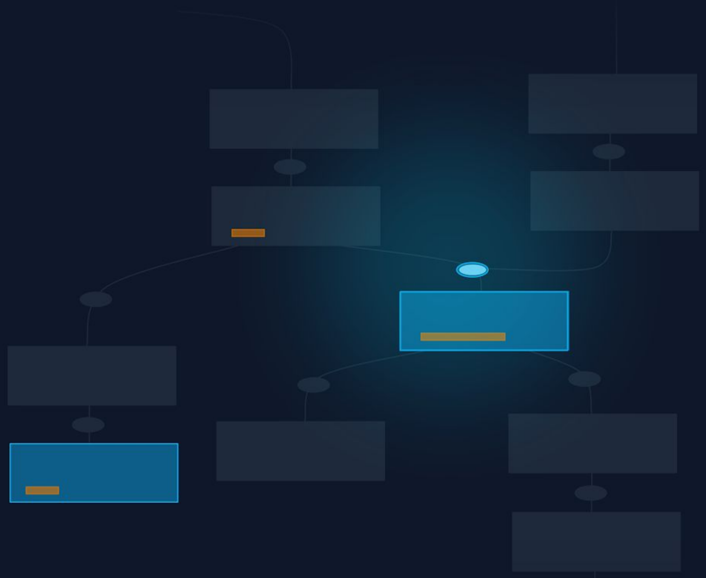


The Missing Manual

Everything you need to know about Snowflake optimization

Ian Whitestone & Niall Woodward

Data Council - March 28, 2023

SELECT

👋 Hello!

Ian Whitestone
ian@select.dev

Niall Woodward
niall@select.dev

SELECT

Why are we here?

- ~~🌮 Tacos~~
- End of the longest bull run in history
- Data teams are increasingly being asked to better understand, monitor and reduce their warehouse spend
- Snowflake is the market leader, with many cost and performance levers available

"sleeping bull" by Midjourney

SELECT

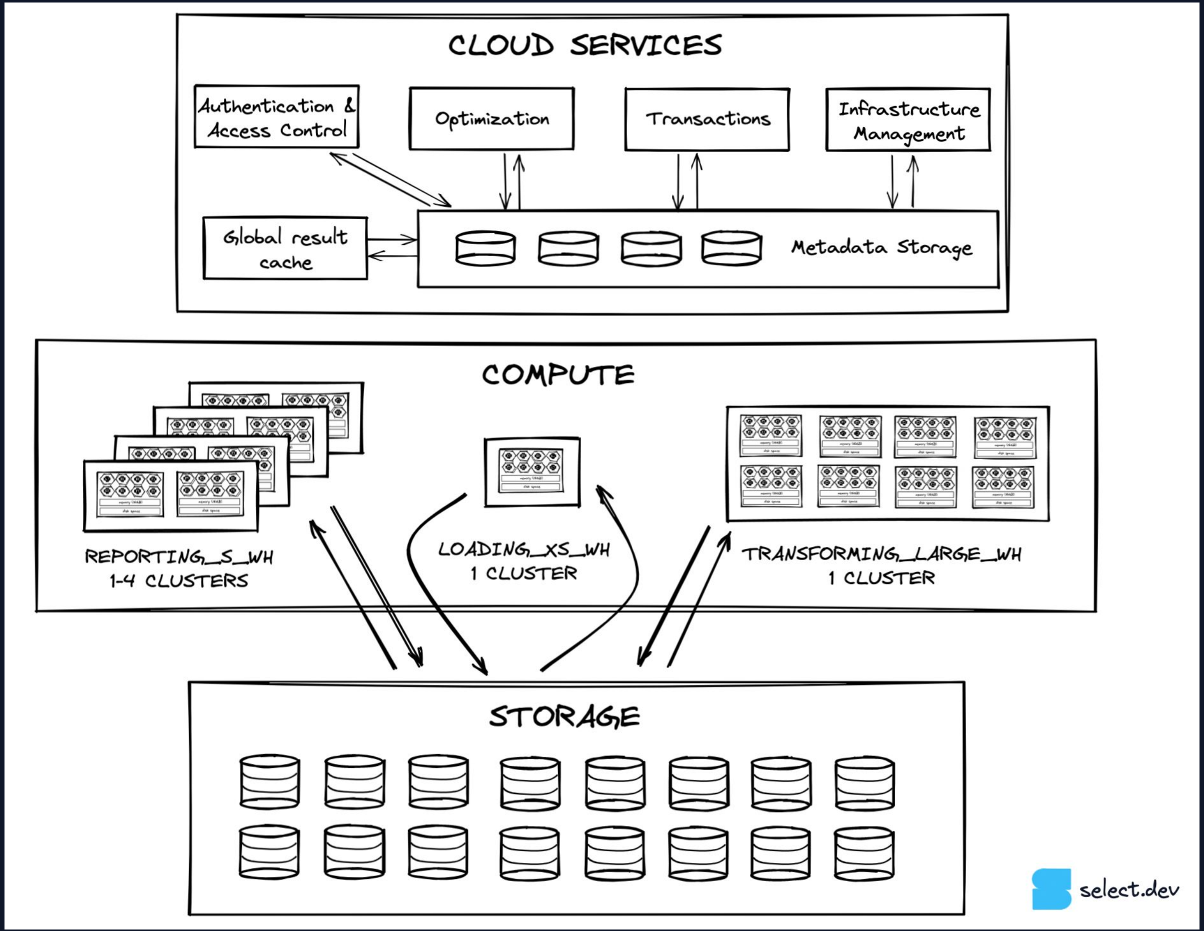
Agenda

- Snowflake architecture overview
- How to lower costs
- How to optimize performance
- Next steps

