UTN Technische Universität Nürnberg

Welcome to Berlin

Pruning in Snowflake: Working Smarter, Not Harder

Andreas Zimmerer, Damien Dam, Jan Kossmann, Juliane Waack, Ismail Oukid, Andreas Kipf

SIGMOD 2025, Berlin

Terminology: Partition Pruning

partition

/ pɑrˈtɪʃ ən, pər- /

noun:

- Table is **split horizontally** into partitions
- With columnar **min/max metadata** (zone maps)
- Independently loadable

synonyms:

(data-)block, row-group (Parquet)

pruning

/ ˈpru nɪŋ /

verb:

- **Removing partitions** from the *scanset*
- Based on external **metadata**
- Before data is fetched from storage

synonyms:

skipping, removing

Why Is It Important?

- Reduced CPU time & reduced (network) IO
  → speedups of more than 100x for real customer queries just due to this
- Better cardinality estimation
  → better join ordering and resource allocation
- Further query optimizations
  → e.g., subquery elimination, join elimination, constant folding, …

Disclaimer

1. Everything presented here **also works with Iceberg tables** in Snowflake.
2. All presented pruning techniques only require **basic min/max metadata**.
3. We look **only at SELECT queries** in this presentation (no DMLs).

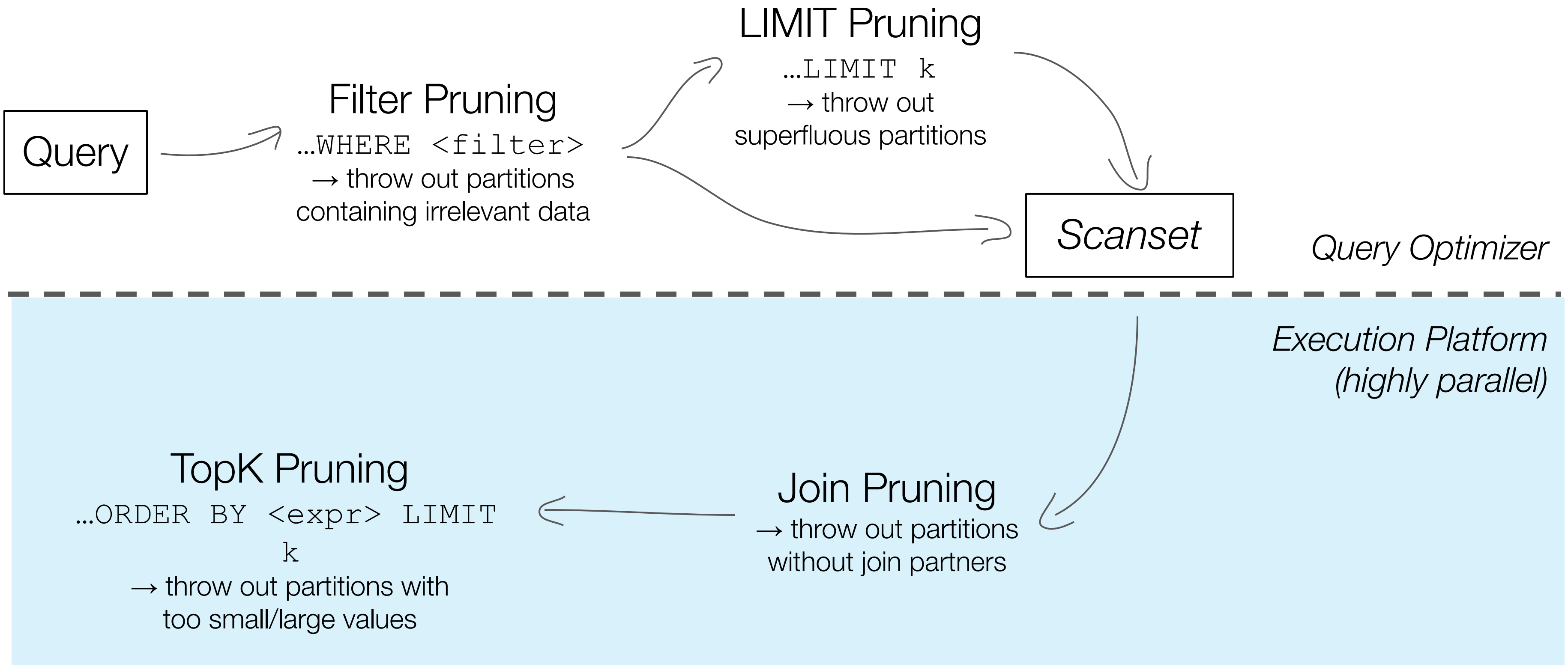
Quiztime!

