Star-Joins in Query Processing and Optimization

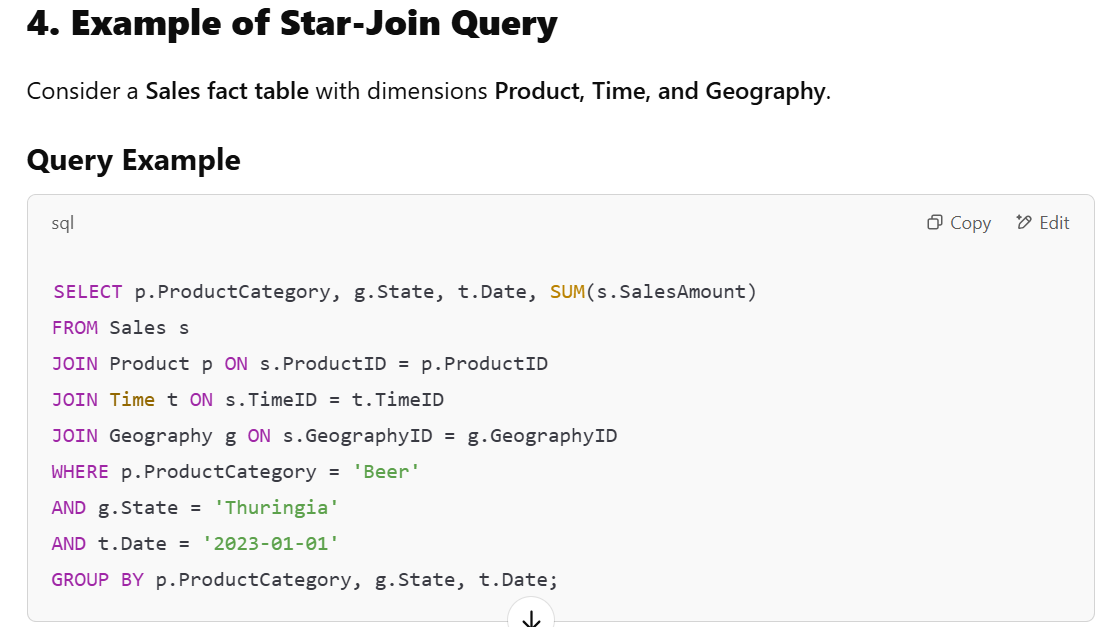
**1. Introduction to Star-Joins**

* A **Star-Join** is a common pattern in **Data Warehouse** queries.
* It involves a **large fact table** and **multiple smaller, independent dimension tables**.
* The classical relational query optimizers struggle with **star-joins** because traditional heuristics fail to optimize these effectively.

**2. Characteristics of Star-Schema**

A **Star Schema** consists of:

1. **A large fact table** (e.g., Sales).
2. **Multiple smaller, independent dimension tables** (e.g., Product, Time, Geography).
3. Fact table contains **foreign keys** referencing dimension tables.
4. Queries typically involve **joining fact and dimension tables**.



The query joins **Sales (fact table)** with three **dimension tables**.

Filters are applied **before** the join to reduce data size.

**3. Star Join Execution Process**

1. **Filter Dimension Tables First**
   * Reduce the number of rows by applying filters on dimension tables.
2. **Convert Filters to Indexed Lookups**
   * Fetch relevant dimension keys using **bitmap indexing**.
3. **Join with Fact Table Efficiently**
   * Use **hash joins** or **merge joins** instead of nested loops.
4. **Aggregate Results**
   * Perform calculations like SUM(), COUNT(), or AVG() on the filtered fact table.

**5. Benefits of Star Join**

✅ **Faster Query Performance** – Reduces full table scans.  
✅ **Efficient for Large Datasets** – Optimized for millions/billions of rows.  
✅ **Supports OLAP Operations** – Used in BI tools and data warehouses.

**6. Semi-Join Optimization for Star-Joins**

**What is a Semi-Join?**

* A **semi-join** reduces the number of rows in the **fact table** before the actual join.
* Instead of joining the entire fact table, **only relevant rows** are selected.

**Steps of Semi-Join Optimization**

1. **Indexing**: Create indexes on the **fact table** for each dimension.
2. **Filtering via Semi-Joins**:
   * Use **dimension table filters** to create **lists of relevant fact table rows** (Tuple Identifiers, TIDs).
3. **Intersection of TID Sets**:
   * Compute the **intersection** of relevant TIDs across dimensions.
4. **Final Join Execution**:
   * Perform **normal joins** using only the relevant TID sets.