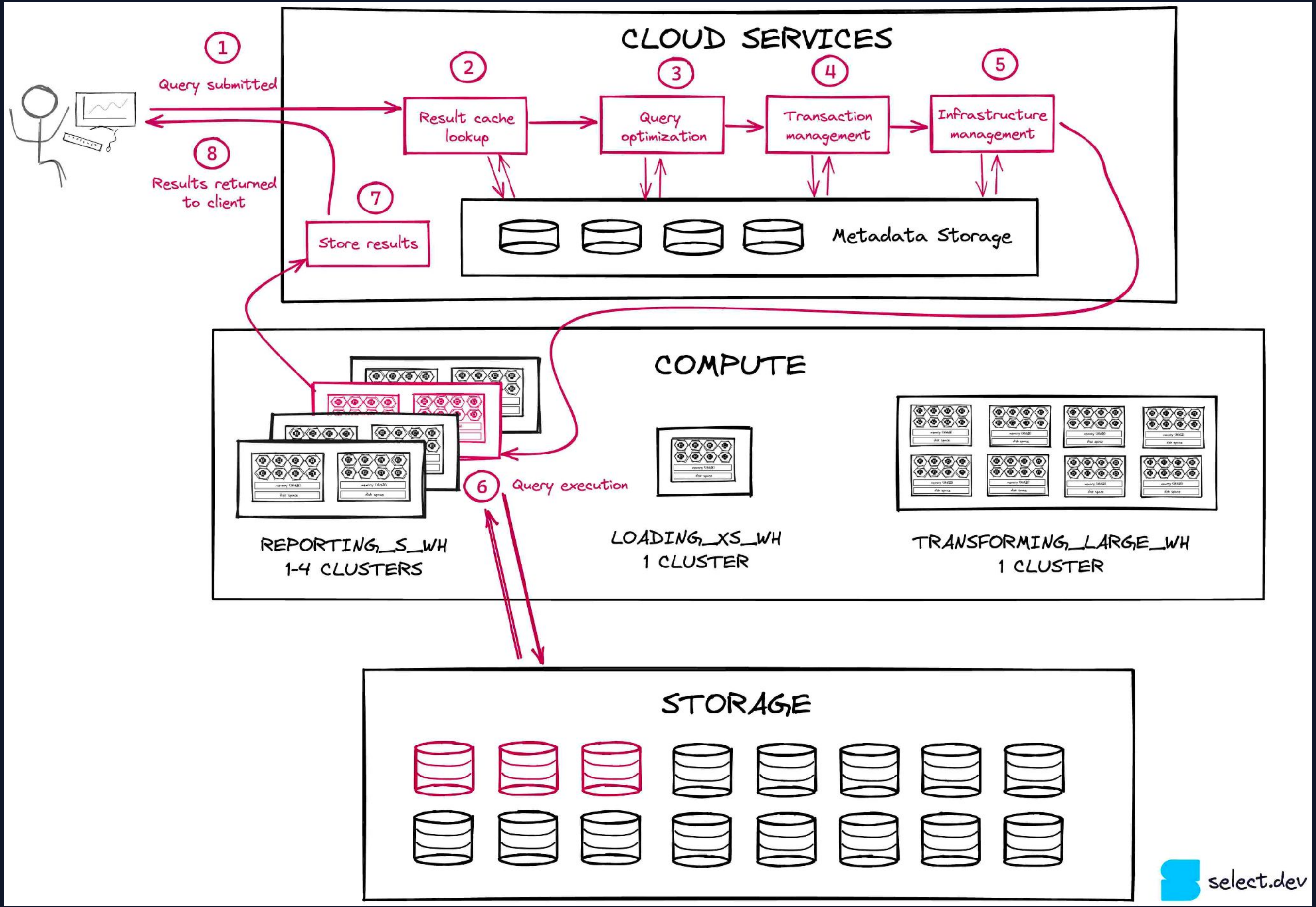
Snowflake Architecture

“arctic cloud data warehouse” by Midjourney

SELECT

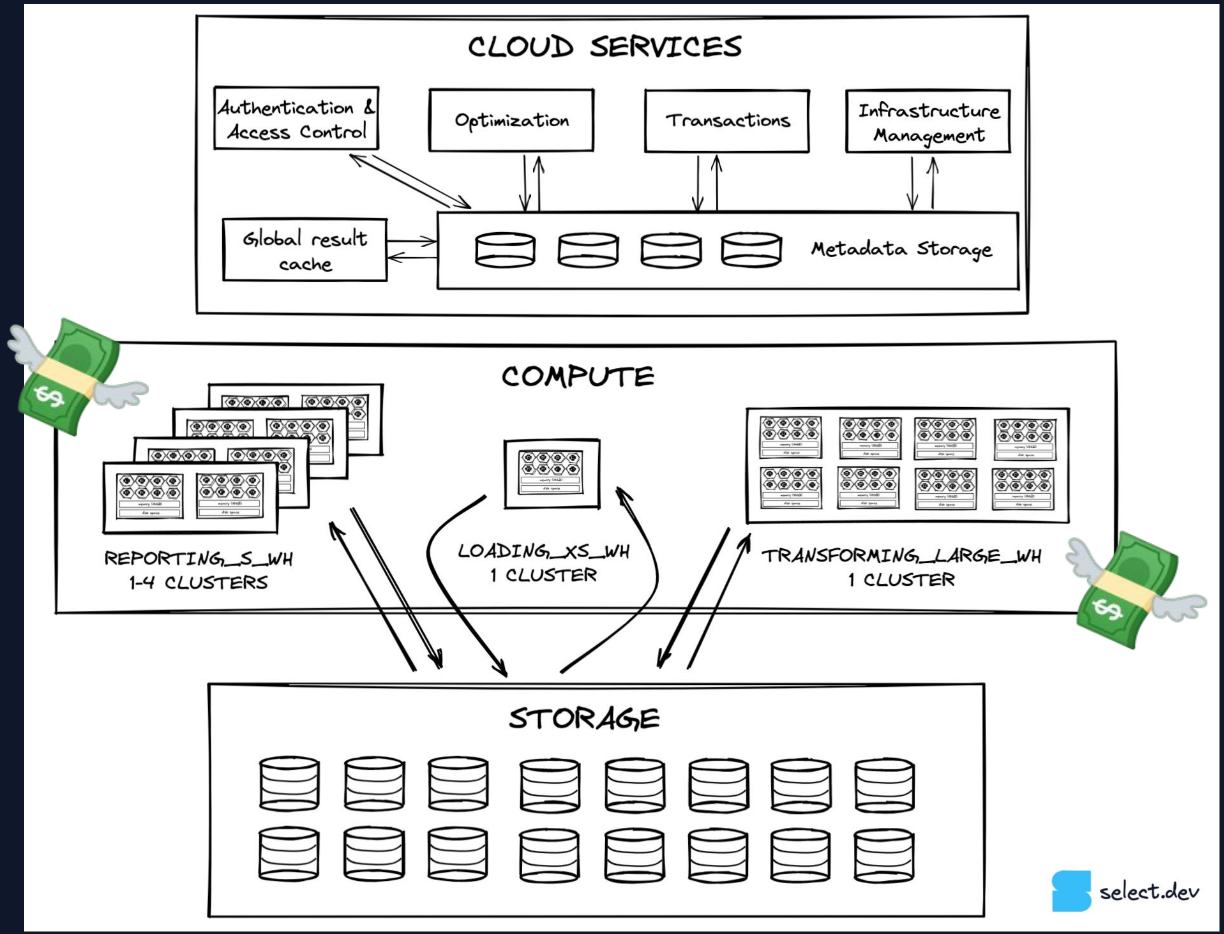

CLOUD SERVICES
Authentication & Access Control
Optimization
Transactions
Infrastructure Management
Global result cache
Metadata Storage
COMPUTE
REPORTING_S_WH
1-4 CLUSTERS
LOADING_XS_WH
1 CLUSTER
TRANSFORMING_LARGE_WH
1 CLUSTER
STORAGE
select.dev