How many % of partitions are pruned in Snowflake?
(across all customers and all query types)

99.4%

How?
(only with zone maps
and smartness)

The Pruning Flow



Filter Pruning

Common technique for simple predicates.
But it quickly becomes complex:

```
name LIKE 'Marked-%-Ridge'
AND IF(unit='feet', altit * 0.3048, altit) > 1500
```

- Imprecise filter rewrites
- Support complex predicate trees

$$\boxed{\mathtt{name}_{min} \texttt{ <= 'Marked-' <= } \mathtt{name}_{max}} \land ($$

$$\boxed{\begin{array}{l} ((\mathtt{altit}_{min} = \texttt{'feet'} \land \mathtt{altit}_{max} = \texttt{'feet'}) \rightarrow \\ \quad \mathtt{altit}_{max} * 0.3048 > 1500) \lor \\ ((\mathtt{altit}_{max} < \texttt{'feet'} \lor \mathtt{altit}_{min} > \texttt{'feet'}) \rightarrow \\ \quad \mathtt{altit}_{max} > 1500) \lor \\ ((\mathtt{altit}_{max} \geq \texttt{'feet'} \land \mathtt{altit}_{min} \leq \texttt{'feet'}) \rightarrow \\ \quad \max(\mathtt{altit}_{max} * 0.3048, \mathtt{altit}_{max}) > 1500)) \end{array}}$$

Problem: Predicates can be complex.
→ Pruning can take a significant amount of time during query optimization.

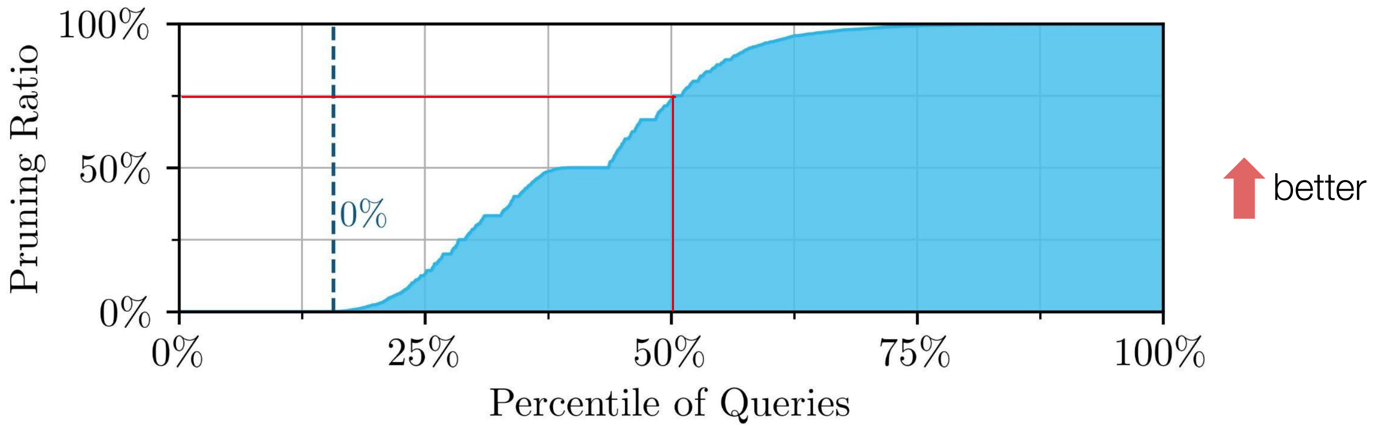
Is pruning worth it?

Solution:

- Adaptive **re-ordering** of pruning steps, prioritizing fast and effective expressions
- **Defer pruning** with slow expressions to highly parallel execution stage

Filter Pruning - Impact

Pruning ratios of SELECT queries with predicates:



50% of queries with predicates prune > 75%

Zimmerer et al. “Pruning in Snowflake: Working Smarter, Not Harder”, SIGMOD’25, Berlin

LIMIT Pruning

We all know:

1. Send query to execution platform
2. Execute query
3. Stop execution when k rows are in the result

Hidden costs:

