

**WCET**

WARNER  
CONTENT &  
ENTERPRISE  
TECHNOLOGY

# Snowflake Best Practices

# Virtual Warehouses

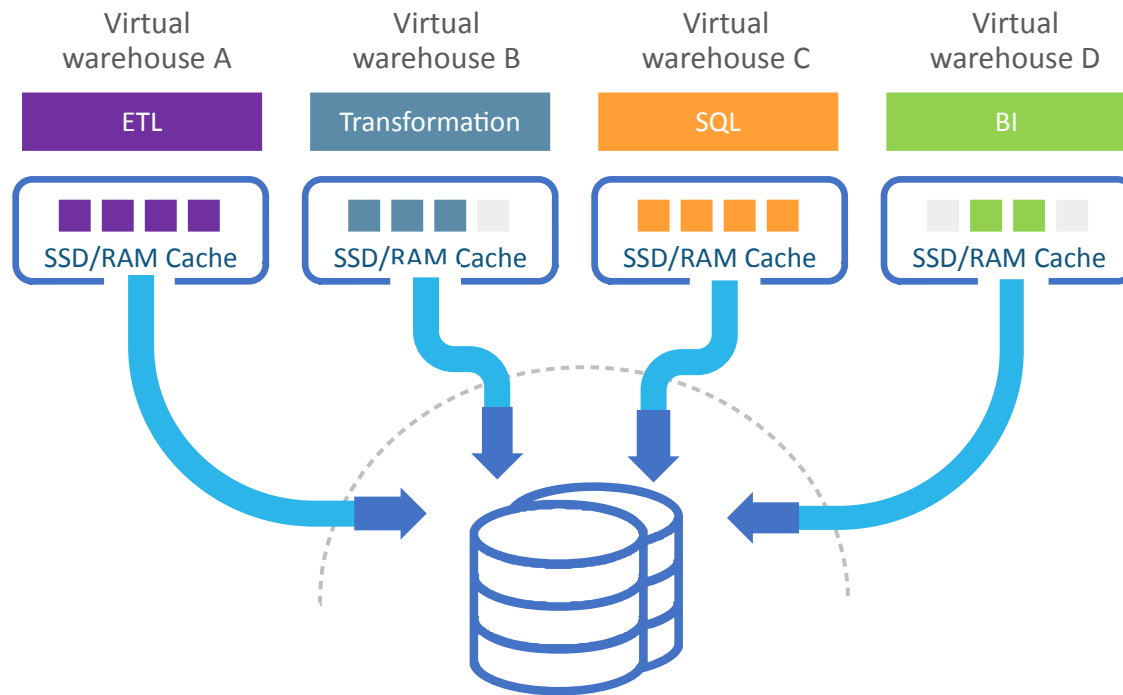
WCET

WARNER  
CONTENT &  
ENTERPRISE  
TECHNOLOGY

# PROCESSING DATA

How to allow concurrent workloads to run without impacting each other?

## Virtual warehouses



A warehouse is one or more MPP compute clusters

Use multiple warehouses to segregate workload

- e.g., ETL warehouse versus query warehouse

Resizable on the fly

- Adjust cluster size (up/down) based on data size and/or query complexity
- Automatically add/remove clusters as level of concurrency varies

Able to access data in any database

Transparently caches data accessed by queries

Transaction manager synchronizes data access

Automatic suspend/resume when needed



# ADDRESSING QUERY PERFORMANCE



**Warehouse Sizing** – Increasing warehouse size frequently improves large/complex query performance, but not always - **SCALE UP**

**Warehouse Sizing** - Using a Multi-Cluster Warehouse or increasing the warehouse max cluster limit can help improve performance when large number of queries are executing/queuing - **SCALE OUT**

**SQL Rewrites** – With modern SQL, there are multiple ways to write a query that produces the same solution. Not all approaches produce equal performance. The optimizer helps, however it does not solve all problems.



# Warehouse size impact on query performance

- Large or Complex queries can be affected by the warehouse size\*
- Performance can see improvement by increasing warehouse size
- “Disk spills” are an indicator that a query is resource constrained
- Test on larger warehouse

\*Increasing warehouse size will not resolve queuing of queries. Use larger multi-cluster warehouses to address (see next slide)

**Configure Warehouse**

Name ALPHA\_WH

Size **X-Large (16 credits / hour)**

Maximum Clusters Small (2 credits / hour)

Scaling Policy Medium (4 credits / hour)

Auto Suspend Large (8 credits / hour)

X-Large (16 credits / hour)

2X-Large (32 credits / hour)

3X-Large (64 credits / hour)

4X-Large (128 credits / hour)

☒ Auto Resume ?

Comment YourName Warehouse

[Show SQL](#) Cancel Finish

# What to Look for - Warehouse Disk Spilling

- When Snowflake cannot fit an operation in memory, it starts spilling data first to disk, and then to remote storage.
- These operations are slower than memory access and can slow down query execution a lot
- To see if a query is spilling to disk, look at the right-hand side of the **Profile** tab in the web interface
- In this example, 37.59 GB of data spilled to disk.

## Profile Overview (Aborted)

Total Execution Time (8m 20.81s)



## Total Statistics

IO	
Scan progress	100.00 %
Bytes scanned	5.61 GB
Percentage scanned from cache	100.00 %
Bytes written	0.81 MB
Pruning	
Partitions scanned	1,328
Partitions total	1,328
Spilling	
Bytes spilled to local storage	37.59 GB



# What to Look for - Warehouse Disk Spilling

- To find out more, click on a node in the **Query Profile** graph. As spilling to disk is so slow, the node of interest is most likely one where the query is spending significant time.
- Here's an example of a sort operator spilling to disk:
- In this case, the amount of data spilled is low, just about 2GB.
- This fits in the local disk cache, and this is indicated as bytes spilled to **local** storage. When the local storage is not enough to hold the data, another level of spilling occurs: the data is spilled to **remote** storage. This is significantly slower than spilling to local storage.

## Sort [2]

### Node Execution Time



### Statistics

#### Spilling

Bytes spilled to local storage



2.07 GB



# Approaches to resolve slow queries caused by disk spillage

- **Use a bigger warehouse.** This increases the total available memory and node/cpu parallelism, and is therefore often a good way to make this type of query faster.
- **Reduce the number of rows that must processed.** Storage Data clustering is often very effective for this as it allows Snowflake in many cases to only read the appropriate subset of the data that is really needed.

## Sort [2]

### Node Execution Time



### Statistics

#### Spilling

Bytes spilled to local storage



2.07 GB





# VIRTUAL WAREHOUSE

## SCALE *UP* FOR PERFORMANCE

### ELASTIC PROCESSING POWER (CPU, RAM, SSD)

- Raw performance boost for complex queries or ingesting large data sets
- Typically more complex queries on larger datasets require larger warehouses

### WAREHOUSE SIZING GUIDELINES

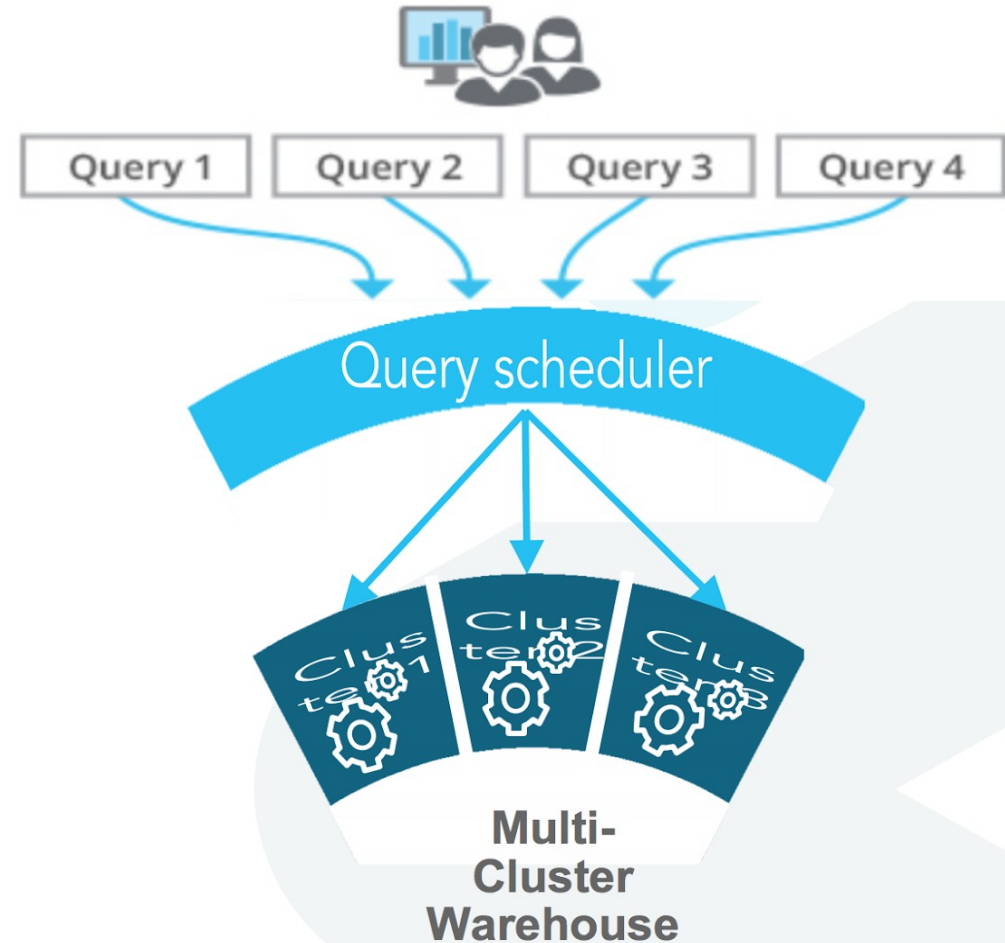
- Snowflake utilizes per-second billing: Use larger warehouses (L, XL, 2XL, etc) for more complex workloads and (auto) suspend when not in use
- Keep queries of similar size and complexity on the same warehouse to simplify compute resource sizing



