Writing Efficient Queries, Part 2

Advanced SQL and Cloud

Objectives

1. Understand the role of table statistics in table planning
2. Apply query design best practices for analysts
3. Use temporary tables effectively
4. Identify and fix common performance gotchas
5. Analyze and optimize queries, demonstrating measurable improvements in query performance

Query Planner

Databases don't just blindly scan tables. They use a **query planner** to decide how to retrieve the results of a query.

The query planner examines different possible plans and goes with the one it thinks will be the most efficient.

Query Planner

- This query planner relies on statistics, or metadata about the data: number of rows, value distributions, how common NULL values are, etc.
- Signs that the statistics may be out of date or inaccurate:
  - The number of estimated rows is very different from the number of actual rows in the output of EXPLAIN ANALYZE
  - Inefficient join methods are chosen
- While not usually the job of a data analyst, it is possible to use the ANALYZE function to update the statistics.
  - See https://www.postgresql.org/docs/current/planner-stats.html and https://www.postgresql.org/docs/current/planner-stats-details.html for more information.

Query Design Tips

- Filter as early as possible. The less rows that have to be processed, the better.
    - Note: This can often be done automatically by the query planner, but it never hurts to structure your query to eliminate as many rows of possible quickly, particularly if you can make use of an index column to do so.

Query Design Tips

- Avoid accidental cartesian joins which can occur when you forget an ON condition or accidentally join a column with itself

```
SELECT jc.job_candidate_id,
       j.job_name
FROM job_candidate AS jc
INNER JOIN job j
       ON jc.job_id = jc.job_id
```

- Avoid accidental many-to-many joins. Ensure that at least one of the columns you are joining on is a unique identifier for the row or you are joining on a unique combination of columns.

Query Design Tips

Avoid unnecessary DISTINCT or ORDER BY calls.

Sorting is slow in general, particularly if what is being sorted doesn't fit into memory and must spill to disk, where reading/writing is much slower.

If sorting is necessary, use a LIMIT when possible, which can allow a much faster algorithm for sorting: Top-N heapsort, which can often be done in memory.

Query Design Tips

AI can be a great tool for writing code and creating SQL queries, but it must be used with caution.

It is common for AI to make mistakes in writing queries or to write very inefficient queries.

Do not turn off your critical thinking skills and always inspect the output to ensure that it cannot be streamlined or made more efficient.

Temporary Tables

A **temporary table (temp table)** is a table that you create during your session. It disappears when you disconnect.

Temp tables can be used to store intermediate results from a query, which is great for efficiency if you're going to be using that same results set multiple times during that session.

Temp tables look and feel like a normal table when you use them.

Temporary Tables

Advantages:

- Simplify complex queries. Temp tables can break up complex queries into smaller, easier-to-read steps.
- Improve performance. Work with a smaller dataset instead of having to scan the full dataset potentially multiple times.
- Reuse results. Run a filter/aggregation once, then join or query it multiple times.

Temporary Tables

How to Create:

```
CREATE TEMP TABLE active_orders AS
SELECT *
FROM orders
WHERE status = 'Shipped'
    AND order_date > CURRENT_DATE - INTERVAL '90 days';
```

How to Use:

```
SELECT c.customer_id, COUNT(*)
FROM customers  AS c
JOIN active_orders AS ao
    ON c.customer_id = ao.customer_id
GROUP BY c.customer_id;
```