Multiple Query Optimization

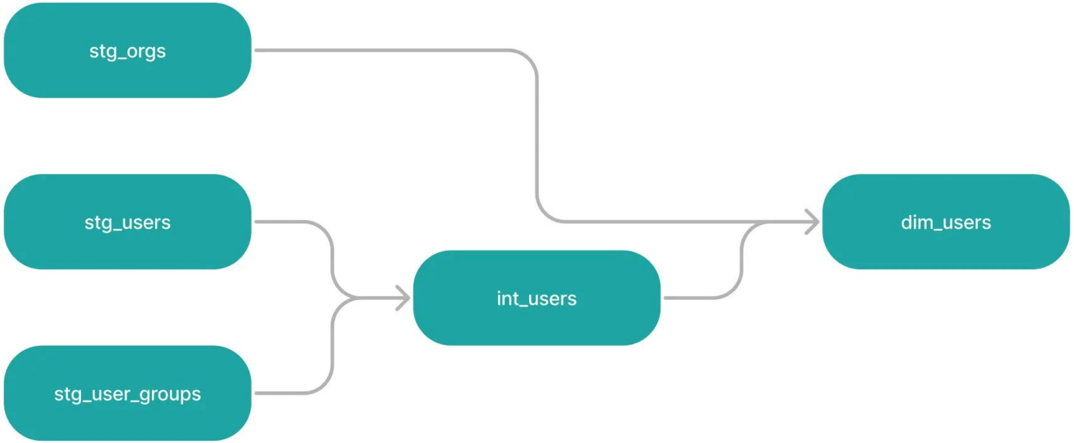
Logical DAG Rewriter

Guide (Yuttapichai), Yizhou, Frank

CMU 15-799 Final Presentation

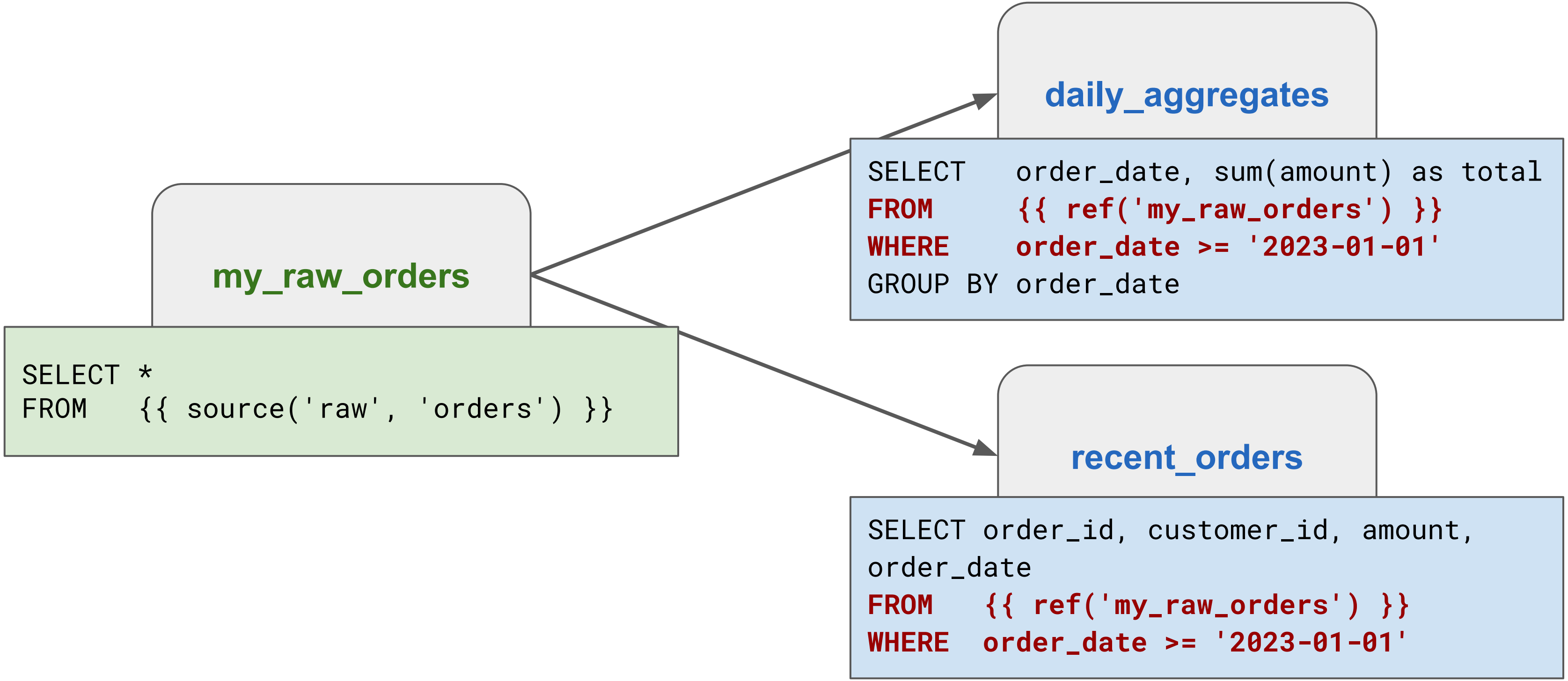
Multiple queries running on dbt

- dbt (Data Build Tool) is a tool for doing data transformation
  - Allow for creating a dependency graph (a node is a transformation stage)
  - Each stage can be written as an SQL statement

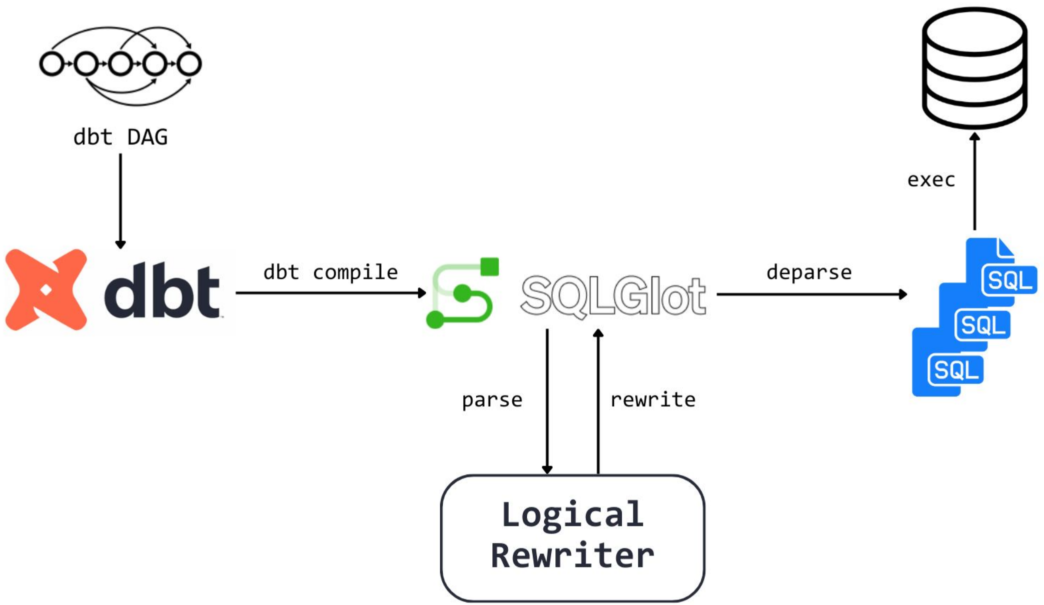


Ref: https://www.getdbt.com/blog/dag-use-cases-and-best-practices

Redundant computations among multiple queries



Solution: (Extensible) Logical DAG Rewriter



Current Status: Most of the things are done + Statistics

75% Goal

- Generate a workload for benchmarking dbt's DAGs ✅
- Implement a DAG rewriter with at least the predicate pushdown heuristic ✅

100% Goal

- Implement all the proposed logical query optimizations 🆗
  (We decided to drop some optimizations)
- Evaluate the DAG using the benchmark ✅

125% Goal

- Explore and implement Physical query optimization (e.g. materialized views from ancestors, cache & reuse of intermediate "subresults") ❌
- Inject statistics from DBMS to decide when to optimize ✅

Three supported optimization heuristics

- Predicate Pushdown
    - Push-able
    - Not Push-able
- Common CTE Elimination
- Naive Projection Pushdown

Two metrics to evaluate our approach

- **Correctness**
  - The output of the rewritten DAG must be the same as the original DAG

- **Performance**
  - Compare the overall execution time between the original and the rewritten DAGs
  - Microbenchmark for each optimization rule to see its performance benefit

