aws

Actividades de mantenimiento de las bases de datos de PostgreSQL en Amazon RDS y Amazon Aurora para evitar problemas de rendimiento

AWS Orientación prescriptiva

AWS Orientación prescriptiva: Actividades de mantenimiento de las bases de datos de PostgreSQL en Amazon RDS y Amazon Aurora para evitar problemas de rendimiento

Table of Contents

Introducción 1
Resultados empresariales específicos 1
Control de simultaneidad multiversión (MVCC) 1
Succión y análisis automáticos de las tablas 3
Parámetros de autovacuum relacionados con la memoria 5
Ajuste de los parámetros de autovacuum 6
A nivel de instancia o clúster 6
Nivel de tabla 6
Uso de una configuración de autovacuum agresiva a nivel de tabla 6
Ventajas y limitaciones 8
Succión y análisis manuales de las tablas 9
Ejecución de operaciones de succión y limpieza en paralelo 10
Reescritura de una tabla completa con VACUUM FULL 14
Eliminación de la sobrecarga con pg_repack 16
Nueva creación de índices 18
Creación de un nuevo índice 22
Nueva creación de un índice 22
Ejemplos 23
Ejemplo: Recuperación de espacio mediante autovacuum y VACUUM FULL 25
Recursos 30
Historial de documentos 31
..... xxxii

Actividades de mantenimiento de las bases de datos de PostgreSQL en Amazon RDS y Amazon Aurora para evitar problemas de rendimiento

Anuradha Chintha, Rajesh Madiwale y Srinivas Potlachervoo, Amazon Web Services (AWS)

Agosto de 2025 (historia del documento)

La edición de Amazon Aurora compatible con PostgreSQL y Amazon Relational Database Service (Amazon RDS) para PostgreSQL son servicios de bases de datos relacionales completamente administrados para bases de datos de PostgreSQL. Estos servicios administrados liberan al administrador de la base de datos de muchas tareas de mantenimiento y administración. Sin embargo, algunas tareas de mantenimiento, como `VACUUM`, requieren una estrecha supervisión y una configuración que se ajuste al uso de la base de datos. En esta guía se describen las actividades de mantenimiento de PostgreSQL en Amazon RDS y Aurora.

Resultados empresariales específicos

El rendimiento de las bases de datos es una medida clave que sustenta el éxito de una empresa. Las actividades de mantenimiento que realiza en las bases de datos de Aurora compatible con PostgreSQL y Amazon RDS para PostgreSQL ofrecen las siguientes ventajas:

- Ayuda a lograr un rendimiento óptimo de las consultas
- Libera el espacio sobrecargado para que lo reutilicen en futuras transacciones
- Evita el reinicio de las transacciones
- Ayuda al optimizador a generar buenos planes
- Garantiza un uso adecuado del índice

Control de simultaneidad multiversión (MVCC)

El mantenimiento de la base de datos de PostgreSQL requiere comprender el control de simultaneidad multiversión (MVCC), que es un mecanismo de PostgreSQL. Cuando se procesan varias transacciones simultáneamente en la base de datos, el MVCC garantiza la atomicidad

y el aislamiento, que son dos características de las transacciones con atomicidad, coherencia, aislamiento y durabilidad (ACID). En el MVCC, cada operación de escritura genera una nueva versión de los datos y almacena la versión anterior. Los lectores y los escritores no se bloquean entre sí. Cuando una transacción lee datos, el sistema elige una de las versiones para aislar la transacción. PostgreSQL y algunas bases de datos relacionales utilizan una adaptación del MVCC llamada aislamiento de instantáneas (SI). Por ejemplo, Oracle implementa SI mediante segmentos de reversión. Durante una operación de escritura, Oracle escribe la versión antigua de los datos en un segmento de reversión y sobrescribe el área de datos con la versión nueva. Las bases de datos de PostgreSQL implementan SI mediante reglas de comprobación de la visibilidad para evaluar las versiones. Cuando se colocan datos nuevos en una página de tabla, PostgreSQL utiliza estas reglas para seleccionar la versión de los datos adecuada para una operación de lectura.

Al modificar los datos de una fila de la tabla, PostgreSQL utiliza el MVCC para mantener varias versiones de la fila. Durante las opciones `UPDATE` y `DELETE` que realiza en la tabla, la base de datos conserva las versiones antiguas de las filas para otras transacciones en ejecución que puedan necesitar una vista de los datos coherente. Estas versiones antiguas se llaman filas inactivas (tuplas). Las colecciones de tuplas inactivas producen sobrecarga. Una gran cantidad de sobrecarga en la base de datos puede provocar una serie de problemas, como una creación deficiente de planes de consultas, un rendimiento lento de las consultas y un mayor uso del espacio en disco para almacenar las versiones antiguas.

Para eliminar la sobrecarga y mantener una base de datos en buen estado, se tiene que realizar un mantenimiento periódico con estas actividades, que se analizan en las siguientes secciones:

- Succión y análisis automáticos de las tablas
- Succión y análisis manuales de las tablas
- Eliminación de la sobrecarga con pg_repack
- Nueva creación de índices

Succión y análisis automáticos de las tablas

Autovacuum es un daemon (es decir, se ejecuta en segundo plano) que succiona (limpia) automáticamente las tuplas inactivas, recupera el almacenamiento y recopila estadísticas. Comprueba si hay tablas hinchadas en la base de datos y borra esa sobrecarga para poder reutilizar el espacio. Supervisa las tablas y los índices de las bases de datos y los agrega a una tarea de succión cuando alcanzan un límite específico de operaciones de actualización o eliminación.

Autovacuum administra la succión mediante la automatización de los comandos `VACUUM` y `ANALYZE` de PostgreSQL. `VACUUM` elimina la sobrecarga de las tablas y recupera espacio, a la vez que `ANALYZE` actualiza las estadísticas que permiten al optimizador elaborar planes eficientes. `VACUUM` también lleva a cabo una importante tarea denominada congelación por succión para evitar problemas relacionados con los ID de las transacciones en la base de datos. Cada fila que se actualiza en la base de datos recibe un ID de transacción de parte del mecanismo de control de transacciones de PostgreSQL. Estos IDs controlan la visibilidad de la fila para otras transacciones simultáneas. El ID de la transacción es un número de 32 bits. Dos mil millones IDs siempre se guardan en el pasado visible. Los restantes (unos 2.200 millones) IDs se conservan para transacciones que se realizarán en el futuro y se ocultan de la transacción actual. PostgreSQL necesita que se realicen limpiezas y congelaciones de las filas antiguas de forma ocasional para evitar que las transacciones se acumulen y hagan invisibles las filas antiguas existentes cuando se crean nuevas transacciones. Para obtener más información, consulte Preventing Transaction ID Wraparound Failures en la documentación de PostgreSQL.