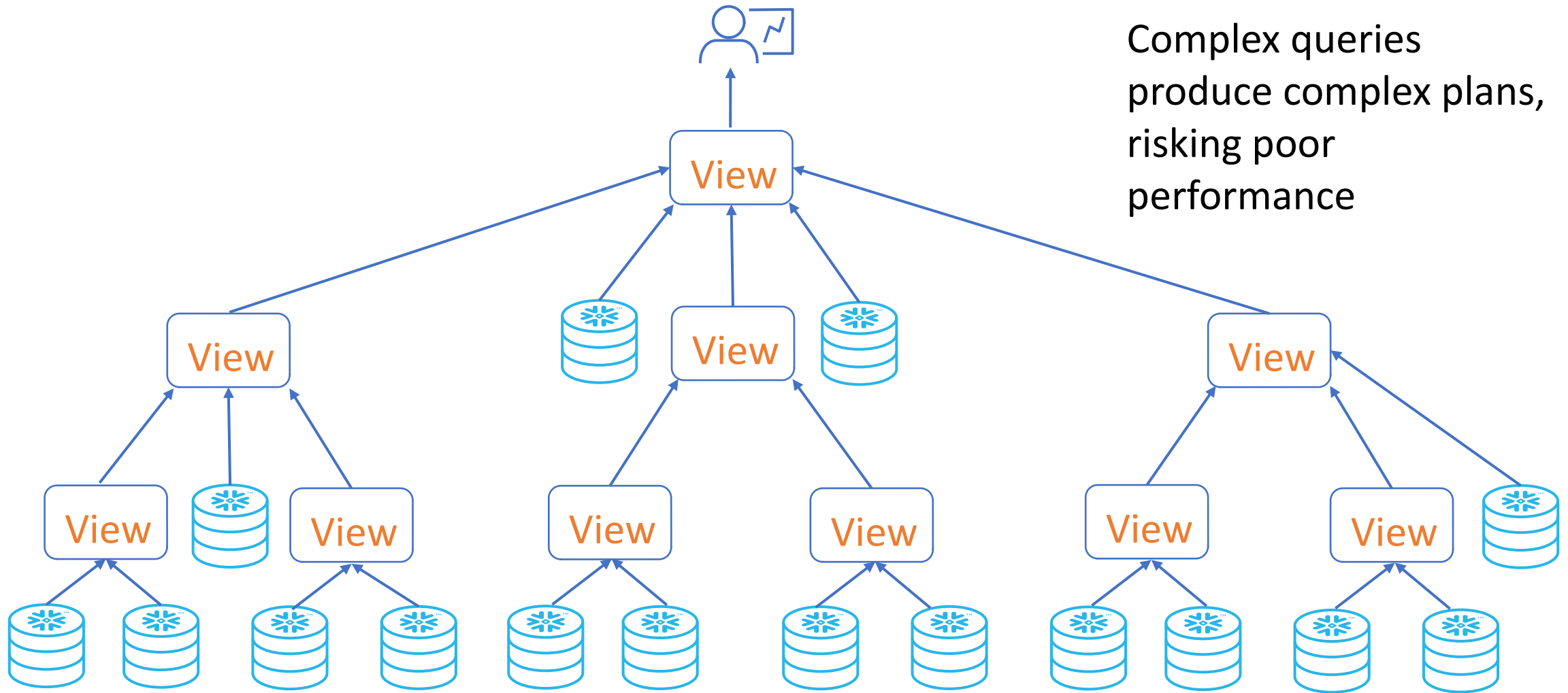
# VIRTUAL WAREHOUSE

## SCALE OUT FOR CONCURRENCY

- Single virtual warehouse with a number of compute clusters
- Deliver consistent SLA, automatically adding and removing compute clusters based on concurrent usage
- Scale out during peak times and scale down during slow times for cost savings
- Queries are load balanced across the clusters in a virtual warehouse
- Compute clusters deployed across availability zones for high availability



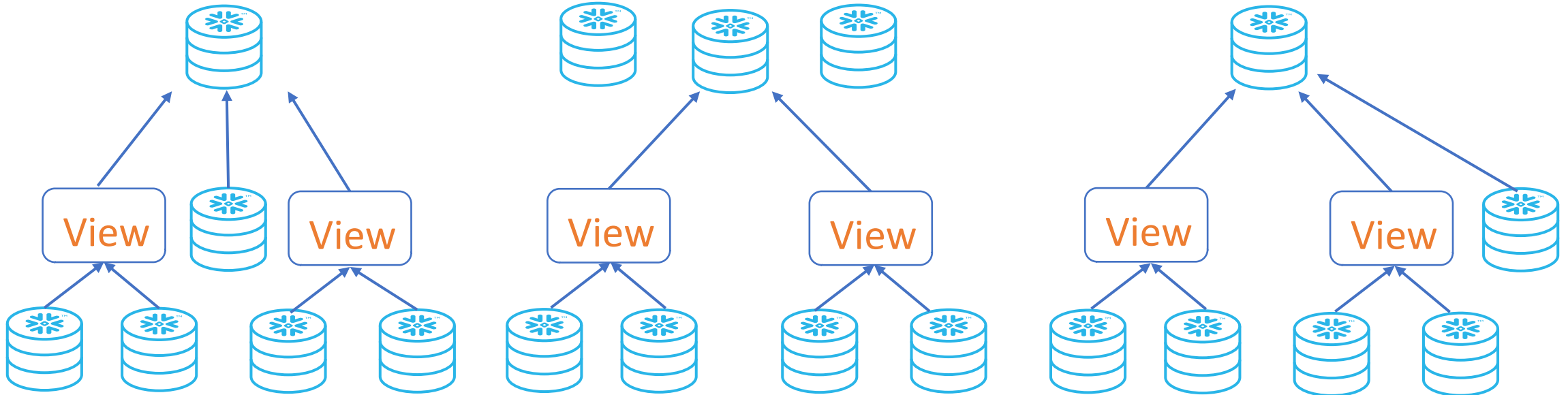
# KEEP QUERIES SIMPLE



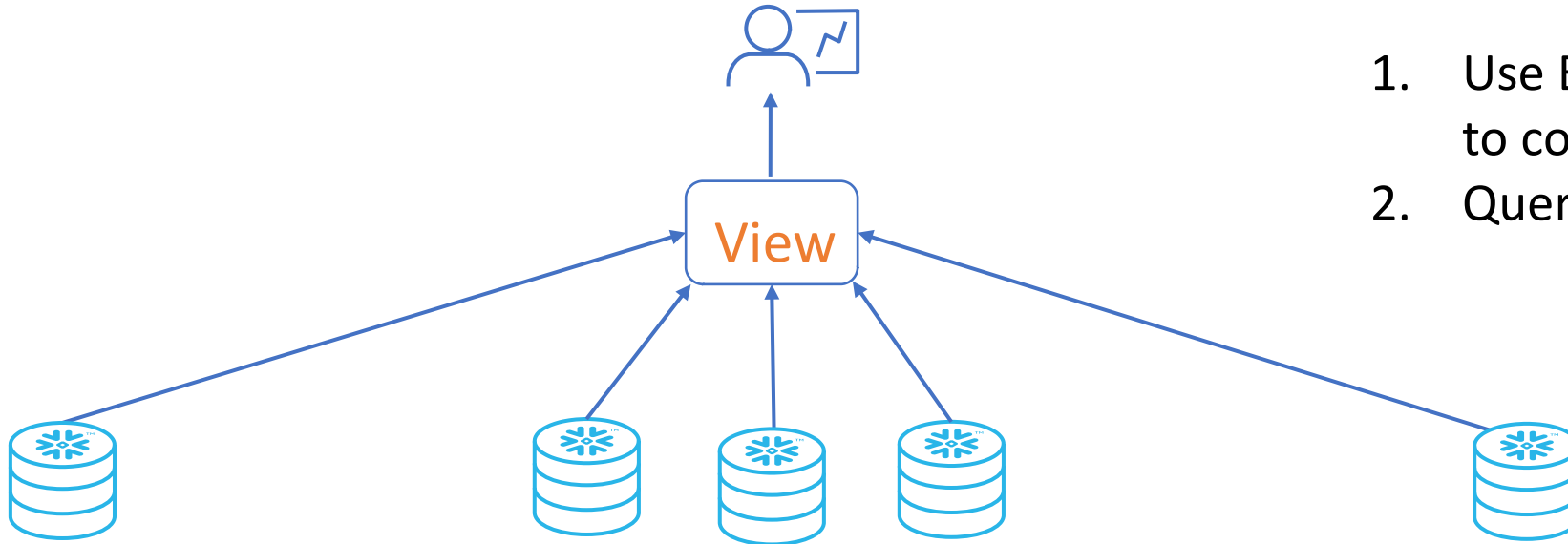
# SIMPLIFIED SOLUTION – STEP 1



1. Use ELT operation to combine results



# SIMPLIFIED SOLUTION – STEP 2



1. Use ELT operation to combine results
2. Query Results Set



- We pay for warehouse running time, use concurrent pipeline execution as much as possible
- Unless required, turn off the DAGs during non-essential hours
- If Data being used by various teams, see if the same warehouse can help as cache can be reused
- Auto suspension for warehouses can be relooked if increasing the suspension can avoid re-generation of cache for queries
- Check for queuing situations and assess the chances to increase the cluster size/scale up for allowing concurrency

- Reporting

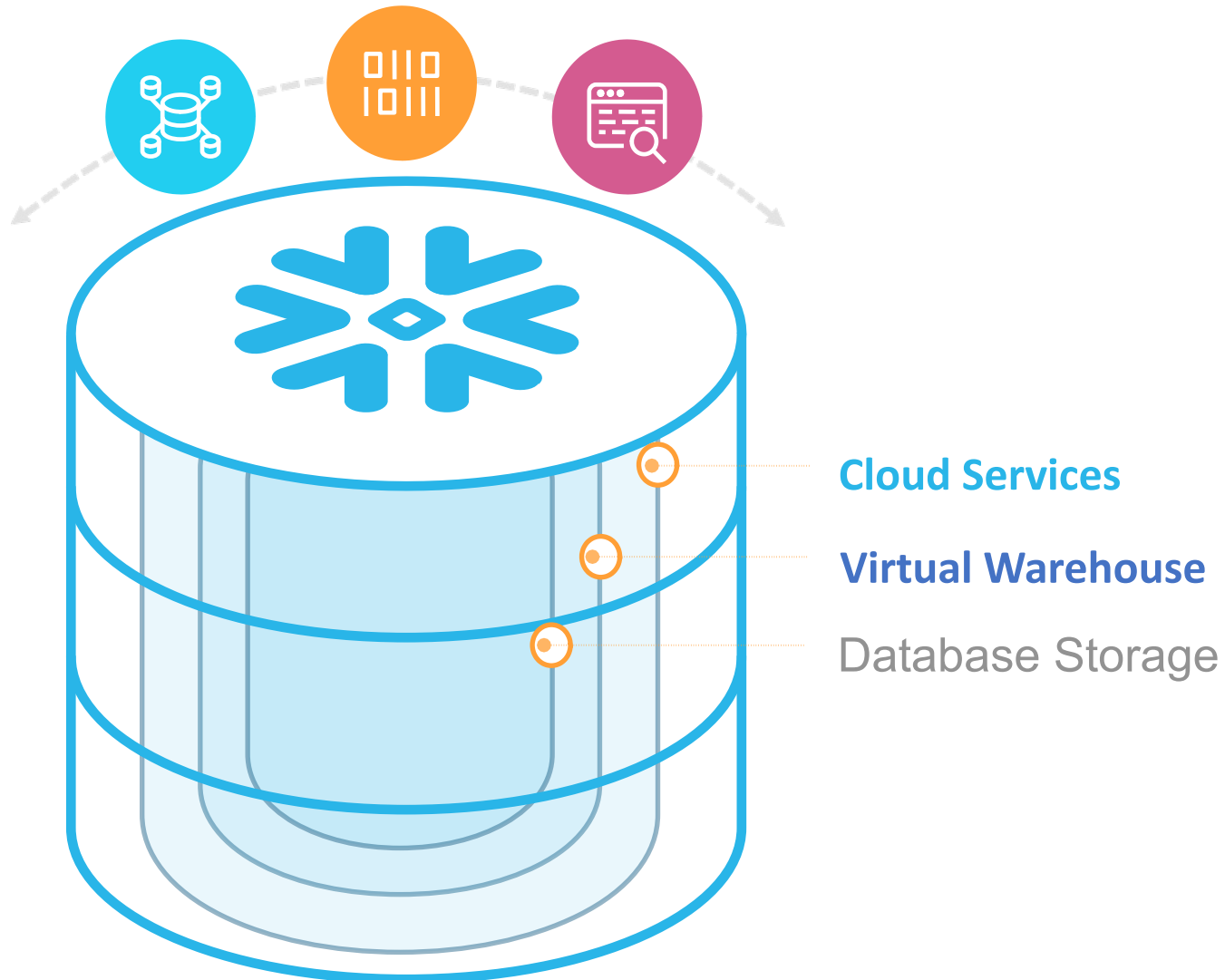
- Use Initial SQL for query tags, so performance can be measured
- Check for Live connection in place of extracts
- Refer to best practices white paper(  
<https://www.tableau.com/learn/whitepapers/designing-efficient-workbooks>)

# CACHING





# CACHING AND REUSE



## Query Data

Active working set transparently  
cached on virtual warehouse SSD

## Query results

Results sets cached for reuse  
without requiring compute  
(e.g., static dashboard queries)