Scanset (de)serialization
+ scheduling "heavy" query

Start X compute nodes
+ communication + fast abort

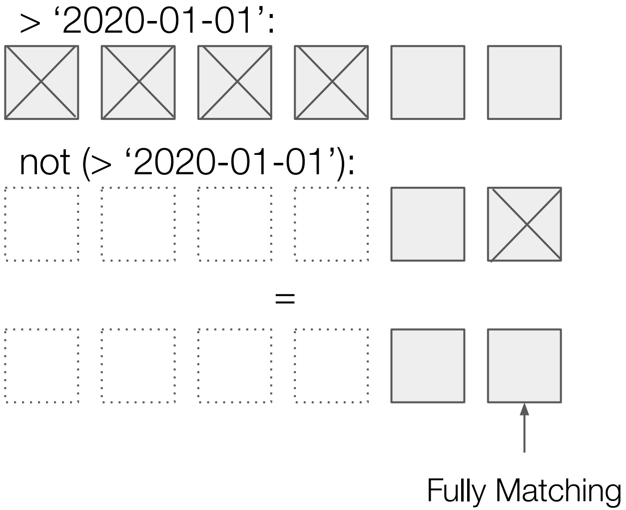
We *might* need to do this for selective predicates,
but we can do better if we know better.

LIMIT Pruning

2.60% of SELECT queries are LIMIT queries, e.g.:

```
… WHERE timestamp > '2020-01-01' LIMIT 10
```

- Filter pruning will remove partitions < 2020
- Thousands of partitions might still remain
- We need only one “sufficient” partition to answer the query!



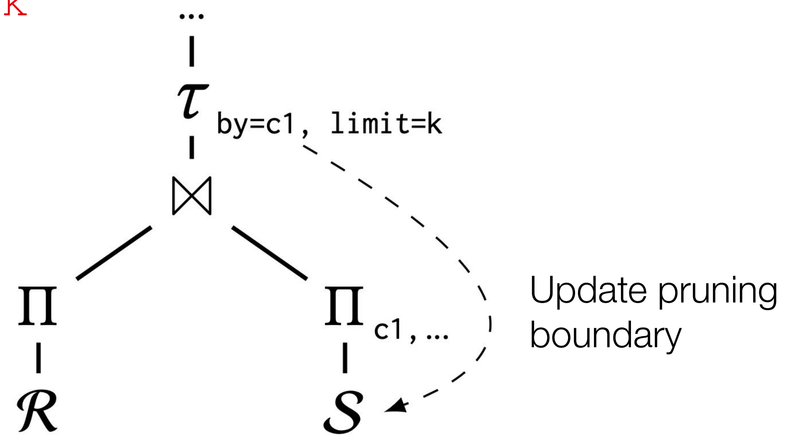
Zimmerer et al. “Pruning in Snowflake: Working Smarter, Not Harder”, SIGMOD’25, Berlin

TopK Pruning

5.55% of all SELECT queries are top-k queries, e.g.:

```
… [WHERE <filter>] ORDER BY c LIMIT k
```

- Heap-based TopK processing:
  O(n log k) ≈ O(n)
- Prune with smallest value in TopK-heap
  → push down to table scan



TopK Pruning

Challenges:

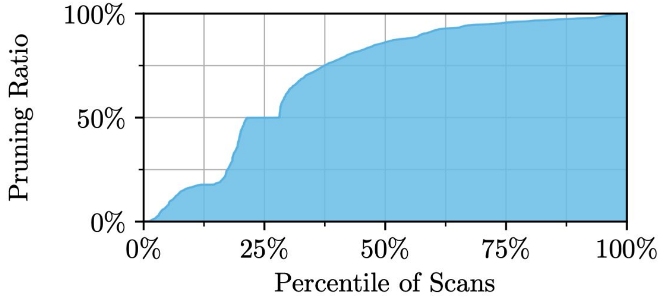
- Through which operators can we push this information (correctness)?
  Probe side, build side of left outer join, filters, …
- Can we support GROUP BY … ORDER BY … LIMIT k?
  Yes, in parts: GROUP BY c1, c2 ORDER BY c1 LIMIT k
- Taking things further: Smart scan order, pre-initialize the boundary value, ….

