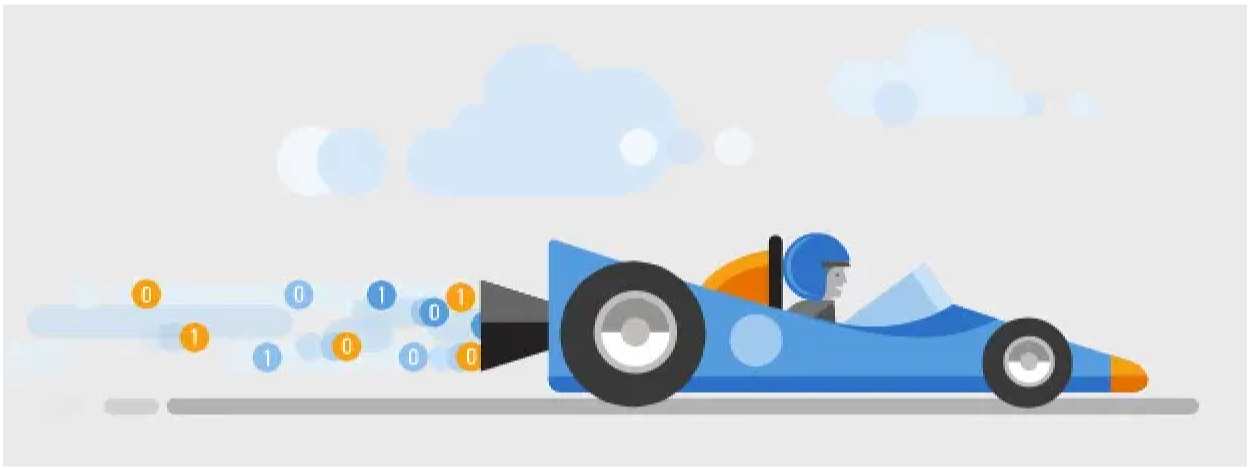


Turbocharging Data Pipelines: How Hybrid Tables Cut Runtimes by 50%

Anurag Sharma 4 min read · Jun 4, 2025

Imagine your data pipelines chugging along like a tired old steam engine — huffing, puffing, and burning through compute resources just to get the job done. Now picture a sleek, turbo-charged sports car zipping through the same workload in half the time. That’s the kind of transformation we achieved by introducing **hybrid tables** in our data engineering project.

The Problem: Joins That Just Wouldn't Quit 😩

In the world of data engineering, joins are like the glue that holds your data together — but when you're gluing massive, high-volume tables over and over, things get sticky.

Our Spark pipelines were doing frequent, heavy-duty joins across the same big tables for multiple downstream applications. Each join was like asking a chef to chop the same vegetables from scratch for every meal. Not only was it time-consuming, but it was also burning through our cloud compute budget faster than a foodie at a buffet.

The result? **Pipeline runtimes were ballooning**, compute costs were creeping up, and our team was spending more time on fixing jobs than innovating. We needed a smarter way to streamline this process without reinventing the wheel.

The Lightbulb Moment: A Join Map to Rule Them All 💡

To tackle this, I decided to play Excel-Excel. I grabbed Excel and created a **join map**. This was a detailed spreadsheet that laid out:

• **All the tables** in our data ecosystem.

• **Their relationships** (think: table A loves table B, but only on specific columns).

• **The most frequently used joins** across our pipelines.

This join map was like a treasure map for optimization. It revealed which joins were the usual suspects slowing down our pipelines.

Spoiler alert: Some joins were showing up more often!

The Game Plan: Enter Hybrid Tables 🛠️

I proposed a bold idea: **hybrid tables**. These are pre-joined, materialized tables stored in Amazon S3 in parquet format, designed to handle the heavy lifting of repetitive joins. Instead of Spark re-computing the same joins for every pipeline, we would do the work once, store the results, and let downstream jobs grab the pre-processed data.

Here is how we made it happen:

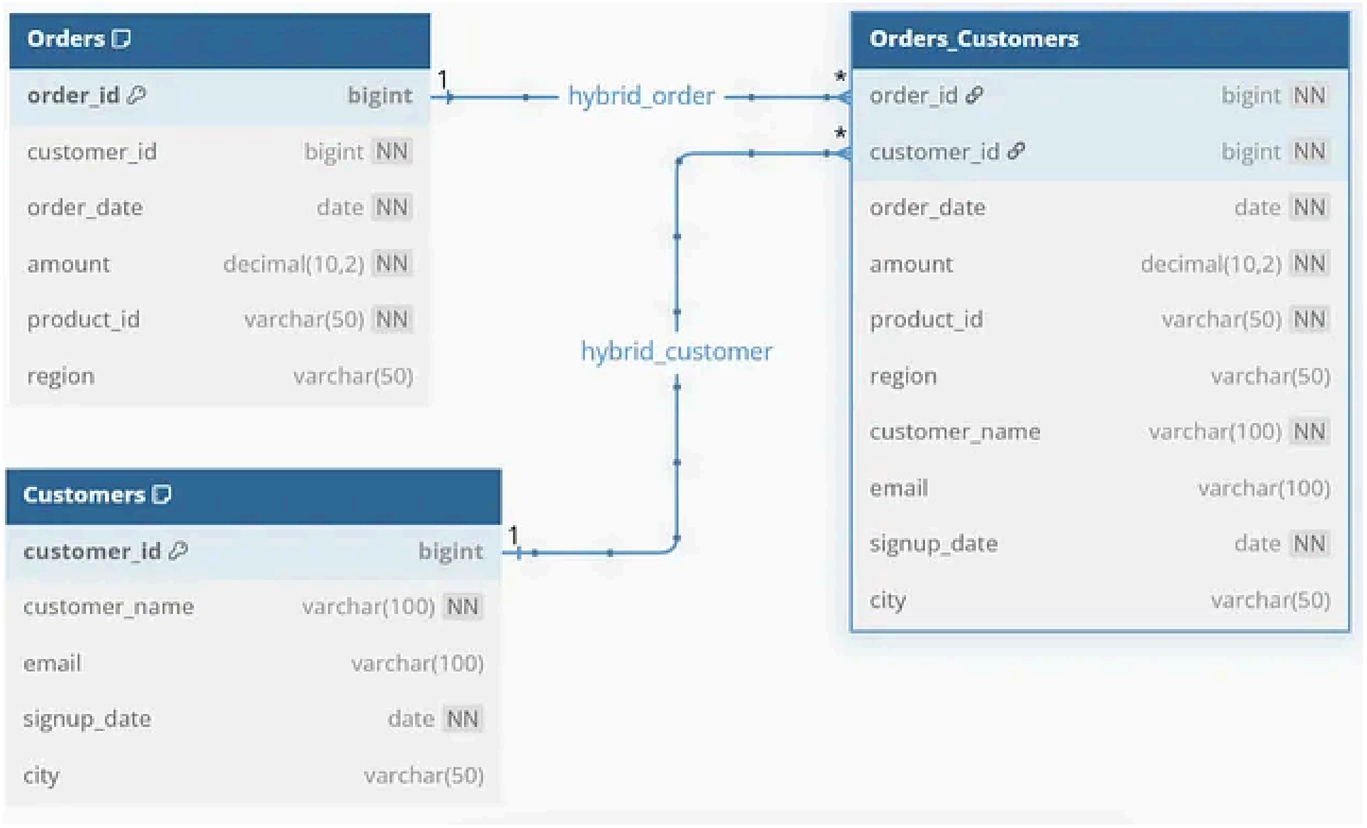
1. Identified High-Impact Joins: Using the join map, we pinpointed 30–40 frequently used joins that were critical to multiple pipelines. These were the VIPs of our data world — think customer data joined with transaction records or product data merged with inventory details.

2. Designed Hybrid Tables: We created hybrid tables by pre-joining these high-volume tables on their most commonly used columns. For example, if every pipeline needed to join Orders and Customers on customer_id, we would materialize that join into a hybrid table and save it in S3.

3. Optimized Storage: To keep things efficient, we used **Parquet** format for its columnar storage magic, which reduced storage costs and sped up read times. We also partitioned the hybrid tables by key columns (like date or region) to make queries even snappier.

4. Integrated with Pipelines: We updated our Spark jobs to read directly from these hybrid tables instead of performing runtime joins.

Example For a Hybrid Table:



The diagram shows three tables and their relationships:

- **Orders Table** (Left): Contains order details like order_id (primary key), customer_id, order_date, amount, product_id, and region. The customer_id links to the Customers table.
- **Customers Table** (Bottom Left): Stores customer information with customer_id (primary key), customer_name, email, signup_date, and city.
- **Orders_Customers Table** (Right): A hybrid table that pre-joins Orders and Customers on customer_id, combining their columns (order_id, customer_id, order_date, amount, product_id, region, customer_name, email, signup_date, city) to avoid repetitive joins in Spark pipelines.

The Payoff: 50% Faster Pipelines and Happier Wallets 🎉
The results were nothing short of spectacular. By introducing hybrid tables, we slashed **pipeline execution time by 50%**.

Here is what that meant for us:

• **Faster Pipelines:** Jobs that once took hours were now finishing in half the time, freeing up compute resources and letting our team focus on building new features.

• **Lower Costs:** Less compute time in Spark meant lower cloud bills. Our DevOps team was thrilled.

• **Happier Downstream Apps:** Teams relying on our data got their insights faster, making everyone from the Data Science to the Data Analytics Team smile a little wider.

Plus, the hybrid tables were reusable across multiple pipelines as we created documentation to explain each hybrid table in detail, sharing the exact usage for the downstream process, so we didn't have to keep reinventing the wheel. It was like creating a universal remote for our data ecosystem — one click, and everything just worked.

Lessons Learned: Work Smarter, Not Harder 🧠

This experience taught us a few key lessons that every data engineer should keep in their toolbox:

• **Map It Out:** A simple join map can reveal hidden bottlenecks and guide your optimization strategy.

• **Pre-Process When Possible:** Materializing frequent joins into hybrid tables can save massive amounts of compute time and cost.

• **Think Reusability:** Designing solutions that multiple pipelines can leverage maximizes efficiency and scalability.

• **Storage Matters:** Using efficient formats like Parquet and smart partitioning can supercharge your performance.

Summary

✓ **Problem:** Frequent, compute-heavy joins in Spark pipelines slowed down runtimes and increased costs.

✓ **Solution:** Created hybrid tables by pre-joining high-volume tables and storing them in S3.
✓ **Impact:** Reduced pipeline execution time by 50%, lowered compute costs, and boosted downstream efficiency.

Data Engineering Data Engineer Optimization Spark Automation

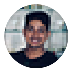
Written by Anurag Sharma

82 followers · 3 following

Edit profile

Data Engineering Specialist with 10+ exp. Passionate about optimizing pipelines, data lineage, and Spark performance and sharing insights to empower data pros!

No responses yet



Anurag Sharma him/he

What are your thoughts?

More from Anurag Sharma