## Metadata

Metadata cached for fast access  
during query planning





## Proven Practice:

- Data analysts run similar queries on the same virtual warehouse to maximize **query** cache reuse
- DBAs size virtual warehouse optimized for **data** cache reuse

## What query data is being cached on virtual warehouse SSD?

- Only the selected columns
- Only the partitions meeting the query filters (WHERE predicates and join predicates)

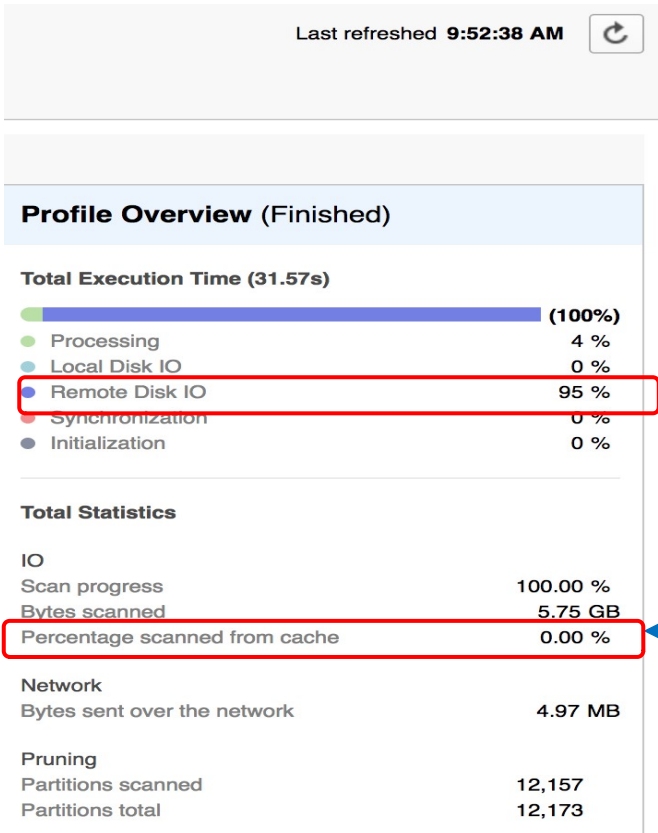
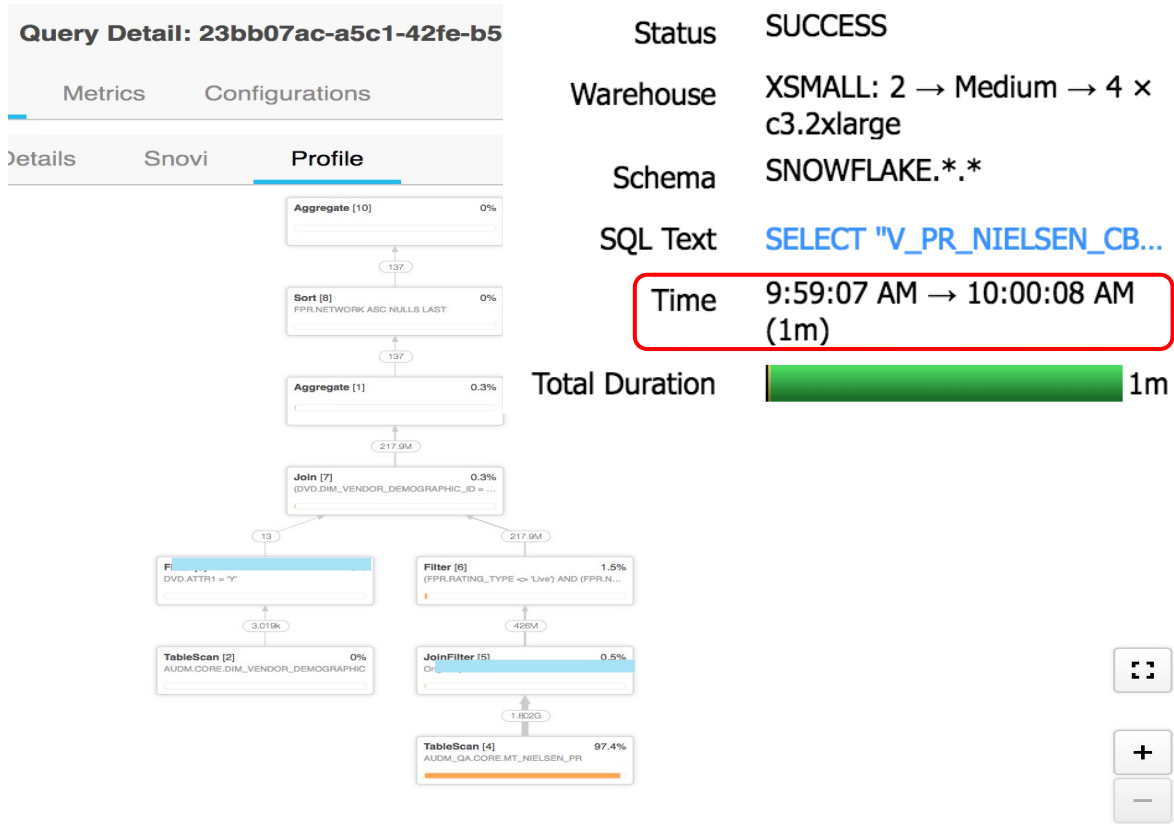
```
SELECT network
FROM RATING
WHERE (data_stream = 'Live')
GROUP BY 1
ORDER BY 1;
```

### Query **data** cache contains:

- Only selected column:  
[**network**]
- **Micro-partitions** satisfying the  
**predicate**: data\_stream =  
'Live'



# DATA CACHE ON VIRTUAL WAREHOUSE, COLD

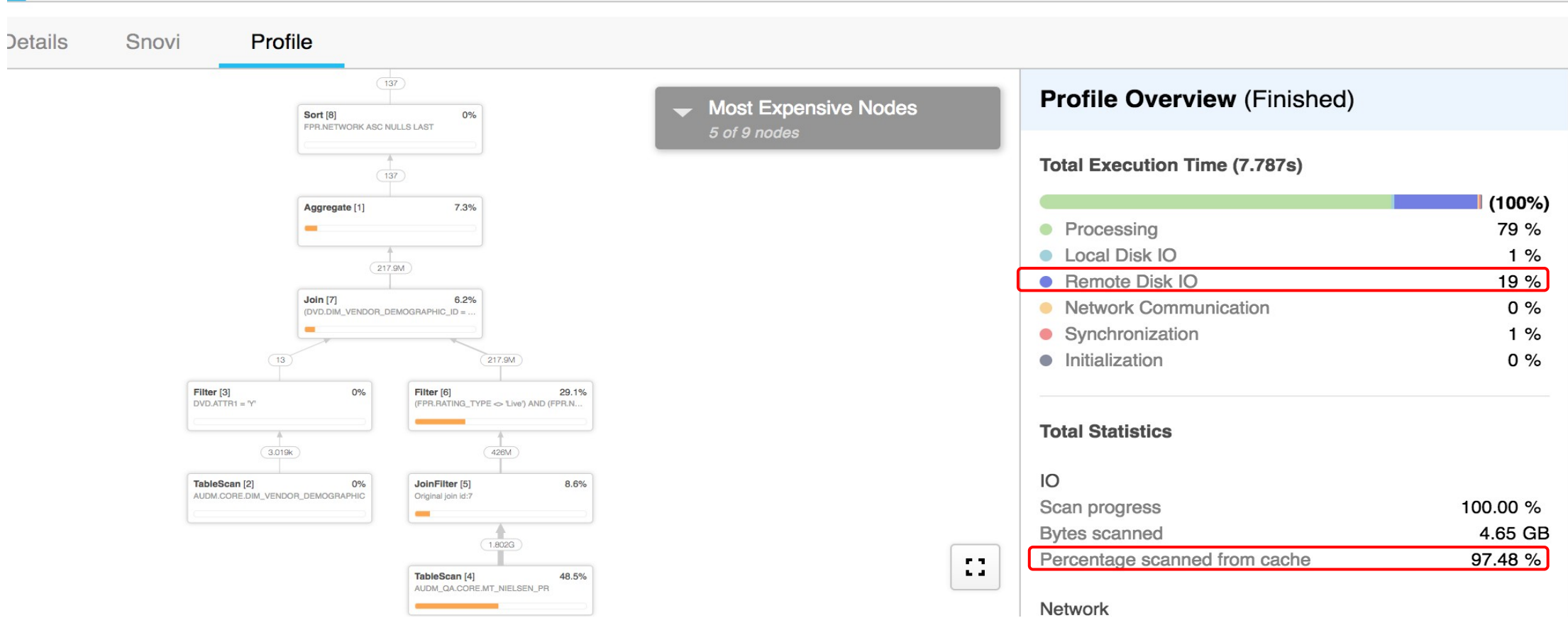


95 % of the query cost was spent on remote IO

0% of data was scanned from cache.

Warehouse cache is cold at start. Data is fetched from S3 and processed to answer the query. Query data is then cached in the local

# DATA CACHE ON VIRTUAL WAREHOUSE, WARM



Remote IO drops to 19% from previous 95%

Most query data are read from local cache

Greatly reduced the cost of scanning the large table. The Table Scan operator cost fell from 97% to 45%. Execution time fell from 1min to

# QUERY RESULT CACHE



Profile

▼ Most Expensive Nodes  
1 of 1 nodes

**QUERY RESULT REUSE [0]** 100%  
db628f44-218b-4d65-8980-afa52bfd4956

## Profile Overview (Finished)

Total Execution Time (13ms)



**Available for 24 hours from last execution**

**Detailed conditions for query result cache ([link](#))**



# Data Ingestion

WCET

WARNER  
CONTENT &  
ENTERPRISE  
TECHNOLOGY

# BATCH VS. CONTINUOUS LOADING

## COPY command

- Throughput determined by virtual warehouse size, multi-cluster option and file size.
- Separate and right-size VWHs for different workloads. Eg. Ingestion and Transformation.
- Good for migration from traditional sources
  - Bulk loading and Backfill processing
- Needed to control transaction boundary:
  - BEGIN / START TRANSACTION / COMMIT / ROLLBACK

## Snowpipe

- Continuously generated data is available for analysis in seconds to minutes
- Serverless model with no user-managed virtual warehouse needed
- Can be most cost efficient than Copy.
- No scheduling (with auto-ingest)
- Ingestion from modern data sources
- Can be combined with Streams/Tasks for continuous loading and transformation pipes



- Files – Split files as much as possible (10 Mb to 100 Mb compressed is ideal), Parquet (try for 1 Gb files)
  - <https://docs.snowflake.com/en/user-guide/data-load-considerations-prepare.html>
- ELT – Snowflake supports varying data formats, explore first before performing transformations using Spark. Switch to Snowflake for COPY/Unload activities as it will reduce the EMR cost
- Explore Snowflake's VARIANT features that supports Semi structured data



# Storage Integration

- Avoid using AWS keys to access S3 locations
- Use Snowflake's Integration feature that allows the use of IAM roles
- Open a ticket with Ops team with info:
  - S3 location for Read/Write
  - Owner of the Stage
  - Database/Schema for the Stage creation
  - Role(s) that needs access to the Stage
- Use of Integrations is secure and allows generic procedure to access Cloud Storage
- A single storage integration can support multiple external stages. The URL in the stage definition must align with the storage location specified for the `STORAGE_ALLOWED_LOCATIONS` parameter.