Evaluate through both micro/macro-benchmark

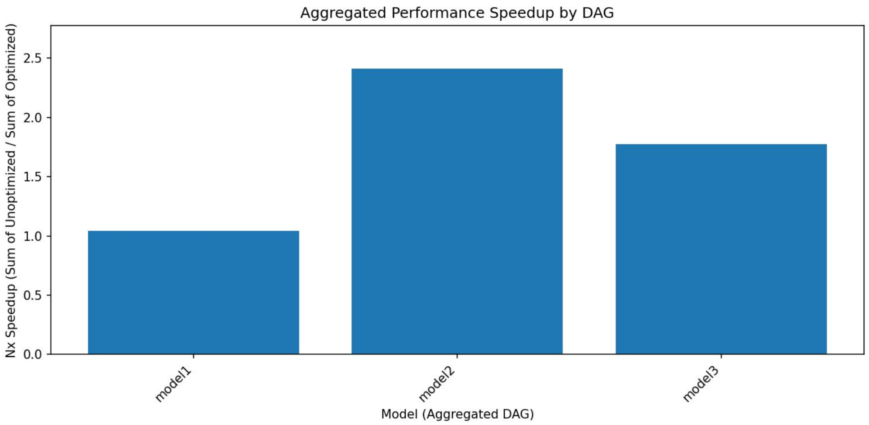
- **Microbenchmark**
  - We synthetically create DAGs (derived from TPC-H) for each specific optimization rule:
    - Predicate Pushdown
    - Common CTE Elimination
    - Projection Pushdown

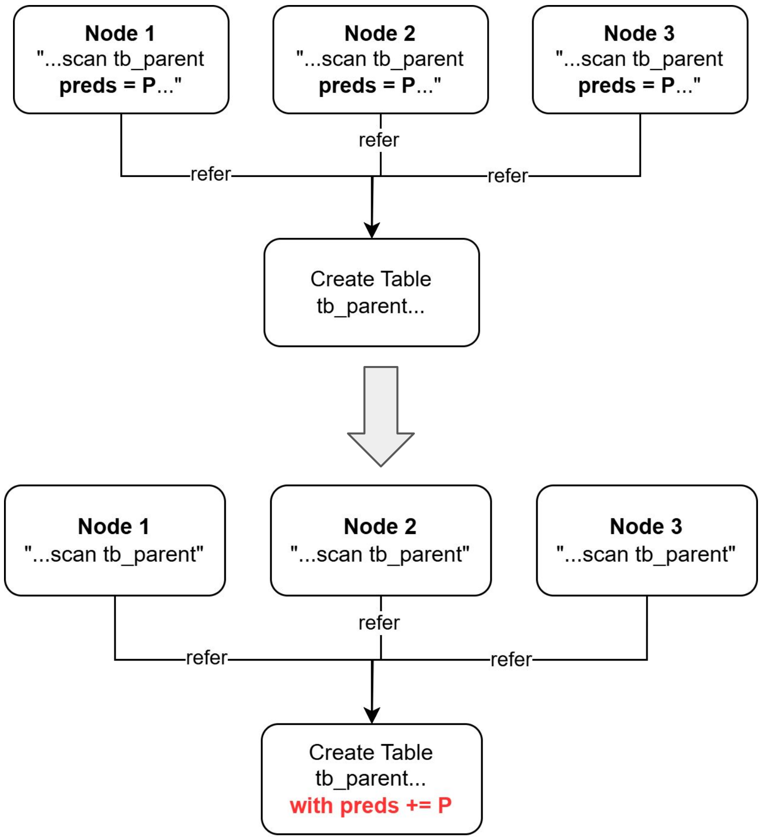
- **More Realistic DAGs**
  - Existing production DAGs from other repositories
    - jaffle-shop

Predicate Pushdown: Push-able

- **Push-able nodes** refer to nodes that we are not interested in their results

- Benchmark results (TPC-H)





Predicate Pushdown: Non-pushable

- When the result of the parent node is required, we need to add an intermediate node
  - Adding a node means we need to materialize the results
- Trade-offs!
  - If we add the intermediate node blindly…

dbt_tpch > tpch > final_benchmark_results.csv

```
TableName,UnoptimizedTimeMs,OptimizedTimeMs,AbsoluteMsDiff (Unopt-Opt),PercentImprovement
simple_multi_pd_child1,289,31,258,89.27335640138409
simple_multi_pd_child2,253,29,224,88.53754940711462
simple_multi_pd_parent,21020,21100,-80,-0.3805899143672693
simple_multi_pd_parent_pushdown,MISSING,4600,,

TOTAL,21562,25760,-4198,-19.47
```