Autovacuum es una funcionalidad recomendada y está activada de manera predeterminada. Acepta los siguientes parámetros:

Parámetro	Descripción	Predeterminado para Amazon RDS	Predeterminado para Aurora
`autovacuum_vacuum_threshold`	El número mínimo de operaciones de actualización o eliminación de tuplas que deben realizarse en una tabla antes	50 operaciones	50 operaciones

	de que autovacuum la succione.		
`autovacuum_analyze_threshold`	El número mínimo de inserciones, actualizaciones o eliminaciones de tuplas que deben producirse en una tabla antes de que autovacuum la analice.	50 operaciones	50 operaciones
`autovacuum_vacuum_scale_factor`	El porcentaje de tuplas que deben modificarse en una tabla antes de que autovacuum la succione.	0.1	0.1
`autovacuum_analyze_scale_factor`	El porcentaje de tuplas que deben modificarse en una tabla antes de que autovacuum la analice.	0.05	0,05
`autovacuum_freeze_max_age`	La edad máxima que debe transcurrir IDs antes de que se vacíe una mesa para evitar problemas relacionados con el identificador de la transacción.	200 000 000 de transacciones	200 000 000 de transacciones

Autovacuum crea una lista de tablas para procesarlas en función de fórmulas de umbral específicas, de la siguiente manera.

- Umbral para ejecutar `VACUUM` en una tabla:

```
vacuum threshold = autovacuum_vacuum_threshold + (autovacuum_vacuum_scale_factor *
 Total row count of table)
```

- Umbral para ejecutar `ANALYZE` en una tabla:

```
analyze threshold = autovacuum_analyze_threshold + (autovacuum_analyze_scale_factor *
 Total row count of table)
```

En el caos de las tablas pequeñas o medianas, los valores predeterminados pueden ser suficientes. Sin embargo, una tabla grande que tenga modificaciones de datos frecuentes tendrá un mayor número de tuplas inactivas. En este caso, es posible que autovacuum procese la tabla con frecuencia para realizar tareas de mantenimiento y que las tareas de mantenimiento de otras tablas se retrasen o se ignoren hasta que se acabe la tabla grande. Para evitar esta situación, puede ajustar los parámetros de autovacuum que se describen en la siguiente sección.

Parámetros de autovacuum relacionados con la memoria

`autovacuum_max_workers`

Especifica el número máximo de procesos de autovacuum (aparte del iniciador de autovacuum) que se pueden ejecutar al mismo tiempo. Este parámetro solo se puede configurar al iniciar el servidor. Si el proceso de autovacuum está ocupado con una tabla grande, este parámetro ayuda a realizar la succión en otras tablas.

`maintenance_work_mem`

Especifica la cantidad máxima de memoria que se utilizará para las operaciones de mantenimiento, como `VACUUM`, `CREATE INDEX` y `ALTER`. En Amazon RDS y Aurora, la memoria se asigna en función de la clase de instancia mediante la fórmula `GREATEST({DBInstanceClassMemory/63963136*1024},65536)`. Cuando se ejecuta autovacuum, se puede asignar hasta `autovacuum_max_workers` veces el valor calculado, así que procure no establecer un valor demasiado alto. Para controlarlo, puede configurar `autovacuum_work_mem` por separado.

`autovacuum_work_mem`

Establece la memoria máxima que puede utilizar cada proceso de trabajo de autovacuum. El valor predeterminado de este parámetro es -1, lo que indica que debe utilizar el valor de `maintenance_work_mem` en su lugar.

Para obtener más información sobre los parámetros de memoria de autovacuum, consulte Asignación de memoria para autovacuum en la documentación de Amazon RDS.

Ajuste de los parámetros de autovacuum

Es posible que los usuarios tengan que ajustar los parámetros de autovacuum en función de sus operaciones de actualización y eliminación. Los parámetros siguientes se pueden configurar por tabla, instancia o clúster.

A nivel de instancia o clúster

Como ejemplo, veamos una base de datos del sector bancario en la que se esperan operaciones continuas con el lenguaje de manipulación de datos (DML). Para mantener el estado de la base de datos, debe ajustar los parámetros de autovacuum a nivel de clúster para Aurora y a nivel de instancia para Amazon RDS, y aplicar también el mismo grupo de parámetros al lector. En caso de que se produzca una conmutación por error, debe aplicar los mismos parámetros al nuevo escritor.

Nivel de tabla

Por ejemplo, en una base de datos relacionada con la entrega de alimentos en la que se esperan operaciones con DML continuas en una sola tabla llamada `orders`, debería valorar si ajustar el parámetro `autovacuum_analyze_threshold` a nivel de tabla mediante el siguiente comando:

```
ALTER TABLE <table_name> SET (autovacuum_analyze_threshold = <threshold rows>)
```

Uso de una configuración de autovacuum agresiva a nivel de tabla

El ejemplo de la tabla `orders` que tiene operaciones de actualización y eliminación continuas se convierte en una buena opción para realizar la succión debido a la configuración de autovacuum predeterminada. Esto provoca que se hagan planes incorrectos y que las consultas sean lentas. Eliminar la sobrecarga y actualizar las estadísticas requiere una configuración de autovacuum agresiva a nivel de tabla.

Para determinar la configuración, lleve un registro de la duración de las consultas que se ejecutan en esta tabla e identifique el porcentaje de operaciones con DML que hacen cambiar los planes. La vista `pg_stat_user_tables` lo ayuda a realizar un seguimiento de las operaciones de inserción, actualización y eliminación.

Ejemplo:

Supongamos que el optimizador hace planes incorrectos cada vez que cambia el 5 % de la tabla `orders`. En este caso, debe cambiar el umbral del factor de escala al 2 %, como se indica a continuación:

```
ALTER TABLE orders SET (autovacuum_vacuum_scale_factor = 0.02)
```

> **Tip**
>
> Seleccione cuidadosamente la configuración de autovacuum agresiva para evitar un consumo elevado de los recursos.

