O'REILLY®

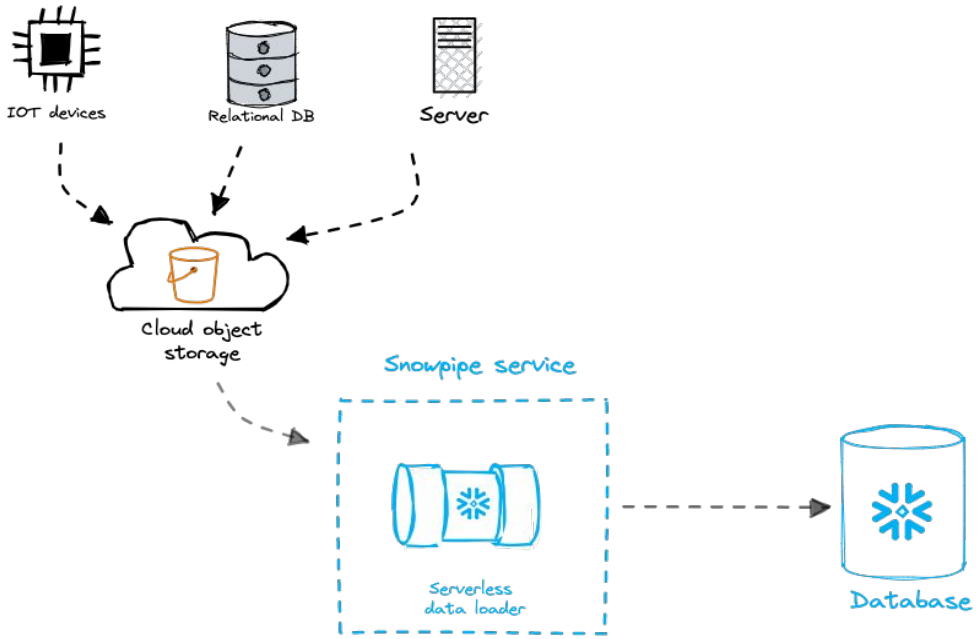
Snowpipes

How to do continuous ingestion in Snowflake

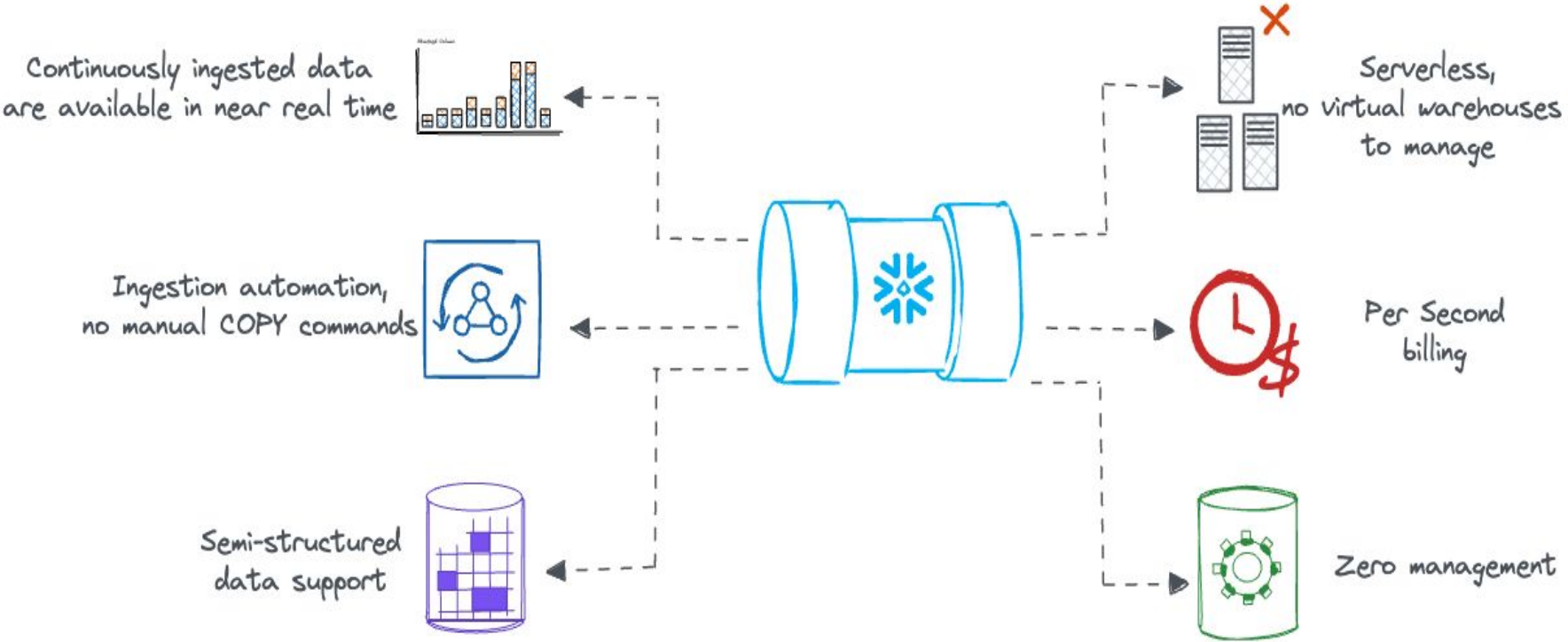
Tomas Sobotik

Snowpipes

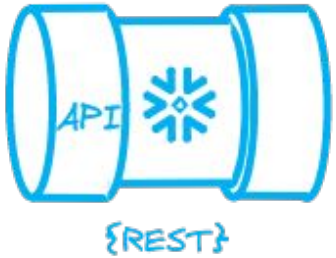
- Automated data loading feature
- Near real time data streaming
- No need to dedicate and suspend virtual warehouse
- Serverless with per second billing
- File (batch) oriented
- Credit consumption is visible under SNOWPIPE warehouse



Faster data ingestion with Snowpipes



Snowpipes – REST API



- Manually call Snowpipe REST API endpoint
- Pass a list of stage files in the stage location
- Works with internal and external stages

Preparing to load data using REST API

1. Create Stage
2. Create a Pipe
3. Security configuration
    - Generate Public-private key pair for making calls to REST API
    - REST endpoint uses JSON Web Token (JWT) for authorization
    - Assign the public key to the user
    - Granting privileges to DB objects in use (DB, schema, pipe, target table…)
    - Best practice – create separate user to handle the REST api calls

4. Stage data files in internal/external stage
5. Call REST API endpoint `insertFiles`

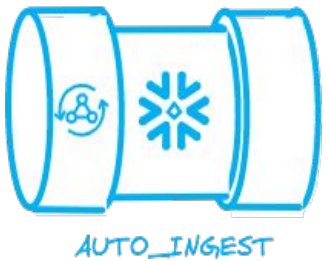
Snowpipe REST API endpoints

- `insertFiles`
  - `POST`
  - Informs Snowflake about files to be ingested into a table
  - `https://{account}.snowflakecomputing.com/v1/data/pipes/{pipeName}/insertFiles?requestId={requestId}`

- `insertReport`
  - `GET`
  - Retrieves a report of files submitted via `insertFiles`
  - The 10000 most recent events
  - Events are retained for max 10 mins

- `loadHistoryScan`
  - `GET`
  - Report about ingested files
  - Return history between two points in time
  - Max 10000 items returned
- For more comprehensive view of history without limits use Information Schema table function – COPY_HISTORY

Snowpipes – AUTO_INGEST

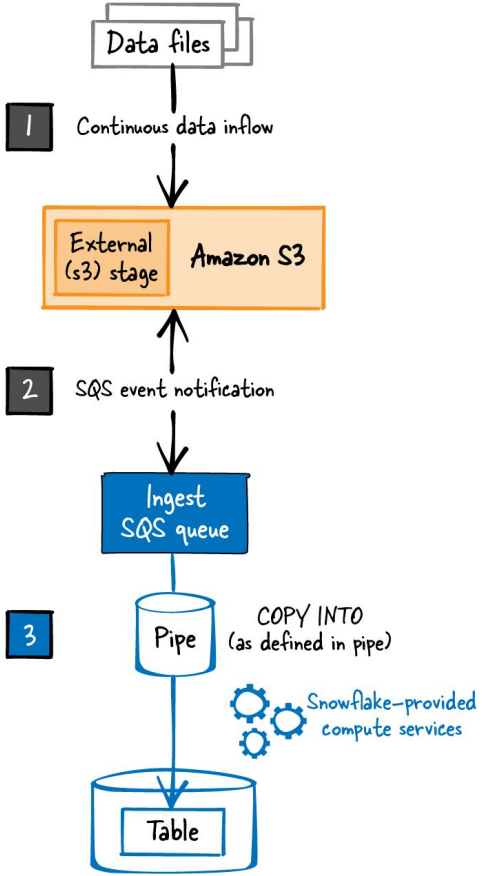


- Receive notifications from cloud provider when files arrive
- Notification trigger the processing
- Only external stages are supported

Snowpipe Cross-Cloud Support

Snowflake account Host	Amazon S3	Google Cloud Storage	Microsoft Azure Blob Storage	Microsoft Data Lake Storage Gen2	Microsoft Azure General-purpose v2
Amazon Web Services	✅	✅	✅	✅	✅
Google Cloud Platform	❌	✅	❌	❌	❌
Microsoft Azure	❌	❌	✅	✅	✅

How to Create Snowpipe with Auto Ingest

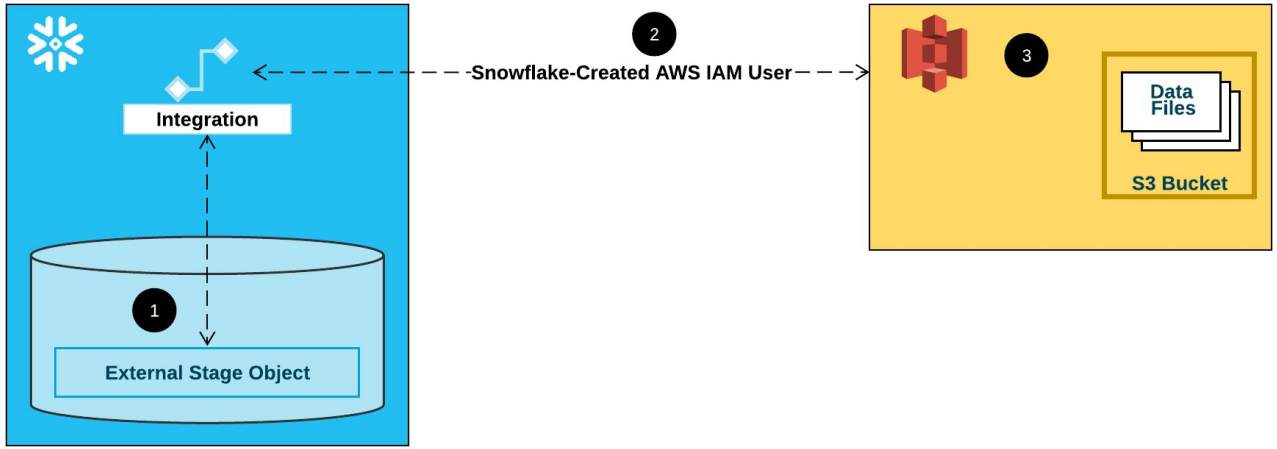



```
create pipe mypipe auto_ingest=true as
  copy into mytable
  from @my_db.public.mystage
  file_format = (type = 'JSON');
```

How to Create Snowpipe with Auto Ingest

- It requires configuration on cloud provider side
  - Bucket access permission
  - IAM role for Snowflake
  - SQS notification for S3 bucket
- Snowflake configuration
  - Storage integration object - encapsulates the access privileges for accessing the S3 bucket

Configure secure Access to cloud Storage



1. External stage references a storage integration object
2. Snowflake associate storage integration with S3 IAM user created for your account
3. AWS admin grants permission to the IAM user to access the bucket referenced in the stage definition.

Create a S3 Event Notification to Automate Snowpipe

SQS setup

1. Execute show pipes to get the notification channel associated with snowpipe

2. Configure event notification for S3 bucket associated with the stage
   - All events related to object creation
   - Send a notification to SQS Queue we got in the first step

Exercise

- Let’s configure integration between Snowflake and AWS for Snowpipe ingestion.
- Once we have the integration ready we can create a landing table in Snowflake and Snowpipe object.
- After that we are going to automatically ingest data into Snowflake via Snowpipe.

How to work with query profile

Query Plan / Query Profile

- DAG consists of operators connected by links
- Operators – process a set of rows
  - Table scan, filter, join, order, aggregate, etc.
- Links
  - Exchanging data between operators
  - Handle parallel redistribution of data at run time

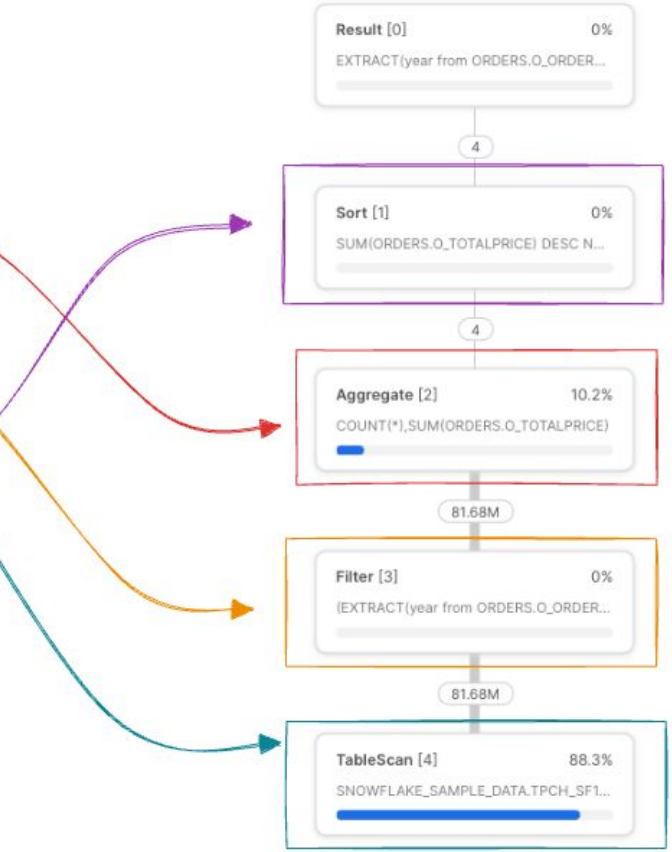
How to access it?

- Query history
- Query Details in Worksheet
- Programmatically

```
GET_QUERY_OPERATOR_STATS()
```

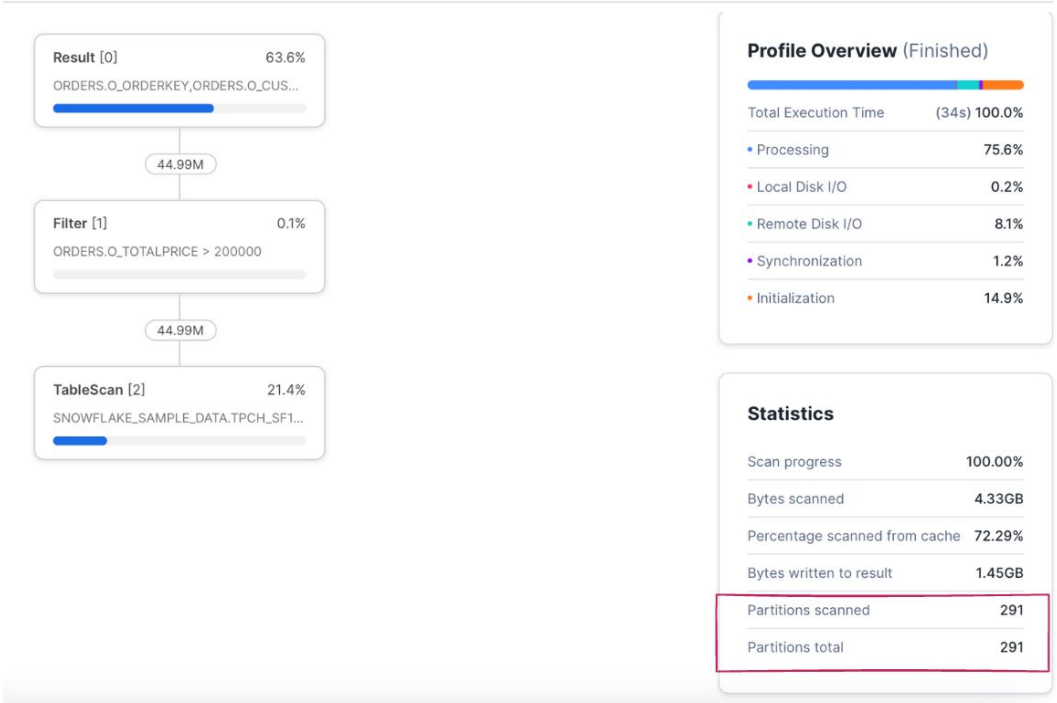
Query Mapping to Operators

```
select
  year(o_orderdate) as year,
  count(*) order_count,
  sum(o_totalprice) total_amount
from orders
where year(o_orderdate) > 1994
group by year
order by 3 desc;
```



Identifying Query Pruning Performance

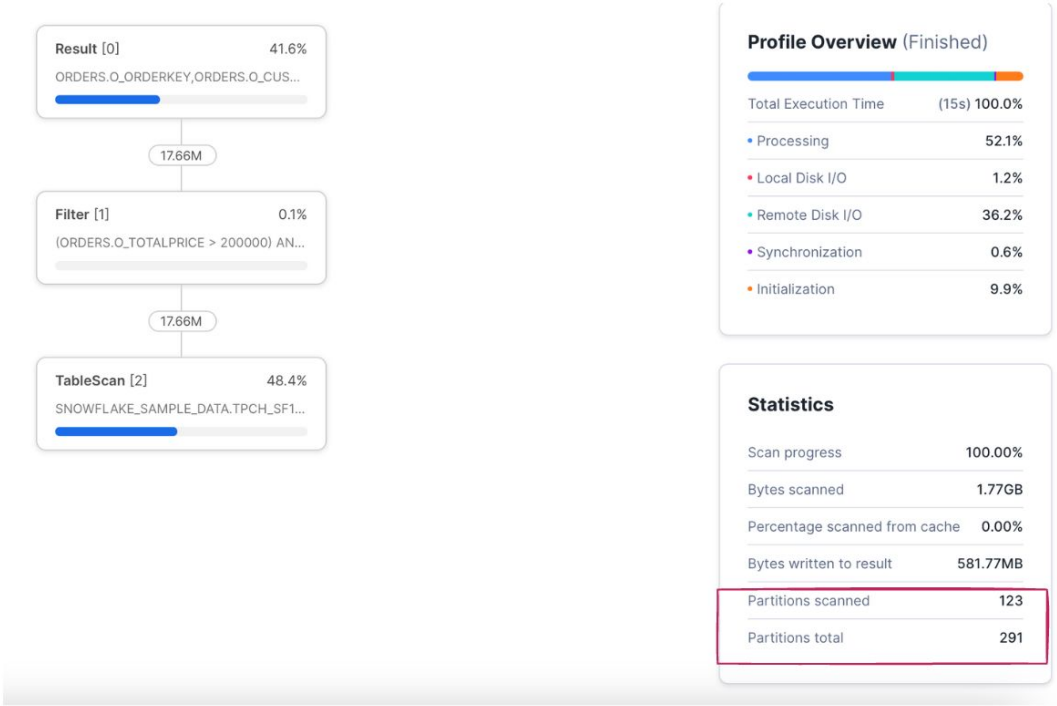
```
select * from orders where o_totalprice > 200000;
```