SELECT



CLOUD SERVICES
1
Query submitted
2
Result cache lookup
3
Query optimization
4
Transaction management
5
Infrastructure management
8
Results returned to client
7
Store results
Metadata Storage
COMPUTE
6
Query execution
REPORTING_S_WH
1-4 CLUSTERS
LOADING_XS_WH
1 CLUSTER
TRANSFORMING_LARGE_WH
1 CLUSTER
STORAGE
select.dev

SELECT


CLOUD SERVICES
Authentication & Access Control
Optimization
Transactions
Infrastructure Management
Global result cache
Metadata Storage
COMPUTE
REPORTING_S_WH
1-4 CLUSTERS
LOADING_XS_WH
1 CLUSTER
TRANSFORMING_LARGE_WH
1 CLUSTER
STORAGE
select.dev

SELECT

How to lower costs

“different sized computers in a row” by Midjourney

SELECT

How to lower costs

1. Understand Snowflake billing model
2. Optimize virtual warehouse configuration
3. Consolidate warehouses

Compute Billing Model

- Only pay while virtual warehouses are active
- Per-second billing ($2-$4/credit)
  - X-Small consumes 1 credit / hour
  - Small consumes 2 credit / hour
  - …doubles with each size
- Minimum 60-seconds billed each time warehouse is resumed

SELECT

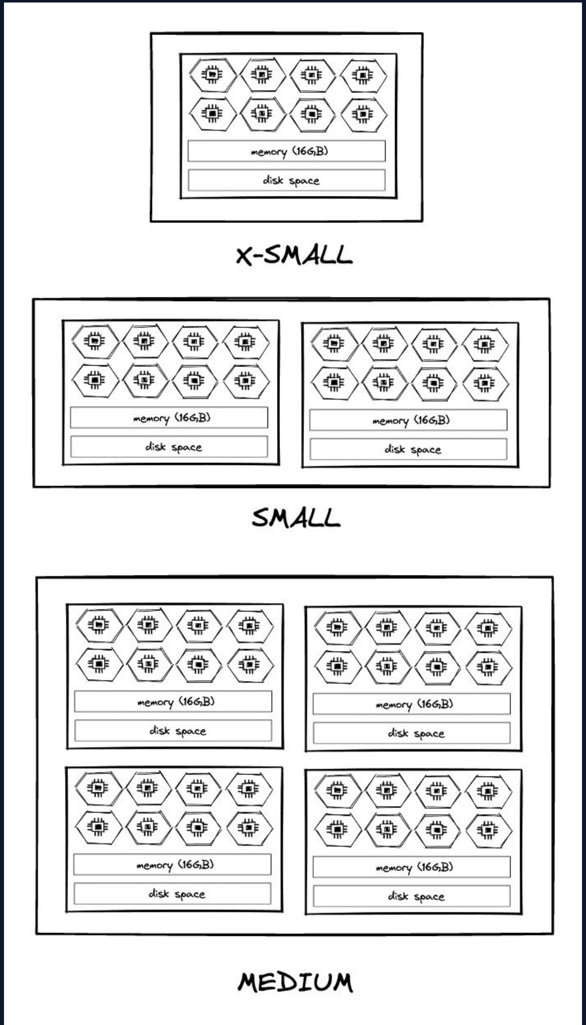
How to lower costs

1. Understand Snowflake billing model
2. Optimize virtual warehouse configuration
3. Consolidate warehouses

SELECT

What are virtual warehouses?

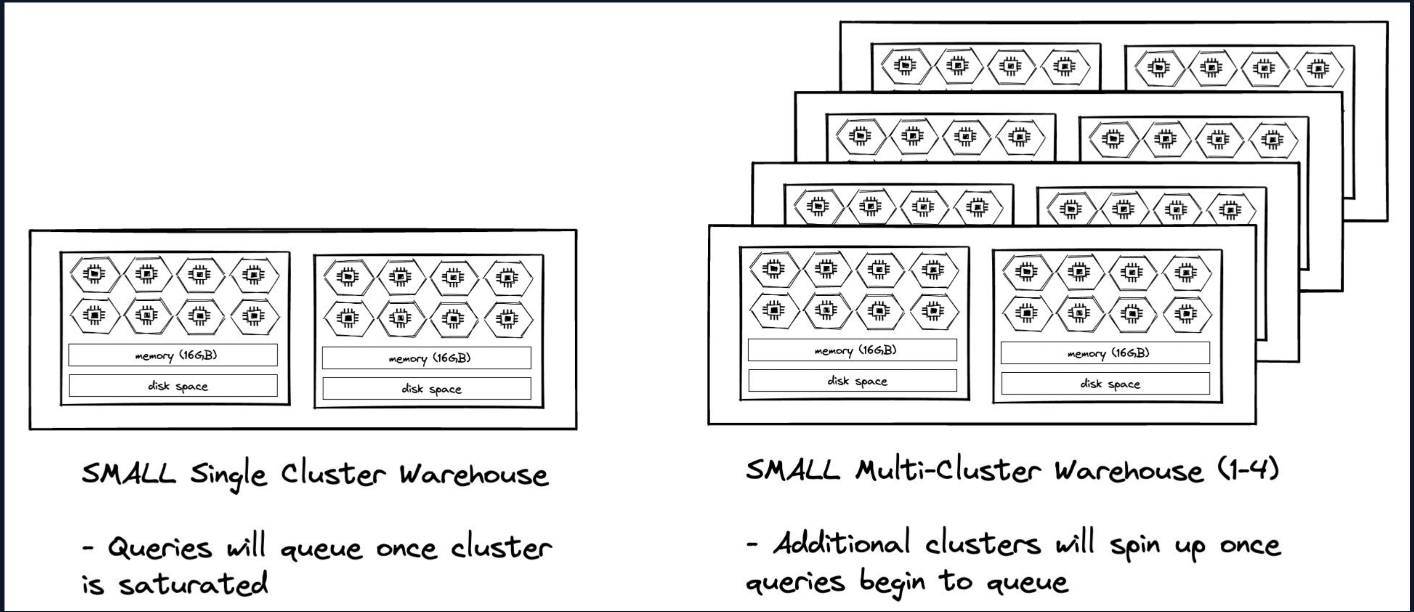
- Abstraction over compute instances
- Each instance has 8 cores/threads, 16GB of RAM, and local SSD
- T-shirt sizes - XS -> 6XL
- Each size doubles compute resources and cost - scaling ‘up’



