# Step 1: Column Pruning

## What is Column Pruning?

**Column Pruning** is the optimization where the query engine reads only the specific columns requested in a `SELECT` statement, ignoring all others.

## Why Columnar Formats Enable It

In a columnar format like Parquet, each column is stored in its own separate block or set of files. The storage layer has a "map" (metadata) that tells the engine exactly where the `amount` column starts and ends on the disk.

## Why Row-Based Formats Cannot

Row-based formats (CSV, JSON, Avro) store data as a sequence of records. To reach the 6th column in a row, the computer must physically scan through the first 5 columns to find the delimiter. Even if you only need 2 columns, you pay the **I/O cost** for the entire row.

## Example Scenario

**Table:** `orders` (9 columns: `order_id`, `customer_id`, `order_date`, `region`, `product_id`, `amount`, `discount`, `tax`, `total_amount`) **Query:** `SELECT order_id, amount FROM orders;`

- **Total Columns:** 9
- **Columns Needed:** 2
- **Columnar (Parquet):** 2 columns read.
- **Row-based (CSV):** 9 columns read (the whole file).
- **I/O Savings: ~78% reduction** in data transferred from disk to memory.

---

# Step 2: Predicate Pushdown

## What is Predicate Pushdown?

**Predicate Pushdown** is the optimization where the filtering logic (the `WHERE` clause) is "pushed" down to the storage layer. The engine evaluates whether a block of data is relevant *before* actually opening and reading it.

### How Parquet Metadata Helps

Parquet files are divided into **Row Groups**. For every row group, Parquet stores metadata (statistics) in the file footer, including:

- **Min/Max values** for every column.
- **Null counts**.

### Why it Reduces Data Scanned

If you filter `WHERE order_date = '2026-02-21'`, the engine checks the Min/Max metadata for each Row Group. If a group's `order_date` range is `2026-01-01` to `2026-02-15`, the engine **skips that entire block** without reading a single row.

### Analysis

- **Predicate:** `order_date = '2024-03-01'`
- **Metadata Help:** Engine looks at the `order_date` min/max for each block.
- **Data Skipped:** Every row group where '2024-03-01' is outside the min/max range.
- **I/O Reduction:** Massive. If you have 3 years of data, you might skip 99% of the file.

---

## Step 3: Query Scenario Analysis

**Query:** `SELECT region, SUM(amount) FROM sales WHERE order_date = '2026-02-21' GROUP BY region;`

### Column Analysis

- **Table Columns:** `order_id`, `customer_id`, `order_date`, `region`, `product_id`, `amount`, `discount`.
- **Needed Columns:** `order_date` (for filter), `region` (for grouping), `amount` (for sum).
- **Columns Read:** 3 (`order_date`, `region`, `amount`).
- **Columns Skipped:** 4 (`order_id`, `customer_id`, `product_id`, `discount`).

### Optimization Breakdown

| Step | Action | Benefit |
|---|---|---|
| **1. Pushdown** | Check footer metadata for `order_date` min/max. | Skips non-matching row groups entirely. |
| **2. Pruning** | Locate only `order_date`, `region`, and `amount` offsets. | Skips 4 unused columns within matching groups. |
| **3. Reading** | Pull only the 3 columns from matching groups into RAM. | Minimizes network/disk traffic. |
| **4. Processing** | Aggregate `amount` by `region` in CPU. | Operates on a tiny, pre-filtered subset of data. |

# Step 4: Optimization Summary

## The Synergy

Parquet provides the **structure** (Columnar), Column Pruning provides the **focus** (Selective Read), and Predicate Pushdown provides the **filter** (Skipping). Together, they allow a query engine to act like a laser rather than a floodlight.

## Scale Impact Example

- **Table Size:** 1 Billion Rows, 20 Columns (Assume 100GB total size).
- **Query Needs:** 2 columns (10% of width), filters to 1% of rows (Date filter).
- **CSV Reading:** 100 GB (Must read everything).
- **Parquet Reading:** ~100 MB (10% of the columns $\times$ 1% of the rows).
- **Result: 1000x improvement** in I/O and significant cost savings in cloud environments like AWS (S3/Athena).