

Database Query Optimization

Database Query Optimization – Quick Reference Notes

1. Why Optimize Queries?

- **Performance:** Faster response times, lower latency.
 - **Scalability:** Handles larger data volumes and more concurrent users.
 - **Resource Efficiency:** Reduces CPU, memory, I/O, and network load.
 - **Cost Savings:** Less hardware and cloud?resource expenditure.
-

2. Core Concepts

Concept	Description	Typical Impact
Cost Model	DBMS estimates I/O, CPU, and memory needed for each execution plan.	Guides the optimizer's choice.
Selectivity	Fraction of rows a predicate returns ($\text{rows} / \text{total_rows}$).	High selectivity ? early filtering, index use.
Cardinality	Expected number of rows produced by an operation.	Influences join order & join method.
Statistics	Histograms, density, distinct values, column correlations.	Accurate stats ? accurate cost estimates.
Execution Plan	Tree of operators (scan, join, sort, etc.) that the DB will run.	Visual map for tuning.
Hints/Directives	Manual overrides (e.g., <code>USE INDEX</code> , <code>FORCE ORDER</code>).	Fine?tune when optimizer mis?chooses.

3. Optimization Workflow

1. Capture the Problem

- Identify slow query (long execution time, high CPU, blocking locks).

Database Query Optimization

- Collect baseline metrics: `EXPLAIN ANALYZE`, `SET STATISTICS IO ON`, `SET STATISTICS TIME ON`.

2. Analyze the Execution Plan

- Look for:
 - Full table scans on large tables.
 - Nested-loop joins with high row counts.
 - Missing or unused indexes.
 - Large sort/aggregate operations.
 - Repeated scans of the same table.

3. Check Statistics

- Ensure up-to-date stats: `ANALYZE`, `UPDATE STATISTICS`, `DBCC UPDATEUSAGE`.
- Verify histogram accuracy for skewed data.

4. Apply Tuning Techniques

- **Indexing** (see Section 4).
- **Rewrite the query** (simplify, push predicates, use set-based logic).
- **Adjust join order** (explicit `JOIN` hints or subqueries).
- **Limit result set** (`TOP`, `LIMIT`, `FETCH FIRST`).

5. Validate

- Rerun `EXPLAIN ANALYZE`.
- Compare execution time, I/O, and CPU before/after.
- Test with realistic data volumes.

6. Monitor in Production

- Use DMVs (SQL Server), `pg_stat_statements` (PostgreSQL), `V$SQL` (Oracle), etc.
- Set alerts for regression.

4. Index-Centric Techniques

Index Type

When to Use

Key Points

Database Query Optimization

B-Tree	Equality, range predicates, ordering.	Composite index order matters; include columns to cover the query.
Bitmap	Low-cardinality columns, data warehousing.	Great for many-to-many joins, but heavy on concurrent DML.
Hash	Exact equality lookups (e.g., in-memory engines).	No ordering; not useful for range queries.
Clustered/Primary Key	Defines physical row order.	Good for range scans; only one per table.
Covering Index	All columns needed are in the index (key + included).	Eliminates need to read the base table (index-only scan).