SELECT

Multi-cluster warehouses

Scale ‘out’ to process variable query volumes, e.g. peak hours



SELECT

Recommended Warehouse Configuration

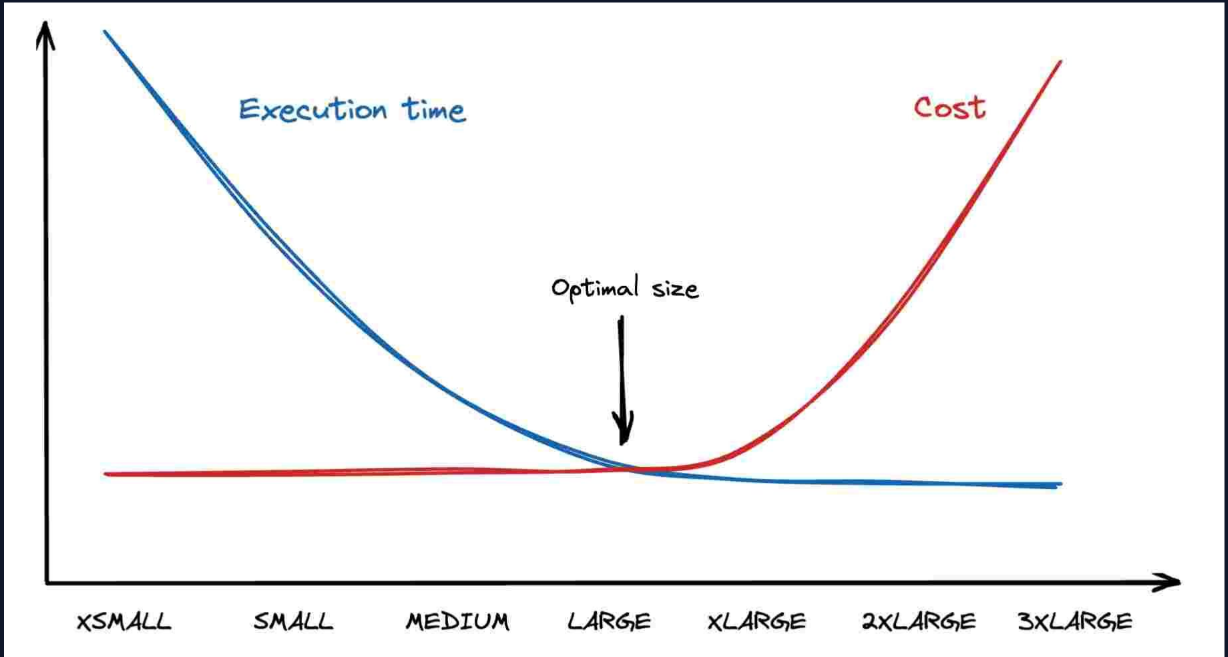
- Start with an X-Small, single cluster warehouse
- Set max_cluster_count to satisfy peak concurrency needs
- 60s auto-suspend
- Set a query timeout (default is 2 days!)
- Resource monitor to alert on spikes

Warehouse Sizing

- Reduce warehouse size and max_cluster_count for workloads which can tolerate some queueing e.g. data loading
- Use per-model warehouse configuration in dbt vs increasing warehouse size for entire project
- Larger warehouses can improve performance at minimal additional cost, especially with remote disk spillage

Warehouse Sizing

- Larger warehouses improve performance at low additional cost – up to a point

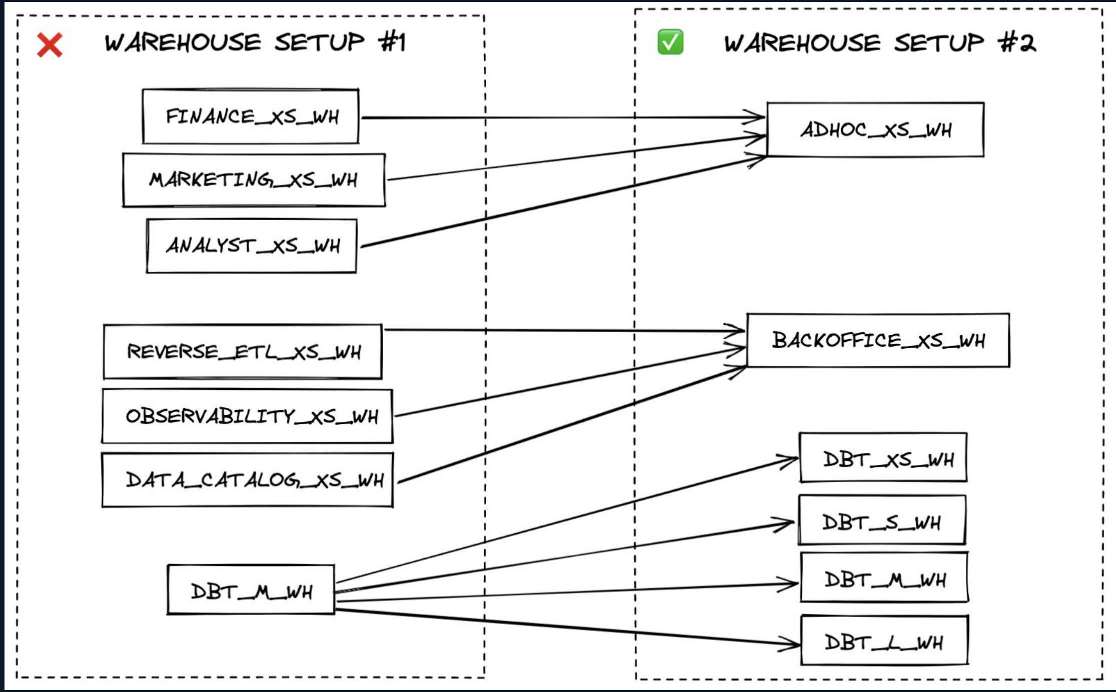


How to lower costs

1. Understand Snowflake billing model
2. Optimize virtual warehouse configuration
3. Consolidate warehouses

Consolidate Warehouses

- Fewer warehouses -> less idle time
- Speeds up queries due to caching
- Separate by workload requirements, not domain