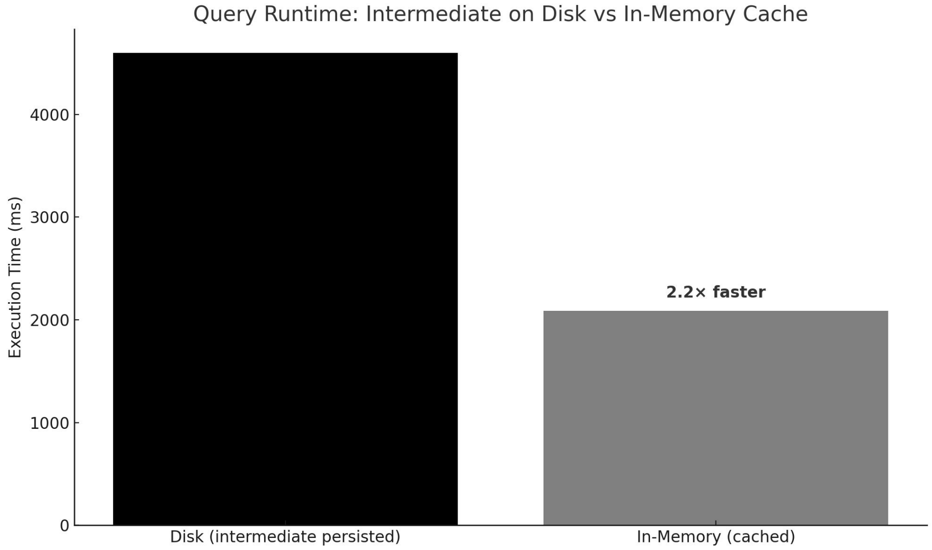
Worse than unoptimized!

Solution: In-Memory Temporary Table

- Use in-memory storage instead of disk storage for new intermediate nodes
    - Experiment: Memory vs Disk (400K rows)

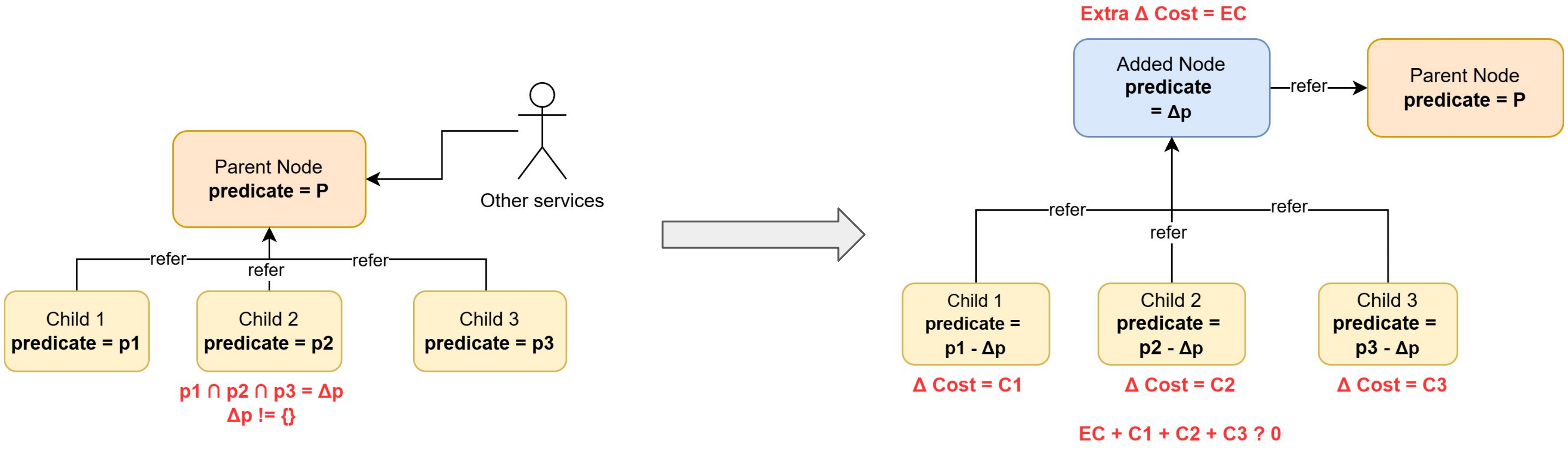
```sql
CREATE TABLE dev.main."parent_pushdown" AS
SELECT *
FROM   "dev"."main"."parent"
WHERE  l_shipdate <  DATE '1995-01-01'
  AND  l_shipdate >= DATE '1994-01-01';
```

```sql
CREATE TEMPORARY TABLE temp.main."parent_pushdown" AS
SELECT *
FROM   "dev"."main"."parent"
WHERE  l_shipdate <  DATE '1995-01-01'
  AND  l_shipdate >= DATE '1994-01-01';
```