- Bad pruning
- Full table scan
- Table is not clustered on that column
- Add your data knowledge

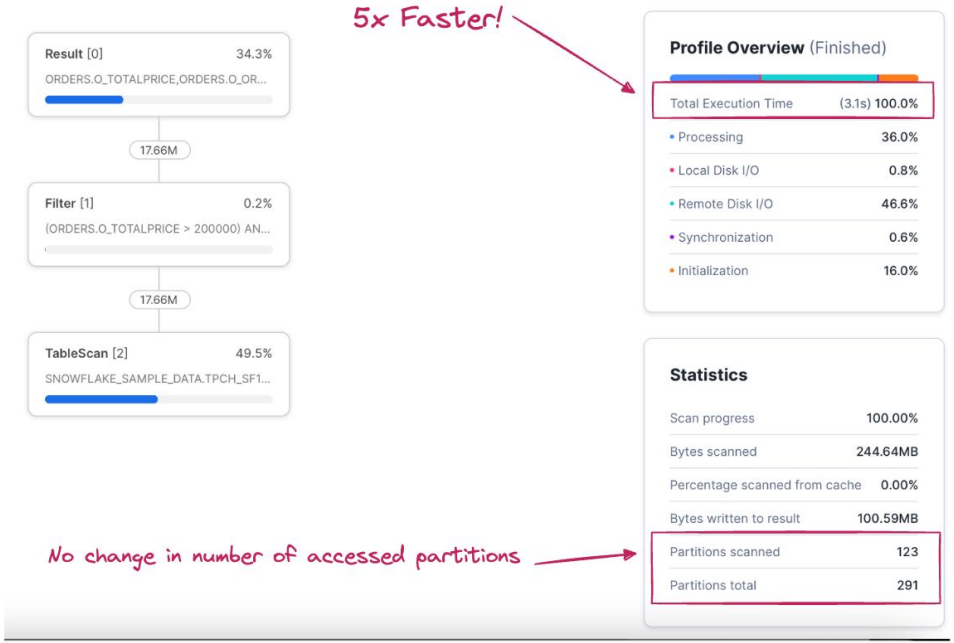
Identifying Query Pruning Performance

```
select * from orders where o_totalprice > 200000 and o_orderdate > '1996-01-01';
```



- Good pruning
- Table organized along date column
- What else can we improve?

Identifying Query Pruning Performance



- Pruning on column level
- Add only needed/relevant columns

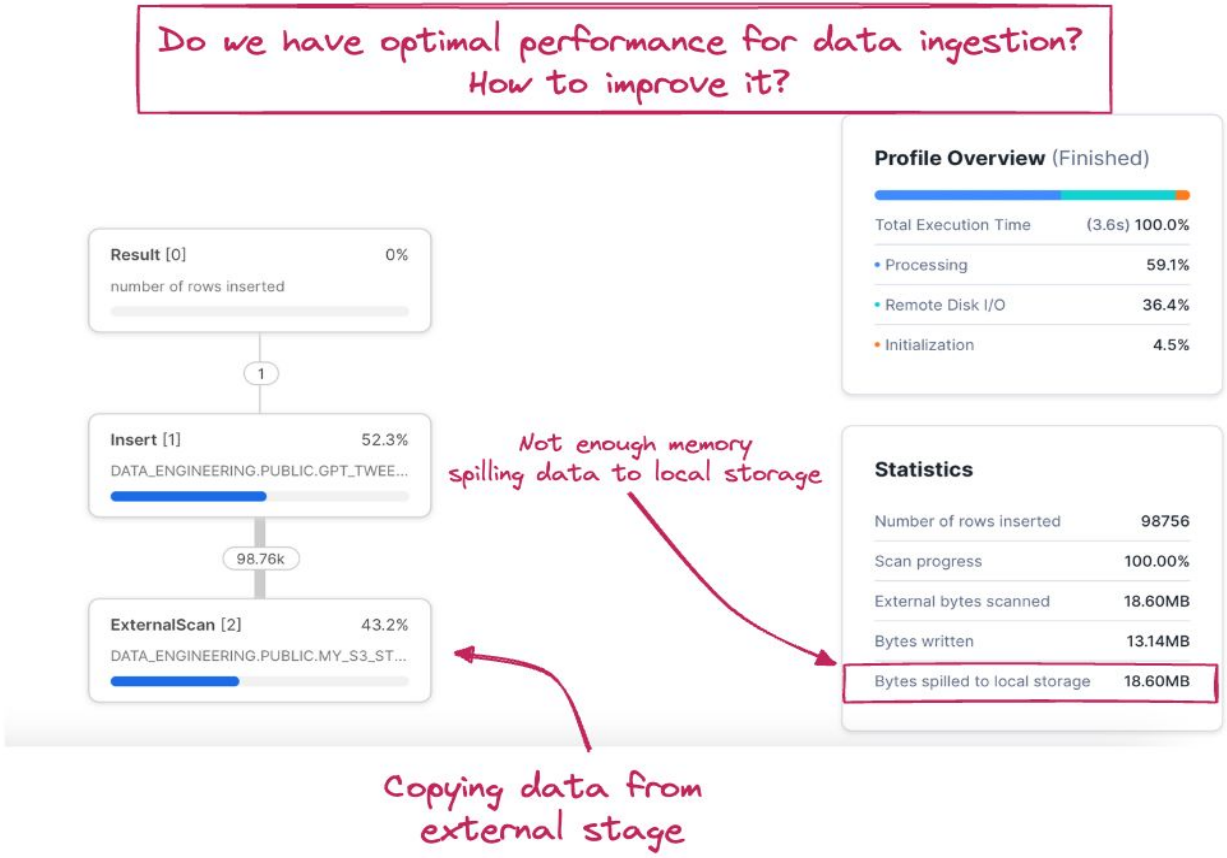
```
select
  o_totalprice,
  o_orderdate,
  o_orderpriority
from orders
where o_totalprice > 200000 and o_orderdate > to_date('1996-01-01', 'YYYY-MM-DD');
```

Query Profile and Data Spilling

Data Spilling

- Data spilling – out of memory issue
- Common performance issue related to long execution times
- WH doesn’t have enough memory to process your request
- Many root causes
- How to use query profile to identify such issues?
- How to check for old queries?

Data Spillage – Query Profile



- Not enough memory/CPU power on WH
- Local disk spilling
- Remote disk spilling

Data Spillage – Root Causes

- Large amount of data for processing
- Wrong warehouse size
  - Not fitting the workload
- Concurrency of queries
- Large intermediate results & memory intensive operations
  - Sorting

Data Spillage – How to Solve

- Increase warehouse size (more memory)
- Reduce concurrency
  - Split the workload between more WH
  - Configure multi-cluster WH
- Improve Pruning
  - Auto clustering
  - Intelligent data insertion

Knowledge check

1.What does operators do in query plan?
a] exchanging data between operations
b] handle parallel data distribution
c] processing a set of row
d] helps understand the query performance

2. How to identify full table scan in Query Plan
a] There is FullTableScan operator used
b] All micropartitions are scanned
c] Local spilling is used
d] Nothing is scanned from cache

3. What is data spilling issue?
a] not enough memory to process request
b] missing cluster key
c] cache is full
d] exploding join condition with non unique keys

Snowflake Caches

Basics

Snowflake has multiple caches

Proper caching understanding have impact on performance and cost

Caches resides in different architecture levels

Cache types

Metadata cache

- Object definitions
- DB objects statistics
- Micro partition statistics
- Part of cloud service layer

Local / SSD / Virtual WH cache

- Raw data from tables (not aggregated)
- The size of the cache is determined by the number of servers in the warehouse
- Warehouse suspended = cache dropped
- Partial data can be used

Result cache

- Exact results from exact queries
- 24 hours
- Underlying data cannot change
- Some functions cause expirations
- Different users but same role

How to use metadata cache

- Queries for count, min, max values, show command …
- No virtual WH needed -> queries are for free!
- Used automatically by optimizer
- Metadata are stored in cloud service layer
- Micro partition metadata used during query planning
  - MIN/MAX values
  - DISTINCT values

How to use local cache

- Data stored directly on SSD of virtual WH
- Active working data set - not the result
- When a similar query is run, Snowflake will use as much data from Cache as possible
- Available for all queries run on same virtual warehouse
- WH first read local data and then fetch the rest from remote cloud storage

Best practises

- Group similar queries to same virtual warehouse to maximize usage of local cache, get better performance and lower cost
- Do not suspend your WH too early = cache is destroyed
- Plan suspension times wisely based on use case

How to use query result cache

- Query results are stored in cloud service layer
- When identical query runs again -> retrieve the data from query result cache -> no cost, superb performance

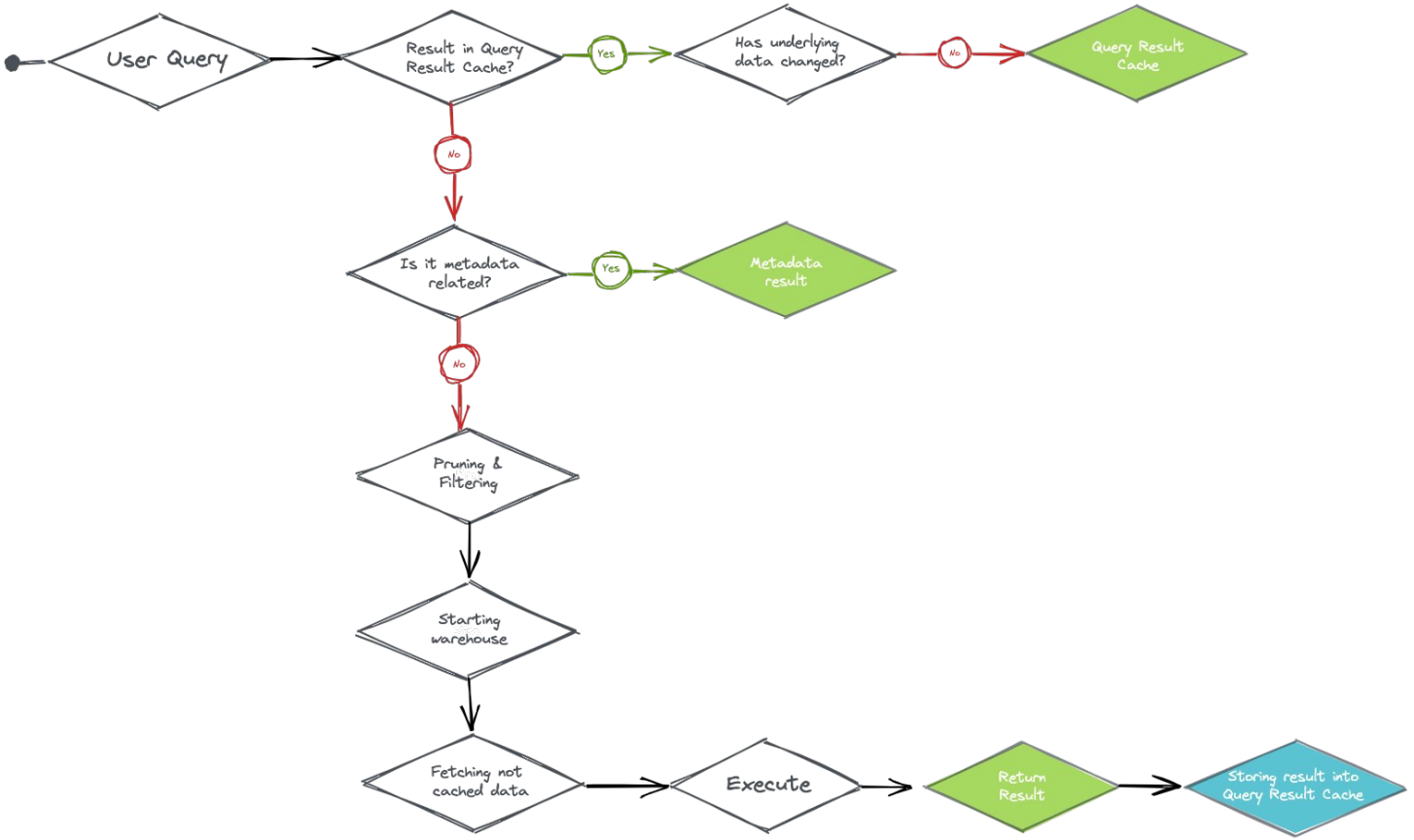
Conditions

- Underlaying data has not been changed
- Query does not use functions which do not support result cache (CURRENT_TIMESTAMP)
  - Query needs to be deterministic, not random
- Result stored for 24 hours, if reused then the counter resets. Up to 31 days

Typical use cases

- Static dashboards
- Dashboard filters

Query life cycle



Cache summary

	Metadata Cache	Query Result Cache	Data Cache
Stored	Cloud Service Layer	Cloud Service Layer	Virtual Warehouse SSD
Storage	Metadata and statistics for micropartitions and tables	Results set for each query	Data used in query
Used	By the optimiser service	Identical query executed again Base table has not changed	Query using some or all of the same data Data has not changed
Time	Continuously updated	24 hours Resets after every run	When WH runs
Who	Everyone	Users with SELECT permissions on all tables	Anyone with the same warehouse access

Knowledge check

1. What is not Snowflake cache

a) Metadata Cache

b) Cloud service layer cache

c) SSD cache

2. Which Snowflake cache is reseted after 24 hours

a) Metadata Cache

b) SSD Cache

c) Query result cache

3. Which Snowflake cache keep underlying raw data

a) SSD Cache

b) Metadata Cache

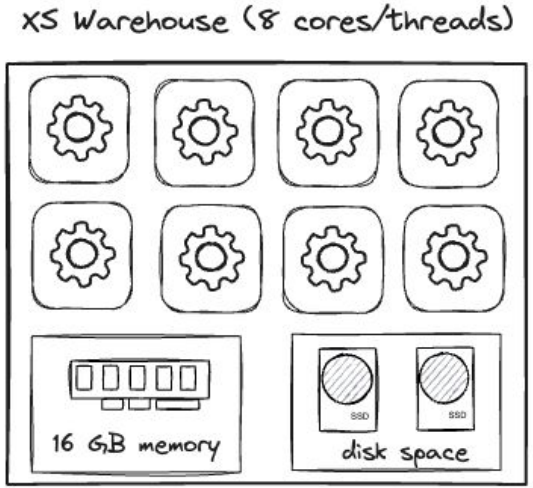
c) Query result cache

Virtual Warehouse scaling

Warehouse Sizing

- How to find out the right warehouse size?
  - Experiment with typical queries on different WH sizes
- Start with an X-SMALL size
  - Increasing the size until query duration stops halving

    → not fully utilizing WH anymore
  - Or meet your SLAs

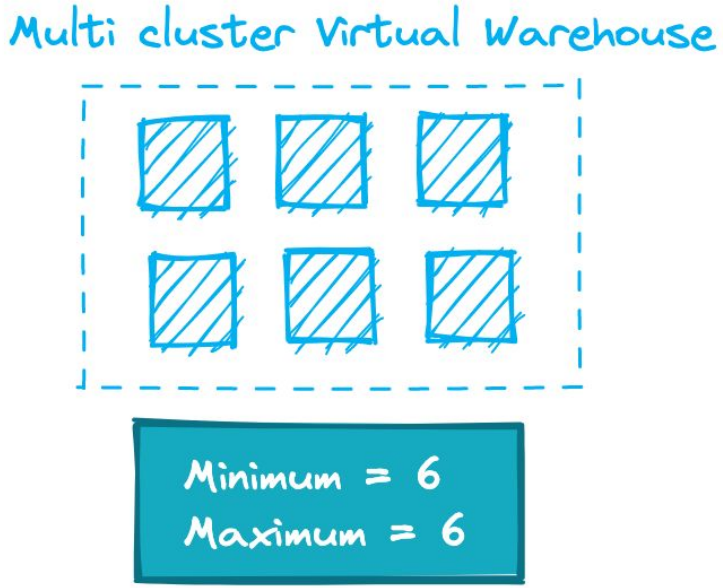


Query Timeouts

- Automatic cancellation of running queries
- Default value is 2 days!
  - Can lead to unintentional high cost
- Configurable at different levels
  - Account
  - Session
  - User

Multi-Cluster Warehouse

- Single virtual warehouse with multiple compute clusters
- Two new properties
  - MIN_CLUSTER_COUNT – minimum number of clusters in
  - MAX_CLUSTER_COUNT – maximum number of clusters
- Allocation of resources could be done statically or dynamically
  - scaling modes & policies



Scaling Modes for Multi-Cluster Warehouse
Maximized

- Static control of compute resources
- Enabled by specifying same value for minimum and maximum number of clusters
  - Value must be larger than 1
- Specified number of clusters is immediately started once warehouse is resumed
- You have large number of concurrent users/queries

Scaling Modes for Multi-Cluster Warehouse

Auto-scale

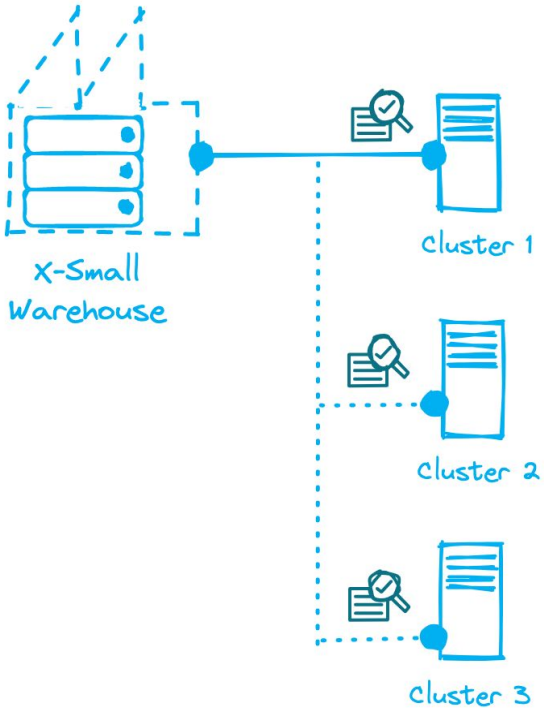
- Dynamic control of compute resources
- Enabled by specifying different values for min and max number of clusters
- Clusters are started/suspended based on the load
- Strategy is controlled by scaling policy

