O'REILLY

Snowflake Architecture

Recap of the architecture

Tomas Sobotik

Learning objectives

By the end of this live, hands-on 3-part series, you'll understand:

- Data loading strategies
- Data ingestion workflow and semi-structured data ingestion
- Continuous data pipelines
- Building and monitoring data pipelines
- Using SQL API and external functions
- Using Snowpark for data transformation
- Building CI/CD pipelines

You will be familiar with almost all data engineering features which Snowflake currently offers.

Tomas Sobotik

- Based in Czech Republic
- 2 kids
- 15+ years in data industry
- Data engineer and architect
- Big Snowflake fan and enthusiast
- Snowflake Data Superhero
- Snowflake Subject Matter Expert

Snowflake in a nutshell

- Founded in 2012 and publicly launched in 2014
- Software as a Service (SaaS)
  - No license to buy, no HW to maintain
- Originally with DWH in the cloud workload
- Only public cloud (AWS, Azure, GCP)
- Separated storage and compute
  - Possibility to scale each part independently
- Per second billing (pay as you go)

ARCHITECTURE

Platform requirements

Fast for any Workload

Run any number or type of job across all users and data volumes quickly and reliably

It just works

Replace manual with automated, optimize costs and minimize downtime

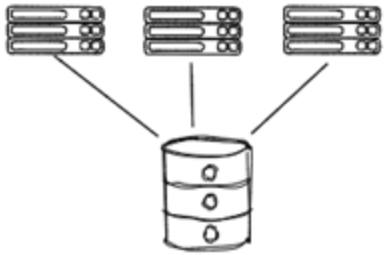
Connected to what matters

Extend access and collaboration across teams, workloads, clouds - seamlessly and securely

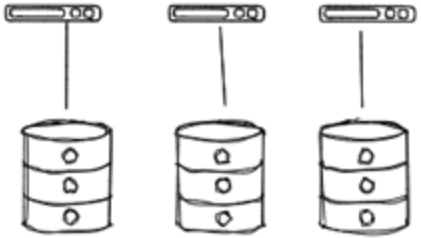
Next level of DWH architecture

Storage and compute separated

Traditional architectures



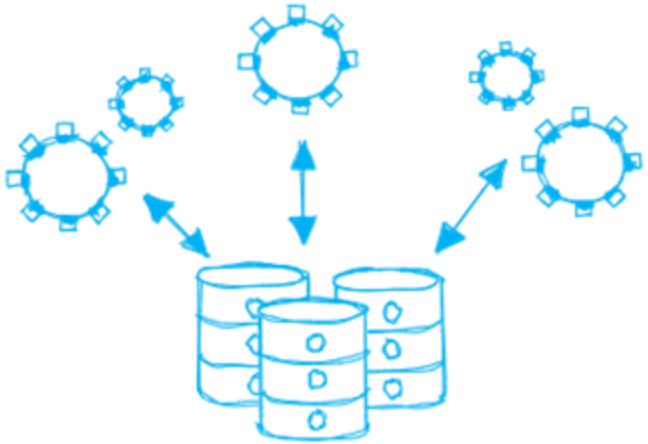
"Shared-disk"
Shared storage
Single Server