# DATA FILE MANAGEMENT

**Within Stages (landing zone), organize files via logical path:**

**Copying from:**

```
United_States/California/Los_Angeles/2016/06/01/11/  
United_States/New_York/New_York/2016/12/21/03/  
United_States/California/San_Francisco/2016/08/03/17/
```

**will be more efficient than copying from:**

```
United_States/
```

**or even from:**

```
United_States/California/  
United_States/New_York/New_York/
```

**Within loads, speed up searching for files using path:**

```
COPY INTO t1 FROM @my_stage/us/California/2020/09/*;
```



# EXTRACT LOAD AND TRANSFORM

## Use ELT instead of ETL

- Snowflake is optimized for bulk load via COPY command or batch insert
- Take advantage of parallel processing and elastic compute capabilities
- Data Ingestion Flow  
Data source --(load)--> Staging table --(validate data then load)--> Target table

## Staging Tables

- Use temporary and transient table type
- Set staging table Time Travel retention period to 0 or 1 day

## Data Validation

- Identify and resolve source data issues to keep data warehouse tables clean

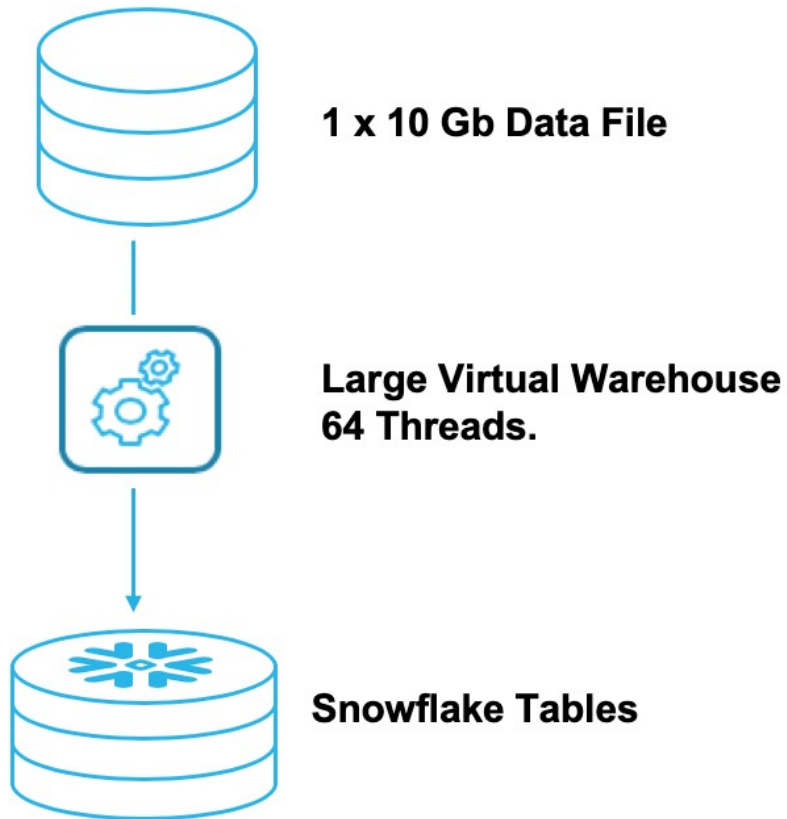
# BULK LOAD VS ROW-BY-ROW

## Snowflake is optimized for bulk load and batched DML

- Use COPY INTO to load data over INSERT with SELECT where possible
- The COPY INTO command supports column reordering, column omission, and casts using a SELECT statement
  - NOT SUPPORTED: Joins, filters, aggregations
  - Can include SEQUENCE columns, current\_timestamp(), or other column functions during data load
  - The VALIDATION\_MODE parameter does not support transformations in COPY statements
- When COPY INTO cannot be used and INSERT is required, batch INSERT statements
  - INSERT w/ SELECT
  - CREATE AS SELECT (CTAS)
  - Minimize frequent single row DMLs



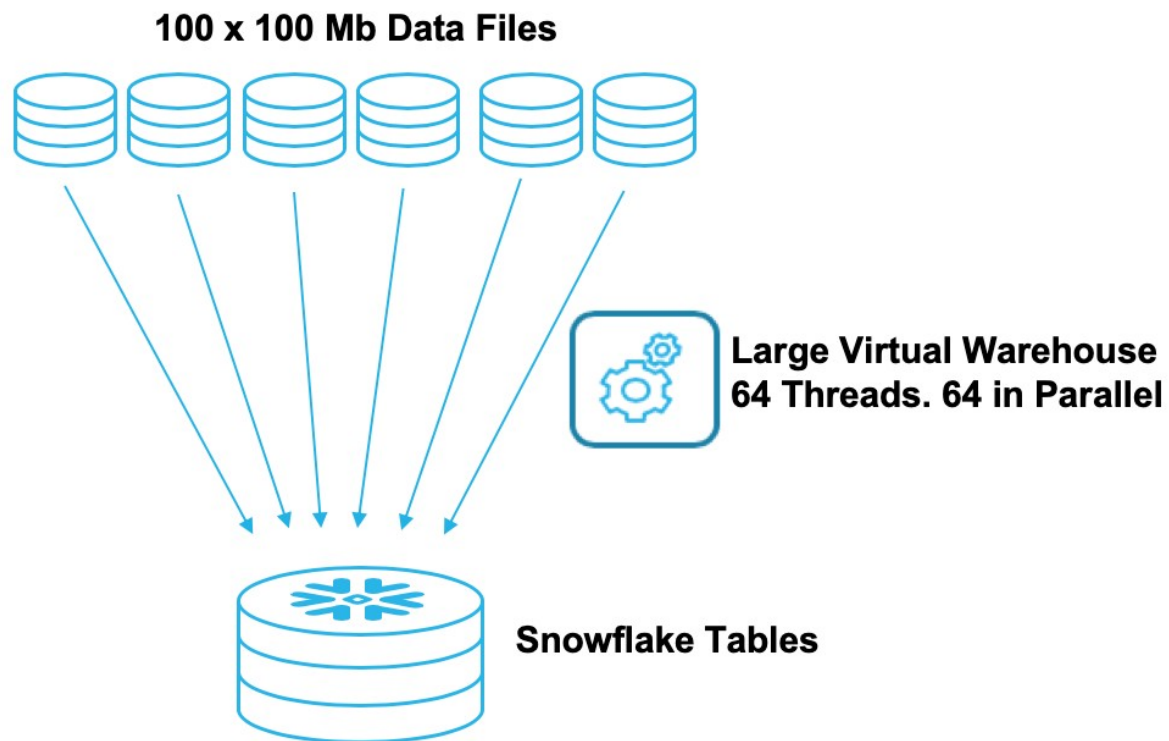
# SERIAL DATA LOADING



- **Bulk loading Anti-Pattern**
  - Loading a single 10Gb File
  - Using a Large Warehouse
  - Potential for 64 files in parallel
  - Single Copy command – serial load
  - Very inefficient
- **Recommendation**
  - Split large files (optimal size 10 -100Mb)
  - Maximize parallel loading
  - Supports ability scale up from 8-1024 parallel loads (XS to 4XL)



# PARALLEL DATA LOADING



- **Fast Parallel Bulk Loading**

- Files split to 100 MB files
- Single copy operation
- 64 Files load in parallel
- Extremely fast and efficient

- **Outcome**

- 64 x 100 MB files in parallel (L)
- 64 times faster than before
- 100% VWH utilization
- Potential for 15 TB / Hour



## PARALLEL LOADS BY T-SHIRT SIZE

T-Shirt Size	Parallel Loads
X-Small	8
Small	16
Medium	32
Large	64
X-Large	128
2X-Large	256
3X-Large	512
4X-Large	1024

- **Insight**
  - A single 100 TB file will load as fast on an XS as a 4XL
  - But will be 128 times less expensive
- **Key Takeaway**
  - Size files 10-100 MB
  - Size the Virtual Warehouse by typical number files available

## BENCHMARK LOAD PERFORMANCE

Format	Target Layout	* TB/Hr
CSV (GZipped)	Structured	15.4
Parquet (Snappy)	Semi	4.8
Parquet (Snappy)	Structured	5.4
ORC (Snappy)	Semi	4.4
ORC (Snappy)	Structured	6.0

\* Uncompressed File Sizes. IE. 15 TB of raw CSV will be stored in columnar compressed format. Approx. 3.5 TB

- **Key Takeaway**
- CSV Fastest Load Rate – 15 TB / Hr
- Not worth converting to/from CSV as Parquet & ORC load fast





# Storage

# WCET

---

WARNER  
CONTENT &  
ENTERPRISE  
**TECHNOLOGY**

# DATA PROTECTION

**Snowflake provides simple but powerful maintenance of historical data using time travel and fail-safe storage**

Time Travel: query, clone, or restore objects to a point-in-time data within the retention window

- Available for database, schema, and table objects
- Default retention period of 1 day
  - Controlled by the `DATA_RETENTION_TIME_IN_DAYS` parameter at the Account, Database, Schema, or Table object level
  - Can be set up to 90 days for permanent tables (Enterprise edition or higher)
- Support for up to 1 day for temporary and transient tables

Fail-safe Storage: Non-configurable, 7-day retention for historical data after Time Travel expiration

- Only accessible by Snowflake personnel under special circumstances
- Admins can view Fail-safe use in the Snowflake Web UI under Account > Billing & Usage
- Not supported for temporary or transient tables

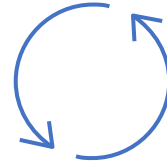


# TABLE TYPES



## Temporary

Tied to an individual session and persists only for the duration of the session. Used for storing non-permanent, transitory data (e.g. ETL data, session-specific data).



## Transient

Specifically designed for transitory data that needs to be maintained beyond each session (in contrast to temporary tables) but does not need the same level of data protection and recovery provided by permanent tables.



## Permanent

Designed for data that requires the highest level of data protection and recovery with both a Time-Travel and Fail-Safe period and is the default for creating tables.

**Time-Travel**



**Fail-Safe**



# TABLE TYPE DETAILS

Type	Subject	Fail Safe Period	Time Travel (days)	Specifics
Temporary	Table	0	0 or 1 (1 is default)	Exists only for the duration of the user session and not visible for other users. Please note: namespace for temp tables can overlap with permanent tables. Temporary will be accessed first
Transient	Schema, Database	0	0 or 1 (1 is default)	Database / Schema objects will not have a Fail-safe period. Leading to less storage cost but no protection in the event of data loss.  Please note: this applies only to newly created objects. If you clone a permanent schema/database with permanent objects into transient schema/database they will be copied as <b>permanent</b> .
	Table	0	0 or 1 (1 is default)	Table will not have a Fail-safe period. Leading to less storage cost but no protection in the event of data loss.  Cloning a table from permanent -> transient result in destination table being transient
Permanent	Table	7	Standard edition: 0 - 1 Enterprise: 0 - 90	Default table type. Time Travel days default to 1 if not specified.



