actual time=2825.891..2825.892 rows=1 loops=3)
      -> Parallel Bitmap Heap Scan on pedidos (cost=4814.63..422714.95 rows=790452 width=0) (actual time=6.056..2356.601 rows=6339900 loops=3)
        Recheck Cond: ((fecha_pedido >= '2018-04-22 06:00:00+02'::timestamp with time zone) AND (fecha_pedido <= '2019-07-06 12:30:00+02'::timestamp with time zone))
        Rows Removed by Index Recheck: 8217
        Heap Blocks: lossy=49048
        -> Bitmap Index Scan on idx_pedidos_fecha_brin (cost=0.00..71.91 rows=18987735 width=0) (actual time=10.898..10.899 rows=1400320 loops=1)
          Index Cond: ((fecha_pedido >= '2018-04-22 06:00:00+02'::timestamp with time zone) AND (fecha_pedido <= '2019-07-06 12:30:00+02'::timestamp with time zone))
          Planning Time: 0.354 ms
          Execution Time: 2922.665 ms
(13 filas)
```

Índices multi-columnas

- Un índice puede ser definido para más de una columna.
 - Limitado a 32 columnas por defecto.
 - Actualmente, los tipos B-Tree, GiST, GIN y BRIN.
 - Se deben usar con moderación.
 - Laboriosos de mantener actualizados.

Índices multi-columnas

```
CREATE TABLE test2 (  
    major int,  
    minor int,  
    name varchar  
);
```

```
CREATE INDEX test2_mm_idx ON test2 (major, minor);
```


Índices ordenados

- Se puede ajustar el orden de un índice B-tree
 - Opciones ASC, DESC, NULLS FIRST, y/o NULLS LAST.

```
CREATE INDEX prueba2_nulls ON prueba (foo  
NULLS FIRST);
```

```
CREATE INDEX prueba2_desc ON prueba (id  
DESC NULLS LAST);
```

Índices ordenados

- Esto ahorrará tiempo de ordenación en la propia consulta.
- Por defecto, los índices B-Tree almacenan sus entradas en orden ascendente, con los nulos al final (NULLS LAST).

Índices únicos

- Para verificar la singularidad/unicidad del valor de una columna.
- La singularidad de valores combinados de más de una columna.

```
CREATE UNIQUE INDEX name ON table  
(column [...]);
```

Índices únicos

- Actualmente, sólo los índices B-tree pueden declararse como únicos.
- PostgreSQL crea automáticamente un índice único:
 - Cuando se define una restricción de valor único (unique constraint).
 - Cuando se define una clave primaria.

Índices funcionales

- Se puede crear un índice sobre un valor (expresión) calculado en las columnas de la tabla.
- Es útil para obtener de manera rápida, los datos basados en los resultados de la ejecución de esas funciones.

Índices funcionales

- Las expresiones de los índices son relativamente caras de mantener.
- Los índices sobre expresiones son útiles cuando la velocidad de obtención de datos es más importante que la velocidad de inserción y actualización de los mismos.
- También se pueden crear a partir de funciones definidas por el usuario (con CREATE FUNCTION).

Índices funcionales

- Ejemplo, búsqueda sin importar mayúsculas y minúsculas.

```
CREATE INDEX test1_lower_col1_idx ON test1 (lower(col1));
```

```
SELECT * FROM test1 WHERE lower(col1) = 'value';
```

Índices funcionales

- También puede ser que necesitemos utilizar más de una columna para crear la expresión buscada.

```
SELECT * FROM people WHERE (first_name || ' ' || last_name) = 'John Smith';
```

```
CREATE INDEX people_names ON people ((first_name || ' ' || last_name));
```


Índices parciales

- Un índice parcial es un índice construido a partir de un subconjunto de los datos de una tabla.
- El índice contiene entradas sólo para aquellas filas que cumplan el predicado.
- Muy utilizado cuando se realizan muchas búsquedas de valores comunes.

Índices parciales

- Ocupa menos que un índice total (dependiendo del filtro), es más eficiente.
- Son útiles cuando un determinado sub-conjunto de datos tiene un tamaño considerable
- Por ejemplo, valores NULL ...

Índices parciales

- Disponemos de unos pedidos que se encuentran facturados y no facturados.
- El mayor porcentaje de la tabla son pedidos facturados.
- El mayor número de consultas son sobre pedidos sin facturar.

```
SELECT * FROM orders WHERE billed is not true AND order_nr < 10000;
```

```
CREATE INDEX orders_unbilled_index ON orders (order_nr)  
WHERE billed is not true;
```

Index-Only Scan

- En un análisis de índice ordinario, cada recuperación de fila requiere obtener datos tanto del índice como del montón.
- Ocasiona que sean lentos.
- Para resolver este problema, PostgreSQL admite Index-Only Scan.

Index-Only Scan

- Con esto, puede responder consultas sin acceder al área de datos principal de la tabla (montón de la tabla).
- Soporta índices B-Tree, GiST y SP-GiST.

Mantenimiento

- Comando CLUSTER:
 - Ordena físicamente las filas siguiendo el índice.
CLUSTER tabla USING indice.

¿Se usan?

- Es difícil dar un procedimiento genérico con el que determinar qué índices crear.
- Requiere de cierta observación.
- Ejecutar primero siempre antes de pensar.
- Usar datos reales para experimentar.

¿Se usan?

- Cuando no se usan índices, puede ser útil hacer pruebas que obliguen su uso.
- El comando `EXPLAIN ANALYZE` puede ser útil en este caso (para determinar qué está haciendo el planificador).

Cursos de PostgreSQL para DBA y Developers



**Todo
PostgreSQL**
by Abatic

TODOPOSTGRESQL.COM