Para obtener más información, consulte los siguientes temas:

- Descripción del autovacuum en entornos Amazon RDS paraAWS PostgreSQL (entrada del blog)
- Automatic Vacuuming (documentación de PostgreSQL)
- Ajuste de los parámetros de PostgreSQL en Amazon RDS y Amazon AWS Aurora (guía prescriptiva)

Para asegurarse de que autovacuum funcione de forma eficaz, supervise las filas inactivas, el uso del disco y la última vez que autovacuum o `ANALYZE` se ejecutó de forma periódica. La vista `pg_stat_all_tables` proporciona información sobre cada tabla (`relname`) y sobre cuántas tuplas inactivas (`n_dead_tup`) hay en la tabla.

Supervisar la cantidad de tuplas inactivas que hay en cada tabla, especialmente en las que se actualizan con frecuencia, permite determinar si los procesos de autovacuum eliminan periódicamente las tuplas inactivas para reutilizar el espacio en disco y mejorar el rendimiento. Puede utilizar la siguiente consulta para comprobar la cantidad de tuplas muertas y la hora de la última ejecución de autovacuum en las tablas:

```
SELECT
relname AS TableName,n_live_tup AS LiveTuples,n_dead_tup AS DeadTuples,
last_autovacuum AS Autovacuum,last_autoanalyze AS Autoanalyze_FROM
pg_stat_user_tables;
```

Ventajas y limitaciones

Autovacuum ofrece las siguientes ventajas:

- Elimina automáticamente la sobrecarga de las tablas.
- Evita el reinicio de los ID de transacción.
- Mantiene actualizadas las estadísticas de la base de datos.

Limitaciones:

- Si las consultas utilizan el procesamiento paralelo, es posible que la cantidad de procesos de trabajo no sea suficiente para el autovacuum.
- Si autovacuum se ejecuta durante las horas pico, es posible que aumente la utilización de los recursos. Debe ajustar los parámetros para solucionar este problema.
- Si las páginas de la tabla están ocupadas en otra sesión, es posible que autovacuum las omita.
- Autovacuum no puede acceder a las tablas temporales.

Succión y análisis manuales de las tablas

Si la base de datos se succiona mediante el proceso de autovacuum, se recomienda evitar realizar succiones manuales en toda la base de datos con demasiada frecuencia. Una aspiradora manual podría provocar I/O cargas innecesarias o picos de CPU, y también podría no eliminar las tuplas muertas. Utilice las aspiradoras manuales table-by-table únicamente si es realmente necesario, por ejemplo, cuando la proporción de tuplas activas y muertas sea baja o cuando haya espacios largos entre las aspiradoras automáticas. Además, debe realizar succiones manuales cuando la actividad del usuario sea mínima.

Autovacuum también mantiene actualizadas las estadísticas de una tabla. Al ejecutar el comando `ANALYZE` manualmente, vuelve a crear estas estadísticas en lugar de actualizarlas. La nueva creación de las estadísticas mediante el proceso habitual de autovacuum cuando ya están actualizadas puede provocar la utilización de los recursos del sistema.

Se recomienda que ejecute los comandos VACUUM y ANALYZE manualmente en las siguientes situaciones:

- Durante las horas de menor actividad y en las tablas más utilizadas, es posible que autovacuum no sea suficiente.
- Inmediatamente después de cargar datos de forma masiva en la tabla de destino. En este caso, si ejecuta `ANALYZE` manualmente, las estadísticas se vuelven a crear completamente, lo cual es una mejor opción que esperar a que autovacuum comience.
- Para succionar tablas temporales (autovacuum no puede acceder a ellas).

Para reducir el I/O impacto de la ejecución de los `ANALYZE` comandos `VACUUM` y en la actividad simultánea de la base de datos, puede utilizar el parámetro. `vacuum_cost_delay` En muchas situaciones, los comandos de mantenimiento como `VACUUM` y `ANALYZE` no tienen por qué finalizarse rápidamente. Sin embargo, estos comandos no deberían interferir con la capacidad del sistema para realizar otras operaciones en la base de datos. Para evitarlo, puede activar los retardos de succión basados en los costos mediante el parámetro `vacuum_cost_delay`. Este parámetro está desactivado de forma predeterminada para los comandos `VACUUM` que se ejecutan manualmente. Para activarlo, configúrelo en un valor distinto de cero.

Ejecución de operaciones de succión y limpieza en paralelo

La opción PARALLEL del comando VACUUM utiliza trabajos paralelos para las fases de succión y limpieza de índices y está desactivada de forma predeterminada. El número de trabajos paralelos (el grado de paralelismo) viene determinado por el número de índices de la tabla, y el usuario puede especificar este número. Si ejecuta operaciones VACUUM paralelas sin un argumento entero, el grado de paralelismo se calcula en función del número de índices de la tabla.

Los siguientes parámetros lo ayudan a configurar la succión paralela en Amazon RDS para PostgreSQL y Aurora compatible con PostgreSQL:

- max_worker_processes establece el número máximo de procesos de trabajo que se ejecutan simultáneamente.
- min_parallel_index_scan_size establece la cantidad mínima de datos de índice que se deben escanear para poder considerar la posibilidad de realizar un escaneo paralelo.
- max_parallel_maintenance_workers establece el número máximo de trabajos paralelos que se pueden iniciar con un único comando de utilidad.

> **Note**
>
> La opción PARALLEL se utiliza únicamente para la succión. No afecta al comando ANALYZE.

En el siguiente ejemplo se ilustra el comportamiento de la base de datos cuando se usa VACUUM y ANALYZE manualmente en una base de datos.

Esta es una tabla de ejemplo en la que se ha desactivado autovacuum (solo con fines ilustrativos; no se recomienda desactivarlo):

```
create table t1 ( a int, b int, c int );
alter table t1 set (autovacuum_enabled=false);
```

```
apgl=> \d+ t1
Table "public.t1"
Column | Type | Collation | Nullable | Default | Storage | Stats target | Description
--------+---------+-----------+----------+---------+---------+--------------
+-------------
```

```
a | integer | | | | plain | |
b | integer | | | | plain | |
c | integer | | | | plain | |
Access method: heap
Options: autovacuum_enabled=false
```

Agregue 1 millón de filas a la tabla t1:

```
apgl=> select count(*) from t1;
count
1000000
(1 row)
```

Estadísticas de la tabla t1:

```
select * from pg_stat_all_tables where relname='t1';
-[ RECORD 1 ]-------+--------
relid         | 914744
schemaname    | public
relname       | t1
seq_scan      | 0
seq_tup_read  | 0
idx_scan      |
idx_tup_fetch |
n_tup_ins     | 1000000
n_tup_upd     | 0
n_tup_del     | 0
n_tup_hot_upd | 0
n_live_tup    | 1000000
n_dead_tup    | 0
n_mod_since_analyze | 1000000
last_vacuum      |
last_autovacuum  |
last_analyze     |
last_autoanalyze |
vacuum_count     | 0
autovacuum_count | 0
analyze_count    | 0
autoanalyze_count | 0
```