SELECT

Optimizing performance

Pruning and clustering

“thousands of tiny files” by Midjourney

SELECT

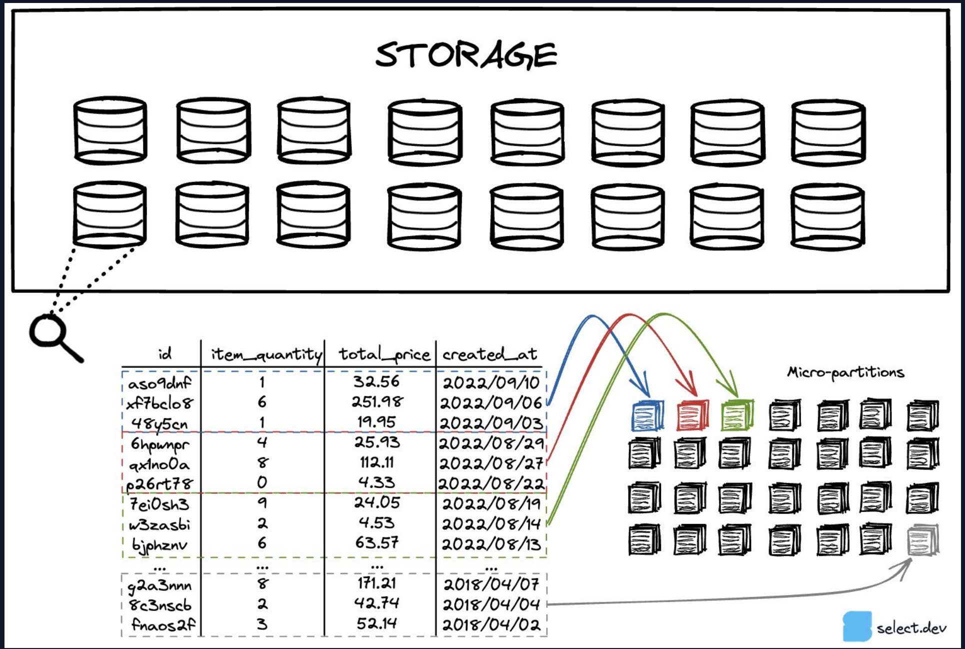
Optimizing performance
Pruning and clustering

1. Micro-partitions
2. Pruning
3. Clustering

SELECT

Micro-partitions

- Tables are stored in cloud storage as micro-partitions
- Micro-partitions are a proprietary, closed-source file format created by Snowflake
- Heavily compressed and ~16MB each
- DML operations (updates/inserts/deletes) add/remove entire files



Micro-partition metadata

Snowflake stores column level statistics in the cloud services layer

Metadata stored for a single micro-partition in cloud services layer

select.dev

	id	item_quantity	total_price	created_at
count distinct	3	2	3	3
min	48y6cn	1	19.95	2022/09/01
max	xf7bclo8	6	251.98	2022/09/10

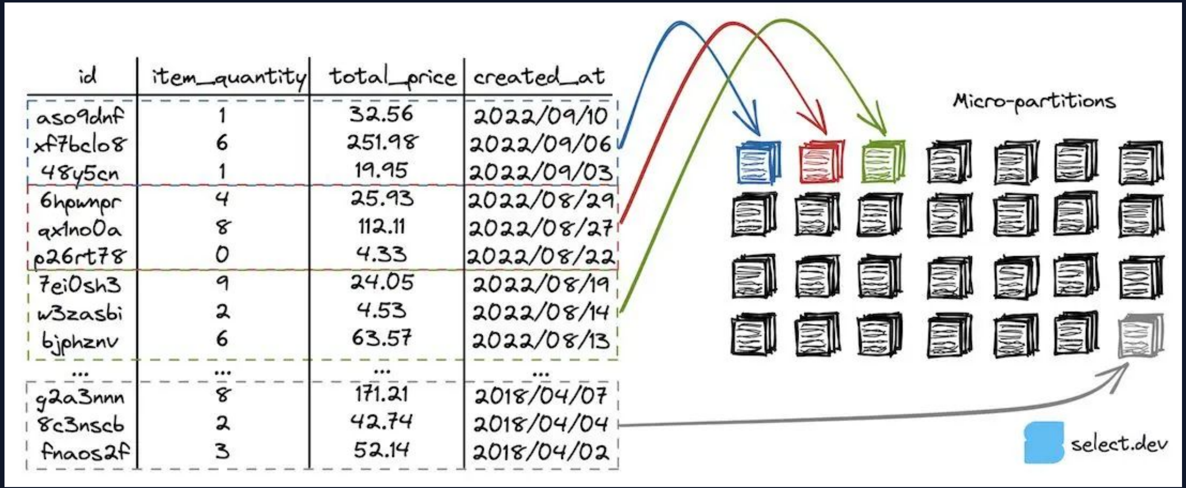
Optimizing performance
Pruning and clustering

1. Micro-partitions
2. Pruning
3. Clustering

SELECT

Pruning - every fast query's secret

```sql
select *
from orders
where created_at > '2022/08/14'
```