# ZERO-COPY CLONING



# DATA LOADED INTO A TABLE

Immutable (no changes)

Source  
xxx



```
INSERT INTO xxx ...
```

Partition:

a.1

b.1

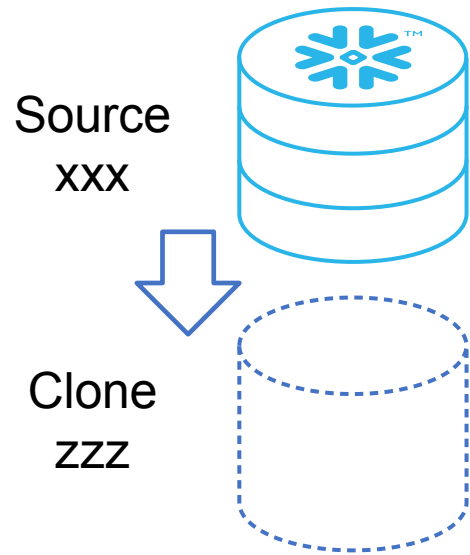
c.1

All entries at version 1

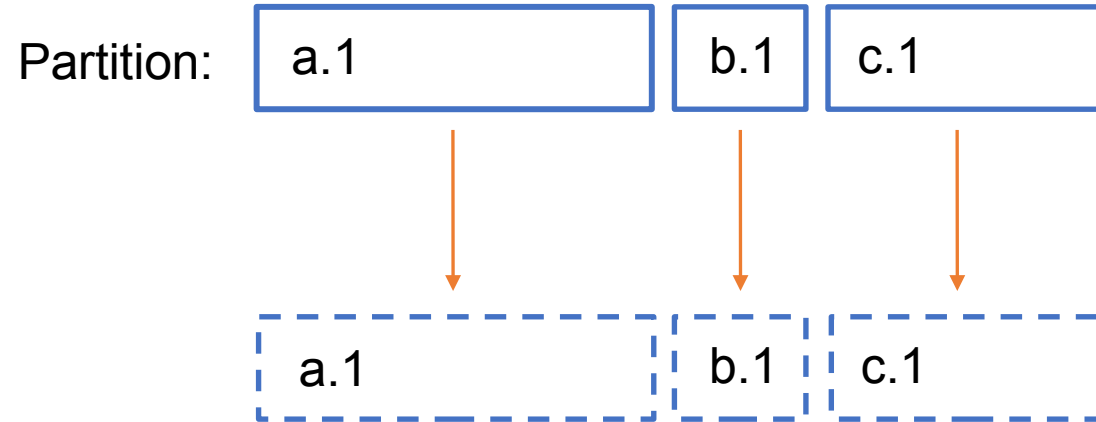


# ZERO-COPY CLONE CREATED

Updates create new micro-partition versions on source



```
CREATE OR REPLACE TABLE zzz  
CLONE xxx;
```

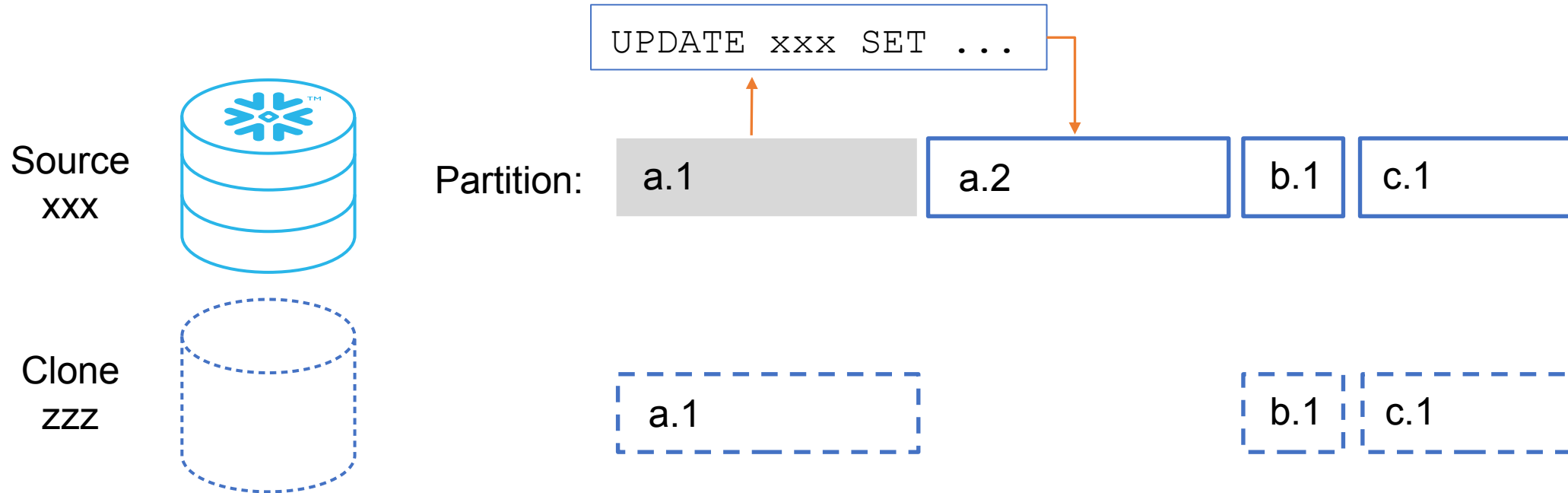


Only Metadata Pointers Copied



# CHANGES TO SOURCE ARE ISOLATED

Updates create new micro-partition versions on source



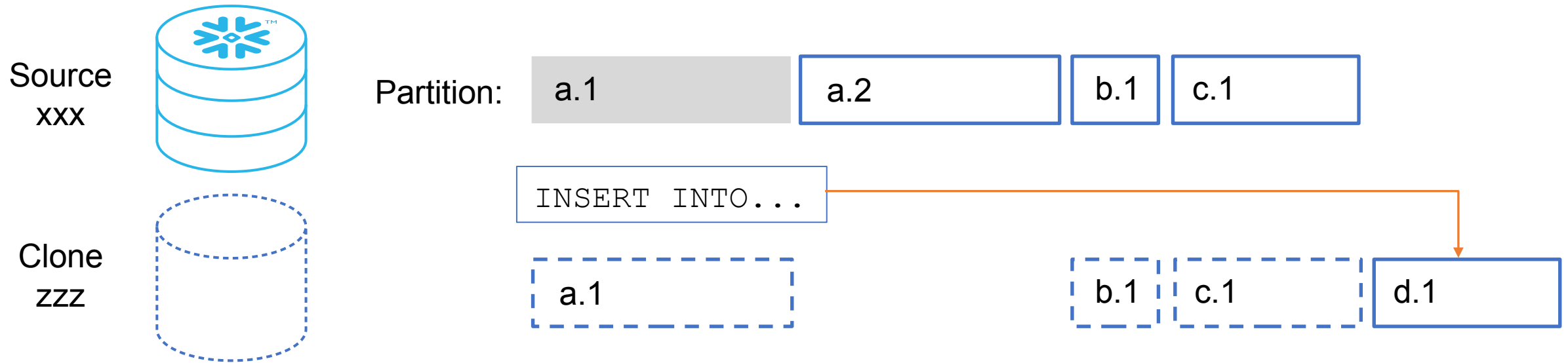
**Changes to Source DO NOT affect Clone**





# CHANGES TO TARGET ARE ISOLATED

Updates create new micro-partition versions on clone

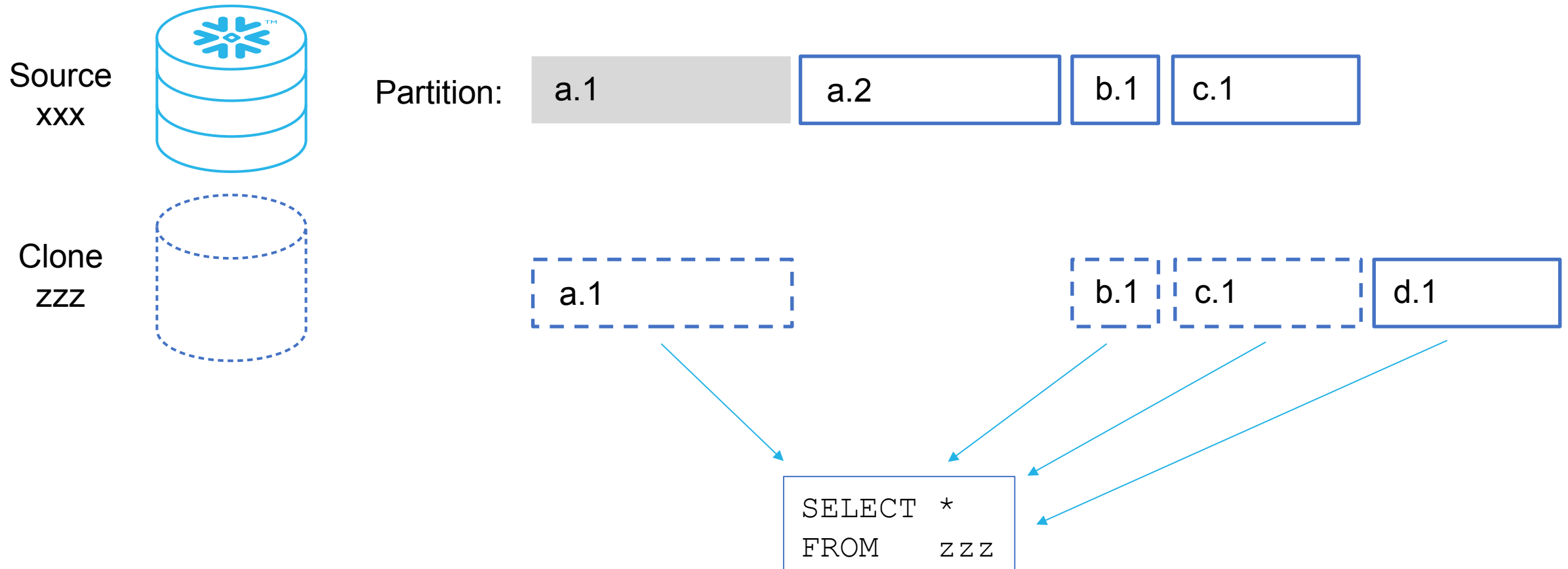


**Changes to clone do not affect source**  
**Additional storage costs incurred**



# QUERY AGAINST THE CLONE

Returns only the micro-partitions local to the Clone Copy

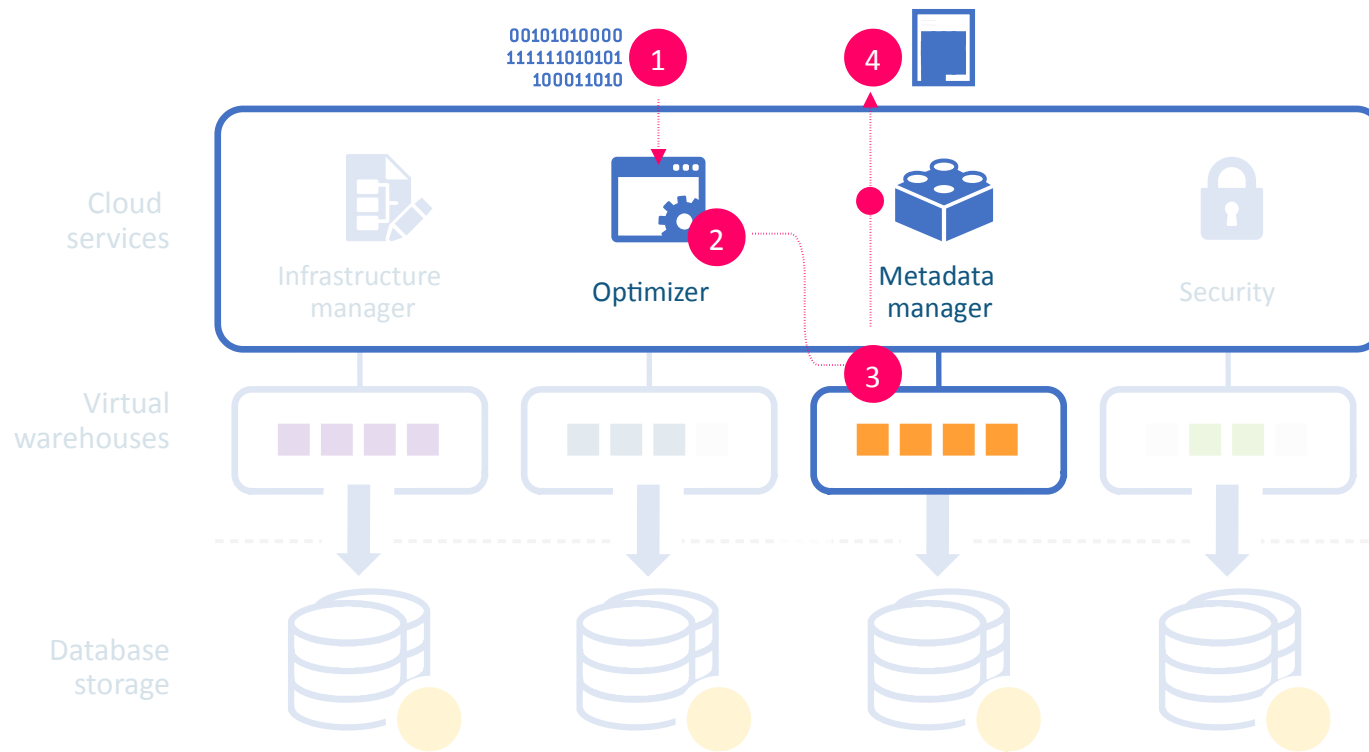


# Sql Query Optimization

WCET

WARNER  
CONTENT &  
ENTERPRISE  
TECHNOLOGY

# QUERYING DATA overview



## 1 Query received by Snowflake

Sent via standard ODBC, JDBC, or web UI interfaces

## 2 Planner and optimizer process query

Prune and filter, then use metadata to identify exact data to be processed (or retrieved from result cache)