Agregue un índice:

```
create index i2 on t1 (b,a);
```

Ejecute el comando EXPLAIN (plan 1):

```
Bitmap Heap Scan on t1 (cost=10521.17..14072.67 rows=5000 width=4)
Recheck Cond: (a = 5)
# Bitmap Index Scan on i2 (cost=0.00..10519.92 rows=5000 width=0)
Index Cond: (a = 5)
(4 rows)
```

Ejecute el comando EXPLAIN ANALYZE (plan 2):

```
explain (analyze,buffers,costs off) select a from t1 where b = 5;
QUERY PLAN
Bitmap Heap Scan on t1 (actual time=0.023..0.024 rows=1 loops=1)
Recheck Cond: (b = 5)
Heap Blocks: exact=1
Buffers: shared hit=4
# Bitmap Index Scan on i2 (actual time=0.016..0.016 rows=1 loops=1)
Index Cond: (b = 5)
Buffers: shared hit=3
Planning Time: 0.054 ms
Execution Time: 0.076 ms
(9 rows)
```

Los comandos EXPLAIN y EXPLAIN ANALYZE muestran planes diferentes, ya que autovacuum estaba desactivado en la tabla y el comando ANALYZE no se ejecutó manualmente. Ahora vamos a actualizar un valor de la tabla y volveremos a crear el plan EXPLAIN ANALYZE:

```
update t1 set a=8 where b=5;
explain (analyze,buffers,costs off) select a from t1 where b = 5;
```

El comando EXPLAIN ANALYZE (plan 3) ahora muestra:

```
apgl=> explain (analyze,buffers,costs off) select a from t1 where b = 5;
QUERY PLAN
Bitmap Heap Scan on t1 (actual time=0.075..0.076 rows=1 loops=1)
Recheck Cond: (b = 5)
Heap Blocks: exact=1
```

```
Buffers: shared hit=5
# Bitmap Index Scan on i2 (actual time=0.017..0.017 rows=2 loops=1)
Index Cond: (b = 5)
Buffers: shared hit=3
Planning Time: 0.053 ms
Execution Time: 0.125 ms
```

Si compara los costos entre el plan 2 y el plan 3, verá las diferencias en el tiempo de planificación y ejecución, ya que aún no hemos recopilado estadísticas.

Ahora ejecutemos ANALYZE manualmente en la tabla y, a continuación, comprobemos las estadísticas y volvamos a crear el plan:

```
apgl=> analyze t1
apgl# ;
ANALYZE
Time: 212.223 ms

apgl=> select * from pg_stat_all_tables where relname='t1';
-[ RECORD 1 ]-------+------------------------------
relid          | 914744
schemaname     | public
relname        | t1
seq_scan       | 3
seq_tup_read   | 1000000
idx_scan       | 3
idx_tup_fetch  | 3
n_tup_ins      | 1000000
n_tup_upd      | 1
n_tup_del      | 0
n_tup_hot_upd  | 0
n_live_tup     | 1000000
n_dead_tup     | 1
n_mod_since_analyze | 0
last_vacuum       |
last_autovacuum   |
last_analyze      | 2023-04-15 11:39:02.075089+00
last_autoanalyze  |
vacuum_count      | 0
autovacuum_count  | 0
analyze_count     | 1
autoanalyze_count | 0
```

```
Time: 148.347 ms
```

Ejecute el comando `EXPLAIN ANALYZE` (plan 4):

```
apgl=> explain (analyze,buffers,costs off) select a from t1 where b = 5;
QUERY PLAN
Index Only Scan using i2 on t1 (actual time=0.022..0.023 rows=1 loops=1)
Index Cond: (b = 5)
Heap Fetches: 1
Buffers: shared hit=4
Planning Time: 0.056 ms
Execution Time: 0.068 ms
(6 rows)

Time: 138.462 ms
```

Si compara todos los resultados del plan después de analizar manualmente la tabla y recopilar estadísticas, observará que el plan 4 del optimizador es mejor que los demás y, además, reduce el tiempo de ejecución de las consultas. En este ejemplo se muestra la importancia de realizar actividades de mantenimiento en la base de datos.

Reescritura de una tabla completa con VACUUM FULL

Al ejecutar el comando `VACUUM` con el parámetro `FULL`, se reescribe todo el contenido de una tabla en un nuevo archivo de disco sin espacio adicional y el espacio no utilizado se devuelve al sistema operativo. Esta operación es mucho más lenta y requiere un bloqueo `ACCESS EXCLUSIVE` en cada tabla. También requiere espacio adicional en el disco, ya que escribe una nueva copia de la tabla y no libera la copia anterior hasta que se completa la operación.

`VACUUM FULL` puede ser de utilidad en los siguientes casos:

- Cuando desee recuperar una cantidad importante de espacio de las tablas.
- Cuando desee recuperar el espacio de sobrecarga en las tablas que no contienen claves principales.

Si la base de datos puede tolerar el tiempo de inactividad, le recomendamos que utilice `VACUUM FULL` cuando tenga tablas sin claves principales.

Como `VACUUM FULL` requiere más bloqueos que otras operaciones, es más caro ejecutarlo en las bases de datos cruciales. Para reemplazar este método, puede usar la extensión `pg_repack`, que se describe en la siguiente sección. Esta opción es similar a `VACUUM FULL`, pero requiere un bloqueo mínimo, y es compatible tanto con Amazon RDS para PostgreSQL como con Aurora compatible con PostgreSQL.

Eliminación de la sobrecarga con pg_repack

Puede usar la extensión `pg_repack` para eliminar la sobrecarga de las tablas y los índices con un bloqueo mínimo de la base de datos. Puede crear esta extensión en la instancia de base de datos y ejecutar el cliente `pg_repack` (si la versión del cliente y la versión de la extensión coinciden) desde Amazon Elastic Compute Cloud (Amazon EC2) o desde un ordenador que se pueda conectar a la base de datos.

A diferencia de `VACUUM FULL`, `pg_repack` no necesita tiempo de inactividad ni un periodo de mantenimiento y no bloqueará otras sesiones.

`pg_repack` es útil en las situaciones en que es posible que `VACUUM FULL`, `CLUSTER` o `REINDEX` no funcionen. Crea una tabla nueva que contiene los datos de la tabla sobrecargada, realiza un seguimiento de los cambios con respecto a la tabla original y, a continuación, reemplaza la tabla original por la nueva. No bloquea la tabla original para operaciones de lectura o escritura mientras crea la nueva tabla.