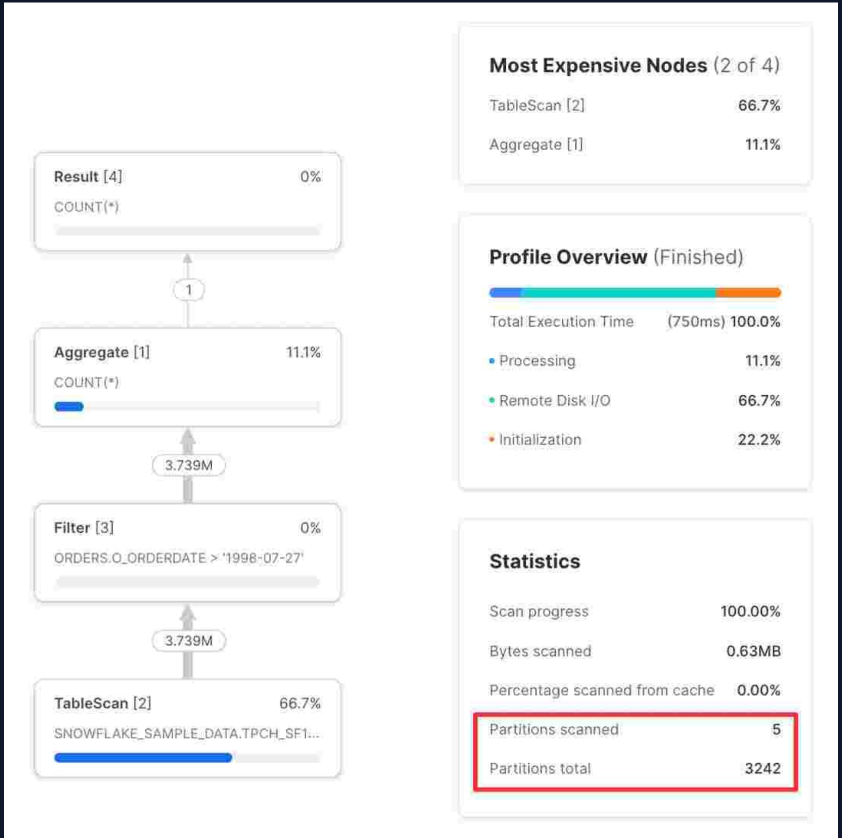


id	item_quantity	total_price	created_at
aso9dnf	1	32.56	2022/09/10
xf7bclo8	6	251.98	2022/09/06
48y5cn	1	19.95	2022/09/03
6hpwnpr	4	25.93	2022/08/29
ax1no0a	8	112.11	2022/08/27
p26rt78	0	4.33	2022/08/22
7ei0sh3	9	24.05	2022/08/19
w3zasbi	2	4.53	2022/08/14
bjphznv	6	63.57	2022/08/13
...	...	...	...
g2a3nnn	8	171.21	2018/04/07
8c3nscb	2	42.74	2018/04/04
fnaos2f	3	52.14	2018/04/02

- Snowflake checks which partitions contain the relevant data
- In this example, only three 3 micro-partitions are read

Check for pruning using the Query Profile

- Query profile shows only 5 partitions scanned out of the 3242 present for the table
- Info also available in query history view



SELECT

Optimizing performance

Pruning and clustering

1. Micro-partitions
2. Pruning
3. Clustering

Clustering

- Describes the distribution of data across a table's micro-partitions
- A 'well-clustered' column has a small range of values per micro-partition for that column
- Snowflake can prune well when queries filter on that column

Clustering methods

- Natural Clustering
  - Leverage wherever possible
- Automatic Clustering Service
  - Use where a table is commonly filtered by a column which isn't the 'natural' clustering key
- Manual Sorting
  - Useful for one-off clustering at lowest cost

Finding good candidates for clustering

- Columns used frequently in ‘where’ clauses
- Column should have a large enough number of distinct values to enable effective pruning on the table
    - i.e. clustering on a categorical column with 2 distinct values will only achieve ~50% pruning
- Use the query history + access history views to determine usage patterns

Optimizing performance

Query design

"fast running computer" by Midjourney

SELECT

Optimizing performance
Query design

1. Before you begin...
2. Fastest way to process data? Don’t!
3. Use clustered columns in join predicates
4. Explicitly list columns in CTEs
5. Filter early

Before you begin...

- What's the expected ROI?
- Does your query need to run every hour?
    - Is anyone looking at the dashboard multiple times per day?
    - If a data models costs $10,000/year running hourly, switching to daily can drop costs by ~95%

Optimizing performance
Query design

1. Before your begin...
2. Fastest way to process data? Don't!
3. Use clustered columns in join predicates
4. Explicitly list columns in CTEs
5. Filter early

Fastest way to process data? Don’t!

1. Ensure query is pruning out unneeded micro-partitions
    - Pruning works with CTEs & subqueries
    - Can fail when applying functions on predicates, type conversions, deeply nested views, table has degraded clustering health
    - Always validate by checking query profile/history
2. Use incremental materializations for larger datasets

Optimizing performance
Query design

1. Before your begin…
2. Fastest way to process data? Don't!
3. Use clustered columns in join predicates
4. Explicitly list columns in CTEs
5. Filter early

Use clustered columns in join predicates

- Snowflake uses values from one side of join to enable pruning
- Applies to joins and merges

```
merge into orders
using orders_tmp
on target.order_key=orders_tmp.order_key
and target.order_date=target.order_date -- additional predicate enables pruning
when matched then
  update set orders.total_price=orders_tmp.total_price
```

Optimizing performance

Query design

1. Before your begin...
2. Fastest way to process data? Don't!
3. Use clustered columns in join predicates
4. Explicitly list columns in CTEs
5. Filter early

Column pruning doesn't always work with CTEs

- Column pruning prevents unneeded columns from being read
- Column pruning stop working when CTEs are referenced more than once or when used in join
  - Ensure required columns are explicitly listed in CTEs

```
with active_users as (
  select *
  from users
  where is_active
)
...
```

⚠️

```
with active_users as (
  select
    id,
    created_at
  from users
  where is_active
)
...
```

✅

SELECT

select.dev/posts/should-you-use-ctes-in-snowflake

Optimizing performance
Query design

1. Before your begin…
2. Fastest way to process data? Don’t!
3. Use clustered columns in join predicates
4. Explicitly list columns in CTEs
5. Filter early

SELECT

Filter early

- Most of the time, Snowflake pushes down filters
- In certain cases it can't
  - Qualify filter happens post join, but should be applied before in a CTE

```sql
SELECT
    client_inventory.is_active,
    client_inventory.quantity,
    client_inventory.supplier_cost,
    client_inventory.client_sku,
    client_inventory.provider,
    client_inventory.client_id,
    sku_mapping.internal_sku,
    inventory.updated_at
FROM client_inventory
LEFT JOIN sku_mapping
    ON
        client_inventory.client_id = sku_mapping.client_id
        AND client_inventory.client_sku = sku_mapping.client_sku
LEFT JOIN products
    ON
        client_inventory.client_id = products.client_id
        AND client_inventory.client_sku = products.client_sku
-- Pick the latest value for each SKU
QUALIFY
    ROW_NUMBER() OVER (
        PARTITION BY
            client_inventory.client_sku, client_inventory.client_id
        ORDER BY client_inventory.updated_at DESC
    ) = 1
ORDER BY sku_mapping.internal_sku
```