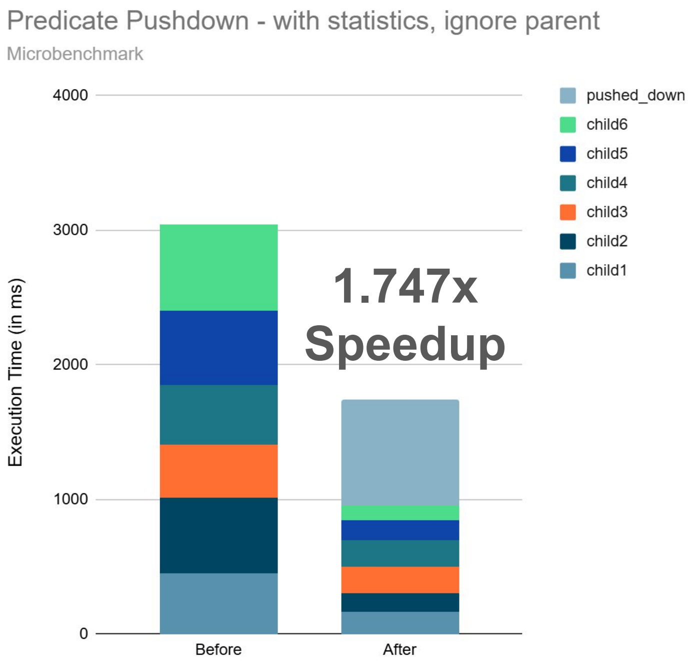
Solution: Use Statistics to Determine

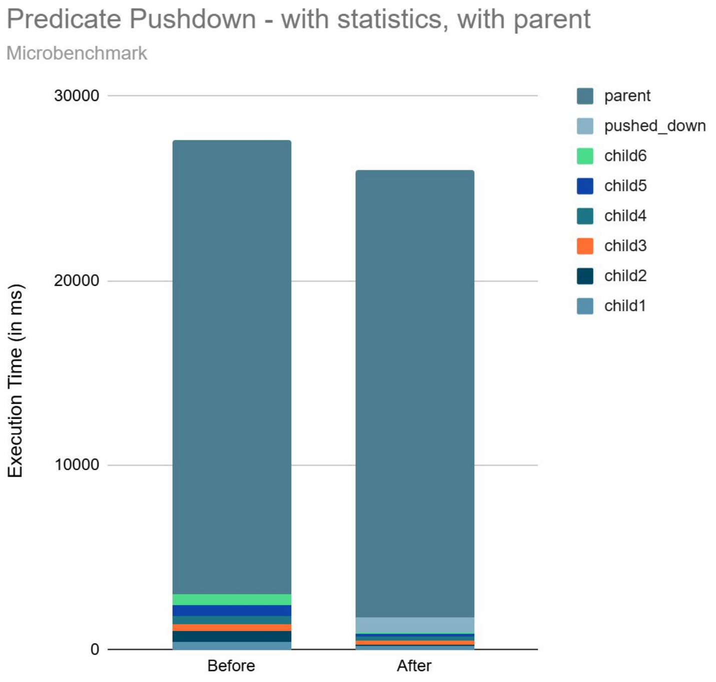
- **Still, adding an intermediate node yields an extra cost**
  - We may want to add an intermediate node only when predicate push-down yields a lower total cost (i.e., if adding predicates saved overall costs, safe to add intermediate node)
- Approximate cost reduction with selectivity of Δp and the number of children
  - Query DuckDB for statistics