Puede utilizar `pg_repack` para una tabla completa o para un índice. Para ver una lista de tareas, consulte la documentación de pg_repack.

Limitaciones:

- Para ejecutar `pg_repack`, la tabla debe tener una clave principal o un índice único.
- `pg_repack` no funcionará con las tablas temporales.
- `pg_repack` no funcionará en las tablas que tengan índices globales.
- Mientras `pg_repack` esté en curso, no podrá realizar operaciones de DDL en las tablas.

En la siguiente tabla se describen las diferencias entre `pg_repack` y `VACUUM FULL`.

`VACUUM FULL`	`pg_repack`
Comando integrado	Una extensión que ejecuta desde Amazon EC2 o desde el ordenador local
Requiere un bloqueo `ACCESS EXCLUSIVE` mientras se utiliza en una tabla	Requiere un bloqueo `ACCESS EXCLUSIVE` solo durante un breve periodo de tiempo

Funciona con todas las tablas	Solo funciona en las tablas que tienen claves principales y únicas
Requiere el doble del espacio de almacenamiento que consumen la tabla y los índices	Requiere el doble del espacio de almacenamiento que consumen la tabla y los índices

Para ejecutar `pg_repack` en una tabla, use el siguiente comando:

```
pg_repack -h <host> -d <dbname> --table <tablename> -k
```

Para ejecutar `pg_repack` en un índice, use el siguiente comando:

```
pg_repack -h <host> -d <dbname> --index <index name>
```

Para obtener más información, consulte la entrada en el blog de AWS Remove bloat from Amazon Aurora and RDS for PostgreSQL with pg_repack.

Advertencia

El error `error-on-invalid-index` de `pg_repack` suele indicar que uno o varios índices de la tabla están dañados o no son válidos. `pg_repack` no puede funcionar de forma segura en las tablas que no tienen índices válidos porque depende de los índices para mantener la coherencia de datos durante el proceso de reempaquetado.
Este error se produce en las siguientes situaciones:

- El índice está marcado como no válido (por ejemplo, debido a un error en una instrucción `CREATE INDEX CONCURRENTLY`).
- El índice está dañado (posiblemente debido a problemas de hardware o a cierres bruscos).

Utilice la siguiente consulta para identificar los índices no válidos y suprimirlos primero si los encuentra.

```
SELECT indexrelid::regclass, indisvalid FROM pg_index WHERE indrelid =
 'orders'::regclass AND NOT indisvalid; Drop the invalid index: DROP INDEX
 index_name;
```

Nueva creación de índices

El comando REINDEX de PostgreSQL vuelve a crear un índice utilizando los datos que hay almacenados en la tabla del índice y reemplazando la copia antigua del índice. Le recomendamos utilizar `REINDEX` en las siguientes situaciones:

- Cuando un índice se dañe y deje de contener datos válidos. Esto puede ocurrir a consecuencia de errores de software o hardware.
- Cuando las consultas que anteriormente utilizaban el índice dejen de usarlo.
- Cuando el índice se sobrecargue con un gran número de páginas vacías o casi vacías. Debe ejecutar `REINDEX` cuando el porcentaje de sobrecarga (`bloat_pct`) sea superior a 20.

