Understanding Bloat in PostgreSQL and How to Manage It

In this article, we'll explore two essential SQL queries for inspecting bloat and discuss how to use the **pg_repack** extension for managing it.

Inspecting Bloat with SQL Queries

Query for Index Bloat

The first query focuses on identifying bloat in indexes. Indexes can become bloated due to frequent insertions, updates, and deletions. Here's a breakdown of the query:

```
-- This query inspects bloat in indexes.
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
FROM (
  SELECT coalesce(1 + ceil(reltuples/floor((bs-pageopqdata-
pagehdr)/(4+nulldatahdrwidth)::float)), 0) 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
  FROM (
    SELECT maxalign, bs, nspname, tblname, idxname, reltuples, relpages, idxoid,
fillfactor,
           (index_tuple_hdr_bm + maxalign - CASE WHEN index_tuple_hdr_bm%maxalign = 0 THEN
maxalign ELSE index_tuple_hdr_bm%maxalign END
            + nulldatawidth + maxalign - CASE WHEN nulldatawidth = 0 THEN 0 WHEN
nulldatawidth::integer%maxalign = 0 THEN maxalign ELSE nulldatawidth::integer%maxalign END
           )::numeric AS nulldatahdrwidth, pagehdr, pageopqdata, is_na
    FROM (
      SELECT n.nspname, i.tblname, i.idxname, i.reltuples, i.relpages, i.idxoid,
i.fillfactor,
             current_setting('block_size')::numeric AS bs,
             CASE WHEN version() ~ 'mingw32' OR version() ~ '64-
bit|x86_64|ppc64|ia64|amd64' THEN 8 ELSE 4 END AS maxalign,
             24 AS pagehdr,
             16 AS pageopqdata,
             CASE WHEN max(coalesce(s.null_frac,0)) = 0 THEN 8 ELSE 8 + ((32 + 8 - 1) / 8)
END AS index_tuple_hdr_bm,
             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;
```

This query calculates the real size, extra size, and bloat percentage for indexes. Key metrics include:

- **real_size**: Actual size of the index.
- **extra_size**: Extra space due to bloat.
- **extra_pct**: Percentage of bloat.
- **bloat_size**: Size of the bloat.
- **bloat_pct**: Percentage of bloat relative to the total size.

Query for Table Bloat

The second query focuses on identifying bloat in tables. Tables can become bloated due to the same reasons as indexes — frequent updates, deletions, and insertions. Here’s the query:

```
-- This query inspects bloat in tables.
SELECT current_database(), schemaname, tblname, bs*tblpages AS real_size,
  (tblpages-est_tblpages)*bs AS extra_size,
  CASE WHEN tblpages > 0 AND tblpages - est_tblpages > 0
    THEN 100 * (tblpages - est_tblpages)/tblpages::float
    ELSE 0
  END AS extra_pct, fillfactor,
  CASE WHEN tblpages - est_tblpages_ff > 0
    THEN (tblpages-est_tblpages_ff)*bs
    ELSE 0
  END AS bloat_size,
  CASE WHEN tblpages > 0 AND tblpages - est_tblpages_ff > 0
    THEN 100 * (tblpages - est_tblpages_ff)/tblpages::float
    ELSE 0
  END AS bloat_pct, is_na
FROM (
  SELECT ceil(reltuples / ((bs-page_hdr)/tpl_size)) + ceil(toasttuples / 4) AS
est_tblpages,
         ceil(reltuples / ((bs-page_hdr)*fillfactor/(tpl_size*100))) + ceil(toasttuples /
4) AS est_tblpages_ff,
         tblpages, fillfactor, bs, tblid, schemaname, tblname, heappages, toastpages, is_na
  FROM (
    SELECT (4 + tpl_hdr_size + tpl_data_size + (2*ma)
            - CASE WHEN tpl_hdr_size%ma = 0 THEN ma ELSE tpl_hdr_size%ma END
            - CASE WHEN ceil(tpl_data_size)::int%ma = 0 THEN ma ELSE
ceil(tpl_data_size)::int%ma END
           ) AS tpl_size, bs - page_hdr AS size_per_block, (heappages + toastpages) AS
tblpages, heappages,
           toastpages, reltuples, toasttuples, bs, page_hdr, tblid, schemaname, tblname,
fillfactor, is_na
    FROM (
      SELECT tbl.oid AS tblid, ns.nspname AS schemaname, tbl.relname AS tblname,
tbl.reltuples,
             tbl.relpages AS heappages, coalesce(toast.relpages, 0) AS toastpages,
             coalesce(toast.reltuples, 0) AS toasttuples,
             coalesce(substring(array_to_string(tbl.reloptions, ' ') FROM 'fillfactor=([0-
9]+)')::smallint, 100) AS fillfactor,
             current_setting('block_size')::numeric AS bs,
             CASE WHEN version()~'mingw32' OR version()~'64-bit|x86_64|ppc64|ia64|amd64'
THEN 8 ELSE 4 END AS ma,
             24 AS page_hdr,
             23 + CASE WHEN MAX(coalesce(s.null_frac,0)) > 0 THEN (7 + count(s.attname)) /
8 ELSE 0::int END
               + CASE WHEN bool_or(att.attname = 'oid' and att.attnum < 0) THEN 4 ELSE 0
END AS tpl_hdr_size,
             sum((1-coalesce(s.null_frac, 0)) * coalesce(s.avg_width, 0)) AS tpl_data_size,
             bool_or(att.atttypid = 'pg_catalog.name'::regtype) OR sum(CASE WHEN
att.attnum > 0 THEN 1 ELSE 0 END) <> count(s.attname) AS is_na
      FROM pg_attribute AS att
      JOIN pg_class AS tbl ON att.attrelid = tbl.oid
      JOIN pg_namespace AS ns ON ns.oid = tbl.relnamespace
      LEFT JOIN pg_stats AS s ON s.schemaname=ns.nspname AND s.tablename=tbl.relname AND
s.inherited=false AND s.attname=att.attname
      LEFT JOIN pg_class AS toast ON tbl.reltoastrelid = toast.oid
      WHERE NOT att.attisdropped
      AND tbl.relkind in ('r','m')
      GROUP BY 1,2,3,4,5,6,7,8,9,10
      ORDER BY 2,3
    ) AS s
  ) AS s2
) AS s3
ORDER BY schemaname, tblname;
```