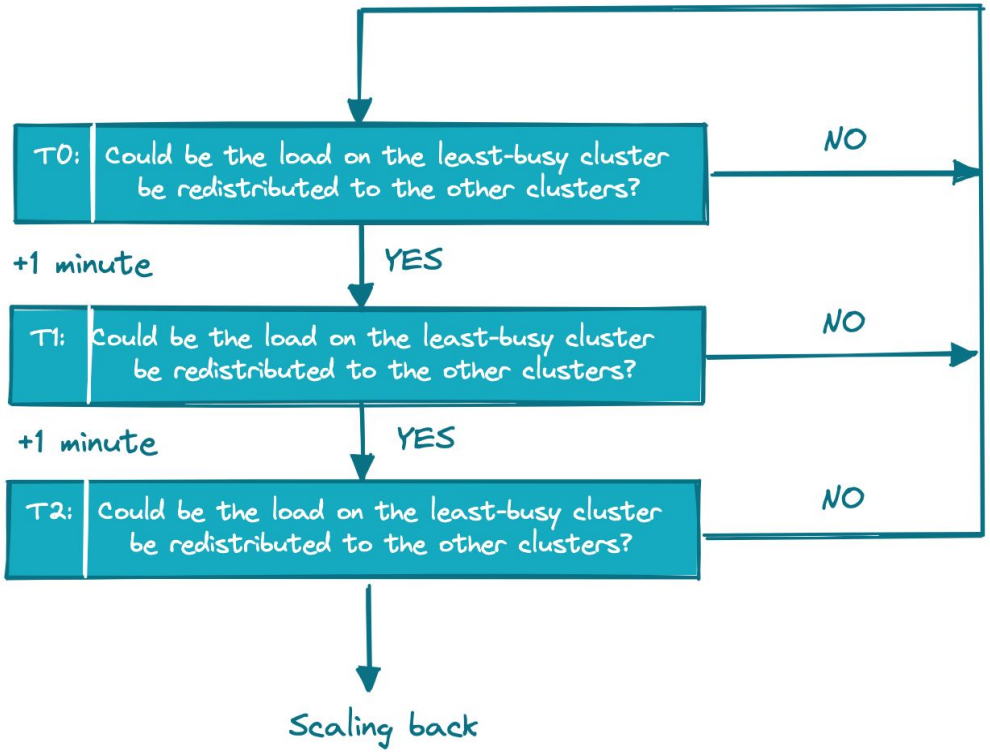
Auto-Scaling Mode for Multi-Cluster Warehouse

Standard scaling policy

Description	Cluster Starts	Cluster Shuts Down
Starts a new cluster as soon as queries start to queue	There is queued query or there is more queries than the currently running cluster can execute	After 2-3 consecutive checks (one minute interval) show that the load on the least-busy cluster could be redistributed to other running clusters

Standard Policy: Scaling Back Workflow



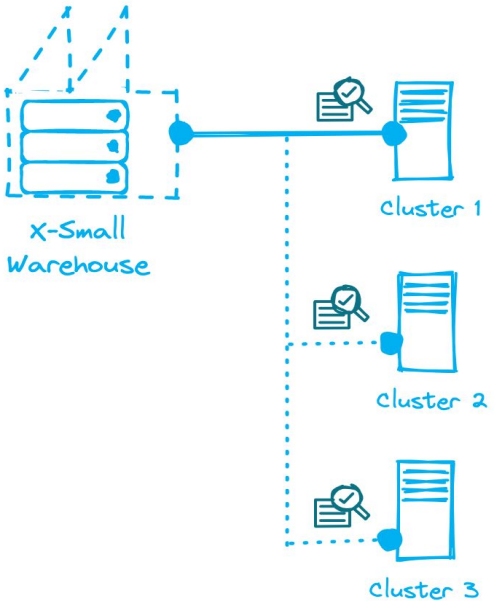


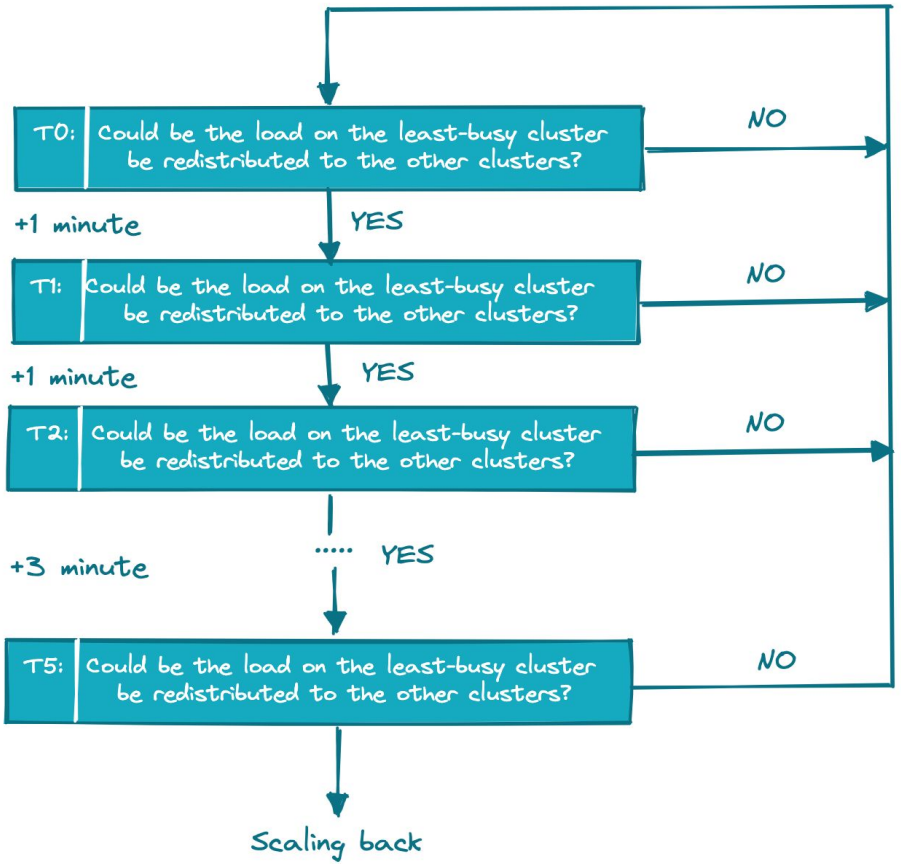
Auto-Scaling Mode for Multi-Cluster Warehouse

Economy scaling policy

Description	Cluster Starts	Cluster Shuts Down
Queries are kept queued up to six minutes - to see if the queue clears. Then another cluster is started	Only if the system estimates there's enough query load to keep the cluster busy for at least six minutes	After 5-6 consecutive checks (one minute interval), show that the load on the least-busy cluster could be redistributed to the others

Economy Policy: Scaling Back Workflow

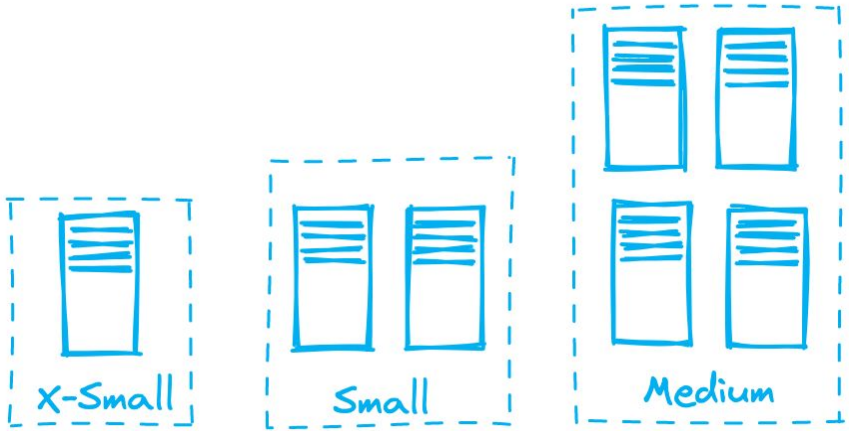




Scaling up vs Scaling out

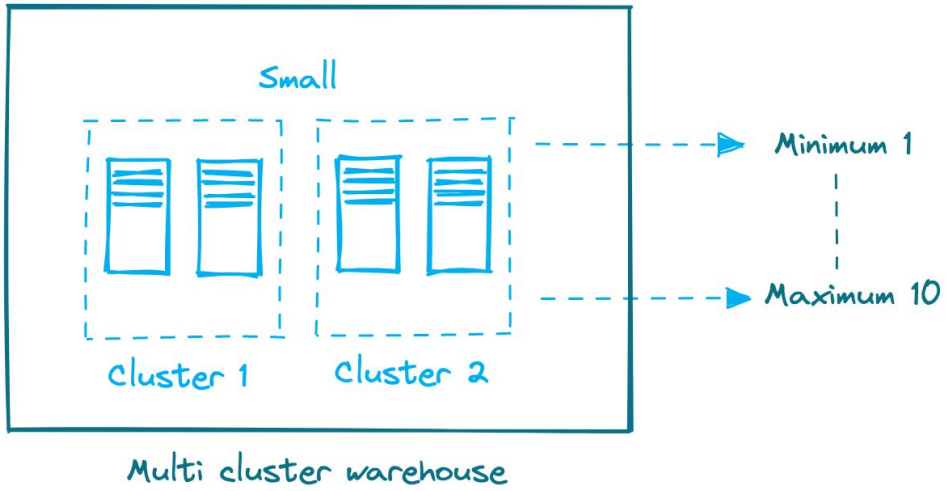
Scaling Up

Resizing the single warehouse in order to handle more complex queries with more data



Scaling Out

Adding more clusters to the warehouse in order to handle more concurrent requests. Size of the warehouse is not changed.



Scaling UP vs. OUT

Credits per hour

	1	2	3	4	5	6	7	8	9	10
4XL	128	256	384	512	640	768	896	1024	1152	1280
3XL	64	128	192	256	320	384	448	512	576	640
2XL	32	64	96	128	160	192	224	256	288	320
XL	16	32	48	64	80	96	112	128	144	160
L	8	16	24	32	40	48	56	64	72	80
M	4	8	12	16	20	24	28	32	36	40
S	2	4	6	8	10	12	14	16	18	20
XS	1	2	3	4	5	6	7	8	9	10

Scale up Summary

- Boost the performance for complex queries or ingesting large datasets
- More complex queries on larger datasets require larger warehouse
- Experiment with warehouse sizes to find the right balance between performance and cost
- Keep similar queries on the same warehouse to simplify compute resource sizing

Scale Out Summary

- Use for concurrency issues (more users/queries)

- Scaling can be controlled statically or dynamically
  - Maximized scaling mode
    - Static load over time
  - Auto-scale scaling mode
    - Load fluctuates over time

- Economy and standard strategies for auto-scaling policy

Knowledge check

1. How are called scaling modes for Multi-Cluster Warehouses
a] static, dynamic
b] maximized, auto-scale
c] economy, standard
d] standard, maximized

2. Scaling up means adding more clusters into multi cluster warehouse
a] true
b] false

3. How would you configure maximized scaling mode for Multicluster warehouse?
a] MIN_CLUSTER_COUNT = 5, MAX_CLUSTER_COUNT = 5
b] MIN_CLUSTER_COUNT = 9, MAX_CLUSTER_COUNT = 10
c] MIN_CLUSTER_COUNT = 1, MAX_CLUSTER_COUNT = 10
d] MAX_CLUSTER_COUNT = 10

Exercise

Let’s create a multi cluster warehouse called MULTI_COMPUTE_WH with following parameters:

- Small size
- Enabled auto resume
- Enable auto suspend
- Suspend after 3 minutes
- It will be suspended after creation
- Min number of clusters: 1
- Max number of clusters: 3
- Economy scaling policy

Table types and impact on cost

Storage basics

- Support for structured, semi-structured and unstructured data
- Always encrypted and compressed
- Stored across 3 availability zones
  - Separate power grids
  - Geographic separation
- Fully online updates and patches
- Billing = actual storage
  - Daily average Terabytes per month

Table Types I.

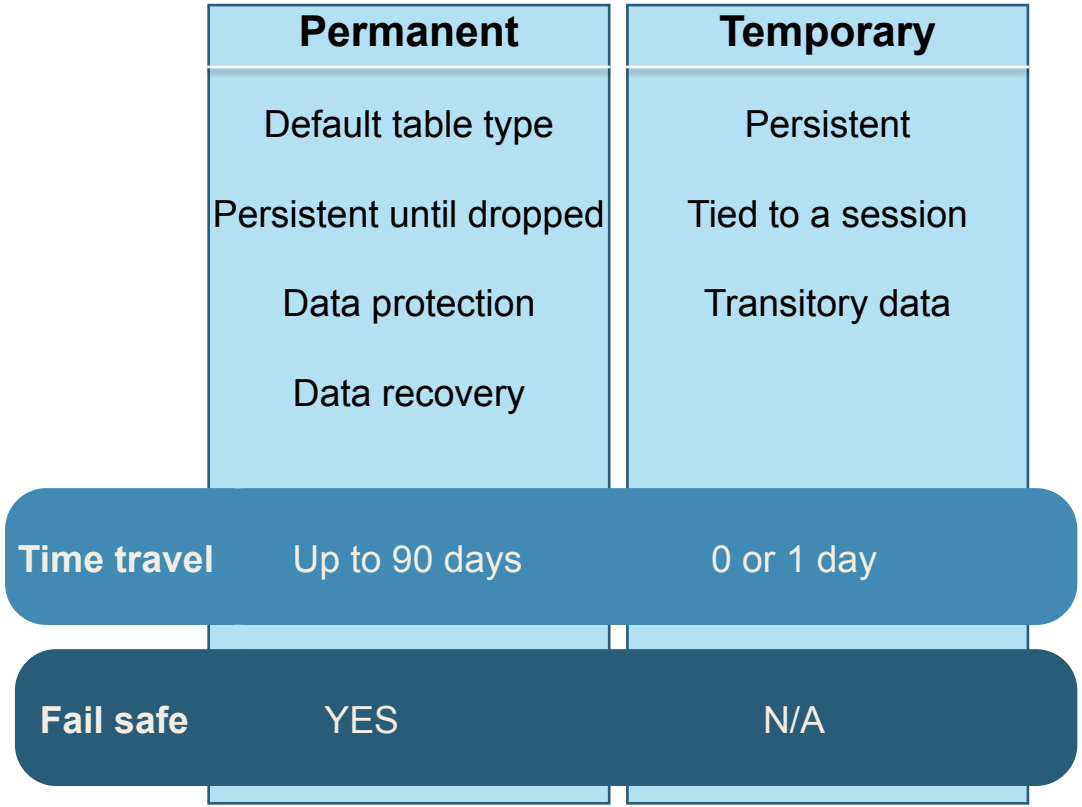
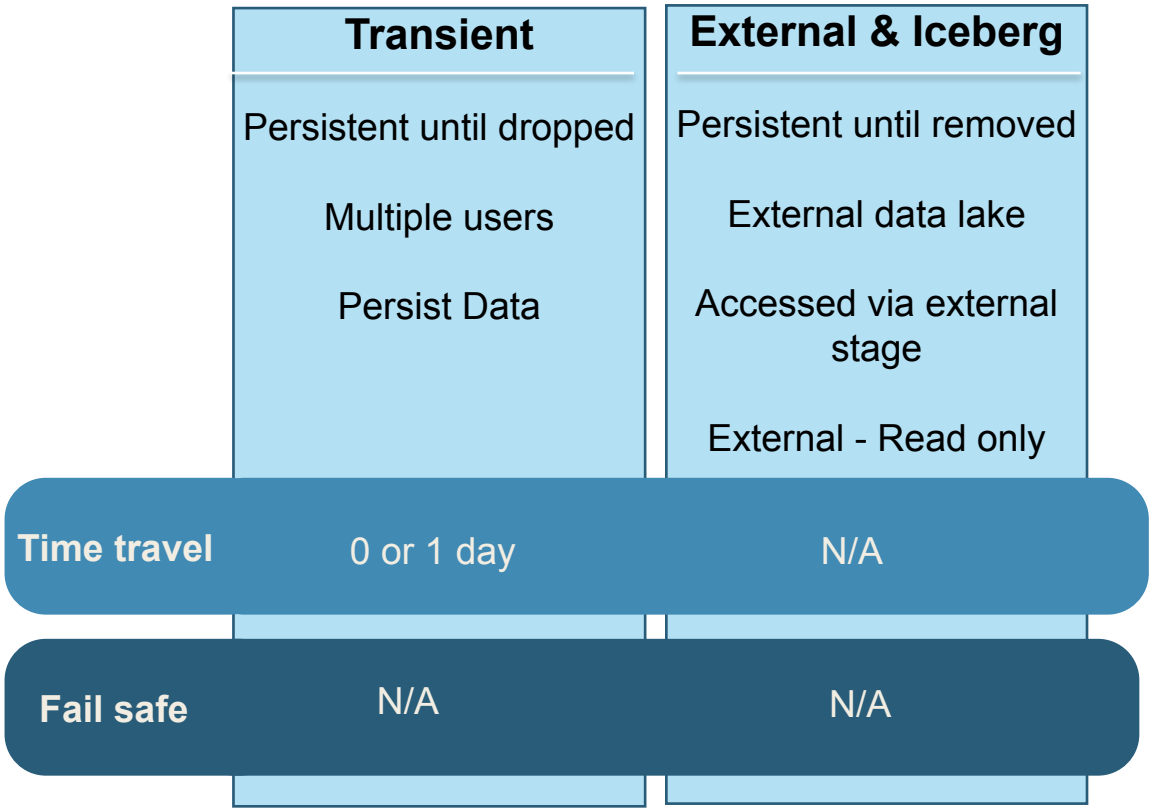


Table Types II.