SELECT

Next Steps

"polar bear on a computer" by Midjourney

SELECT

Bootstrap Cost & Performance Observability

- Understanding virtual warehouse cost drivers is critical
- Use our dbt package dbt-snowflake-monitoring
  - Cost per query, cost per dbt model, etc.
- Create dashboards for monitoring, alerts for big spikes
- Review monthly/quarterly

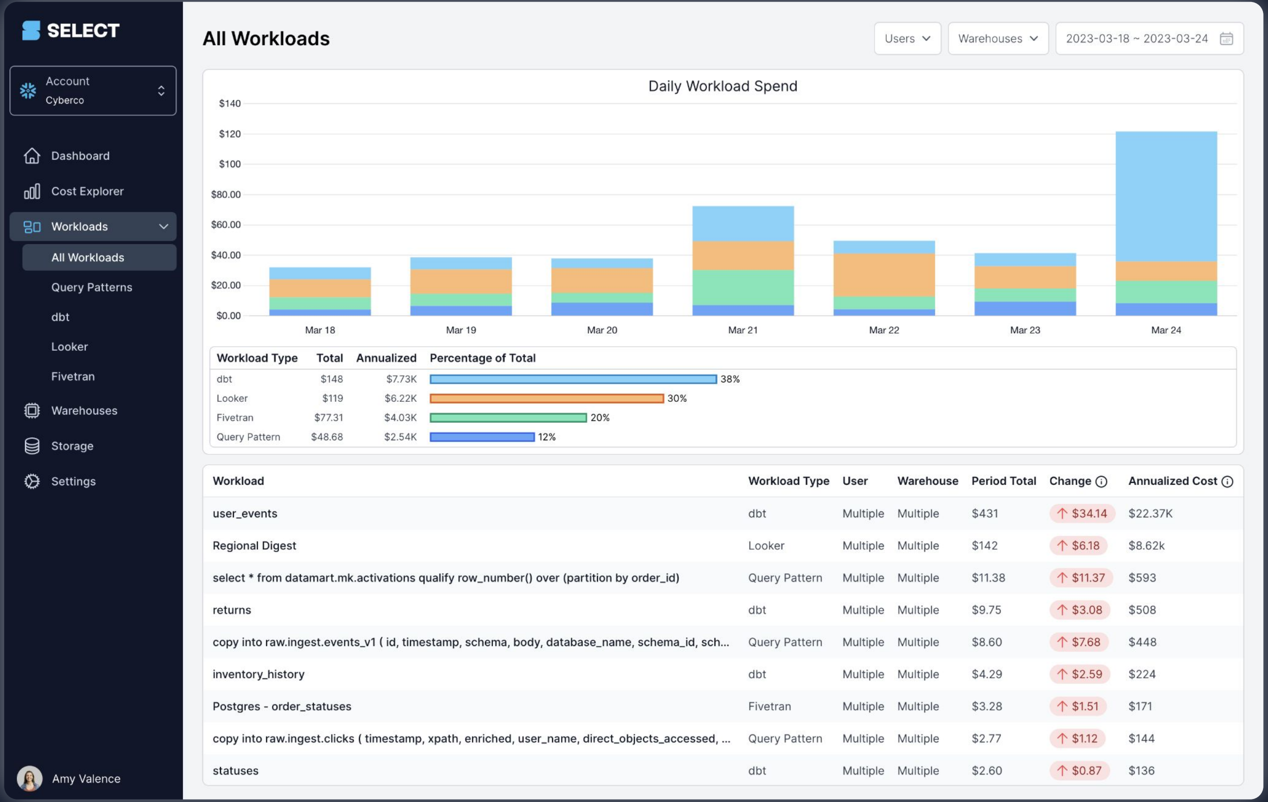
select.dev/posts/cost-per-query
github.com/get-select/dbt-snowflake-monitoring

Use SELECT

Lower Costs

Save Time

Optimize Performance



SELECT

Reach out to join early access or book a demo → **hello@select.dev**

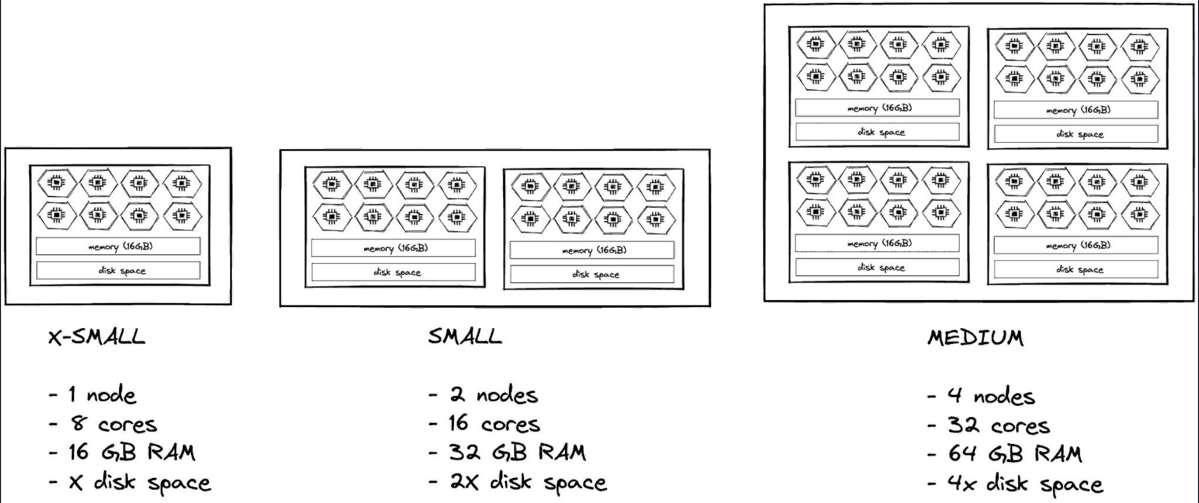
Thanks for listening!