## 3 Virtual warehouse processing

Virtual warehouse scans only needed data from local SSD cache or cloud storage, processes, and returns to cloud services

## 4 Result set return

Final result processed, stored in cache for future use, and returned to client



# SNOWFLAKE QUERY PROCESSING - COST-BASED OPTIMIZER (CBO)

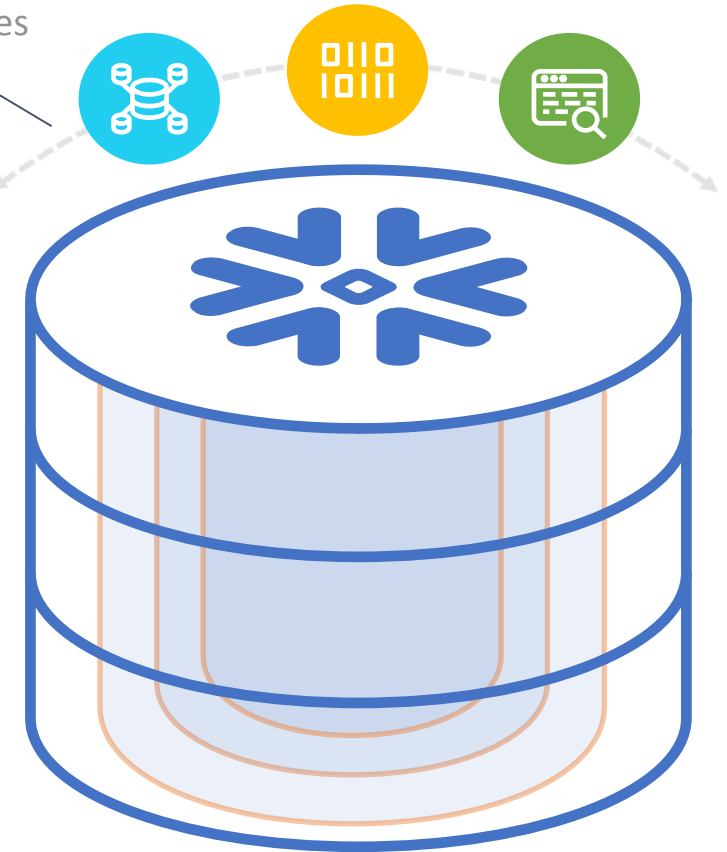
## Cost-Based Optimizer

- Snowflake employs a powerful CBO in the Global Services layer that determines optimal plan for processing SQL statement
- CBO utilizes metadata statistics to prune unnecessary micro-partitions, minimizing I/O
- Execution of CBO happens EVERY TIME a query is submitted to service
- Sometimes compilation time > execution time
- Compilation times are affected by:
  - Query complexity
  - Massive tables, since applying filtering (pruning) against an extremely high number of micro-partitions can be expensive
- When troubleshooting performance issues, it is important to monitor query compilation time

## Optimization Phases (in sequence):

- Parser
- Semantic Analysis
- Logical Writers
- Micro-partition pruning (1)
- Initial plan generation
- Plan rewrites
- Micro-partition pruning (2)
- Cost-based join ordering
- Physical plan generation

Global Services





# SNOWFLAKE SQL TIPS

## The SELECT statement

Although the SELECT clause (also referred to as the "projection") is the FIRST component of a SELECT statement in SQL, it is often helpful to think of SELECT as the LAST component to be processed

Start with the FROM/JOIN/WHERE clauses, since they define the working set of data

Then consider the (optional) GROUP BY clause; follow that with the (optional) HAVING clause

Next comes the (optional) ORDER BY clause that defines a sort that should be applied to results

At this point, the (optional) LIMIT clause should be considered

### Finally, consider the SELECT clause

- Snowflake does provide some "syntactic sugar" by allowing computed expressions to be defined in the SELECT clause and then referenced in the following clauses (GROUP BY, for example)
- But generally, the SELECT clause defines a final "column filter" for the result set





# SNOWFLAKE OPTIMIZATION - SQL TIPS

- The number of columns retrieved on the SELECT statement matters for performance

**Best Practice:** Selecting only the columns you need

- Avoid using SELECT \* to return all columns
- Whenever possible, limit the SELECT statements on long strings or heavy VARIANT columns

Do not use an ORDER BY unless it is required; sorting tends to be a **VERY EXPENSIVE OPERATION**

- Requires significant resources (memory, disk, CPU)

**JOINS:**

- For poorly-performing join, verify that cardinality and uniqueness of values in each column matches expectations; duplicate data should NOT be handled by blindly applying DISTINCT or GROUP BY!






# SNOWFLAKE OPTIMIZATION - AGGREGATES


`COUNT (*)` and `COUNT (1)` are identical in function and performance


`COUNT (<col_name>)` will ignore rows that have a NULL value in the given column

`COUNT (<col_name1>, <col_name2>, ...)` will ignore rows that have a NULL value in ANY of the given columns

Use `DISTINCT` and `GROUP BY` properly; and NEVER use BOTH in the same `SELECT`

 `SELECT DISTINCT  
MY_COL2  
FROM X  
GROUP BY MY_COL2`

 `SELECT  
DISTINCT  
MY_COL2  
FROM X`

 `SELECT MY_COL2  
FROM X  
GROUP BY MY_COL2`

`MIN (<col_name>)` and/or `MAX (<col_name>)` aggregates across an entire table is very efficient, since the answer to this query can be found entirely through the MIN/MAX metadata statistics





# SNOWFLAKE OPTIMIZATION - FILTERS

## FILTER BEST PRACTICES TO ASSIST SQL OPTIMIZER

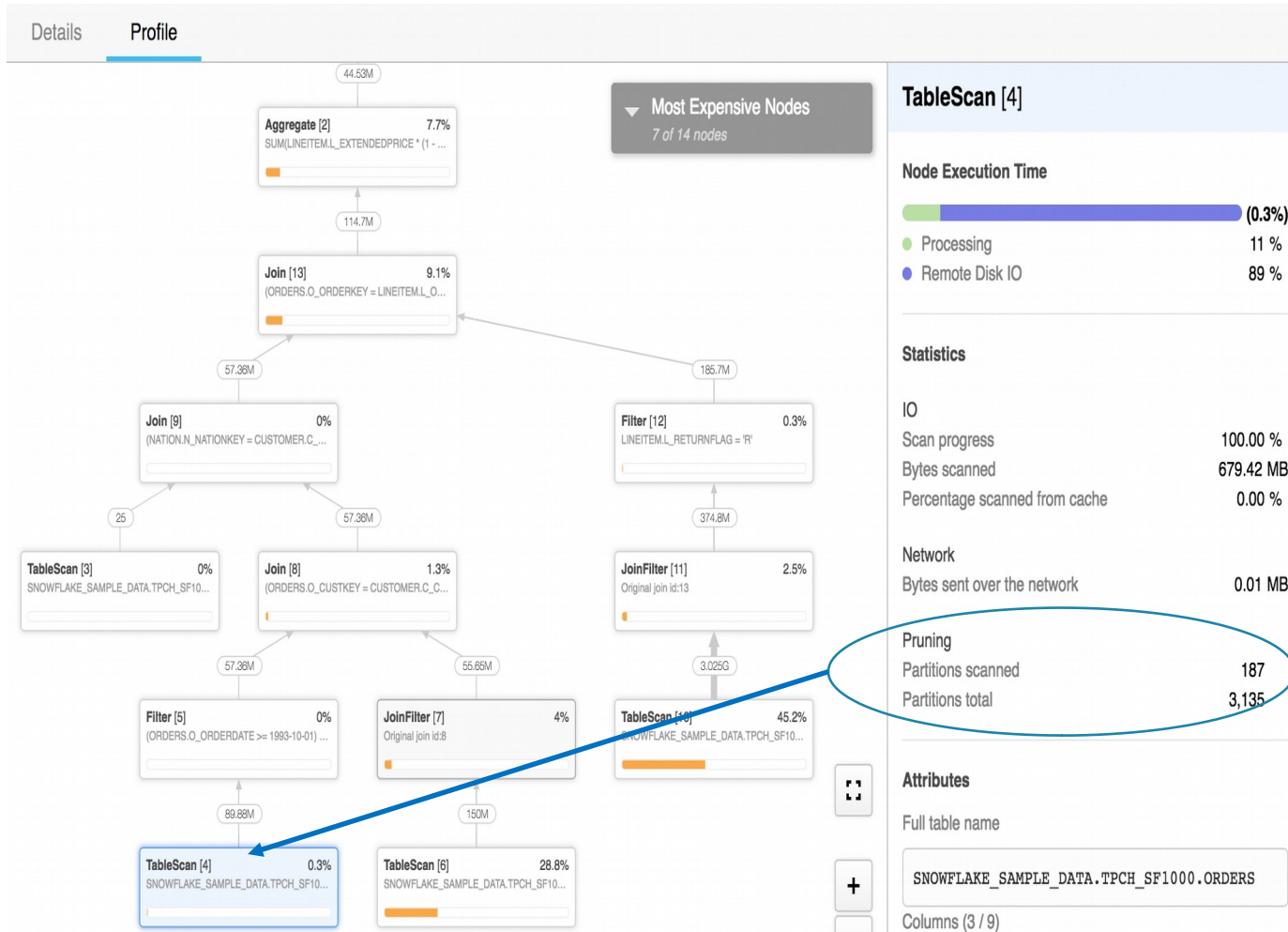
- Apply appropriate filters in WHERE clauses and appropriate join conditions in the query to eliminate as many rows as possible early in the query
  - **SQL pattern matching is faster than regular expressions**
- For natural clustered tables, applying appropriate predicate columns (e.g. date columns) that have high correlation to ingestion order can provide significant performance boost
  - **Pruning is more effective on dates and numbers than strings (especially long ones, like UUID)**

# SNOWFLAKE OPTIMIZATION - FILTERS

## SNOWFLAKE BUILT-IN OPTIMIZATIONS

- Snowflake provides patented micro-partition pruning techniques to optimize query performance:
  - Static partition pruning based on columns in query's WHERE clause
  - Dynamic partition pruning based on join columns of a query