View types

Standard View	Secure view	Materialized View
Default view type	Secure definition and details	Like a table
Owning Role	Owning Role	Results of underlying query stored
DDL available to role with access	Query optimizer bypassed	Auto-refreshed
		Secured Materialized Views

Materialized views

What

Store frequently used queries
To avoid wasting time and money

Results guaranteed to be up to date

Automatic data refresh

MV on tables and External Table

Simplify query logic

Enterprise edition feature

When

Large and stable datasets

Underlying data do not change
so often

Query result contains small number of
Rows/columns relative to base table

Query results contains results which
Requires significant processing

Cost

Serverless feature

Additional storage

Maintenance cost

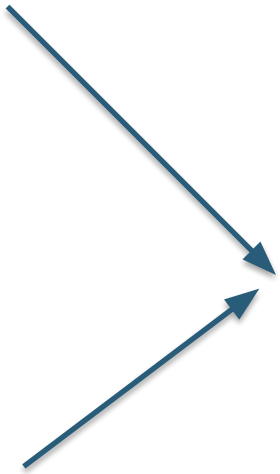
Materialized views limitations

Col 1	Col 2	Col N
	T	
	F	
	F	

Partition 14

Col 1	Col 2	Col N
	F	
	T	
	T	

Partition 28



Materialized view

Partition	Col 2	Sum
14	T	1000
14	T	2000
28	F	3000

- Only single table (no JOIN support)
- Cannot query
  - Another MV
  - View
  - User defined table function
- Cannot include
  - Window functions
  - HAVING clauses
  - ORDER BY clause
  - LIMIT clause
  - Many aggregate functions

Knowledge check

1. What are table types in Snowflake. Mark all of them
    a) Permanent
    b) Fail Safe
    c) Transient
    d) Session
    e) External
    f) User Defined

2. External tables can be updated
    a) True
    b) False

3. Which table type can be used only by single user
    a) Permanent
    b) External
    c) Temporary
    d) Transient
    e) User Defined

4. Secure view can get data from multiple tables
    a) True
    b) False

Exercise

Let’s define normal and secure views and have a look on differences.

Optimizing Query Performance

Performance optimization globally

- Self tuned platform providing leading performance and concurrency
- Instant and cost effective scaling
- Workload separation
- Requires almost no tuning
  - No indexes
  - No partition management

No extra effort is needed

There are still things, features & workflows which is worth to know

- Understand how compute engine executes queries (query profiler)
- SQL best practises in relation to Snowflake architecture
- Caching
- Data loading strategies

Cost Save

Snowflake built-in optimizations

- Partition pruning
  - Patented micro-partition pruning to optimize query performance
  - Static partition pruning based on columns in WHERE clause
  - Dynamic partition pruning based on JOIN columns of a query

- How you can assist to SQL optimizer
  - Apply appropriate filters as early as possible
  - For naturally clustered tables, apply predicate columns with high correlation to the ingestion order (e. g. date columns)

Clustering

- Data organization in micropartition
- Crucial element for partition pruning & performance
- Not for all tables
  - Fastest response times required
  - Large tables (TB in size)
  - Cost for maintaining the clustering!
- Cluster key

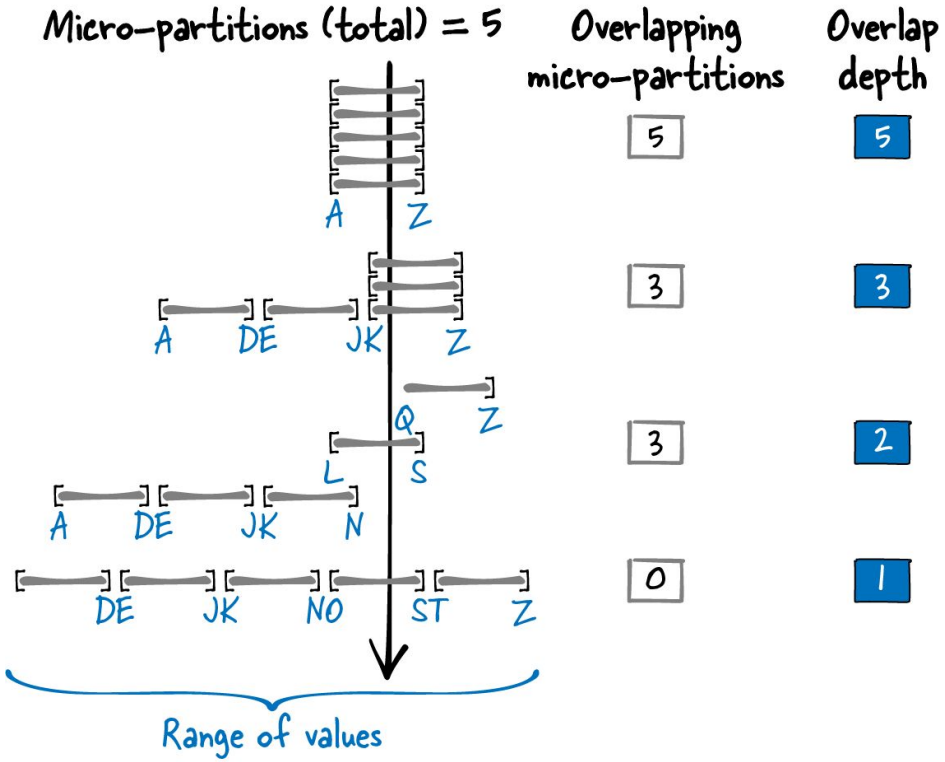
Defining the Cluster Key

- When to consider defining the cluster key
  - Query performance degraded over time and performance is not acceptable
  - Need to improve performance
  - Selective queries
  - Queries using sort operation
- To be cost wise = number of queries should be much higher than DML operations (INSERT, UPDATE,

Automatic Clustering

- Define a cluster key
- Snowflake does the maintenance automatically
- Runs in background on system defined warehouse
- Provide functions to monitor clustering health & cost

Clustering Depth and Overlapping



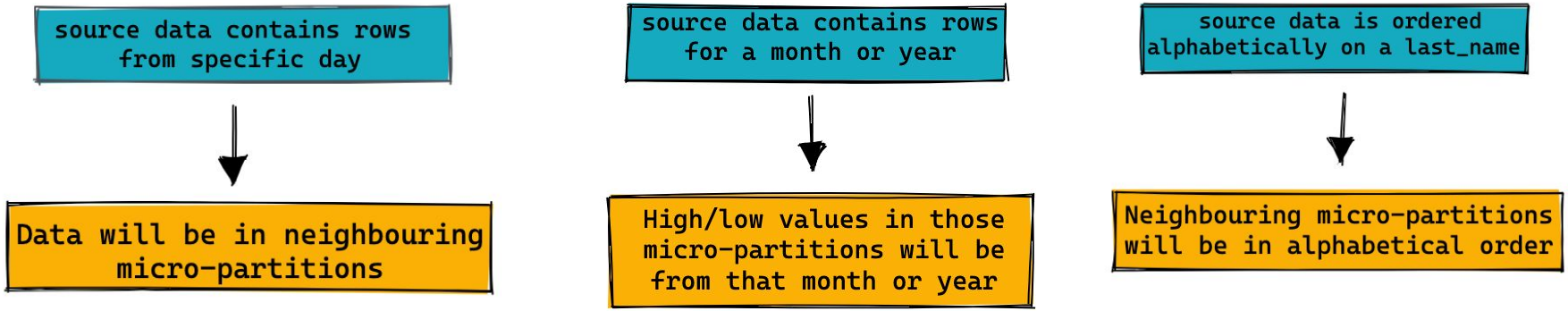
- **Goal:** smallest depth – better clustered table
- Can be used for monitoring of the clustering health
- Empty table – clustering depth – 0

System Functions for Clustering Monitoring

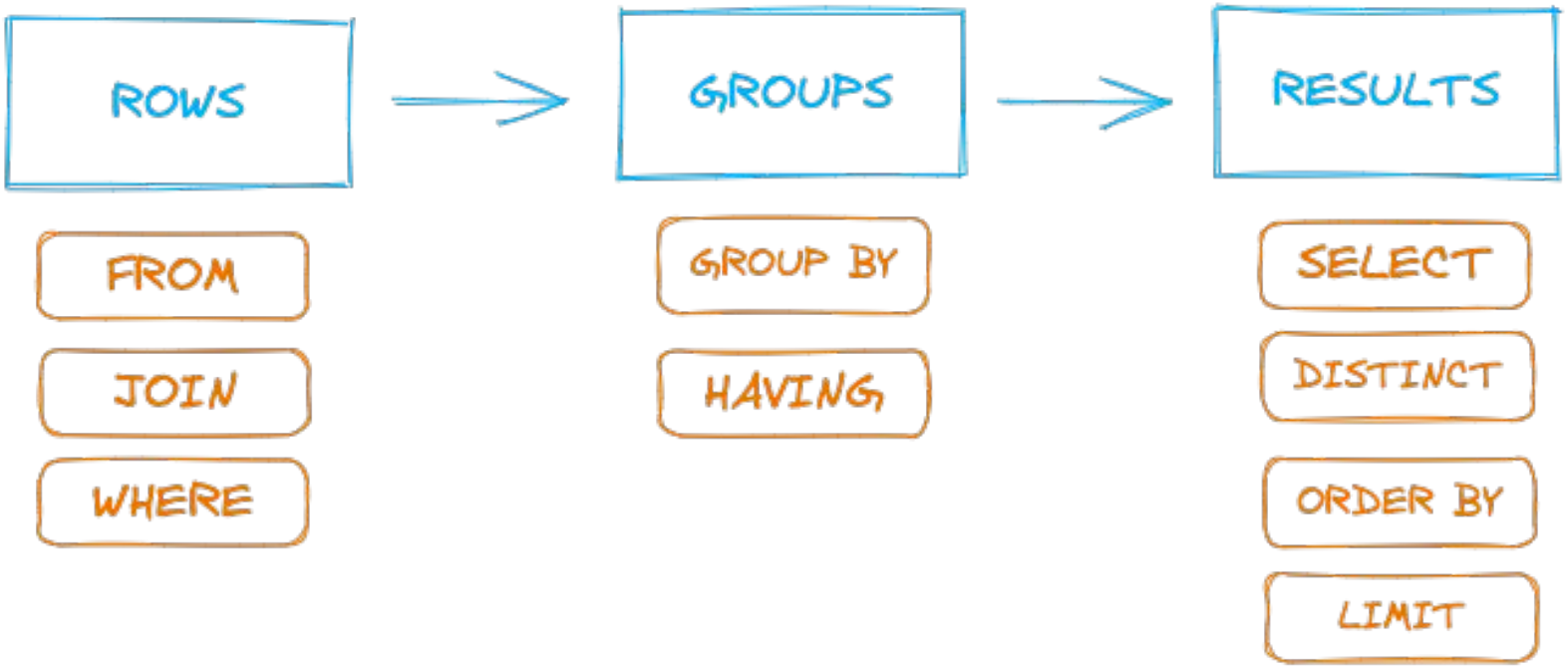
- `SYSTEM$CLUSTERING_DEPTH`
  - Computes average depth of the table based on input columns
  - Can be used to estimate how clustering could improve the pruning
- `SYSTEM$CLUSTERING_INFORMATION`
  - Returns clustering information including the depth, depth_histogram, number of micro-partitions, average overlap, total_constant_partition_count etc.

NATURAL DATA CLUSTERING

- Micro-partitions are created based on ingestion order
- ingestion order may correlate with one or more columns
  - sequential field (ids)
  - date fields
- single data load reads source data and writes it into some number of micro-partitions
- source data order determines what range of values are inserted in each micro-partition



Typical DB execution order

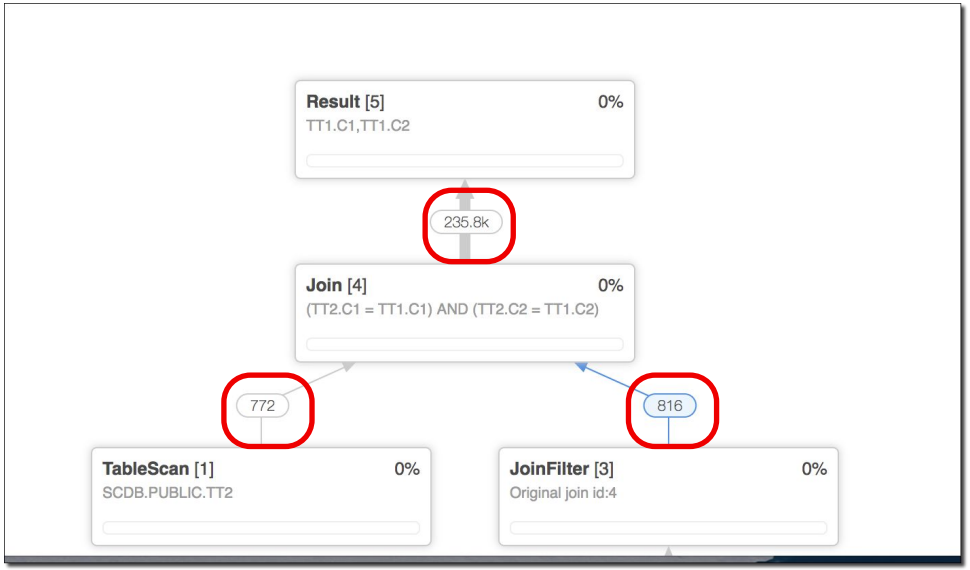


Top performance tips

- Use appropriate filters, as early as possible
- Check row operations first
  - First check FROM and WHERE clauses
  - Then check GROUP BY and HAVING clauses
- Row operations are performed before GROUP operations
- Use caches!
- JOINs
  - Ensure keys are distinct
  - Understand relationship between tables before joining
  - Avoid many to many join
  - Avoid unintentional cross join
    - Joining on non-unique keys can explode your data output - join explosion

Exploding joins

- what happen when you use not unique columns for joins (cartesian join, one record from table A match multiple records in table B)
- it takes a lot of time
- produce huge output
- hundreds records on input produce hundreds of thousands



Knowledge check

1. How is called feature which helps you with data organization in micropartitions?
a] SYSTEM CLUSTERING
b] DATA CLUSTERING
c] AUTO CLUSTERING
d] STORAGE CLUSTERING

2. How to check the clustering information
a] run function SYSTEM$GET_CLUSTERING()
b] run function SYSTEM$GET_DEPTH()
c] run function SYSTEM$CLUSTERING_OVERVIEW()
d] run function SYSTEM$CLUSTERING_INFORMATION()

3. There is no extra cost for Table Clustering
a] Yes
b] No

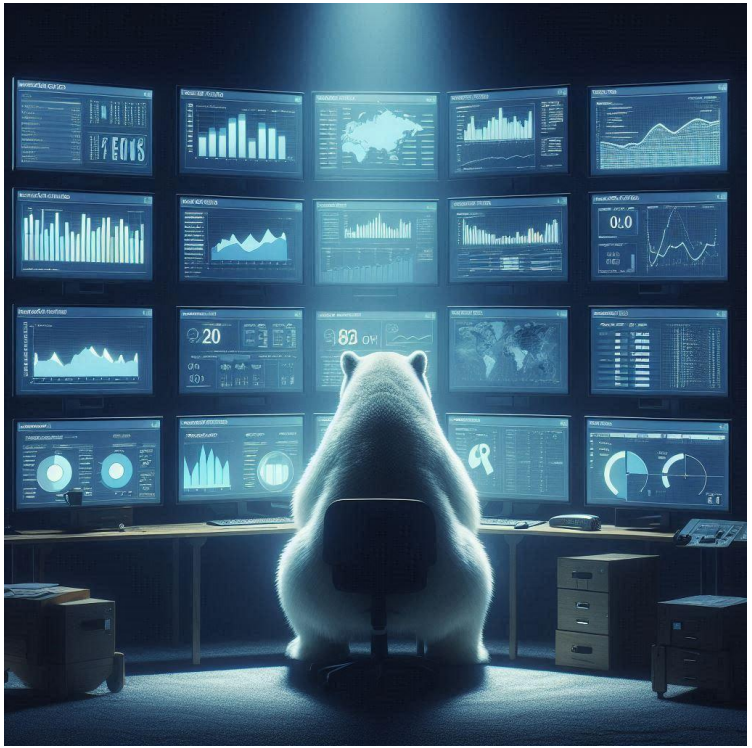
Resource Monitors and Budgets

Use cases

- Automation of monitoring, notification and suspension of warehouses
- Resource monitors helps to keep costs under control (compute costs)
- Prevent all unintentional cost spikes
  - Bad users queries
  - Long running ELT pipelines, ML models or data apps
  - Missing configuration (query time outs)
- Ability to track costs per domains, teams

Resource monitors intro

- Account level object
- Help to control costs
- Properties
  - Credit quota
  - Monitor type
    - Account monitor
    - Warehouse monitor
  - Schedule
    - Daily, weekly, monthly, etc.
    - Start
    - End
  - Actions



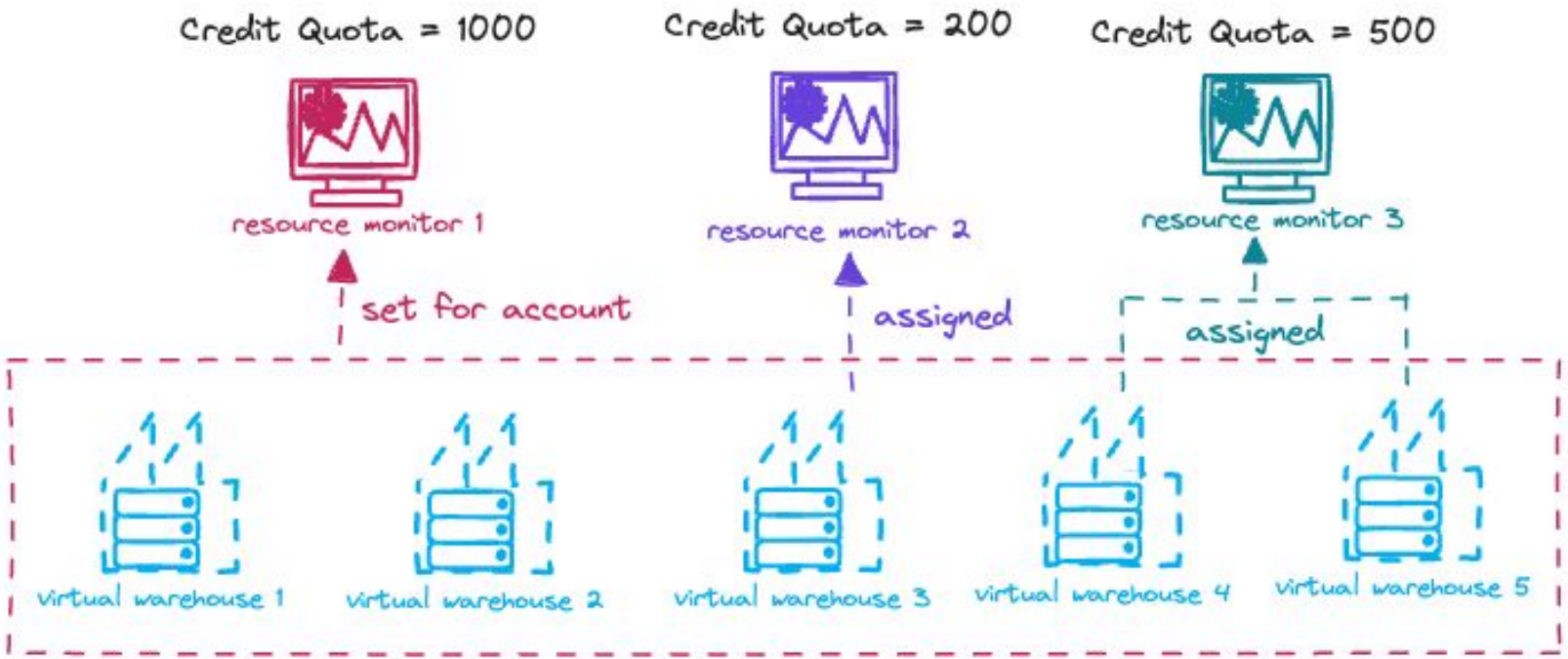
Resource Monitors assignment

- Account level
  - Control costs for whole account

- Warehouse level
  - Single or more warehouses could be assigned to resource monitor