"data nerds socializing" by Midjourney

Choosing the right warehouse size

- Start with X-SMALL warehouse
- Test with representative production queries
- If execution time is within SLO, leave as is. Otherwise, increase warehouse until SLO is met.
- If on enterprise, configure maximum cluster count on warehouse to meet peak concurrency needs. Simulate using historical production data if available.

Impact of warehouse size on query execution time



- Compute, memory, and disk space (cache size + space available for local spillage) double with each size increase
- Generally speaking, query execution time will also halve, until…
  - A certain point where performance will either stop improving (Snowflake won't parallelize further) or gets worse due to added communication costs outweighing performance benefits

Before you start, can you reduce the frequency?

- Does your query need to run every hour?
  - Is anyone looking at the dashboard multiple times per day?
- If a data models costs $10,000/year running hourly, switching to daily can drop costs by ~95%

Include additional join predicate to force pruning

- Static pruning vs. dynamic pruning
- During a join, Snowflake creates a hash table on the "build side" (smaller table, on the left of the query profile)
- Statistics are collected for the distribution of join keys in build-side records
- These are pushed to the probe side (bigger table) and can be used to filter or skip entire files

Regular merge forces join

- A merge results in a join
- Table is well clustered with order timestamp, not order key

```sql
MERGE INTO db.public.orders as target
USING orders_to_update as source
ON target.o_orderkey = source.o_orderkey
WHEN matched THEN
    UPDATE SET target.o_totalprice = source.o_totalprice
```

Regular merge forces a full table scan!

Most Expensive Nodes (3 of 6)

TableScan [2]	97.1%
JoinFilter [3]	1.6%
Merge [5]	1.1%

Profile Overview (Finished)

Total Execution Time	(11s) 100.0%
• Processing	17.4%
• Local Disk I/O	4.8%
• Remote Disk I/O	77.7%
• Initialization	0.1%

Statistics

Number of rows updated	3
Scan progress	99.89%
Bytes scanned	6.23GB
Percentage scanned from cache	0.08%
Bytes written	32.79MB
Bytes sent over the network	0.28MB
Partitions scanned	2772
Partitions total	2775

Takes 11 seconds

```
MERGE INTO db.public.orders as target
USING orders_to_update as source
ON target.o_orderkey = source.o_orderkey
WHEN matched THEN
    UPDATE SET target.o_totalprice = source.o_totalprice
```

SELECT

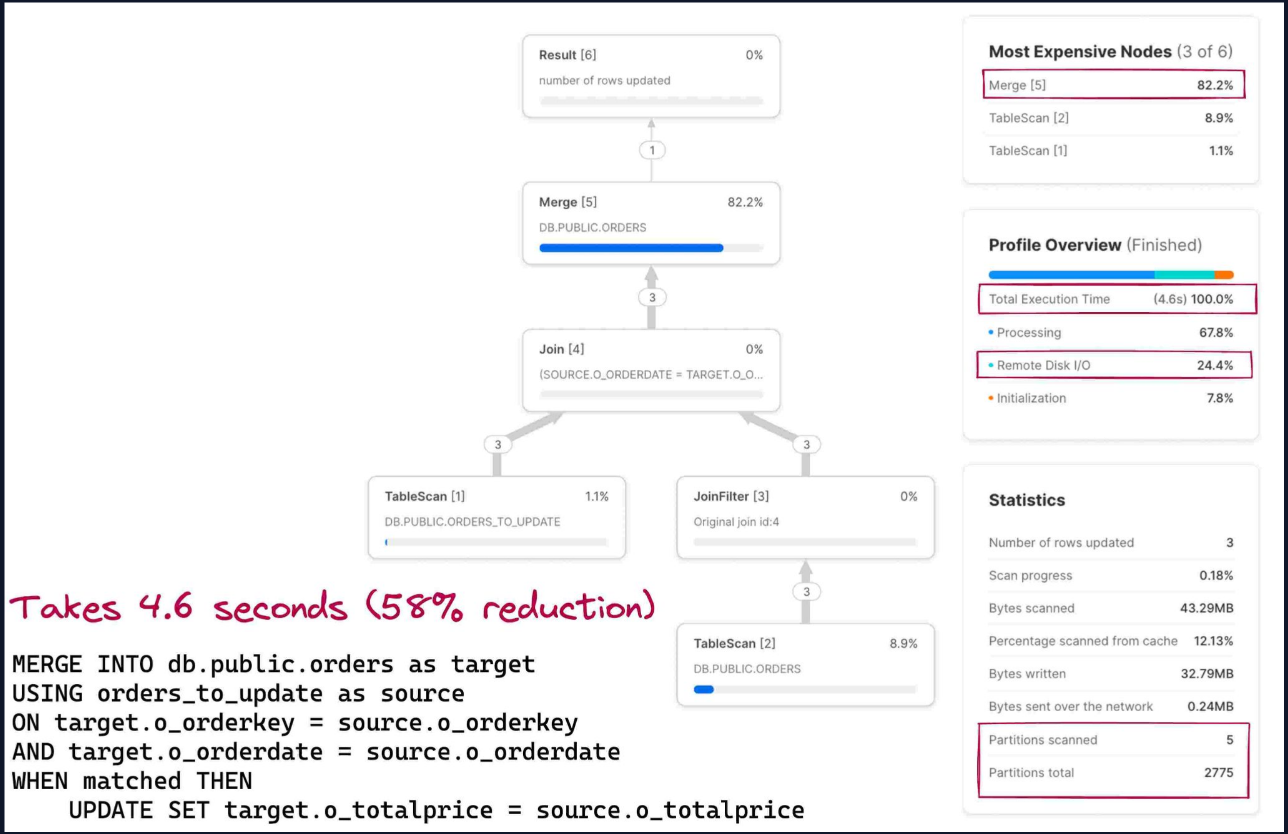
Add additional join condition on our clustered column

- Table is well clustered on order date
- Adding additional join key doesn't change validity of join

```sql

MERGE INTO db.public.orders as target
USING orders_to_update as source
ON target.o_orderkey = source.o_orderkey
AND target.o_orderdate = source.o_orderdate -- additional predicate!
WHEN matched THEN
    UPDATE SET target.o_totalprice = source.o_totalprice
```

Adding date predicate to join forces dynamic pruning, query now scans <0.2% of table!



SELECT

Should you use CTEs?

- Yes
- CTEs are computed once in Snowflake
- In certain scenarios where CTE is referenced more than once, can be faster to repeat logic in subqueries rather than use a CTE