# STATIC PARTITION PRUNING - TPCH Q10

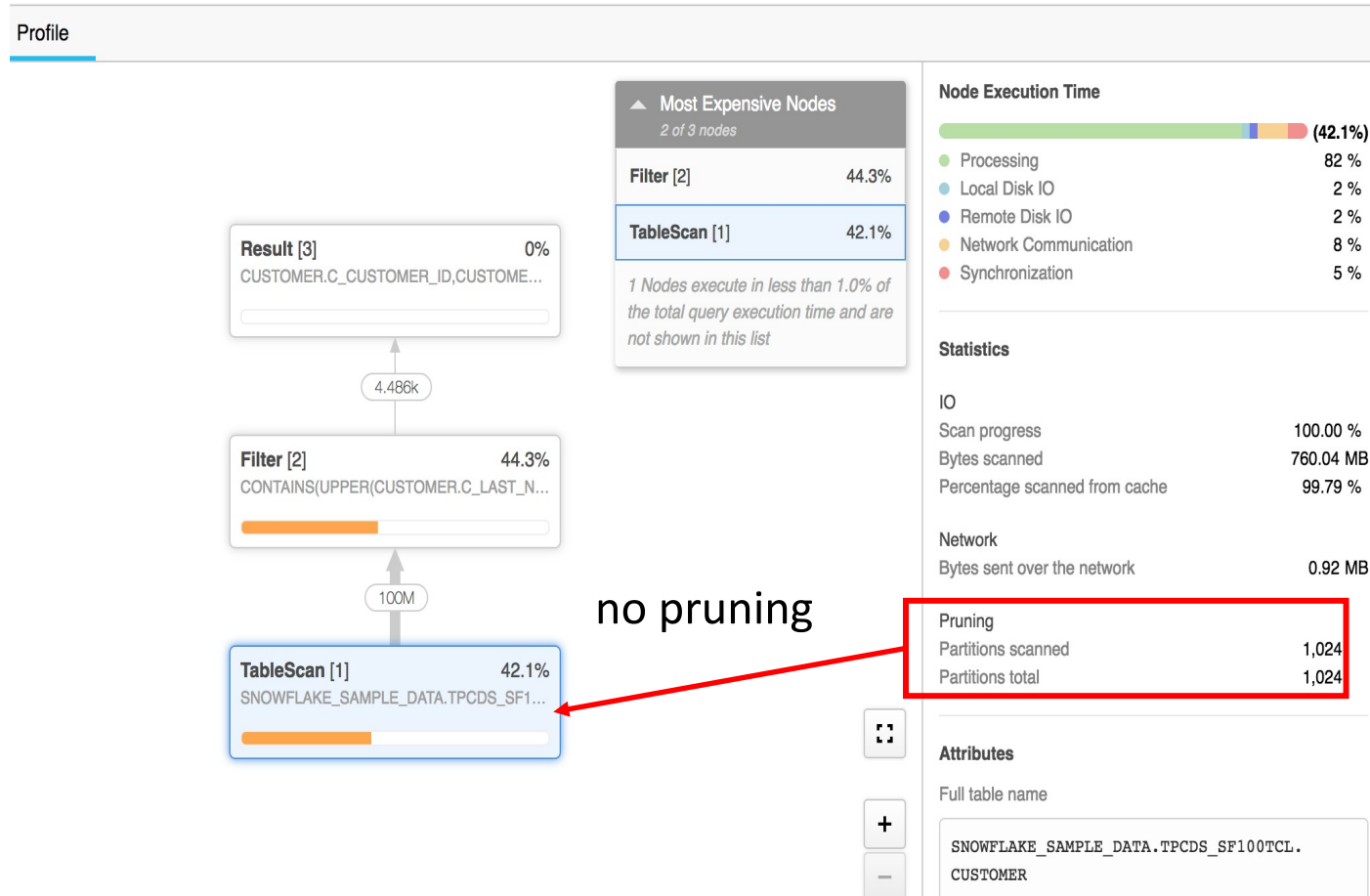


```
SELECT ... FROM order, ... WHERE  
o_orderdate >= to_date('1993-10-01')  
and  
o_orderdate < dateadd( month, 3,  
to_date('1993-10-01'))
```

Effective static pruning: query's filter  
column matches table's natural  
clustering order

- Order table has natural clustering based on data ingestion order
- Query's filter column is o\_orderdate and highly correlated to ingestion date

# PITFALLS OF NON-PERFORMING PREDICATES



Some WHERE predicates provide no partition pruning optimization opportunity:

- Predicate columns not matching the clustering (sort) key of table

```
SELECT c_customer_id, c_last_name
FROM tpcds.customer
WHERE UPPER (c_last_name) LIKE '%KROLL
%';
```

- Predicate column evaluated to result of subquery

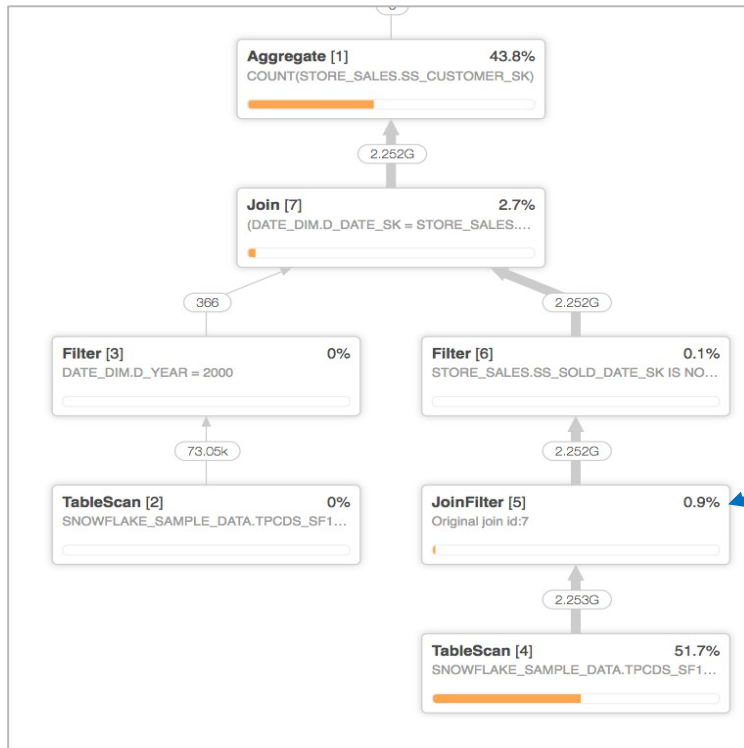
```
SELECT ...
FROM table1
WHERE c_date > (SELECT max(b_date)
FROM table3 where key = 10)
```

# DYNAMIC PARTITION PRUNING

```
SELECT count(ss_customer_sk) FROM store_sales JOIN
date_dim d
ON ss_sold_date_sk = d_date_sk
WHERE d_year = 2000 GROUP BY ss_customer_sk;
```

## Assisting SQL Optimizer:

- Join column, `ss_sold_date_sk`, matches larger table's clustering
- Effective dynamic pruning in larger table of hash join (e.g. `store_sales`)
- Pushdown via join filter to larger table
- Data type of join columns can impact effectiveness of pruning



<b>Pruning</b>	
Partitions scanned	16,912
Partitions total	86,546
<b>Attributes</b>	
Full table name	
SNOWFLAKE_SAMPLE_DATA.TPCDS_SF10TCL.STORE_SALES	
Columns (2 / 24)	
SS_SOLD_DATE_SK	
SS_CUSTOMER_SK	

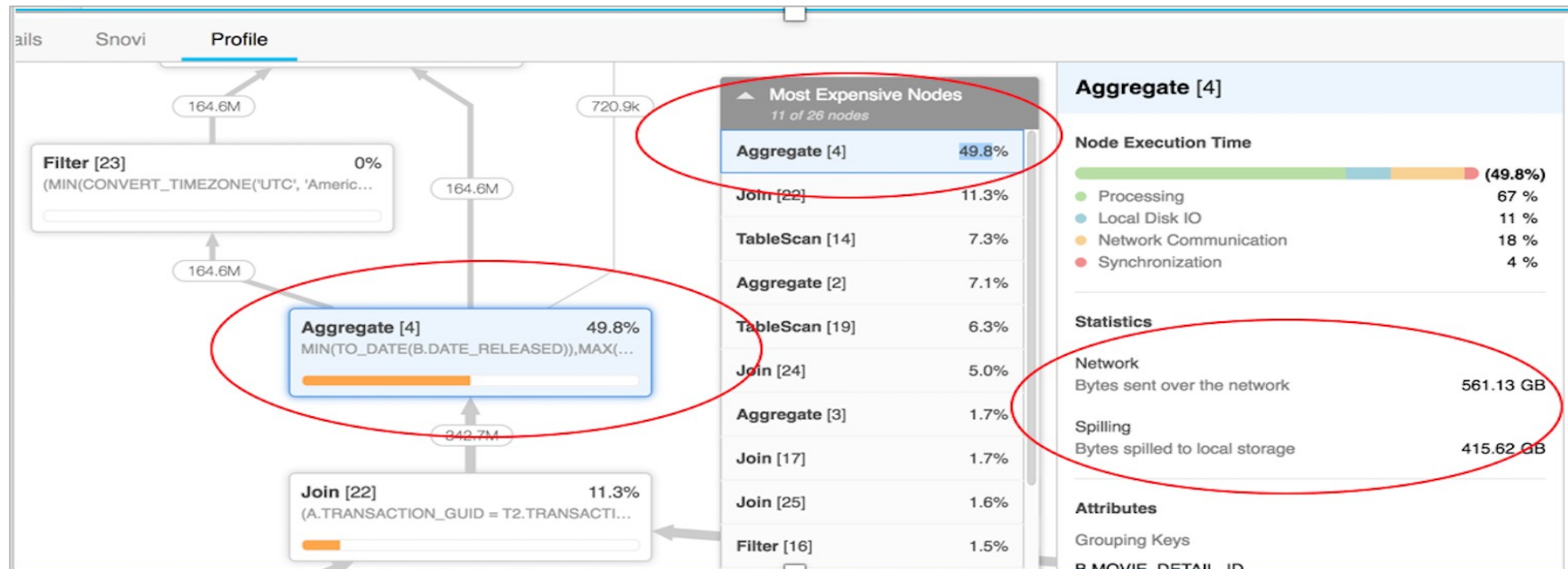
# AVOID COMPLEX FUNCTIONS AND UDFS IN PREDICATES

- Built-in functions and UDFs are very useful
- Can impact performance when used in query predicate
- **Best Practice:** use the right functions in the right place
- Materialize the intermediate result using temporary table

```
select l_orderkey
from lineitem l, orders o
where l_orderkey=o_orderkey and
      log(10, l_extendedprice) > 4.5 and
      log(10, o_totalprice - l_tax) > 4.5|
```

# HAVE A LARGE GROUP BY?

- If a Query has group by columns with large number of distinct values, this task will be **memory intensive**
- This task will risk of large spilling to disk and high network data, resulting on **suboptimal performance**
- Work with your admin to size up to bigger virtual warehouse if better performance is desired



# USE ORDER BY ONLY IN RESULT SET

## Best Practice:

### Use ORDER BY in top level SELECT only

- ORDER BY in subqueries and common table expressions has no effect
- [https://en.wikipedia.org/wiki/Order\\_by](https://en.wikipedia.org/wiki/Order_by)
- To avoid performance impact and wasting compute resource

```
select
    o_orderpriority,
    count(*) as order_count
from
    orders
where
    o_orderdate >= to_date('1993-07-01')
    and o_orderdate < dateadd(month, 3, to_date('1993-07-01'))
    and exists (
        select
            *
        from
            lineitem
        where
            l_orderkey = o_orderkey
            and l_commitdate < l_receiptdate
            order by l_orderkey
        )
group by
    o_orderpriority
order by
    o_orderpriority;
```