- Warehouse could be assigned only to single monitor
- Account level monitor does not override monitor assigned to individual warehouse
- Account level monitor does not control credit usage of Snowflake provided warehouses

Resource Monitors assignment



Actions

- What happen when the threshold from credit quota is reached (within specified interval)
- Defined as percentage of the credit quota
- Thresholds greater than 100 are supported

Notify

Notify & Suspend

Notify & Suspend immediately

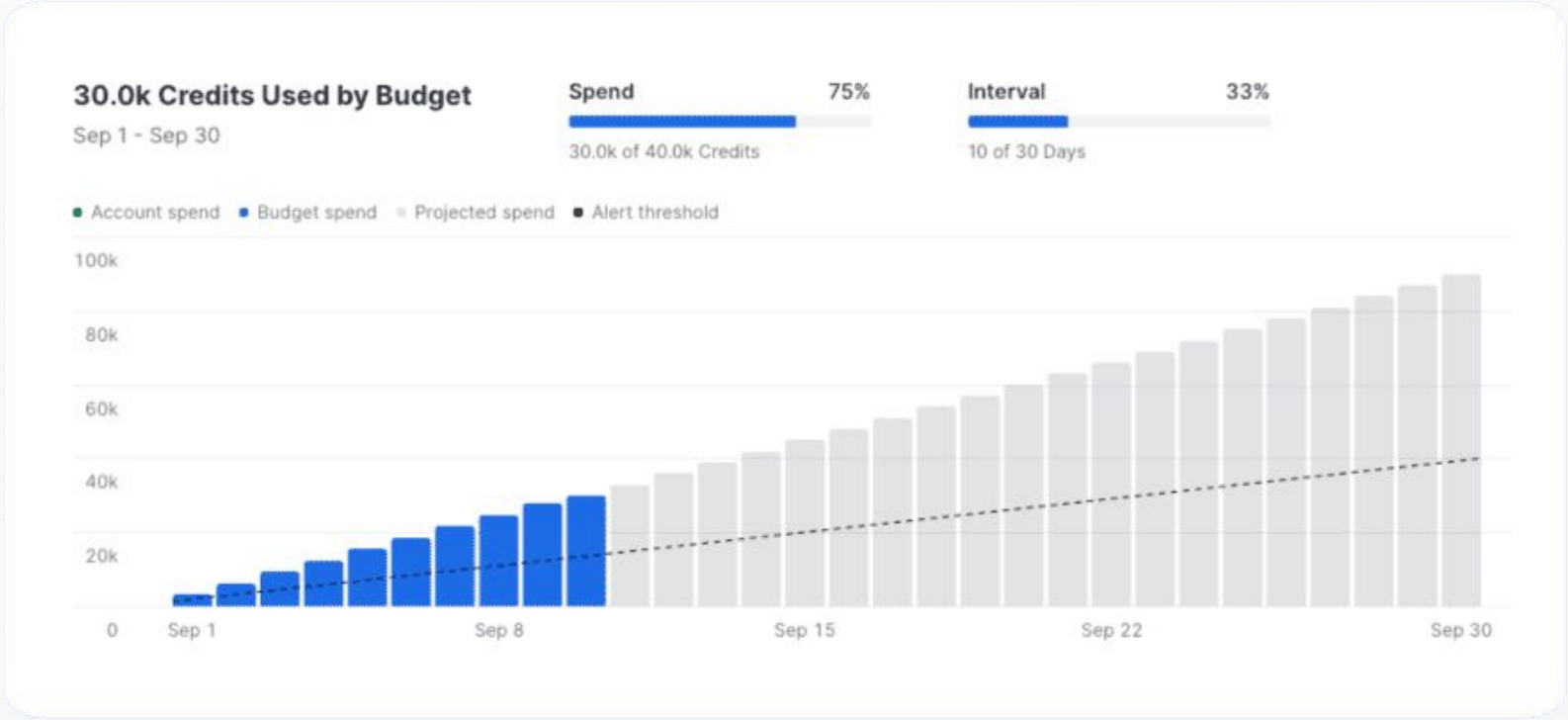
Scheduling options

- Default
  - Start monitoring credits immediately after creation and reset at the beginning of the month
- Custom
  - Interval is relative to specified start date
  - Frequency
    - Daily
    - Weekly
    - Monthly
    - Yearly
    - Never
  - Start
    - Immediately
    - At specified timestamp in the future

Budgets

- Define a monthly limit for account or custom group of objects (e. g. containers, pipes, MVs, tasks)
- Provide more granular overview than resource monitors
- When limit is reached notification is sent (email, SQS, webhook – slack, teams)
- Informative character – can’t shutdown any resources
- Provide an UI for controlling

Budgets UI


30.0k Credits Used by Budget
Sep 1 - Sep 30
Spend
75%
30.0k of 40.0k Credits
Interval
33%
10 of 30 Days
Account spend
Budget spend
Projected spend
Alert threshold
100k
80k
60k
40k
20k
0
Sep 1
Sep 8
Sep 15
Sep 22
Sep 30

Exercise

We are going to create a different resource monitors:

1. account level resource monitor
2. warehouse level resource monitor

Stored procedures and User defined functions

Stored procedures

- Extend the system with procedural code executing SQL
- Support for SQL, JavaScript, Python, Java
- Bundle multiple SQL commands into single callable script
- Variables, loops, conditions, cursors
- Transaction support
- Error handling
- Dynamic creation of SQL
- Called as independent statements
  - CALL myStoredProcedure(argument);
- One SP can call another one
- Can return a value
- Cannot return a set of rows

Use cases

- Task automation
  - Requires combination of multiple sql statements or additional logic
- DB clean up
  - Remove data from DB older than XY days
  - multiple DELETE statements bundled into single SP, pass the date as parameter
  - Could be scheduled as a task or run separately
- DB backup
- ML models calculations
- Data compliance verification
- ETL pipeline

Stored procedure example

```
Create or replace procedure myProcedure()
returns varchar
language sql
as $$
    --Snowflake scripting code
    declare
      r float;
      area_of_circle float;
    begin
      radius_of_circle := 3;
      area_of_circle := pi() * r
      return area_of_circle;
    end;
$$
;
```

How to choose the right language for SP and UDF

- Own preference
- You already have some other code in that language – consistency
- Language has capabilities which other does not have
- Language has libraries which can help you with data processing
- Do you want to keep your stored procedure code in-line or externally (file on stage)?

Language	Handler Location
Java	In-line or staged
JavaScript	In-line
Python	In-line or staged
Scala	In-line or staged
SQL	In-line

In-line handler example

```
create or replace procedure my_proc(from_table string, to_table string, count int)
returns string
language python
runtime_version = '3.8'
packages = ('snowflake-snowpark-python')
handler = 'run'
as
$$
def run(session, from_table, to_table, count):
session.table(from_table).limit(count).write.save_as_table(to_table)
return "SUCCESS"
$$;
```

Way of working:

1. Develop & test handler code locally
2. Compile it if necessary (Java, Scala)
3. Copy to Snowsight
4. Create a Stored Procedure

Stage handler example

```
CREATE OR REPLACE PROCEDURE MYPROC(value INT, fromTable STRING, toTable STRING)
RETURNS INT
LANGUAGE JAVA
RUNTIME_VERSION = '11'
PACKAGES = ('com.snowflake:snowpark:latest')
IMPORTS = ('@mystage/MyCompiledJavaCode.jar')
HANDLER = 'MyJavaClass.run';
```

Way of working:

1. Develop & test handler code locally
2. Compile it if necessary (Java, Scala)
3. Upload to internal stage
4. Create a Stored Procedure

Keeping Handler code In-line or on a Stage?

In-line Handler Advantages

- Easier to implement
- Update the code with ALTER command
- Maintain the code directly in Snowsight
- If code needs to be compiled (Java, Scala) it is possible to define a location for output path with TARGET_PATH
- Then code is not compiled with each SP call – faster execution of repeated calls

Stage Handler Advantages

- Can use code which might be too large for in-line handler
- Handler can be reused by multiple stored procedures
- Easier to debug / test in existing external tools – especially for complex and large code

Caller's vs Owner's Rights

Caller's rights

- Runs with privileges of the caller
- Knows caller's current session
- Nothing what caller can't do outside SP can't be even done in SP
- Changes in session persist after end of SP call
- Can view, set, unset caller's session variables and parameters
- Use it when
  - SP operates only on objects owns by caller
  - You need to use caller's environment (session vars)

Owner's rights (default option)

- Runs with privileges of the SP owner
- If SP owner have some privilege (delete data), SP can delete them even if the caller can't
- Can't change session state
- Can't view, set, unset caller's session variables and parameters
- SP does not have access to variables created outside the stored procedure
- Use it when
  - You want to delegate some task to another role without granting such privilege to that role (delete data)
  - You want to prevent callers from viewing the source code of SP

User Defined functions

- Support for SQL, JavaScript, Python, Java
- Develop a reusable logic which is not part of Snowflake
  - Area of circle
  - Profit per department
  - Concatenate names
  - Get bigger number
- Only SQL and JavaScript UDFs can be shared
- Secure version (data sharing)
- Called from SQL query
- DML and DDL is not permitted
- MUST return value

User defined function example

```
Create or replace function addone(i int)
returns int
language python
runtime_version = '3,8'
handler = '3,8'
as $$
  def addone_py(i):
     return i+1
$$;
```

```
SELECT addone(3)
```

* in-line handler

Tabular UDFs

- Scalar = return single value for each input row
- Tabular = return tabular value for each input row
  - RETURN value specify schema of returned table, including data type
  - BODY of UDTF is SQL expression – it must be SELECT statement
  - Max 500 input parameters
  - Max 500 output columns
- INFORMATION_SCHEMA – full of tabular functions

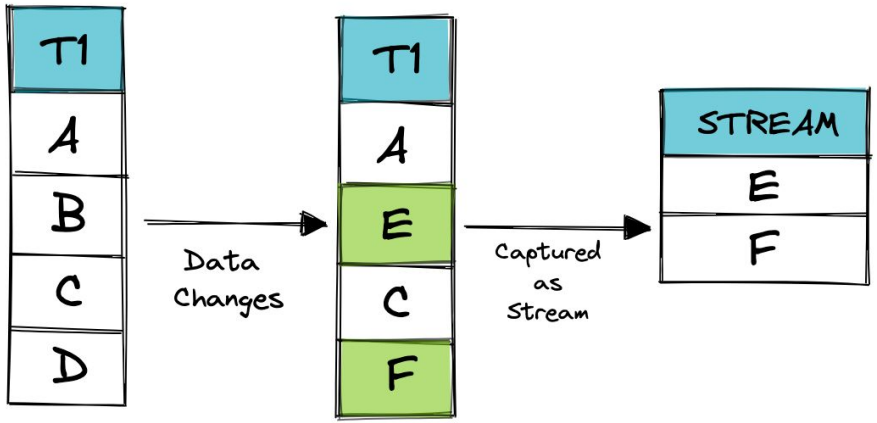
```
SELECT ……
  FROM TABLE ( udtf_name (udtf_arguments) )
```

Exercise

Let’s create a SQL user define function. We will try to extract domain info from emails stored in snowpipe_landing table.

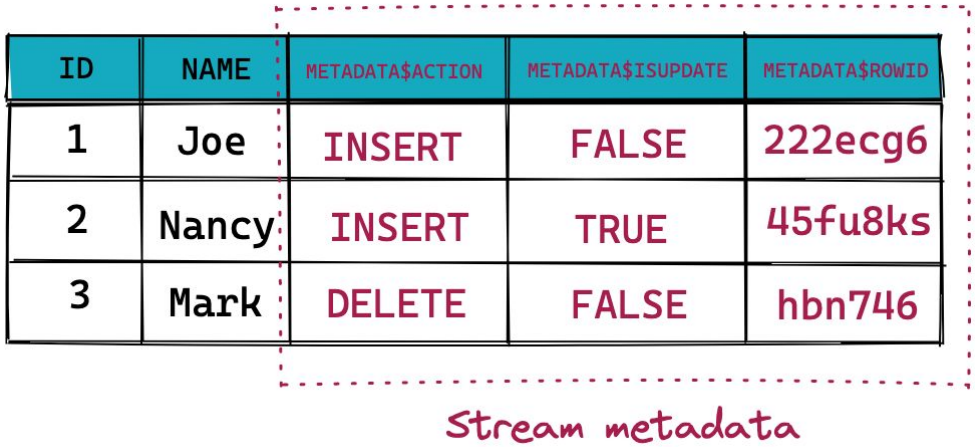
Streams and tasks

Streams



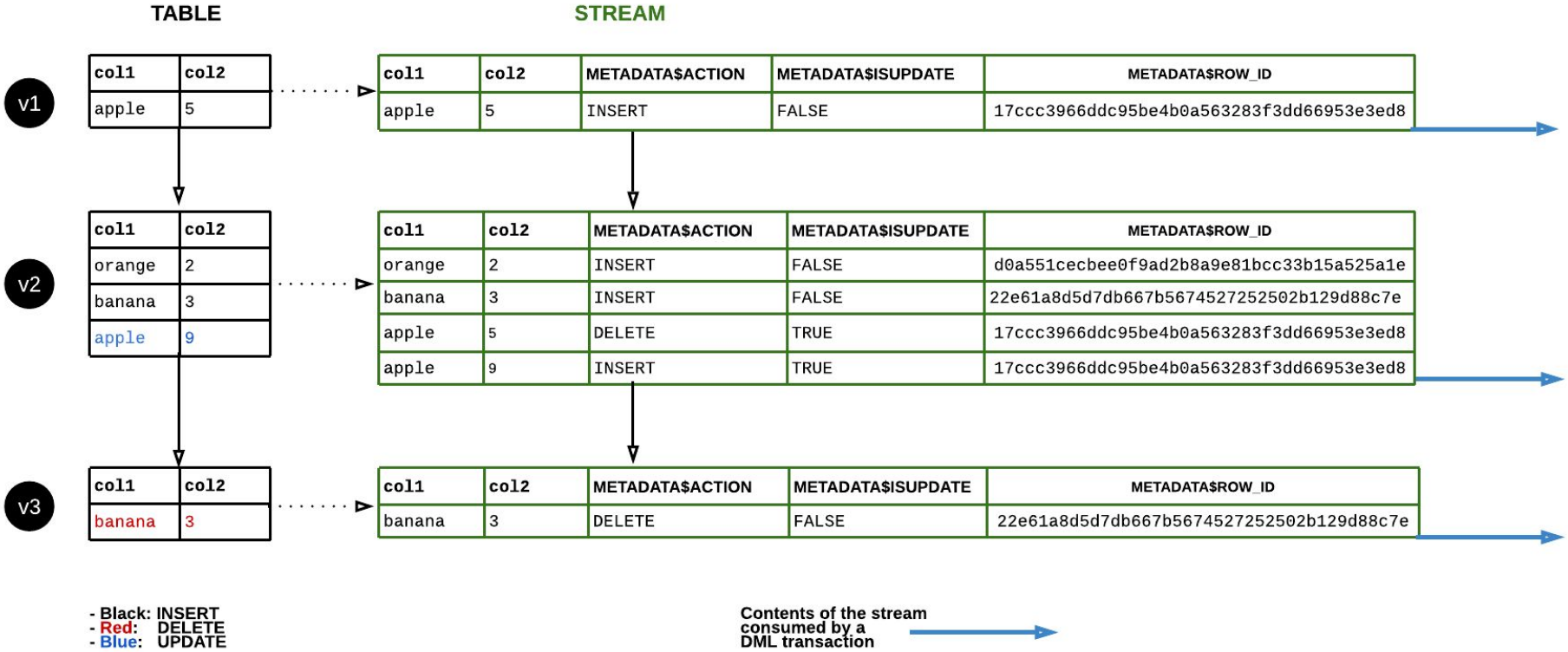
- Changed data capture (CDC)
- Identify and act on changed records
- Stream append 3 metadata columns to the table
  - Tracks the changed data
  - Little storage required
- Can be queried (consumed) as standard table/view
- When consumed as part of DML operation, the stream is cleared
- Stream itself does not contain any data
- Offset = point in time

Streams



- Stream metadata columns
  - METADATA$ACTION
  - METADATA$ISUPDATE
  - METADATA$ROWID
- Type of streams
  - Standard (delta)
  - Append-only
  - Insert-only (external tables)
- Supported objects to track changes
  - tables
  - directory tables
  - external tables
  - views (in preview)

Stream data flow



Data retention and staleness

- If stream is not consumed within defined period of time it becomes stale and data inaccessible
- Default value: 14 days
- Controlled by two parameters
  - DATA_RETENTION_TIME_IN_DAYS
  - MAX_DATA_EXTENSION_TIME_IN_DAYS

- How to check?
  - DESCRIBE STREAM or SHOW STREAMS
    - STALE and STALE_AFTER columns

- When stale => recreate the stream

Streams wrap up

- Multiple consumers needed?
  - Create individual streams per consumer
- Main costs associated with processing time used by virtual warehouse to query the stream
- Think about streams as a “bookmarks”

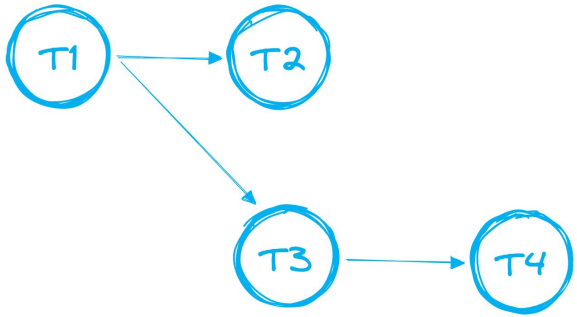
```
create stream mystream on table mytable;
```

Exercise 1 – Practise the streams

We are going to practice working with streams and tasks. For that we need to prepare some data. Let's simulate we are receiving the data with trips details on monthly basis. We will create a new table called `TRIPS_MONTHLY`, load there some data and build some aggregation logic on top of the table. It will be using streams to track the changes. Later we will combine it with task to automatically process the records when new data arrive into the table.

Tasks

- Scheduled execution of SQL operation
  - Single SQL statement
  - Call to a stored procedure
- Often combine with streams for ELT pipelines to process changed table rows
- Trigger
  - Schedule (CRON)
  - Interval (minutes, seconds ..)
  - Predecessor - child task starts when parent task completes
  - Condition = stream contains a new data
  - Manually
- Tasks can be chained together to create complex data pipelines (DAGs) - tree of tasks



Tasks – use cases

Copy data in/out of Snowflake

Complex ELT pipelines

Generate periodic reports

internal cleaning

Keeping aggregations up to date

???