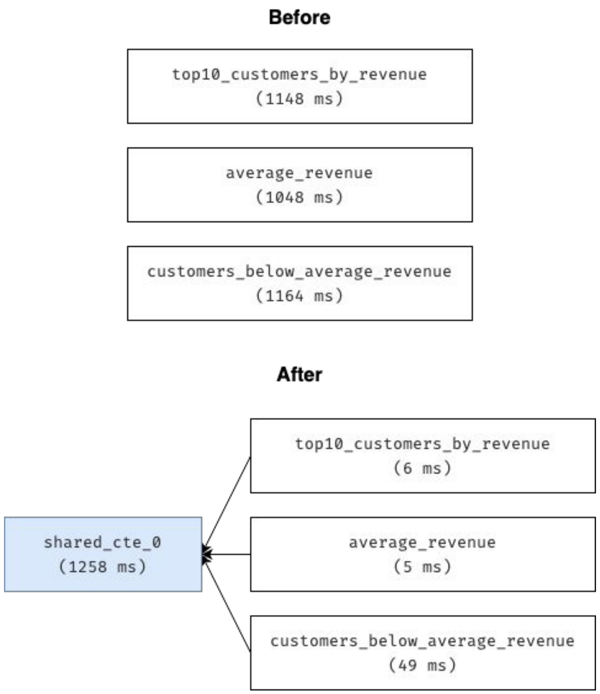
Predicate Pushdown: Non-pushable (Solution Applied)

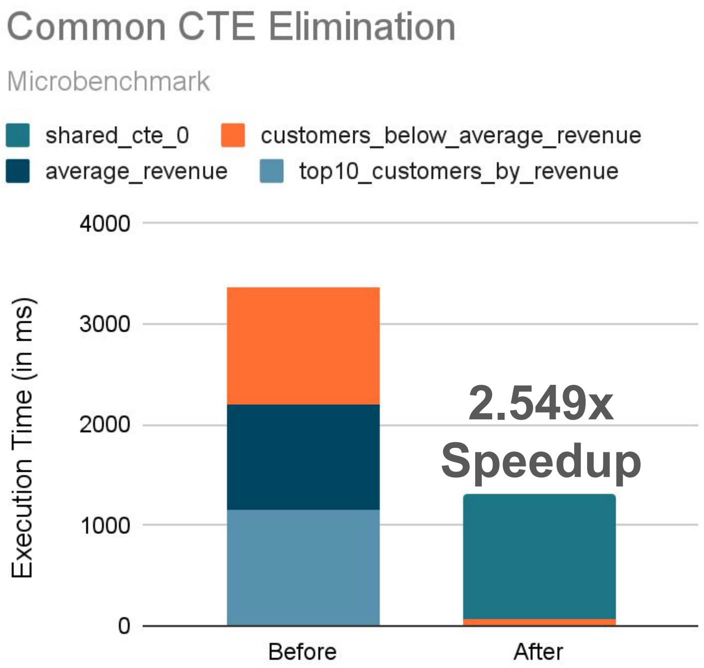
- Performance impact only visible on children





Common CTE Elimination: Result

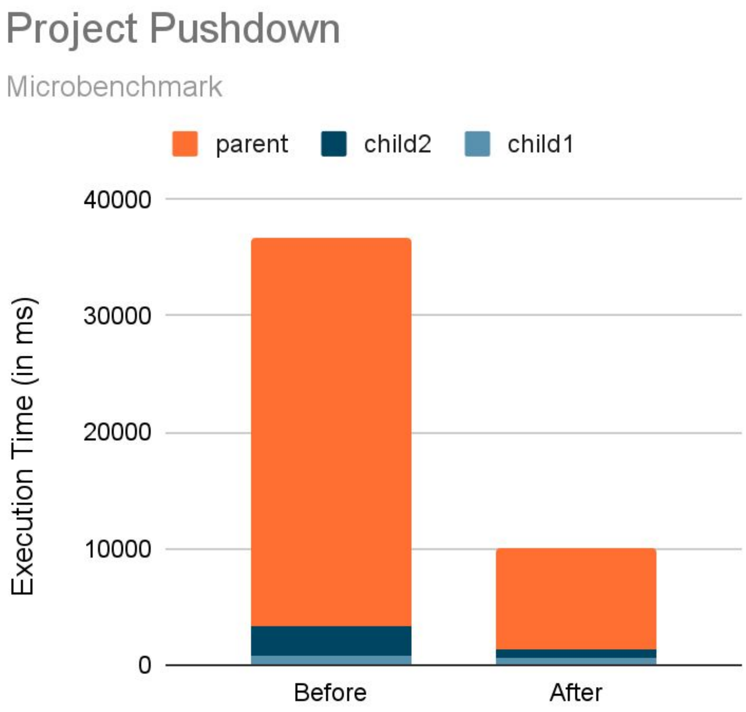




Performance improvement depends on the CTE (Similar to the predicate pushdown)
(i.e., materializing CTE incurs overheads that may make performance worse)

