CHECKLIST

PostgreSQL Index Bloat Detection and Reindexing Checklist

A practical checklist for identifying PostgreSQL index bloat, validating whether it is operationally significant, and choosing a safe reindexing approach for production systems.

PostgreSQL Index Bloat Detection and Reindexing Checklist

Use this checklist to decide whether a PostgreSQL index is bloated enough to matter, and whether reindexing is the right operational response.

---

1) Confirm the business impact

- [ ] Identify the critical queries that depend on the index.
- [ ] Note whether the index is on a hot path for dashboards, user-facing reads, batch jobs, or API requests.
- [ ] Determine whether recent symptoms include slower reads, higher I/O, rising storage use, or less predictable plans.
- [ ] Confirm whether the system has uptime, replication, or maintenance-window constraints.
- [ ] Decide whether the cost of doing nothing is greater than the cost of rebuilding.

---

2) Identify candidate indexes

- [ ] List the largest indexes by size.
- [ ] List indexes with high update or delete churn on the underlying table.
- [ ] Flag indexes that support critical queries but have grown faster than expected.
- [ ] Flag indexes that appear large compared with table row count or live data footprint.
- [ ] Note indexes on tables with frequent updates to indexed columns.

---

3) Check whether bloat is plausible

- [ ] Compare current index size against historical size growth.
- [ ] Check whether table row count is stable while the index continues to grow.
- [ ] Check whether the table has recently shrunk after deletes, archival, or purges.
- [ ] Review whether autovacuum has had time and capacity to keep up.
- [ ] Consider whether the index is simply large because the table is large, the key is wide, or the access pattern truly requires it.

---

4) Validate with operational evidence

- [ ] Inspect relevant pg_stat_* views for usage, churn, and vacuum activity.
- [ ] Review query plans for important workloads that use the index.
- [ ] Confirm whether the index is actually scanned often enough for its inefficiency to matter.
- [ ] Determine whether the index is large and important, not just large.
- [ ] Avoid reindexing based on size alone.

---

5) Assess whether reindexing is justified

- [ ] Reindex only if the bloat is likely real and the operational impact is meaningful.
- [ ] Reindex only if storage growth, cache pressure, or read performance has become a concern.
- [ ] Confirm that rebuilding the index will not create more risk than the current bloat.
- [ ] Consider whether the index should be redesigned or dropped instead of rebuilt.
- [ ] Treat reindexing as a capacity and availability decision, not just housekeeping.

---

6) Choose the least disruptive strategy

- [ ] Use a standard rebuild only if a blocking lock is acceptable.
- [ ] Use a concurrent rebuild if the system cannot tolerate blocking on a busy index.
- [ ] Expect concurrent rebuilds to take longer and create additional write and I/O pressure.
- [ ] Confirm the exact locking and failure behavior for your PostgreSQL version.
- [ ] Select the simplest method that still meets uptime requirements.

---

7) Prepare a safe execution plan

- [ ] Schedule the change during the lowest-risk window available.
- [ ] Confirm backup, replication, and failover readiness.
- [ ] Check available disk space for the rebuild process.
- [ ] Identify whether write load or replication lag could increase during maintenance.
- [ ] Define a rollback or fallback plan if the rebuild fails or causes unacceptable load.
- [ ] Notify stakeholders if the rebuild may affect response times or maintenance SLAs.

---

8) Verify the result after maintenance

- [ ] Confirm the index size has changed as expected.
- [ ] Recheck the query plan for the affected workload.
- [ ] Validate that read latency and cache behavior have improved or remained acceptable.
- [ ] Check for unexpected replication lag, lock contention, or I/O spikes.
- [ ] Document the before-and-after state, including the reason for action and the outcome.

---

9) If you decide not to reindex

- [ ] Record why the index was left alone.
- [ ] Confirm that current impact is acceptable.
- [ ] Review whether autovacuum tuning, fillfactor, or workload changes could reduce future bloat.
- [ ] Revisit the index during the next maintenance review.
- [ ] Consider whether query redesign or index removal would be a better long-term fix.

Quick decision guide

- Reindex now: the index is large, actively used, and causing clear operational pain.
- Reindex later: the index looks suspicious, but evidence is incomplete or the window is not safe yet.
- Do not reindex: the index is large but healthy, rarely used, or not worth the disruption.

Final review

- [ ] I have measured before acting.
- [ ] I have confirmed the index matters to a real workload.
- [ ] I have chosen the least disruptive safe option.
- [ ] I have a plan for validation after the change.
- [ ] I have documented the decision either way.

Use this checklist as an operational guide, not a substitute for PostgreSQL version-specific documentation or change-control procedures.