Task creation

- Key parameters
  - WAREHOUSE
  - SCHEDULE
  - WHEN
  - AFTER
  - SUSPEND_TASK_AFTER_NUM_FAILURES
- Newly created task is suspended
  - Nneeds to be resumed by ALTER command

```
create or replace task my_task
   warehouse = xsmall_vwh
   schedule = '1 minute'
   as
    insert into dim_customers (
        select customer_id, customer_name
        from s_src_customer_stream
    where metadata$action = 'INSERT');
```

Task triggered by new data in stream

```
create or replace task my_task
   warehouse = xsmall_vwh
   schedule = '5 minute'
   when SYSTEM$STREAM_HAS_DATA('s_src_customer_stream')
   as
    insert into dim_customers (
        select customer_id, customer_name
        from s_src_customer_stream
    where metadata$action = 'INSERT');
```

Task history retrieval

- TASK_HISTORY view or table function
- Table function = last 7 days history
- View = last year history

- Key columns
  - NAME
  - STATE
  - QUERY_TEXT
  - ERROR_MESSAGE
  - ROOT_TASK_ID

```
select *
  from table(information_schema.task_history(
    scheduled_time_range_start=>dateadd('hour',-1,current_timestamp()),
    result_limit => 10,
    task_name=>'MYTASK'));

 select *
 from snowflake.account_usage.task_history
 limit 10;
```

Exercise 2 – task creation

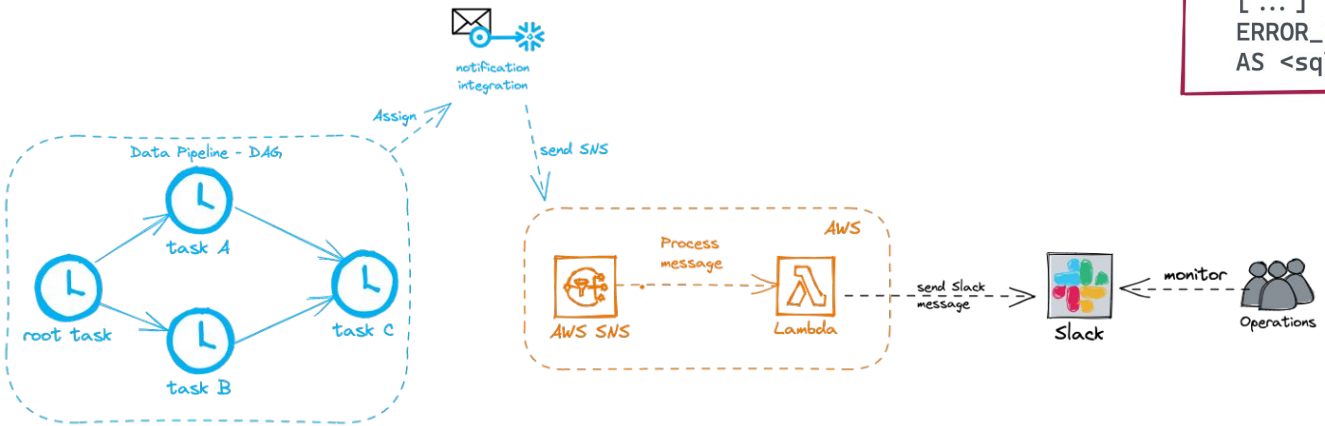
Now we are going to turn the logic into the task. We are going the create a task which will take records from the `STR_TRIPS_MONTHLY`, do the aggregation and store the results into `FACT_RIDES` table. Task should be scheduled to run every minute but the code will be triggered only when new data will be placed into the stream.

1. Write a task `T_RIDES_AGG` with logic and trigger described above.
2. Check the task definition (`SHOW TASKS`)
3. Resume the task. Newly created tasks are suspended and they have to be resumed in order to start operate.
4. Load May 2018 trips data into `TRIPS_MONTHLY` table.
5. Check the `FACT_RIDES`. New row should be inserted there by our task/

Error notifications for tasks

- Receive a notification when task fails
- Snowflake and Cloud Provider integration (AWS, Azure, GCP)
- new Snowflake object **NOTIFICATION INTEGRATION**
  - in preview
  - currently supports only AWS SNS

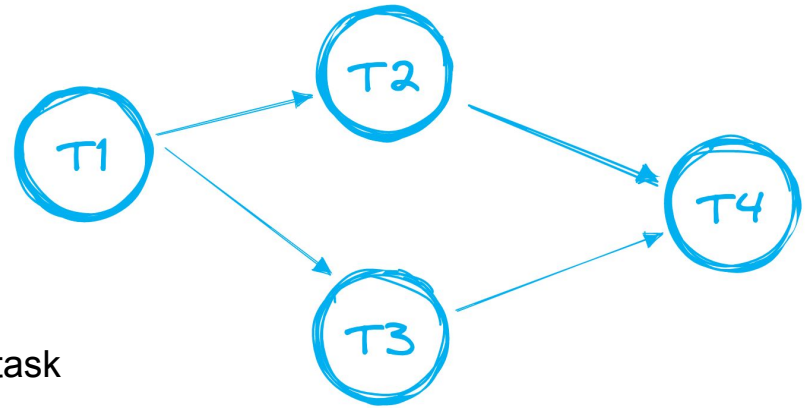
```
CREATE TASK <name>
  [ ... ]
  ERROR_INTEGRATION = <integration_name>
  AS <sql>
```



Complete step by step guide & in detail overview:
https://medium.com/snowflake/error-notifications-for-snowflake-tasks-ca5798884e67

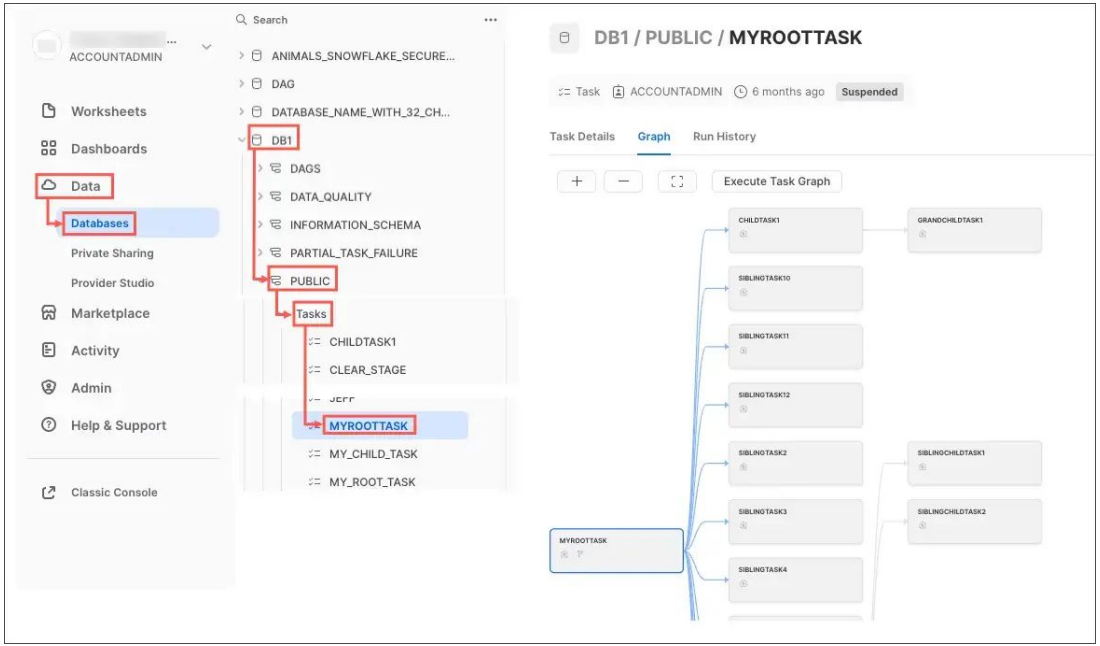
Tree of tasks details

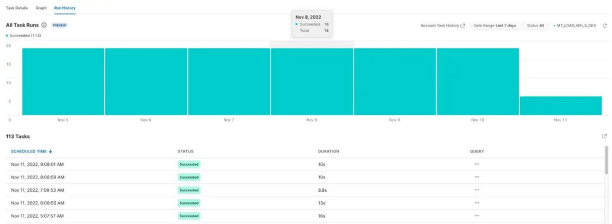
- One root task
- Only single direction - no loop support
- Total numbers
  - 1000 tasks in total for single DAG
  - 100 predecessors tasks
  - 100 child tasks
- If any task in the tree needs to be updated, the root task needs to be always suspended



- How to check the dependencies between tasks
  - TASK_DEPENDENTS table function
  - UI in Snowsight – on account level or in individual DBs

UI for viewing DAG and task history





Streams and tasks best practises

- More stream consumers = multiple streams
- Create a separate custom role for managing the tasks (TASK_ADMIN) with EXECUTE_TASK privilege
- Have a robust operation system around the pipelines
  - Stream stale prevention
  - Task failure notifications
  - Task suspension notifications
  - Pipeline overview
- Version control (GIT integration)
- Do not forget to resume the task

Exercise 3.

Let’s create another task which will be running after T_RIDES_AGG. We would have a data pipeline consisting of two steps. Suppose we need some custom logging of the latest loaded month into fact table. For the sake of simplicity we will be just taking the latest loaded month from FACT_RIDES and load it into our custom logging table called LOG_FACT_RIDES. New task should be linked to the T_RIDES_AGG and run after it. T_RIDES_AGG will become a root task of our pipeline.

Flattening Semi-structured data

Semi structured data types

can hold standard SQL type, arrays and objects

non-native JSON types stored as strings

max length 16 MB

Usage:
Hierarchical data, direct storage of JSON, Avro, ORC, Parquet data

key-value pairs collection

Value = VARIANT

max length 16 MB

no NULL support

Arrays of varying sizes

Value = VARIANT

items accessed by position in the array

dynamic size

nested arrays are supported

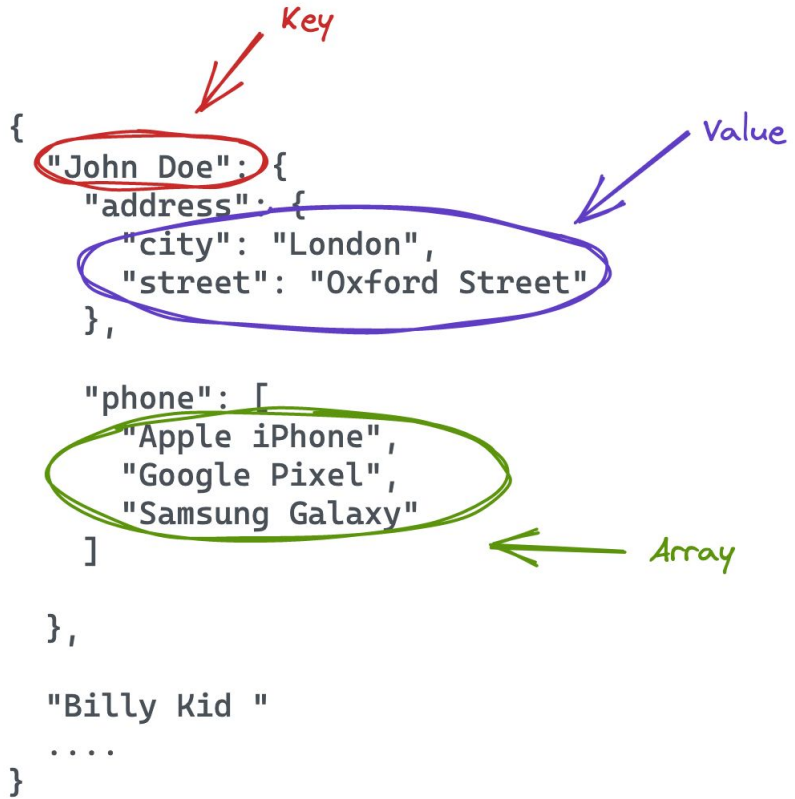
JSON example

```
{
  "John Doe": {
    "address": {
      "city": "London",
      "street": "Oxford Street"
    },

    "phone": [
      "Apple iPhone",
      "Google Pixel",
      "Samsung Galaxy"
    ]

  },

  "Billy Kid "
  ....
}
```




```
[ {"John Doe": {"address": { "city": "London", "street": "Oxford Street" }, "phone": [ "Apple iPhone", "Google Pixel", Samsung Galaxy"] }, {"Billy Kid": ...} ]
```

Load options for JSON data

Store semi structured data in a VARIANT Column

You are not sure about data structure, type of performed operations or how often the structure will be changed

File has many nested levels including arrays and objects

Flatten the nested structures

Flatten if, data contains:

- dates and timestamps
- Numbers within strings
- Arrays

Depends on the use case

My recommendation: Load data as is, flatten them if needed

FLATTENING DATA

- VARIANTs usually contains nested elements (arrays, objects) that contains the data
- FLATTEN function extract elements from nested elements
- It is used together with a LATERAL JOIN
    - Refer to columns from other tables, views, or table functions

```
LATERAL FLATTEN
(input => expression [options] )
```

    - Input => The expression or column that will be extracted into rows
        - It must be VARIANT, OBJECT, or ARRAY

```
SELECT
  value:size::number(10,2) as size
  value:product_name::varchar as product
FROM my_json_data j
  LATERAL FLATTEN(input => j.v:data);
```

FLATTENING DATA

- VARIANTs usually contains nested elements (arrays, objects) that contains the data
- FLATTEN function extract elements from nested elements
- It is used together with a LATERAL JOIN
  - Refer to columns from other tables, views, or table functions

SRC_VAL

```
{
  "data":[
    {
      "size":"10.50",
      "product_name":"A"
    },
    {
      "size":"11.70",
      "product_name":"B"
    },
    {
      "size":"12.30",
      "product_name":"C"
    }
  ]
}
```

LATERAL FLATTEN

ROW	SIZE	PRODUCT
1	10.50	A
2	11.70	B
3	12.30	C

- Input => The expression or column that will be extracted into rows
  - It must be VARIANT, OBJECT, or ARRAY

```
LATERAL FLATTEN
(input => expr [, outer ] [, path ] [, recursive ] [, mode ])

SELECT
    value:size::number(10,2) as size
    value:product_name::varchar as product
FROM my_json_data j
    LATERAL FLATTEN(input => j.v:data);
```

Casting VARIANT data

- Variant data are just strings, arrays, and numbers
- Need to be casted to proper data type otherwise they remain VARIANT object types
- There is special operator for casting **::**

```
SELECT
  value:size::number(10,2) as size
  value:product_name::varchar as product
FROM my_json_data j
  LATERAL FLATTEN(input => j.src_val:data);
```

Row	Size	Product
1	10.50	A
2	11.70	B
3	12.30	C

Semi structured data functions

Array handling	`ARRAY_AGG, TO_ARRAY, ARRAY_SIZE …`
Object handling	`OBJECT_CONSTRUCT, OBJECT_KEYS, OBJECT_DELETE, OBJECT_INSERT`
JSON Parsing	`PARSE_JSON`
Extraction	`FLATTEN`
Casting	`AS_<object_type>`
Type Checking	`IS__<object_type>`

FLATTEN OUTPUT

SEQ	A unique sequence number associated with the input record; the sequence is not guaranteed to be gap-free or ordered in any particular way
KEY	For maps or objects, this column contains the key to the exploded value
PATH	The path to the element within a data structure which needs to be flattened.
INDEX	The index of the element, if it is an array, otherwise NULL
VALUE	The value of the element of the flattened array/object
THIS	The element being flattened (useful in recursive flattening).

FLATTENING DATA

```
SELECT * FROM
   TABLE (FLATTEN(input => PARSE_JSON('["A","B","C"]')));
```

SEQ	KEY	PATH	INDEX	VALUE	THIS
1	NULL	[0]	0	“A”	[“A”,”B”,”C”]
1	NULL	[1]	1	“B"	[“A”,”B”,”C”]
1	NULL	[2]	2	“C”	[“A”,”B”,”C”]

Exercise 3.

Let’s practice the JSON flattening. This will be combined exercise because first we will be creating the JSON by using functions like ARRAY_AGG and OBJECT_CONSTRUCT. Then we can try to flatten such JSON.