Projection Pushdown: Result

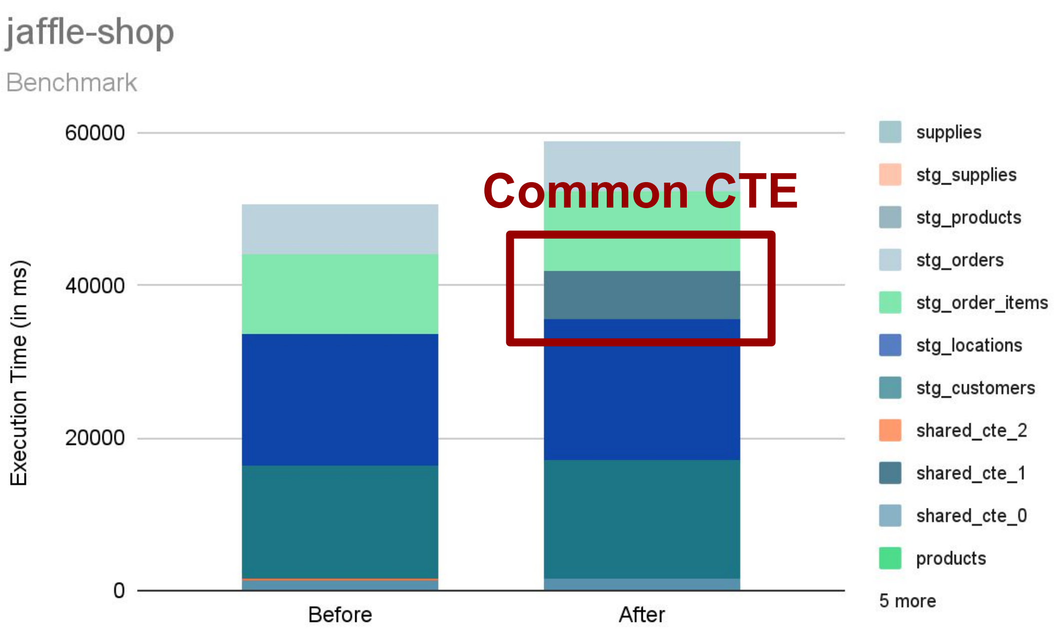
- Assume that the parent-node is push-able
  - Less materialization on parent



Preliminary Result: Work not quite well on jaffle-shop :(

- **Only Common CTE Elimination can be applied**

- However, the common CTEs do not filter any data
    - **The overhead of materializing CTE makes worse execution time**
    - Our rules may help on other workloads, but trade-offs must be weighed carefully



Code Quality Discussion

- **Modularized Components ✅:**
  - Abstract Rules (match(), apply())
    - Very extensible
  - Logical rewrite: stages loosely coupled, easy rule registration
  - Execution (performance + optional correctness check)
    - Flexible (no dependency / hardcoding)

- Need further work to work with dbt execution modules
  - We wrote our own execution module to run the queries

Conclusion

- **Performance can be improved significantly** by carefully rewriting DAGs
  - Rewriting through heuristics such as predicate pushdown, common CTE eliminations, projection pushdown
- **Not all the heuristics should be applied**: adding a node in a DAG **may incur high overheads** that it may not worth doing so
  - We demonstrate that by having simple statistics, we are able to heuristically determine whether we should adding a node or not
- Complex DAG **may require more complicated heuristics** (e.g., join sharing)

Future Work

- **Better heuristics** (e.g., refine projection pushdown)
- **More heuristics** (e.g., join sharing)
- **Cost-based optimization**
  - Intermediate materialization cost vs Saved I/O from all children nodes
- **Demand-driven** (push down if user do not really use that table)
  - Lazy transformation for original parent table
- **Evaluate with larger DAGs** (e.g., Gitlab’s dbt)