Pitfall

Best Practice





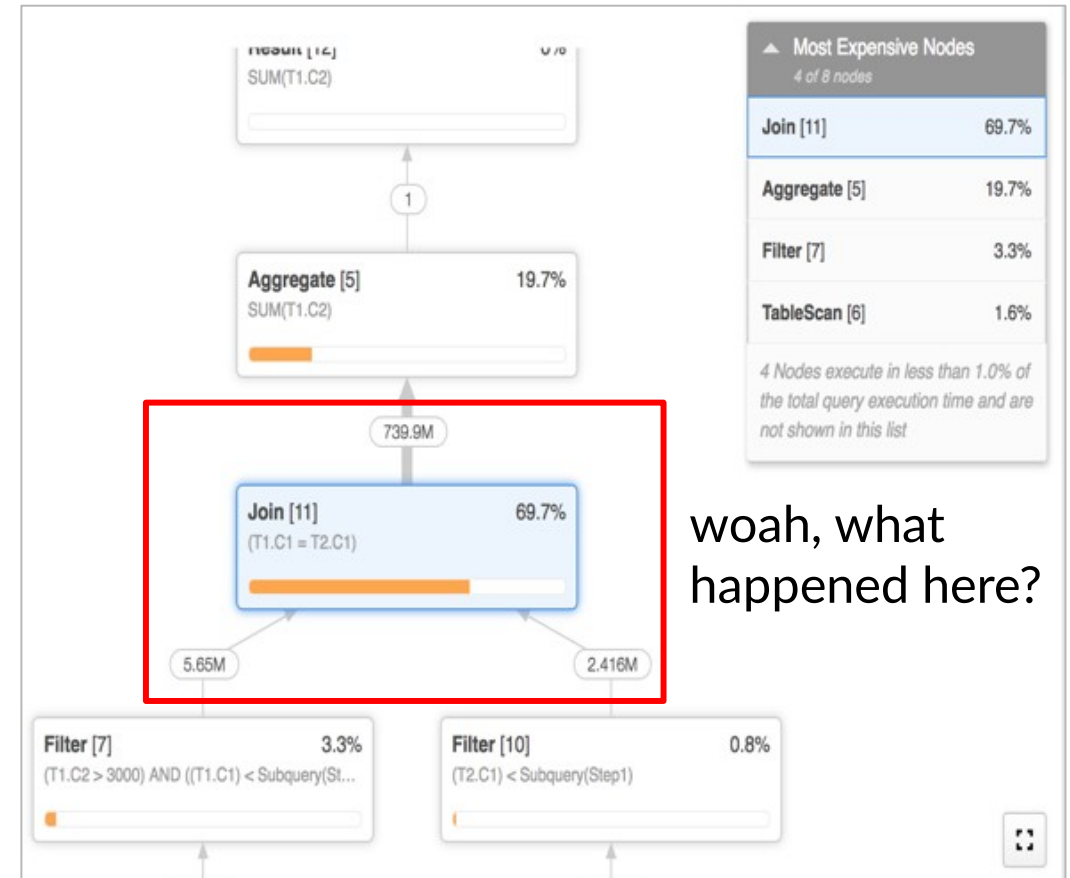
# JOIN ON UNIQUE KEYS

## Best Practices

- Ensure keys are distinct (deduplicate)
- Understand relationships between your tables before joining
- Avoid many-to-many join
- Avoid unintentional cross join

## Troubleshooting Scenario

- Joining on non-unique keys can explode your data output
  - Each row in table1 matches multiple rows in table 2



# USE TEMPORARY TABLE FOR REPETITIVE SUBQUERIES

If using same subqueries in multiple queries:

- Use Snowflake temporary table to materialize subquery results
- Performance improvement from reducing repeated I/O
- Cost saving from reducing repeated computation
- Temporary table exists only within session

```
create temporary table
mydb.public.customer_total_return as (
  select sr_customer_sk as ctr_customer_sk,
         sr_store_sk as ctr_store_sk,
         sum(SR_RETURN_AMT_INC_TAX) as ctr_total_return
  from store_returns
  , date_dim
  where sr_returned_date_sk = d_date_sk
  and d_year = 1999
  group by sr_customer_sk, sr_store_sk
);

select c_customer_id
from mydb.public.customer_total_return ctr1
, store
, customer
where ctr1.ctr_total_return >
      (select avg(ctr_total_return)*1.2
       from mydb.public.customer_total_return ctr2
       where ctr1.ctr_store_sk = ctr2.ctr_store_sk
      )
and s_store_sk = ctr1.ctr_store_sk
and s_state = 'NM'
and ctr1.ctr_customer_sk = c_customer_sk
order by c_customer_id
limit 100;
```

# Warehouse size impact on query performance

- Large or Complex queries can be affected by the warehouse size\*
- Performance can see improvement by increasing warehouse size
- “Disk spills” are an indicator that a query is resource constrained
- Test on larger warehouse

\*Increasing warehouse size will not resolve queuing of queries. Use larger multi-cluster warehouses to address (see next slide)

**Configure Warehouse**

Name ALPHA\_WH

Size **X-Large (16 credits / hour)**

Maximum Clusters Small (2 credits / hour)

Scaling Policy Medium (4 credits / hour)

Auto Suspend Large (8 credits / hour)

X-Large (16 credits / hour)

2X-Large (32 credits / hour)

3X-Large (64 credits / hour)

4X-Large (128 credits / hour)

☒ Auto Resume ?

Comment YourName Warehouse

[Show SQL](#) Cancel Finish

# What to Look for - Warehouse Disk Spilling

- When Snowflake cannot fit an operation in memory, it starts spilling data first to disk, and then to remote storage.
- These operations are slower than memory access and can slow down query execution a lot
- To see if a query is spilling to disk, look at the right-hand side of the **Profile** tab in the web interface
- In this example, 37.59 GB of data spilled to disk.

## Profile Overview (Aborted)

Total Execution Time (8m 20.81s)



## Total Statistics

IO	
Scan progress	100.00 %
Bytes scanned	5.61 GB
Percentage scanned from cache	100.00 %
Bytes written	0.81 MB
Pruning	
Partitions scanned	1,328
Partitions total	1,328
Spilling	
Bytes spilled to local storage	37.59 GB



# What to Look for - Warehouse Disk Spilling

- To find out more, click on a node in the **Query Profile** graph. As spilling to disk is so slow, the node of interest is most likely one where the query is spending significant time.
- Here's an example of a sort operator spilling to disk:
- In this case, the amount of data spilled is low, just about 2GB.
- This fits in the local disk cache, and this is indicated as bytes spilled to **local** storage. When the local storage is not enough to hold the data, another level of spilling occurs: the data is spilled to **remote** storage. This is significantly slower than spilling to local storage.

## Sort [2]

### Node Execution Time



### Statistics

#### Spilling

Bytes spilled to local storage



2.07 GB

# Approaches to resolve slow queries caused by disk spillage

- **Use a bigger warehouse.** This increases the total available memory and node/cpu parallelism, and is therefore often a good way to make this type of query faster.
- **Reduce the number of rows that must processed.** Storage Data clustering is often very effective for this as it allows Snowflake in many cases to only read the appropriate subset of the data that is really needed.

## Sort [2]

### Node Execution Time



### Statistics

#### Spilling

Bytes spilled to local storage



2.07 GB



# VIRTUAL WAREHOUSE

## SCALE *UP* FOR PERFORMANCE

### ELASTIC PROCESSING POWER (CPU, RAM, SSD)

- Raw performance boost for complex queries or ingesting large data sets
- Typically more complex queries on larger datasets require larger warehouses

### WAREHOUSE SIZING GUIDELINES

- Snowflake utilizes per-second billing: Use larger warehouses (L, XL, 2XL, etc) for more complex workloads and (auto) suspend when not in use
- Keep queries of similar size and complexity on the same warehouse to simplify compute resource sizing

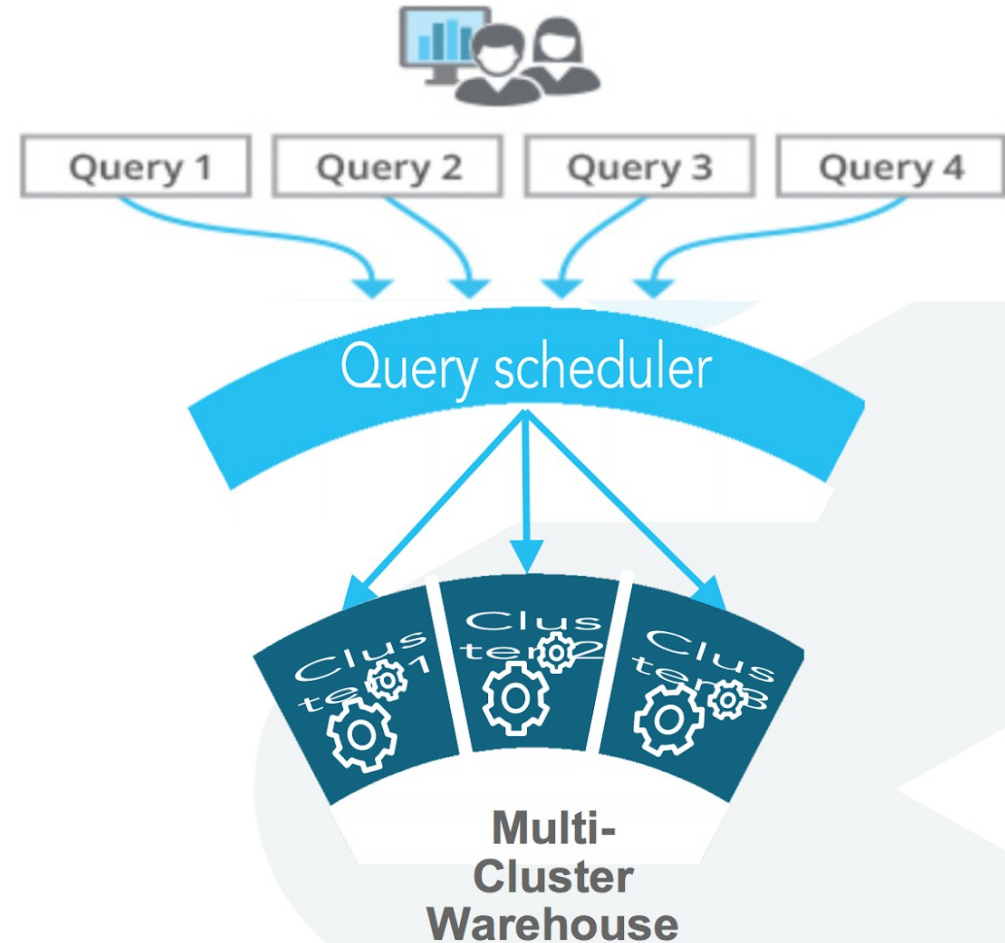




# VIRTUAL WAREHOUSE

## SCALE OUT FOR CONCURRENCY

- Single virtual warehouse with a number of compute clusters
- Deliver consistent SLA, automatically adding and removing compute clusters based on concurrent usage
- Scale out during peak times and scale down during slow times for cost savings
- Queries are load balanced across the clusters in a virtual warehouse
- Compute clusters deployed across availability zones for high availability





# SETUP TEST SESSION

- Result Set Caching

- Query History – tagging queries

Total Duration	Bytes Scanned	Rows	Query Tag
6.0s	3.4GB	4	TEST-Query
59ms			
433ms	496	1	
102ms			

- Control Size of Returned Result Set

```
ALTER SESSION SET  
USE_CACHED_RESULT = FALSE;
```

```
ALTER SESSION SET QUERY_TAG =  
<string>
```

```
ALTER SESSION SET  
ROWS_PER_RESULTSET = <num>
```



# Security

# WCET

---

WARNER  
CONTENT &  
ENTERPRISE  
TECHNOLOGY

# Security

- Unless you are sure, treat every data element as PII
- Check with Snowflake Admins/Ops if a data element is PII
- Snowflake Admins can help you define rules to obfuscate PII
- Careful with grants to others

Thank You