Data Protection features

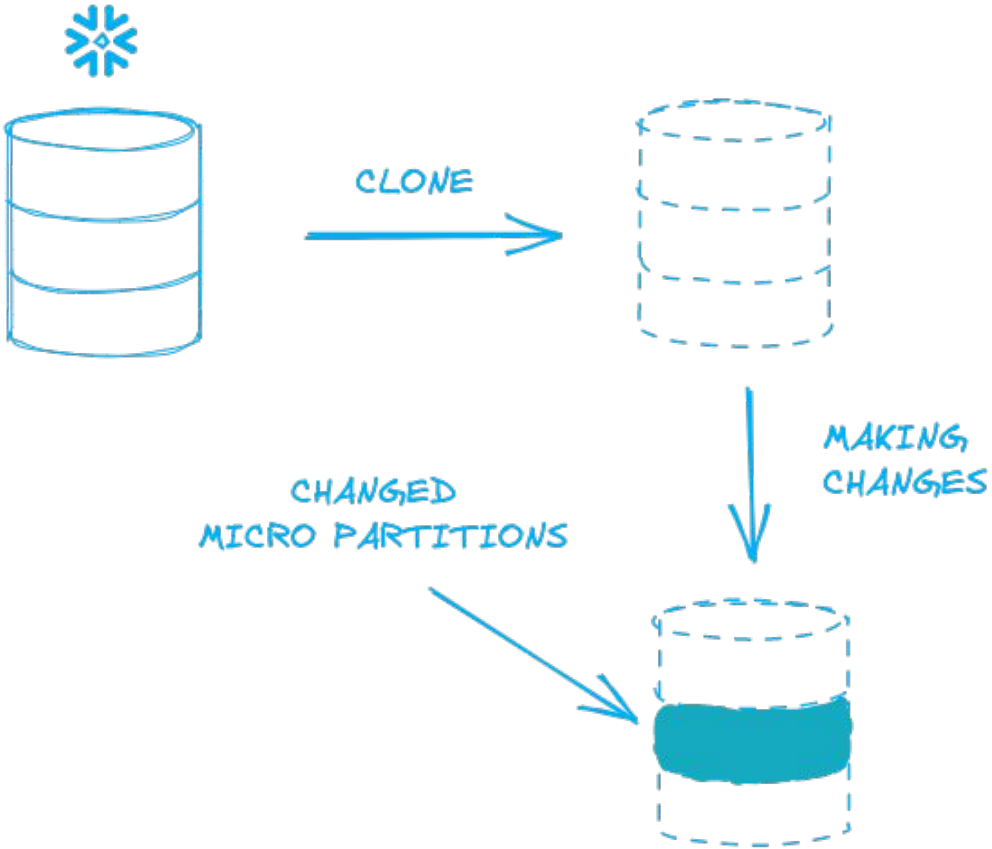
Zero-Copy Cloning

- Quickly take snapshot of any table, schema, or database
- No additional storage cost until changes are made
- Great for spinning up DEV and TEST environments
- Effective and easy “backup” solution
- Cloned objects have their own life-cycle

How Zero-Copy Cloning Works

- When clone is created:
  - Both objects share all micro-partitions
  - Oldest object owns all micro-partitions
  - Clone references them
- Access control

Zero-Copy Cloning Schema



Time Travel

- Access to historical data at any point within defined retention period
- Retention period 1 - 90 days
- Great for investigations when data issues arise
- Easy way to fix unwanted deletes or edits - UNDROP command
- Backup or restore data based on time or ID
- Special SQL commands to get the data in given point of time

Time Travel Retention Control

- Controlled by two parameters
- `DATA_RETENTION_TIME_IN_DAYS`
  - Can be set on different levels – account >> DB >> schema >> table
- `MIN_DATA_RETENTION_TIME_IN_DAYS`
  - Account level parameter

Time Travel SQL Syntax

- AT | BEFORE has the following parameters
  - TIMESTAMP
  - OFFSET (time difference in seconds)
  - STATEMENT (query ID)

Query historical data as of 10 minutes ago

```
SELECT * FROM my_table AT(OFFSET => -60*10);
```

- UNDROP for DBs, schemas and tables

Examples

- Query historical data as of 10 minutes ago

```
SELECT * FROM my_table AT(OFFSET => -60*10);
```

- Query data at given timestamp

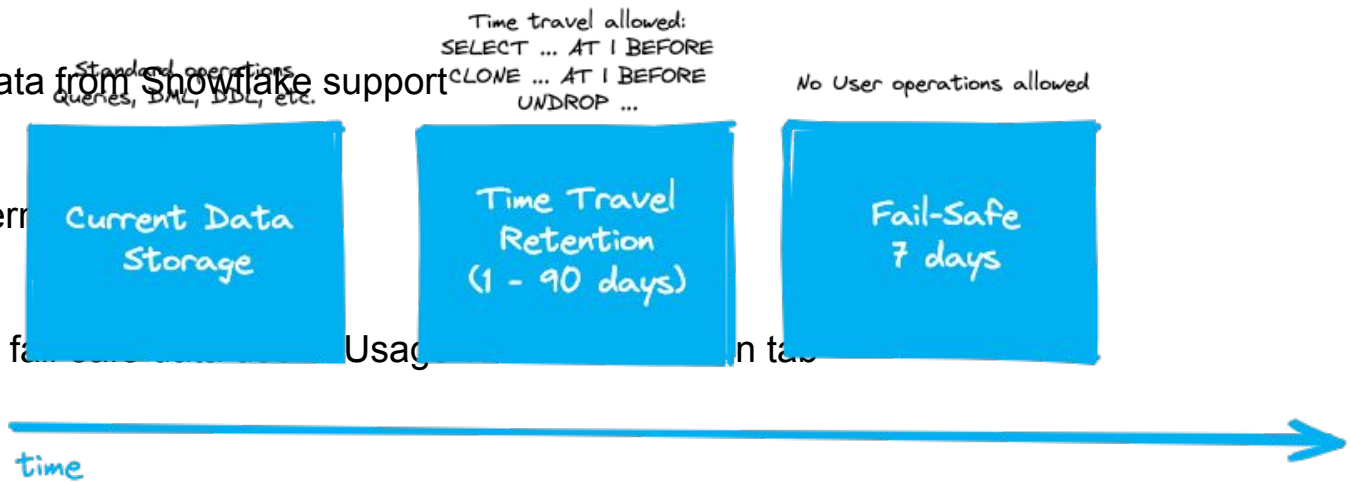
```
SELECT * FROM my_table AT(TIMESTAMP =>
      'Thu, 7 Nov 2024 10:15:00'::timestamp_ntz);
```

- Query data up to changes made by given query

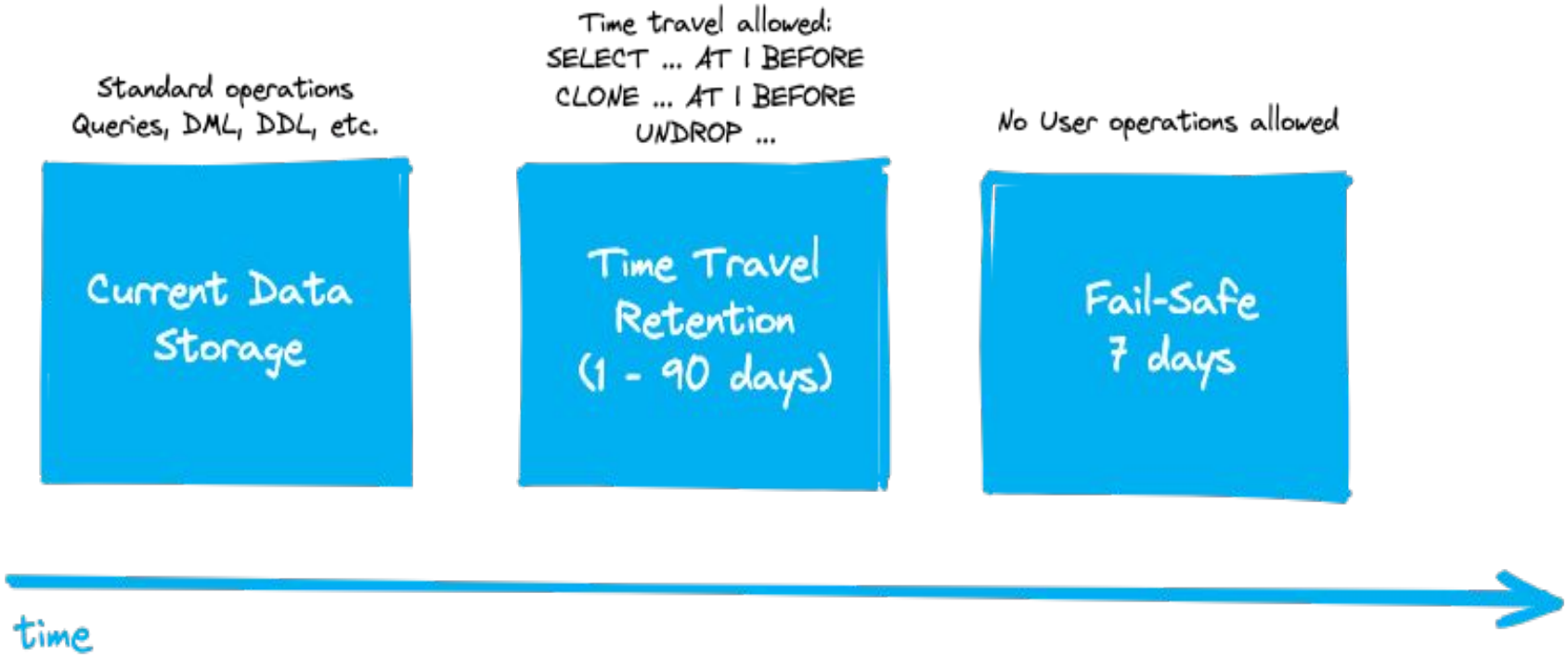
```
SELECT * FROM my_table BEFORE(
     STATEMENT => '8e5d0ca9-005e-44e6-b858');
```

Fail-Safe Storage

- Last instance to get your data in case of disturbance in given cloud region
- Not configurable
  - 7-day retention after time travel expiration
- Request data from Snowflake support
- Only for perr
- Admin can fa… Usag… n ta…



Continuous Data Protection – Overall Lifecycle



Fail–Safe and Cost Optimizations

- Use transient tables for lower environments & intermediate tables
  - Limited time travel (up to 1 day)
  - No fail-safe
- Configure
- Monitor your fail-safe spendings in cost management

Knowledge check

1. What is maximal retention period for Time Travel?

a] 1 day

b] 7 days

c] 30 days

d] 90 days

2. How can be configured retention period for Fail-Safe storage?

a] 1 day

b] 7 days

c] 30 days

d] not configurable [x]

3. How is called parameter controlling Time Travel retention period

a] RETENTION_TIME

b] STORAGE_EXTENSION_TIME

c] DATA_RENTENTION_TIME_IN_DAYS [x]

d] USER_RETENTION_TIME

Replication & Failover

Replication and Failover

- Two types → Database & account replication
- Database replication – standard edition
- Account replication – business critical edition
- Accounts must be part of same organization
- All regions are supported
- Replication group – collection of objects in source account

Primary Database

Read & Write DB

“Main account” with all operations

Secondary Database

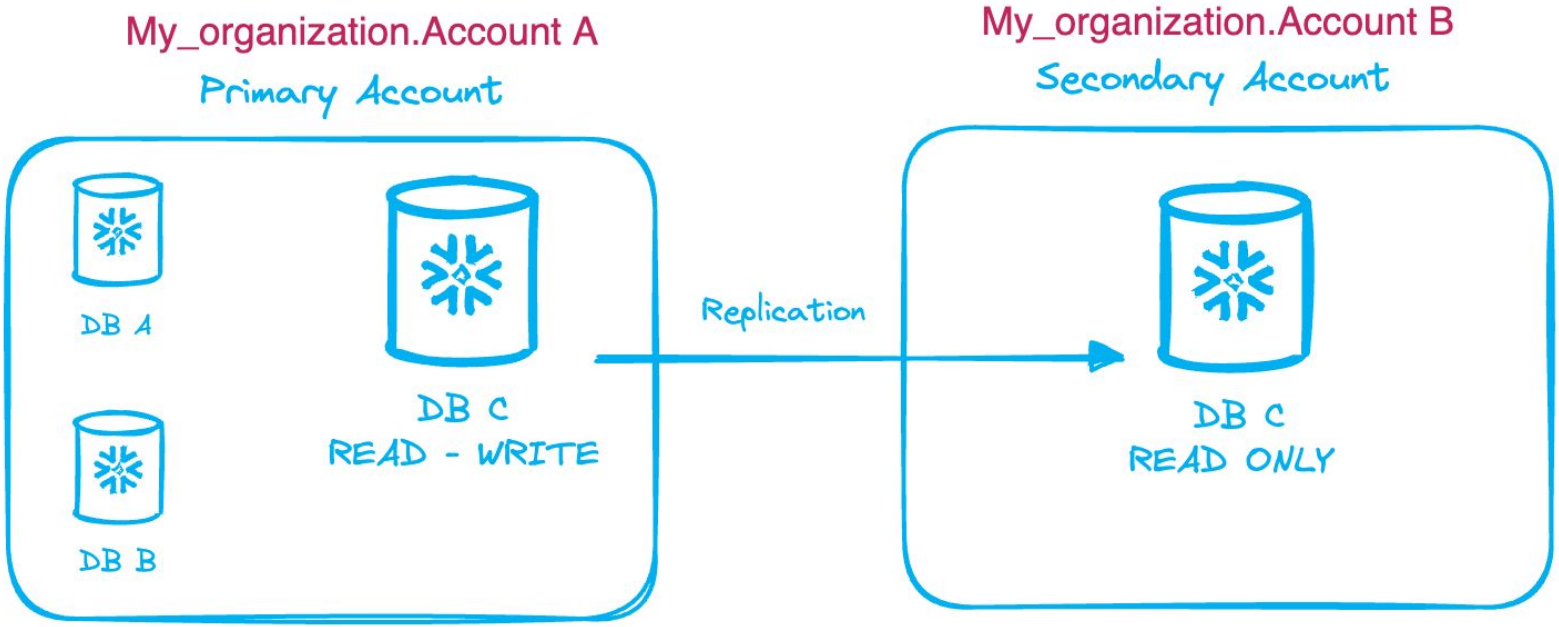
Read DB

- Replication of primary DB
- Same or different region
- Same or different cloud

Replication Feature Matrix

Feature	Standard	Enterprise	Business Critical	VPS
Database replication	X	X	X	X
Share replication	X	X	X	X
Replication group	X	X	X	X
Account object			X	X
Failover group			X	X
Tri Secret secure data protection			X	X

DB Replication



Database Replication

- Not all DB objects are replicated
  - e. g. external tables, hybrid tables, iceberg tables
  - It’s changing over time
- Replication to lower editions cause an error
  - E. g. business critical -> enterprise or standard
- Replication can have different rules and consequences for different

Replication and Failover Group Constraints

DB & shares

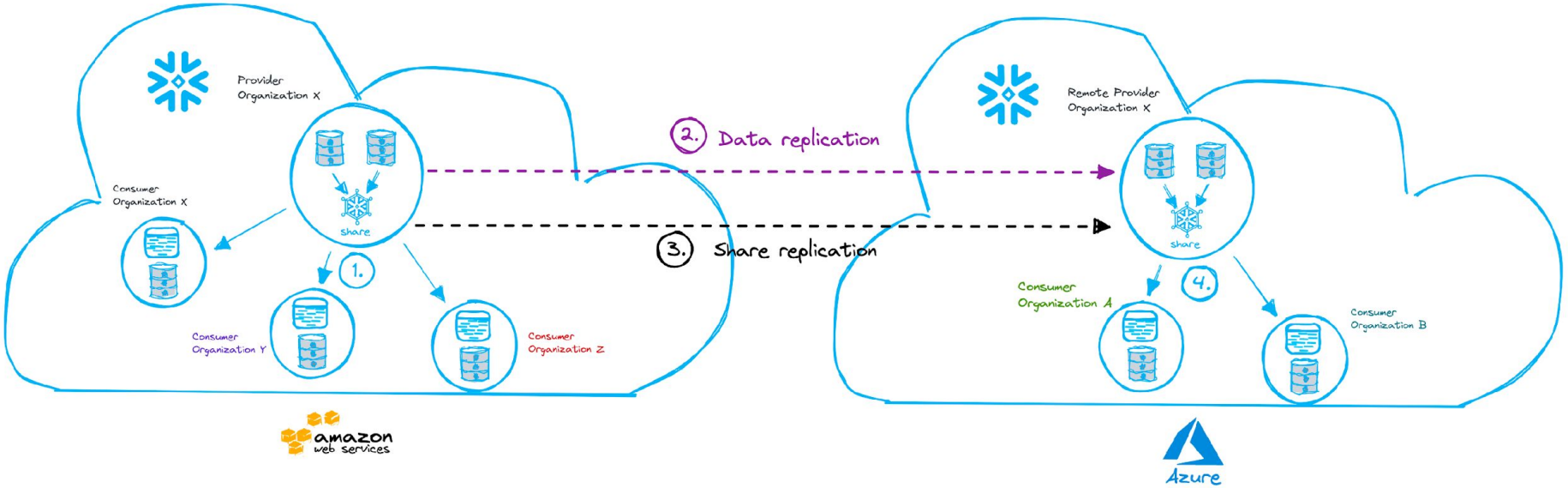
- An object could be part of single group
- An object could be part of multiple groups if they are replicated to different accounts
- Object cannot be in both failover and replication group

Account objects

- Only one replication group containing other objects than databases and shares

Replication and Data Sharing

Cross region



Failover

- Business critical edition feature
- One of the target accounts could be promoted to serve as source account
- In case the replication is currently running the secondary replica can’t be promoted as primary one
  - Wait
  - Suspend future refresh operations
  - Cancel currently running refresh (data inconsistency!)

Knowledge check

1. What Snowflake edition do you need for database replication?

a] Standard

b] Enterprise

c] Business Critical

d] Virtual Private Snowflake

2. Data replication can be done only to another Snowflake account provisioned in same or different region of the same cloud provider?

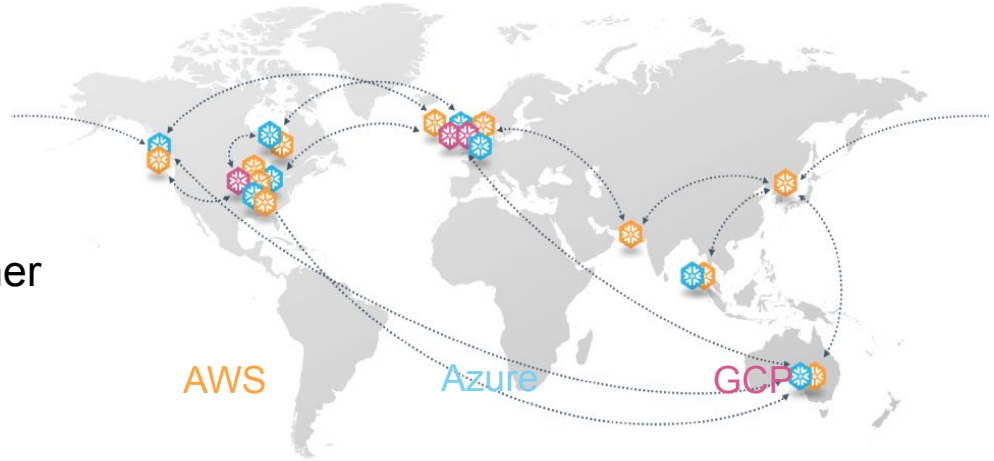
a] True

b] False

Data Sharing in Snowflake

Snowgrid

- Unique, Snowflake technology
- Connecting regions and clouds together
- It allows:
  - Maintain global business continuity
  - Share data with no ETL, silos
  - Share logic and services



Secure Data Sharing

Traditional Methods

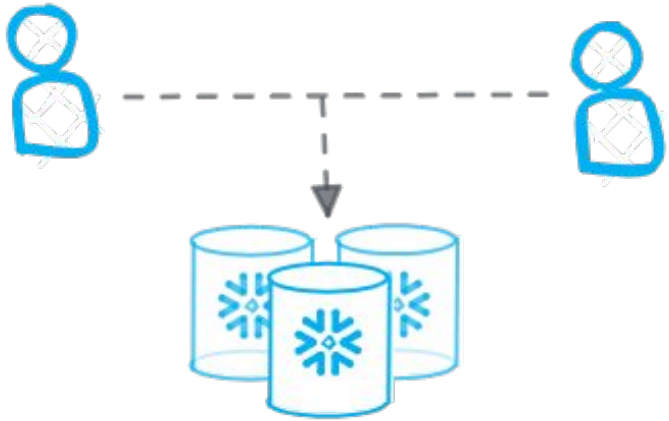
APIs | FTP | email | ETL | Cloud storage

- Copy and move data
- Data is delayed
- Costly to manage and maintain
- Unsecure, once data is

Secure Data Sharing

Snowflake

Secure Data Sharing



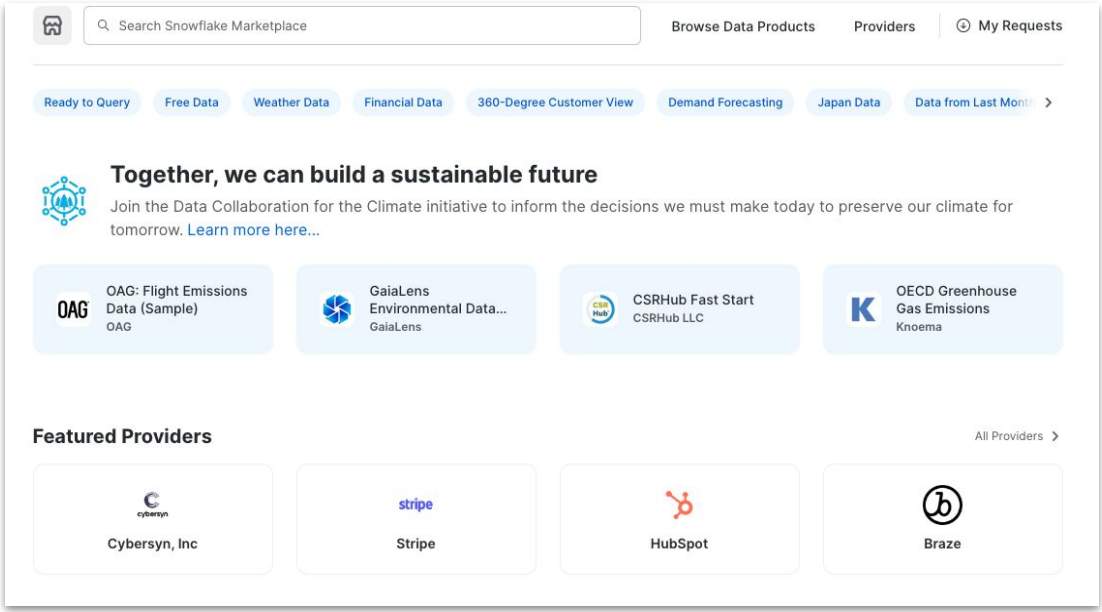
- Single copy of live data, no delays
- No costs of moving, copying, ingestion
- No more data lake silos

Data Marketplace

- The way to monetize your data
- Live access to data
- Cross-cloud availability
- Publish them on public data marketplace
- Data sharing offered

Monetize Your Data

- Discover data, services and apps from 260+ providers across 18+ categories
- Access the most current data available and receive real time updates automatically
- Reduce data integration costs with direct access to live, ready-to-query data; No ETL



Core Concepts

Data Product:

- Curated collections of tables, views, UDFs, etc.

