Database Tip #2 -- How the Optimizer Uses Indexes

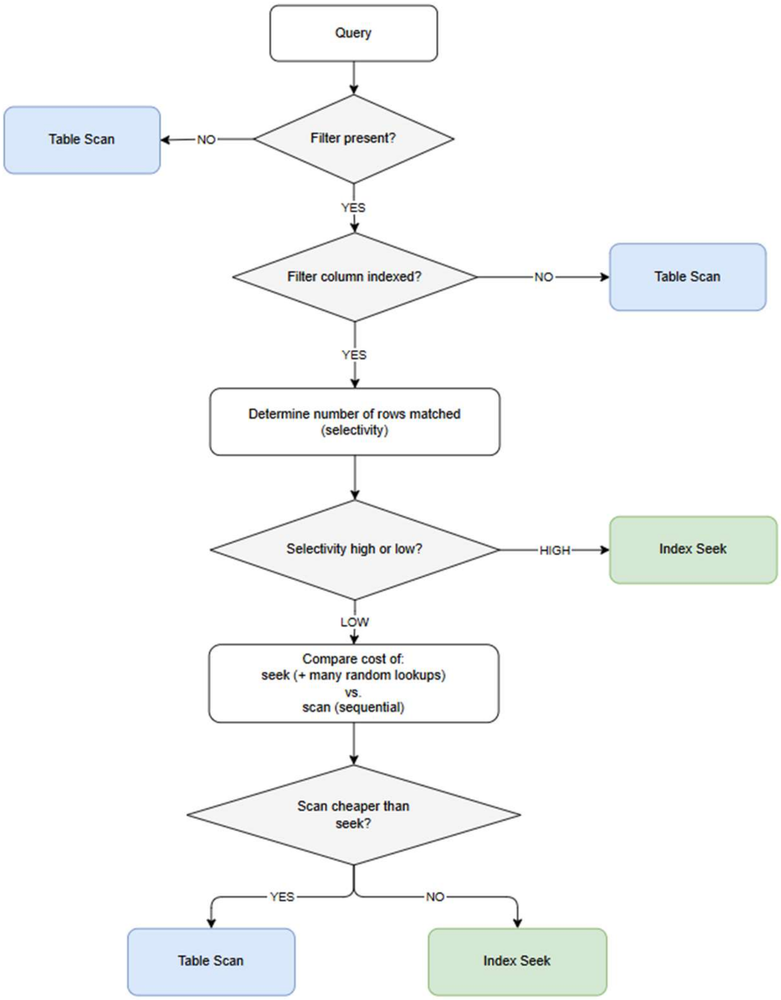
When a query includes a filter, the optimizer must choose the most efficient way to retrieve the required rows. That choice almost always comes down to two access paths:

- Use an index seek to jump directly to matching rows, or
- Perform a table or index scan to read pages sequentially.

The optimizer makes this decision using a simple but powerful concept: selectivity — how many rows the filter is expected to return.

To understand how this works, here's a clear, step-by-step decision flow that mirrors the optimizer's logic.

Optimizer Decision Flow: Seek or Scan?



Note:This diagram shows the optimizer's behavior when evaluating a single filter condition. With multiple predicates, the optimizer considers each indexed column, their combined selectivity, and whether composite indexes exist.

Understanding Selectivity

Selectivity determines whether the index is helpful:

- High selectivity: the filter returns a small percentage of the table
- Low selectivity: the filter returns a large percentage of the table

And importantly…if the query has no filter at all, the optimizer must read the entire table — which always results in a scan, regardless of the table's size.

On small tables, this is fast.

On large tables, it can be expensive — but still cheaper than using an index when all rows are needed.

Why Scans Aren't Always "Bad"

A scan simply means: "Read all data pages sequentially in storage order."

Sequential I/O is extremely efficient as the DBMS reads large amounts of data with minimal overhead, which is why scans can outperform seeks when many rows are needed.

And here's the key nuance: whether a scan is "fast" or "slow" is relative to the size of the table.

- On small tables, scanning every page is often faster than seeking into an index.
- On large tables, a scan can be expensive — but still cheaper than performing thousands or millions of random lookups when many rows are needed.

Indexes help only when they allow the optimizer to avoid reading non-qualifying data. If most of the table must be read anyway, the index provides no benefit.

Takeaway

Indexes improve performance by reducing the amount of data the DBMS must read.

If the filter is highly selective, a seek is efficient.

If the filter returns many rows — or there's no filter at all — a scan becomes the cheapest option.