🔍 Are Redundant Indexes Slowing Down Your SQL Server?

Most DBAs tune queries…

But many forget one silent performance killer:

👉 **Redundant Indexes.**

These are indexes on the same table with the same leading key columns, often duplicated with slight variations. And they cause more harm than we realize:

- 🚫 Extra storage
- 🚫 Slower INSERT/UPDATE/DELETE operations
- 🚫 Increased fragmentation
- 🚫 More memory usage
- 🚫 Longer maintenance windows

The good news?

You can detect *every redundant index across every database* in a single scan.

A full-instance script can help you identify:

- ✓ Indexes that have identical key columns
- ✓ Smaller indexes fully covered by larger ones
- ✓ Redundant include columns
- ✓ "Child indexes" that can be safely dropped
- ✓ Tables suffering from duplicated index patterns

Why does this matter?

Because dropping redundant indexes leads to:

- ⚡ Faster write performance
- ⚡ Reduced storage costs
- ⚡ Healthier buffer pool
- ⚡ Shorter rebuild/reorg cycles
- ⚡ Cleaner, more efficient index strategy

💡 **Pro Tip:**

Before dropping any index, always check usage stats and validate with application teams.

Optimizing indexes isn't just cleanup — it's *smart performance engineering*.

Reference: https://www.madeiradata.com/post/every-redundant-index-in-every-database-all-at-once

Credits To: Eitan Blumin

DBA Checklist — Redundant Index Cleanup

Below is a ready-to-use checklist DBAs can follow.

🔍 **Redundant Index Cleanup Checklist (For DBAs)**

1 Preparation

- Run index usage stats (dm_db_index_usage_stats)
- Generate list of all indexes per database
- Identify tables with more than 5–10 indexes
- Ensure full backups are recent

2 Redundant Index Identification

- Compare key columns between indexes on same table
- Identify indexes where:
  - Leading key columns match
  - Larger index covers smaller index
  - Only included columns differ
- Check for filtered indexes with overlapping filters
- Mark indexes with **0 seeks, 0 scans, 0 lookups**

3 Validation Before Dropping

- Confirm index usage over last 7–30 days
- Review query plans of relevant workloads
- Validate with developers/app owners
- Confirm no replication/distribution dependencies
- Confirm no constraints (PK/unique)

4 Safe Cleanup

- Script “DROP INDEX …” statements
- Drop redundant index during low traffic window
- Update statistics after cleanup
- Rebuild/REORG remaining indexes

5 Post-Cleanup Monitoring

- Track blocking / high CPU / slow queries
- Watch query store regressions
- Capture index usage over next week
- Document the change

Checklist Result:

👉 A cleaner, faster, more efficient SQL Server environment.

Stored Procedure — Detect Redundant Indexes in ALL Databases

https://github.com/PMSQLDBA/DB-Maintenance-Tasks/blob/main/RedundantIndexes-Audit