TopK Pruning - Impact

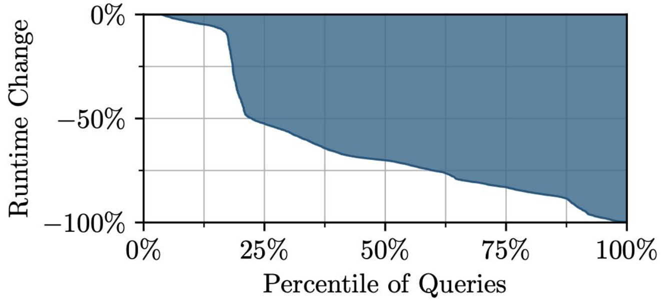
Pruning ratios

SELECT queries with successfully applied TopK pruning

Runtime impact



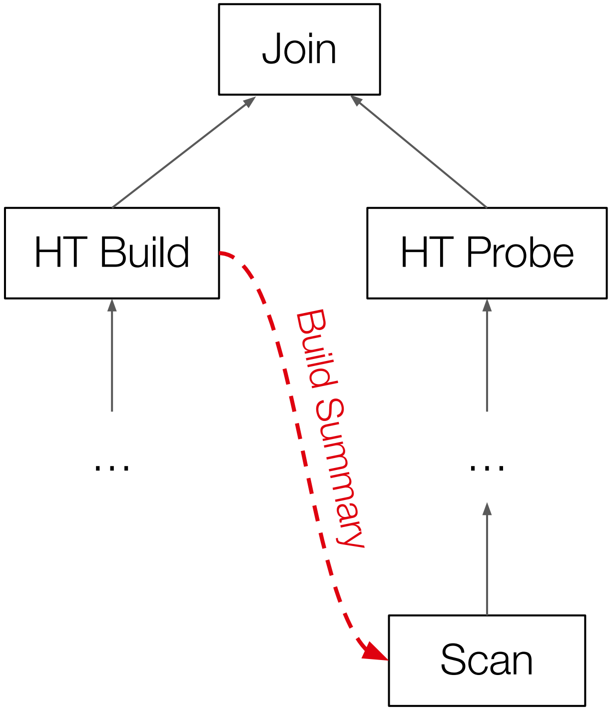
better



better

Zimmerer et al. “Pruning in Snowflake: Working Smarter, Not Harder”, SIGMOD’25, Berlin

Join Pruning



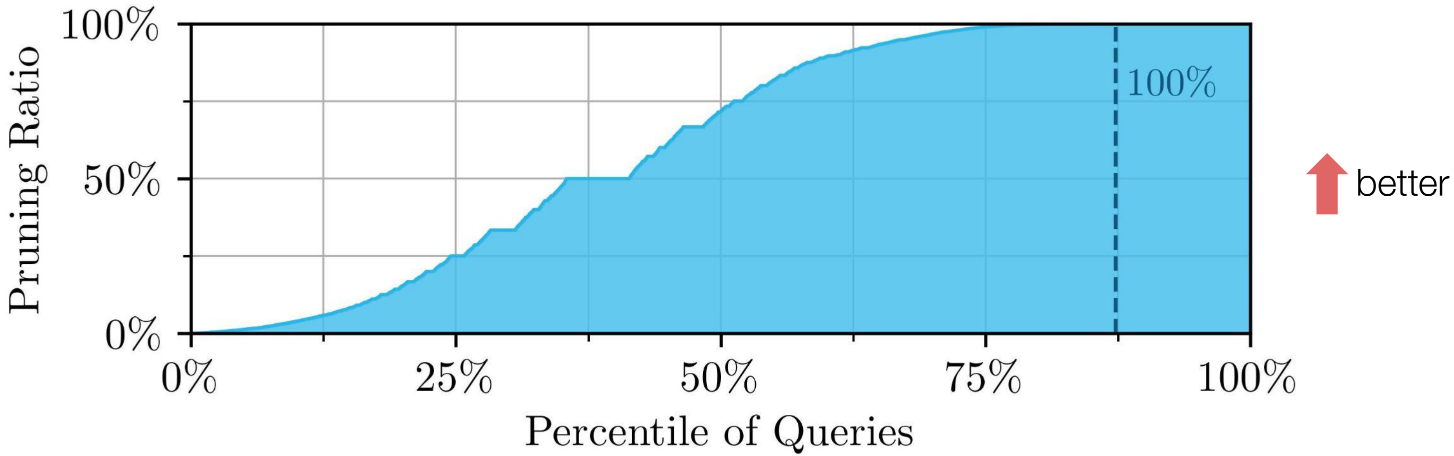
Conceptually similar to SIP with Bloom-filters.

- Bloom-filters only allow row-wise filtering and therefore only save CPU time
- Snowflake summarizes the build side in a way that allows pruning on partition-level ➜ reduces both CPU time <u>and</u> IO

Zimmerer et al. "Pruning in Snowflake: Working Smarter, Not Harder", SIGMOD'25, Berlin

Join Pruning - Impact

Pruning ratios of SELECT queries that successfully used Join Pruning:



Conclusion

Aggressive partition pruning is a major performance driver for query processing.

We propose new specialized pruning techniques for LIMIT, TopK, and Join queries.

We assess the individual and combined impact of pruning techniques.

Contact: andreas.zimmerer@utn.de