Listing:

- Data products enriched by metadata to help make them discoverable
- Type of Listings:
  - Free

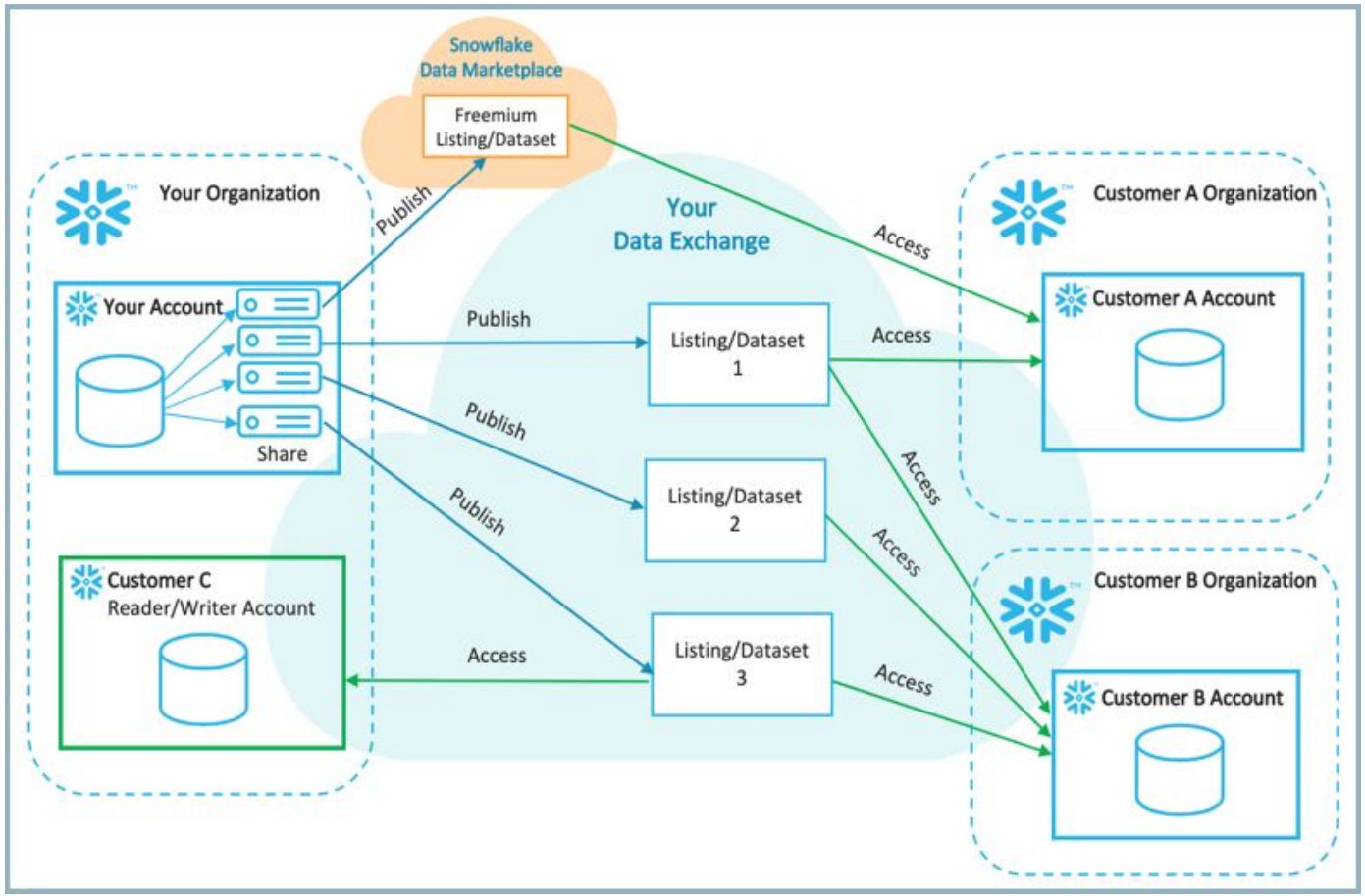
What we already know

- Provider
- Consumer
- Share
- Region
- Replication

Data Exchange

- Private data marketplace
- You publish data and invite consumers
- Use cases:
  - Internal data exchange between business units
  - Collaborate with external parties - vendors, partners, or customers
- Membership managed by data provider
- Roles

Data Exchange



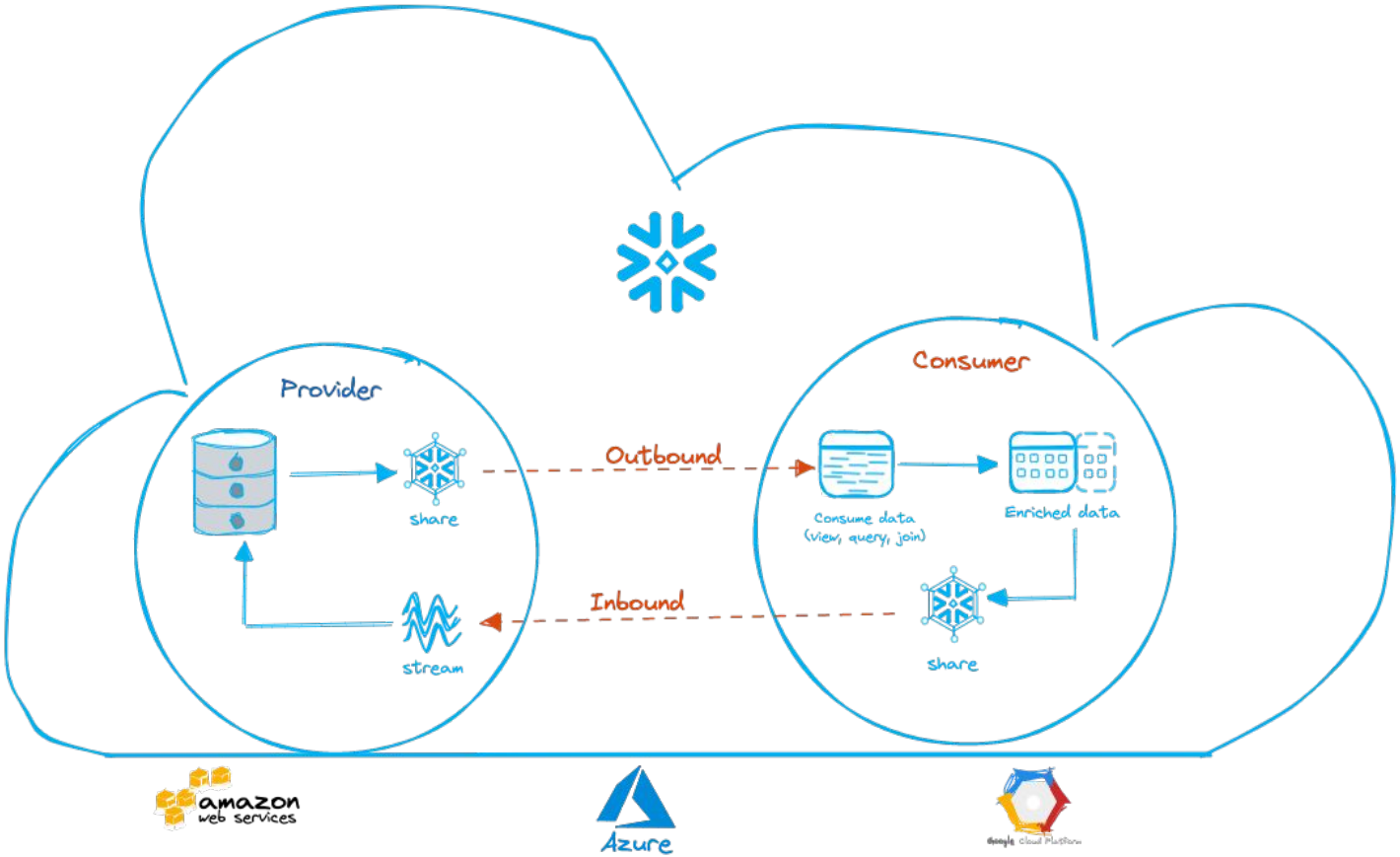
Direct Data Sharing in Detail

Secure Data Sharing

Within a Snowflake region on any cloud

- No data movements
- No data latency within same cloud region
- Provider can instantly revoke object access or drop share
- Shared table/view, UDF

Secure Data Sharing



Share Components

Table

External Table

Secure View

Secure MV

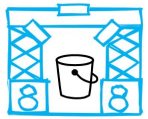
Secure UDF

Share Components

Task

Snowpipe

Stream



Stage

File Format

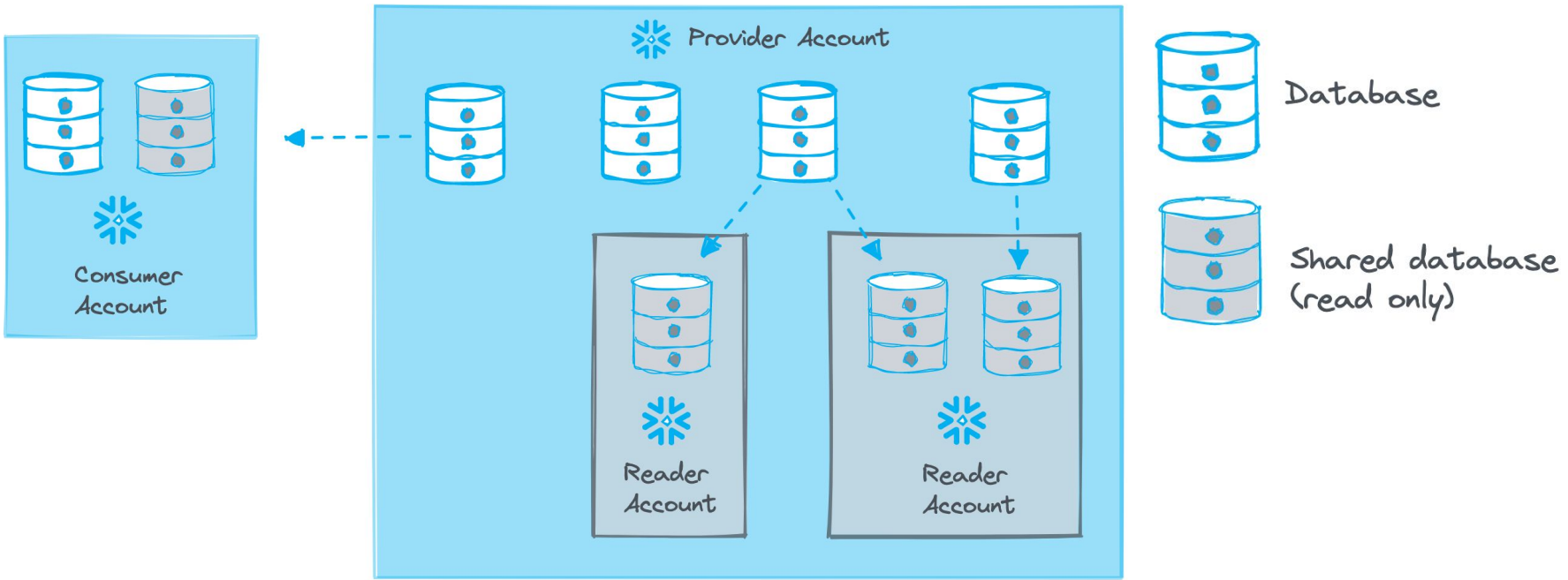
Sharing Data with Non-Snowflake Customers

Reader Accounts

- Data provider can create special accounts
- No requirements for consumers to become Snowflake customers
- Account is owned by data provider
- Credit charges go to provider account
- Provider shares data with reader account
- Reader account can only consume data from provider account - adding

Sharing Data with Non-Snowflake Customers

Reader Accounts



Data Sharing summary

- Modern way to share data
- Cross-cloud, cross-region data sharing - thanks to Snowgrid
- Breaks down data silos
- Single copy of data
- No data movement, copying
- Easy central governance

Knowledge check

1. What does Snowgrid?
a] ingest data from cloud storages (s3, azure blob storage]
b] stores data in internal Snowflake stages
c] connect regions and cloud together [x]
d] allocate resources for multicluster warehouse

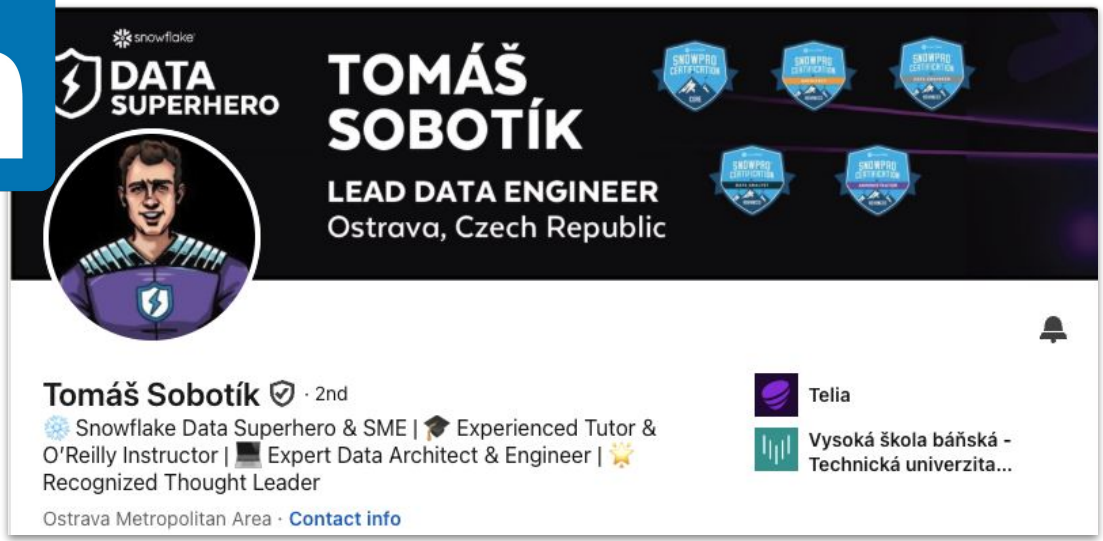
2. When you want to invite & manage consumers to privately share data what feature do you need to use?
a] private data marketplace
b] data marketplace
c] data exchange
d] data sharing

Wrap-up and Tips for Taking the Exam

What Did We Cover?

- Snowflake architecture and layers
- Snowflake security (authentication, authorization)
- Data governance features
- Best practices for loading data into snowflake
- Performance concepts

Where to Find Me



https://www.linkedin.com/in/tomas-sobotik/

Where to Find Me

Medium

Tomáš Sobotík

679 Followers

Lead data engineer & architect, Snowflake Data Superhero and SME, O'Reilly instructor.
Obsessed by cloud & data solutions.

Follow

https://tomas-sobotik.medium.com/

Live Courses

Your Instructor

Tomáš Sobotík

SnowPro Core Certification Bootcamp

Published by O'Reilly Media, Inc.

Beginner to intermediate

Snowflake Fundamentals Bootcamp

Published by O'Reilly Media, Inc.

Beginner

Snowflake for Data Engineering Bootcamp

Published by O'Reilly Media, Inc.

Intermediate

Snowflake Administration

Published by O'Reilly Media, Inc.

Intermediate

Optimizing Snowflake

Published by O'Reilly Media, Inc.

Intermediate

On demand Courses

Your Instructor

Tomáš Sobotík

On-demand course

+ Add to playlist

SnowPro Core Certification Course: Understand the Core Foundations for the Exam

★★★★★ 1 4h 4m

O'Reilly Media, Inc. • June 2025

Quiz

By Tomáš Sobotík

CC in German, English + 6 more

This on-demand course reviews the fundamentals of the Snowflake platform and helps learners prepare for the SnowPro Core certification exam. Through a mix of concise presentations and demonstrations, the course covers architectural concepts of the platform and best practices for building scalable data solution...

Tips and Tricks for exam

- Core exam is less practical and more theoretical – you need to understand the architecture and the features
  - What is XYZ feature for
  - What are the benefits of feature XYZ
  - Can you do XYZ with feature ABC
  - Which feature would you use for XYZ
  - Etc.
- Go through the study guide and read linked documentation pages
- Have a real hands-on experience – If not from projects, then go through quick start guides and level up series:
  - https://quickstarts.snowflake.com
  - https://learn.snowflake.com/en/pages/level-up-track/
- Validate your knowledge through similar courses
- Be careful about trusting “testing questions dumps” – they might vary in quality

Good Luck in your Snowflake journeys! 👋

Share your success! 🥳