La siguiente consulta lo ayuda a encontrar `bloat_pct`:

```
SELECT current_database(), nspname AS schemaname, tblname, idxname,
 bs*(relpages)::bigint AS real_size,
  bs*(relpages-est_pages)::bigint AS extra_size,
  100 * (relpages-est_pages)::float / relpages AS extra_pct,
  fillfactor,
  CASE WHEN relpages > est_pages_ff
    THEN bs*(relpages-est_pages_ff)
    ELSE 0
  END AS bloat_size,
  100 * (relpages-est_pages_ff)::float / relpages AS bloat_pct,
  is_na
  -- , 100-(pst).avg_leaf_density AS pst_avg_bloat, est_pages, index_tuple_hdr_bm,
 maxalign, pagehdr, nulldatawidth, nulldatahdrwidth, reltuples, relpages -- (DEBUG
 INFO)
FROM (
  SELECT coalesce(1 +
         ceil(reltuples/floor((bs-pageopqdata-pagehdr)/(4+nulldatahdrwidth)::float)), 0
 -- ItemIdData size + computed avg size of a tuple (nulldatahdrwidth)
      ) AS est_pages,
      coalesce(1 +
         ceil(reltuples/floor((bs-pageopqdata-pagehdr)*fillfactor/
(100*(4+nulldatahdrwidth)::float))), 0
      ) AS est_pages_ff,
      bs, nspname, tblname, idxname, relpages, fillfactor, is_na
```

```
     -- , pgstatindex(idxoid) AS pst, index_tuple_hdr_bm, maxalign, pagehdr,
 nulldatawidth, nulldatahdrwidth, reltuples -- (DEBUG INFO)
  FROM (
     SELECT maxalign, bs, nspname, tblname, idxname, reltuples, relpages, idxoid,
 fillfactor,
           ( index_tuple_hdr_bm +
               maxalign - CASE -- Add padding to the index tuple header to align on
 MAXALIGN
                 WHEN index_tuple_hdr_bm%maxalign = 0 THEN maxalign
                 ELSE index_tuple_hdr_bm%maxalign
               END
             + nulldatawidth + maxalign - CASE -- Add padding to the data to align on
 MAXALIGN
                 WHEN nulldatawidth = 0 THEN 0
                 WHEN nulldatawidth::integer%maxalign = 0 THEN maxalign
                 ELSE nulldatawidth::integer%maxalign
               END
           )::numeric AS nulldatahdrwidth, pagehdr, pageopqdata, is_na
           -- , index_tuple_hdr_bm, nulldatawidth -- (DEBUG INFO)
     FROM (
         SELECT n.nspname, i.tblname, i.idxname, i.reltuples, i.relpages,
             i.idxoid, i.fillfactor, current_setting('block_size')::numeric AS bs,
             CASE -- MAXALIGN: 4 on 32bits, 8 on 64bits (and mingw32 ?)
               WHEN version() ~ 'mingw32' OR version() ~ '64-bit|x86_64|ppc64|ia64|
amd64' THEN 8
               ELSE 4
             END AS maxalign,
             /* per page header, fixed size: 20 for 7.X, 24 for others */
             24 AS pagehdr,
             /* per page btree opaque data */
             16 AS pageopqdata,
             /* per tuple header: add IndexAttributeBitMapData if some cols are null-
able */
             CASE WHEN max(coalesce(s.null_frac,0)) = 0
                 THEN 8 -- IndexTupleData size
                 ELSE 8 + (( 32 + 8 - 1 ) / 8) -- IndexTupleData size +
 IndexAttributeBitMapData size ( max num filed per index + 8 - 1 /8)
             END AS index_tuple_hdr_bm,
             /* data len: we remove null values save space using it fractionnal part
 from stats */
             sum( (1-coalesce(s.null_frac, 0)) * coalesce(s.avg_width, 1024)) AS
 nulldatawidth,
```

```
            max( CASE WHEN i.atttypid = 'pg_catalog.name'::regtype THEN 1 ELSE 0
END ) > 0 AS is_na
         FROM (
            SELECT ct.relname AS tblname, ct.relnamespace, ic.idxname, ic.attpos,
ic.indkey, ic.indkey[ic.attpos], ic.reltuples, ic.relpages, ic.tbloid, ic.idxoid,
ic.fillfactor,
                coalesce(a1.attnum, a2.attnum) AS attnum, coalesce(a1.attname,
a2.attname) AS attname, coalesce(a1.atttypid, a2.atttypid) AS atttypid,
                CASE WHEN a1.attnum IS NULL
                THEN ic.idxname
                ELSE ct.relname
                END AS attrelname
            FROM (
                SELECT idxname, reltuples, relpages, tbloid, idxoid, fillfactor,
indkey,
                    pg_catalog.generate_series(1,indnatts) AS attpos
                FROM (
                    SELECT ci.relname AS idxname, ci.reltuples, ci.relpages,
i.indrelid AS tbloid,
                        i.indexrelid AS idxoid,
                        coalesce(substring(
                            array_to_string(ci.reloptions, ' ')
                            from 'fillfactor=([0-9]+)')::smallint, 90) AS fillfactor,
                        i.indnatts,
                        pg_catalog.string_to_array(pg_catalog.textin(
                            pg_catalog.int2vectorout(i.indkey)),' ')::int[] AS indkey
                    FROM pg_catalog.pg_index i
                    JOIN pg_catalog.pg_class ci ON ci.oid = i.indexrelid
                    WHERE ci.relam=(SELECT oid FROM pg_am WHERE amname = 'btree')
                    AND ci.relpages > 0
                ) AS idx_data
            ) AS ic
            JOIN pg_catalog.pg_class ct ON ct.oid = ic.tbloid
            LEFT JOIN pg_catalog.pg_attribute a1 ON
                ic.indkey[ic.attpos] <> 0
                AND a1.attrelid = ic.tbloid
                AND a1.attnum = ic.indkey[ic.attpos]
            LEFT JOIN pg_catalog.pg_attribute a2 ON
                ic.indkey[ic.attpos] = 0
                AND a2.attrelid = ic.idxoid
                AND a2.attnum = ic.attpos
        ) i
        JOIN pg_catalog.pg_namespace n ON n.oid = i.relnamespace
```

```
            JOIN pg_catalog.pg_stats s ON s.schemaname = n.nspname
                                      AND s.tablename = i.attrelname
                                      AND s.attname = i.attname
            GROUP BY 1,2,3,4,5,6,7,8,9,10,11
      ) AS rows_data_stats
  ) AS rows_hdr_pdg_stats
) AS relation_stats
ORDER BY nspname, tblname, idxname;
```

Las páginas de índice que están completamente vacías se recuperan para su reutilización. Sin embargo, recomendamos realizar una nueva indexación periódicamente si se han eliminado las claves de índice de una página pero se conserva el espacio asignado.

Volver a crear el índice ayuda a mejorar el rendimiento de las consultas. Puede volver a crear un índice de tres maneras, tal y como se describe en la siguiente tabla.

Método	Descripción	Limitaciones
`CREATE INDEX` y `DROP INDEX` con la opción `CONCURRENTLY`	Crea un índice nuevo y elimina el antiguo. El optimizador genera planes utilizando el índice recién creado en lugar del índice antiguo. Durante las horas de menor actividad, puede eliminar el índice antiguo.	La creación del índice lleva más tiempo cuando se utiliza la opción `CONCURRENTLY`, ya que tiene que hacer un seguimiento de todos los cambios entrantes. Cuando los cambios se congelan, el proceso se marca como completo.
`REINDEX` con la opción `CONCURRENTLY`	Bloquea las operaciones de escritura durante el nuevo proceso de creación. La versión 12 y las versiones posteriores de PostgreSQL ofrecen la opción `CONCURRENTLY`, lo que evita estos bloqueos.	El uso de `CONCURRENTLY` requiere más tiempo para volver a crear el índice.

Extensión `pg_repack`	Limpia la sobrecarga de una tabla y vuelve a crear el índice.	Debe ejecutar esta extensión desde una instancia de EC2 o desde una computadora local que esté conectada a la base de datos.

Creación de un nuevo índice

Cuando los comandos `DROP INDEX` y `CREATE INDEX` se utilizan juntos, vuelven a crear un índice:

```
DROP INDEX <index_name>
CREATE INDEX <index_name> ON TABLE <table_name> (<column1>[,<column2>])
```

La desventaja de este enfoque es el bloqueo de exclusividad que se aplica a la tabla, lo que afecta al rendimiento durante esta actividad. El comando `DROP INDEX` adquiere un bloqueo de exclusividad que bloquea las operaciones de lectura y de escritura en la tabla. El comando `CREATE INDEX` bloquea las operaciones de escritura en la tabla. Permite operaciones de lectura, pero tienen un costo elevado durante la creación del índice.

Nueva creación de un índice

El comando `REINDEX` lo ayuda a mantener un rendimiento uniforme de la base de datos. Al realizar un gran número de operaciones con DML en una tabla, se produce una sobrecarga tanto de la tabla como de los índices. Los índices se utilizan para acelerar las búsquedas en las tablas y mejorar el rendimiento de las consultas. La sobrecarga de los índices afecta al rendimiento de las búsquedas y de las consultas. Por lo tanto, es recomendable volver a indexar las tablas que tienen un gran volumen de operaciones con DML para mantener la coherencia en el rendimiento de las consultas.

El comando `REINDEX` bloquea las operaciones de escritura en la tabla subyacente para volver a crear el índice desde cero, pero permite las operaciones de lectura en la tabla. Sin embargo, sí bloquea las operaciones de lectura en el índice. Las consultas que utilizan el índice correspondiente están bloqueadas, pero las demás no.

