Index Scan vs. Sequential Scan in PostgreSQL

PostgreSQL's **query planner** decides whether to use an **Index Scan** or a **Sequential Scan** based on multiple parameters. The key influencing factors include **cost-based estimation**, available memory, cache settings, and table statistics.

1. Parameters That Influence Index vs. Sequential Scan

Parameter	**Effect on Scan Type**
effective_cache_size	A higher value encourages **Index Scans** by assuming more data is cached in RAM.
random_page_cost	Lowering it makes **Index Scans** cheaper and more preferable.
seq_page_cost	Higher values make **Sequential Scans** more expensive, favoring **Index Scans**.
work_mem	Higher work_mem allows sorting and joins to stay in memory, reducing the need for Sequential Scans.
shared_buffers	Affects how much PostgreSQL can cache internally before relying on OS caching.
Table Size	Smaller tables often get **Sequential Scans**, as reading everything is cheaper than using an index.
Index Selectivity	If an index filters a large portion of the table, an **Index Scan** is preferred.
Query Conditions	If the query returns many rows, a **Sequential Scan** may be more efficient than multiple index lookups.

==

==

Index Scan vs. Sequential Scan in PostgreSQL

1. Sequential Scan (Seq Scan)

A **Sequential Scan** reads the entire table row by row, even if it only needs a few rows. It is **efficient for small tables** or queries that return a large percentage of the table.

✅**Preferred when:**

- The table is **small**, so reading the whole table is fast.
- The query needs to retrieve **a large number of rows** (e.g., SELECT * FROM large_table).
- The table **does not have an index** on the search column.
- **Full table scans** (OLAP queries, reports) are expected.

� **Disadvantage:**

- **Slow for large tables** if only a few rows are needed.

Case 1: Default Cost Settings (random_page_cost = 4, seq_page_cost = 1)

- **EXPLAIN ANALYZE**
- **SELECT * FROM employees WHERE department_id = 10;**

Result (Sequential Scan used):

- Seq Scan on employees (cost=0.00..20000.00 rows=1000 width=50)

- PostgreSQL assumes disk reads are expensive (random_page_cost = 4), so it avoids the index and does a **full table scan**.

2. Index Scan

An **Index Scan** uses a B-tree or other index type to find specific rows quickly **without scanning the entire table**. It is preferred for **selective queries** that fetch only a few rows.

✅**Preferred when:**

- The table is **large**, and the query retrieves **a small number of rows**.
- The column being searched is **indexed**.
- Queries use **conditions on indexed columns** (WHERE column = value).

 Disadvantage:

- If the query retrieves a **large percentage of the table**, using an index can be slower than a sequential scan.

Case 2: Lowering random_page_cost = 1.1 to Favor Index Scan

- SET random_page_cost = 1.1;
- EXPLAIN ANALYZE
- SELECT * FROM employees WHERE department_id = 10;

Result (Index Scan used):

- Index Scan using idx_department_id on employees (cost=0.42..500.00 rows=100 width=50)

- Now, PostgreSQL thinks **random disk accesses are cheap**, so it prefers an **Index Scan**.

3. Tuning for Optimal Performance

For OLTP Workloads (Frequent Small Queries)

1.
1. Increase effective_cache_size (e.g., 75% of RAM).
2. Decrease random_page_cost to **favor Index Scans**.
3. Use EXPLAIN ANALYZE to check if queries are using indexes.
2.

For OLAP Workloads (Large Reports, Full Table Scans)

1. Allow more work_mem for large queries.
2. Keep seq_page_cost low to **favor Sequential Scans**.
3. Ensure parallel_tuple_cost and parallel_setup_cost allow parallel execution.

==

Which One is Preferred?

- **Small tables** → **Sequential Scan** (as reading everything is fast).
- **Large tables with selective queries** → **Index Scan** (for efficiency).
- **Full-table processing (aggregations, reports)** → **Sequential Scan** (because reading everything at once is faster).

⍰ Rule of Thumb:

- If the query **fetches <10%** of a large table → **Index Scan is preferred**.
- If the query **fetches >10%** of a table → **Sequential Scan may be better**.