This query calculates the real size, extra size, and bloat percentage for tables. Key metrics include:

- **real_size**: Actual size of the table.
- **extra_size**: Extra space due to bloat.
- **extra_pct**: Percentage of bloat.
- **bloat_size**: Size of the bloat.
- **bloat_pct**: Percentage of bloat relative to the total size.

Threshold Values for Managing Bloat

When it comes to managing bloat in PostgreSQL, it's crucial to determine when bloat becomes significant enough to warrant intervention. Setting threshold values helps you decide when to use tools like `pg_repack` to optimize your database. Here are some guidelines for establishing these thresholds:

Index Bloat Thresholds

1. **Extra Size**: This is the amount of space taken up by bloat. A general rule of thumb is to consider repacking an index if the extra size exceeds 20% of the real size of the index.
2. **Extra Percentage**: This is the percentage of the index that is bloated. If the extra percentage exceeds 30%, it's usually a good indication that the index should be repacked.
3. **Bloat Size**: This is the size of the bloat itself. If the bloat size exceeds a specific threshold (e.g., 100 MB), it might be time to consider repacking.
4. **Bloat Percentage**: Similar to the extra percentage, if the bloat percentage exceeds 30%, it is typically a sign that the index needs maintenance.

Table Bloat Thresholds

1. **Extra Size**: Like with indexes, if the extra size of a table exceeds 20% of the table's real size, it might be time to consider repacking.
2. **Extra Percentage**: If the extra percentage of the table exceeds 30%, it indicates significant bloat that should be addressed.

3. **Bloat Size**: For tables, a threshold of 100 MB or more in bloat size can be used to decide when to repack.
4. **Bloat Percentage**: If the bloat percentage exceeds 30%, the table likely needs maintenance.

Example Scenario for Using Thresholds

Consider a scenario where you have run the provided SQL queries and obtained the following results for a table:

- Real Size: 500 MB
- Extra Size: 150 MB
- Extra Percentage: 30%
- Bloat Size: 100 MB
- Bloat Percentage: 20%

In this case, both the extra size and extra percentage are at significant levels (30% and 20%, respectively), indicating that it is time to use `pg_repack` to optimize the table.

Using `pg_repack` After Thresholds are Met

Once you have determined that a table or index meets the threshold for bloat, you can use **pg_repack** please read for more details for pg_repack our blog

Regular Monitoring and Maintenance

It's essential to regularly monitor your PostgreSQL database for bloat and perform maintenance as needed. Setting up automated monitoring scripts that run the provided SQL queries can help you track bloat levels and take timely action based on the thresholds you've set. This proactive approach ensures your database remains efficient and performs optimally. In conclusion, managing bloat effectively involves setting appropriate threshold values for intervention and using tools like `pg_repack` when these thresholds are met. Regular monitoring and timely maintenance are key to maintaining a performant PostgreSQL database. For more detailed and technical articles like this, keep following our blog on Medium. If you have any questions or need further assistance, feel free to reach out in the comments below and directly.