La versión 12 de PostgreSQL introdujo un nuevo parámetro opcional, `CONCURRENTLY`, que vuelve a crear el índice desde cero, pero no bloquea las operaciones de lectura ni escritura en la tabla ni en

las consultas que utilizan el índice. Sin embargo, se tarda más en completar el proceso cuando se usa esta opción.

Ejemplos

Creación y eliminación de un índice

Cree un índice nuevo con la opción `CONCURRENTLY`:

```
create index CONCURRENTLY on table(columns) ;
```

Elimine el índice antiguo con la opción `CONCURRENTLY`:

```
drop index CONCURRENTLY <index name> ;
```

Nueva creación de un índice

Para volver a crear un índice único:

```
reindex index <index name> ;
```

Para volver a crear todos los índices de una tabla:

```
reindex table <table name> ;
```

Para volver a crear todos los índices de un esquema:

```
reindex schema <schema name> ;
```

Nueva creación de un índice de forma simultánea

Para volver a crear un índice único:

```
reindex index CONCURRENTLY <indexname> ;
```

Para volver a crear todos los índices de una tabla:

```
reindex table CONCURRENTLY <tablename> ;
```

Para volver a crear todos los índices de un esquema:

```
reindex schema CONCURRENTLY <schemaname> ;
```

Nueva creación o reubicación únicamente de índices

Para volver a crear un índice único:

```
pg_repack -h <hostname> -d <dbname> -i <indexname> -k
```

Para volver a crear todos los índices:

```
pg_repack -h <hostname> -d <dbname> -x <indexname> -t <tablename> -k
```

Ejemplo: Recuperación de espacio mediante autovacuum y VACUUM FULL

Como ejemplo, vamos a crear una tabla emp con 500 000 filas y, a continuación, vamos a actualizar las filas con nuevos valores. Autovacuum está activado, por lo que ejecutará los comandos VACUUM y ANALYZE de esta tabla para eliminar la sobrecarga y recuperar espacio. El espacio recuperado se puede reutilizar, pero no se devolverá al sistema operativo.

La siguiente consulta determina la sobrecarga de la tabla:

```
-- WARNING: When run with a non-superuser role, the query inspects only indexes on
 tables you are granted to read.
-- WARNING: Rows with is_na = 't' are known to have bad statistics ("name" type is not
 supported).
-- This query is compatible with PostgreSQL 8.2 and later.
SELECT current_database(), nspname AS schemaname, tblname, idxname,
 bs*(relpages)::bigint AS real_size,
  bs*(relpages-est_pages)::bigint AS extra_size,
  100 * (relpages-est_pages)::float / relpages AS extra_pct,
  fillfactor,
  CASE WHEN relpages > est_pages_ff
    THEN bs*(relpages-est_pages_ff)
    ELSE 0
  END AS bloat_size,
  100 * (relpages-est_pages_ff)::float / relpages AS bloat_pct,
  is_na
  -- , 100-(pst).avg_leaf_density AS pst_avg_bloat, est_pages, index_tuple_hdr_bm,
 maxalign, pagehdr, nulldatawidth, nulldatahdrwidth, reltuples, relpages -- (DEBUG
 INFO)
FROM (
  SELECT coalesce(1 +
         ceil(reltuples/floor((bs-pageopqdata-pagehdr)/(4+nulldatahdrwidth)::float)), 0
 -- ItemIdData size + computed avg size of a tuple (nulldatahdrwidth)
      ) AS est_pages,
      coalesce(1 +
         ceil(reltuples/floor((bs-pageopqdata-pagehdr)*fillfactor/
(100*(4+nulldatahdrwidth)::float))), 0
      ) AS est_pages_ff,
      bs, nspname, tblname, idxname, relpages, fillfactor, is_na
```

```
     -- , pgstatindex(idxoid) AS pst, index_tuple_hdr_bm, maxalign, pagehdr,
 nulldatawidth, nulldatahdrwidth, reltuples -- (DEBUG INFO)
  FROM (
     SELECT maxalign, bs, nspname, tblname, idxname, reltuples, relpages, idxoid,
 fillfactor,
           ( index_tuple_hdr_bm +
               maxalign - CASE -- Add padding to the index tuple header to align on
 MAXALIGN
                 WHEN index_tuple_hdr_bm%maxalign = 0 THEN maxalign
                 ELSE index_tuple_hdr_bm%maxalign
               END
             + nulldatawidth + maxalign - CASE -- Add padding to the data to align on
 MAXALIGN
                 WHEN nulldatawidth = 0 THEN 0
                 WHEN nulldatawidth::integer%maxalign = 0 THEN maxalign
                 ELSE nulldatawidth::integer%maxalign
               END
           )::numeric AS nulldatahdrwidth, pagehdr, pageopqdata, is_na
           -- , index_tuple_hdr_bm, nulldatawidth -- (DEBUG INFO)
     FROM (
         SELECT n.nspname, i.tblname, i.idxname, i.reltuples, i.relpages,
             i.idxoid, i.fillfactor, current_setting('block_size')::numeric AS bs,
             CASE -- MAXALIGN: 4 on 32bits, 8 on 64bits (and mingw32 ?)
               WHEN version() ~ 'mingw32' OR version() ~ '64-bit|x86_64|ppc64|ia64|
amd64' THEN 8
               ELSE 4
             END AS maxalign,
             /* per page header, fixed size: 20 for 7.X, 24 for others */
             24 AS pagehdr,
             /* per page btree opaque data */
             16 AS pageopqdata,
             /* per tuple header: add IndexAttributeBitMapData if some cols are null-
able */
             CASE WHEN max(coalesce(s.null_frac,0)) = 0
                 THEN 8 -- IndexTupleData size
                 ELSE 8 + (( 32 + 8 - 1 ) / 8) -- IndexTupleData size +
 IndexAttributeBitMapData size ( max num filed per index + 8 - 1 /8)
             END AS index_tuple_hdr_bm,
             /* data len: we remove null values save space using it fractionnal part
 from stats */
             sum( (1-coalesce(s.null_frac, 0)) * coalesce(s.avg_width, 1024)) AS
 nulldatawidth,
```

```
            max( CASE WHEN i.atttypid = 'pg_catalog.name'::regtype THEN 1 ELSE 0
END ) > 0 AS is_na
         FROM (
            SELECT ct.relname AS tblname, ct.relnamespace, ic.idxname, ic.attpos,
ic.indkey, ic.indkey[ic.attpos], ic.reltuples, ic.relpages, ic.tbloid, ic.idxoid,
ic.fillfactor,
                coalesce(a1.attnum, a2.attnum) AS attnum, coalesce(a1.attname,
a2.attname) AS attname, coalesce(a1.atttypid, a2.atttypid) AS atttypid,
                CASE WHEN a1.attnum IS NULL
                THEN ic.idxname
                ELSE ct.relname
                END AS attrelname
            FROM (
                SELECT idxname, reltuples, relpages, tbloid, idxoid, fillfactor,
indkey,
                    pg_catalog.generate_series(1,indnatts) AS attpos
                FROM (
                    SELECT ci.relname AS idxname, ci.reltuples, ci.relpages,
i.indrelid AS tbloid,
                        i.indexrelid AS idxoid,
                        coalesce(substring(
                            array_to_string(ci.reloptions, ' ')
                            from 'fillfactor=([0-9]+)')::smallint, 90) AS fillfactor,
                        i.indnatts,
                        pg_catalog.string_to_array(pg_catalog.textin(
                            pg_catalog.int2vectorout(i.indkey)),' ')::int[] AS indkey
                    FROM pg_catalog.pg_index i
                    JOIN pg_catalog.pg_class ci ON ci.oid = i.indexrelid
                    WHERE ci.relam=(SELECT oid FROM pg_am WHERE amname = 'btree')
                    AND ci.relpages > 0
                ) AS idx_data
            ) AS ic
            JOIN pg_catalog.pg_class ct ON ct.oid = ic.tbloid
            LEFT JOIN pg_catalog.pg_attribute a1 ON
                ic.indkey[ic.attpos] <> 0
                AND a1.attrelid = ic.tbloid
                AND a1.attnum = ic.indkey[ic.attpos]
            LEFT JOIN pg_catalog.pg_attribute a2 ON
                ic.indkey[ic.attpos] = 0
                AND a2.attrelid = ic.idxoid
                AND a2.attnum = ic.attpos
        ) i
        JOIN pg_catalog.pg_namespace n ON n.oid = i.relnamespace
```

```
            JOIN pg_catalog.pg_stats s ON s.schemaname = n.nspname
                                      AND s.tablename = i.attrelname
                                      AND s.attname = i.attname
            GROUP BY 1,2,3,4,5,6,7,8,9,10,11
      ) AS rows_data_stats
  ) AS rows_hdr_pdg_stats
) AS relation_stats
ORDER BY nspname, tblname, idxname;
```

Los resultados de la consulta muestran que la tabla tiene una sobrecarga de alrededor del 51 %:

```
current_database | schemaname | tblname | real_size | extra_size | extra_pct |
 fillfactor | bloat_size | bloat_pct | is_na
------------------+------------+---------+-----------+------------+------------------
+------------+------------+-------------------+-------
apgl | public | emp | 60383232 | 30744576 | 50.91575091575091 | 100 | 30744576 |
 50.91575091575091 | f
```

A continuación, se muestran las estadísticas de la vista `pg_stat_all_tables`:

```
relid           | 914748
schemaname      | public
relname         | emp
seq_scan        | 5
seq_tup_read    | 1500000
idx_scan        | 0
idx_tup_fetch   | 0
n_tup_ins       | 600000
n_tup_upd       | 500000
n_tup_del       | 0
n_tup_hot_upd   | 0
n_live_tup      | 500000
n_dead_tup      | 0
n_mod_since_analyze | 0
last_vacuum         |
last_autovacuum     | 2023-04-15 11:59:54.957449+00
last_analyze        |
last_autoanalyze    | 2023-04-15 11:59:55.016352+00
vacuum_count        | 0
autovacuum_count    | 2
analyze_count       | 0
```

```
autoanalyze_count   | 3
```

Observe que autovacuum actualizó las columnas `last_autovacuum` y `last_autoanalyze` después de ejecutarse.

Ahora, insertemos algunas filas en la tabla y comprobemos `extra_size(bloat_size)`, ya que el espacio vacío también se considera sobrecarga.

```
apgl=> select count(*) from emp;
count | 900000

current_database | schemaname | tblname | real_size | extra_size | extra_pct |
 fillfactor | bloat_size | bloat_pct | is_na
------------------+------------+---------+-----------+------------+------------------
+------------+------------+--------------------+-------
apgl | public | emp | 61349888 | 327680 | 0.5341167044999332 | 100 | 327680 |
 0.5341167044999332 | f
(1 row)
```

La columna `bloat_pct` de la salida indica que hay nuevas inserciones que han ocupado el espacio limpio. Ejecutemos `VACUUM FULL`:

```
apgl=> vacuum full emp ;
VACUUM

current_database | schemaname | tblname | real_size | extra_size | extra_pct |
 fillfactor | bloat_size | bloat_pct | is_na
------------------+------------+---------+-----------+------------+-----------
+------------+------------+-----------+-------
apgl | public | emp | 60792832 | -229376 | 0 | 100 | 0 | 0 | f
(1 row)
```

En esta salida, puede ver que se han eliminado el espacio vacío y la sobrecarga y que el espacio se ha devuelto al sistema operativo.

> **Note**
>
> En lugar de `VACUUM FULL`, puede ejecutar `pg_repack` para obtener los mismos resultados.

Recursos

- Descripción del autovacuum en entornos Amazon RDS paraAWS PostgreSQL (entrada del blog)
- Automatic Vacuuming (documentación de PostgreSQL)
- Asignación de memoria para autovacuum (documentación de Amazon RDS)
- Prevención de errores envolventes en los identificadores de transacciones; (documentación de PostgreSQL)
- Remove bloat from Amazon Aurora and RDS for PostgreSQL with pg_repack (entrada en el blog de AWS)

Historial de documentos

En la siguiente tabla, se describen cambios significativos de esta guía. Si quiere recibir notificaciones de futuras actualizaciones, puede suscribirse a las notificaciones RSS.

Cambio	Descripción	Fecha
Actualizaciones	Se corrigieron errores y se agregó información a las secciones Succión y análisis automáticos de las tablas y Eliminación de la sobrecarga con pg_repack.	22 de agosto de 2025
Se corrigió la sintaxis de `reindex`.	En la sección dedicada a la nueva creación simultánea de un índice, se corrigieron los ejemplos de `reindex`.	30 de junio de 2025
Publicación inicial	—	22 de diciembre de 2023

Las traducciones son generadas a través de traducción automática. En caso de conflicto entre la traducción y la version original de inglés, prevalecerá la version en inglés.