"Shared-nothing"
Decentralized, local storage
Single cluster
MPP APPLIANCE

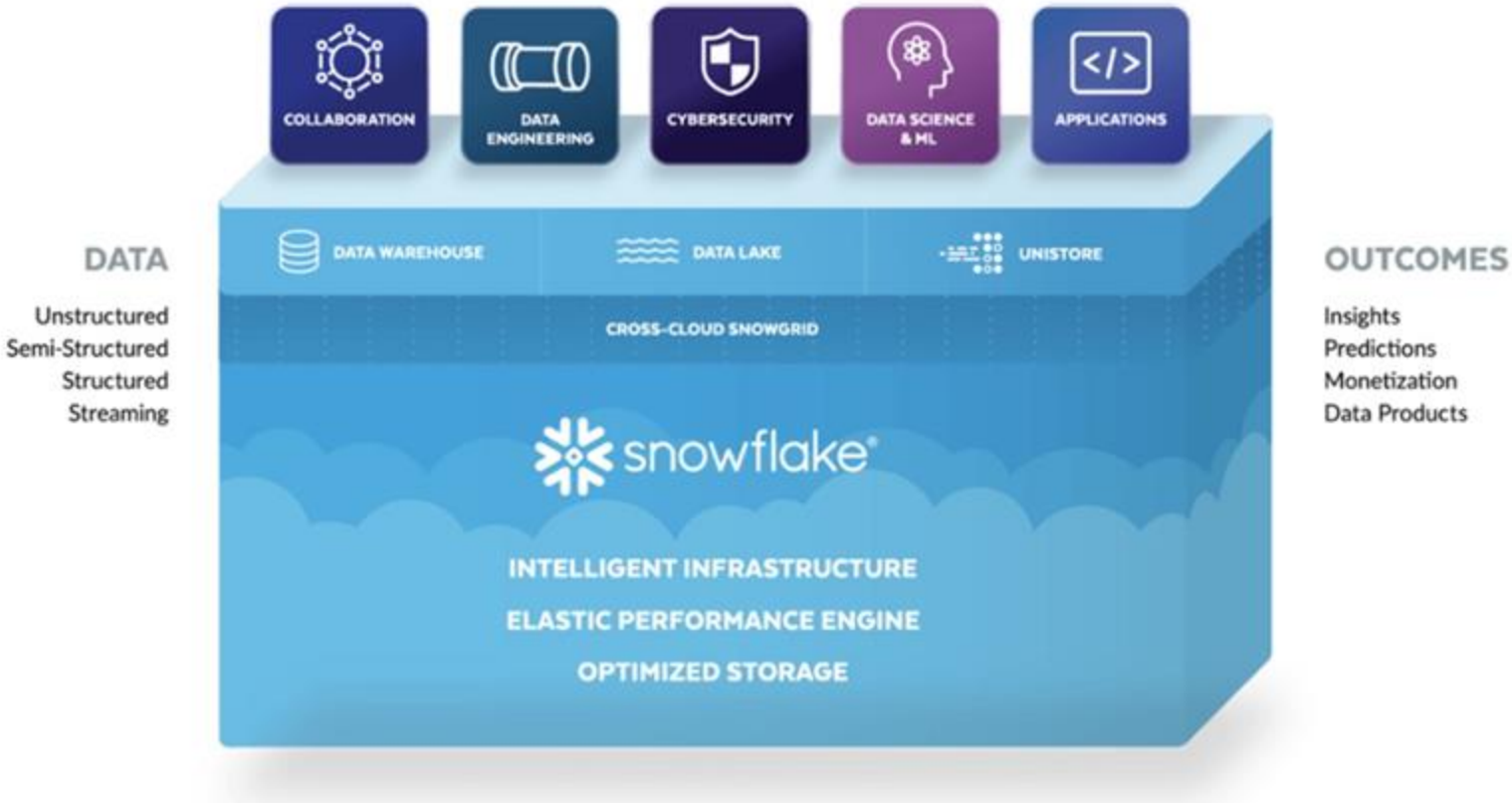
Teradata, Netezza, Vertica
Redshift

Data Cloud

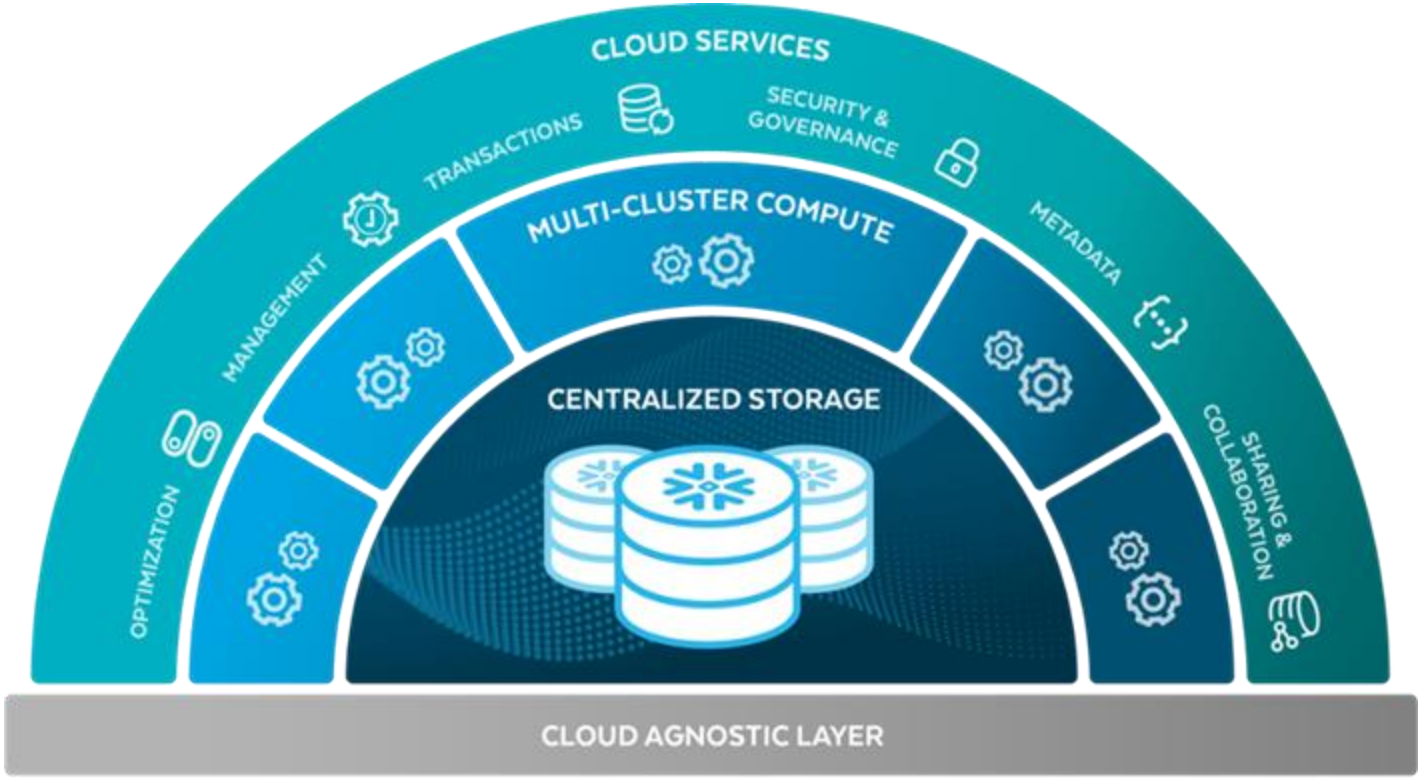


"Multi-cluster, shared data"
Centralized, elastic storage
Multiple MPP, independent compute clusters
UNLIMITED Storage and Compute

Globally


COLLABORATION
DATA ENGINEERING
CYBERSECURITY
DATA SCIENCE & ML
APPLICATIONS
DATA WAREHOUSE
DATA LAKE
UNISTORE
CROSS-CLOUD SNOWGRID
snowflake
INTELLIGENT INFRASTRUCTURE
ELASTIC PERFORMANCE ENGINE
OPTIMIZED STORAGE
DATA
Unstructured
Semi-Structured
Structured
Streaming
OUTCOMES
Insights
Predictions
Monetization
Data Products

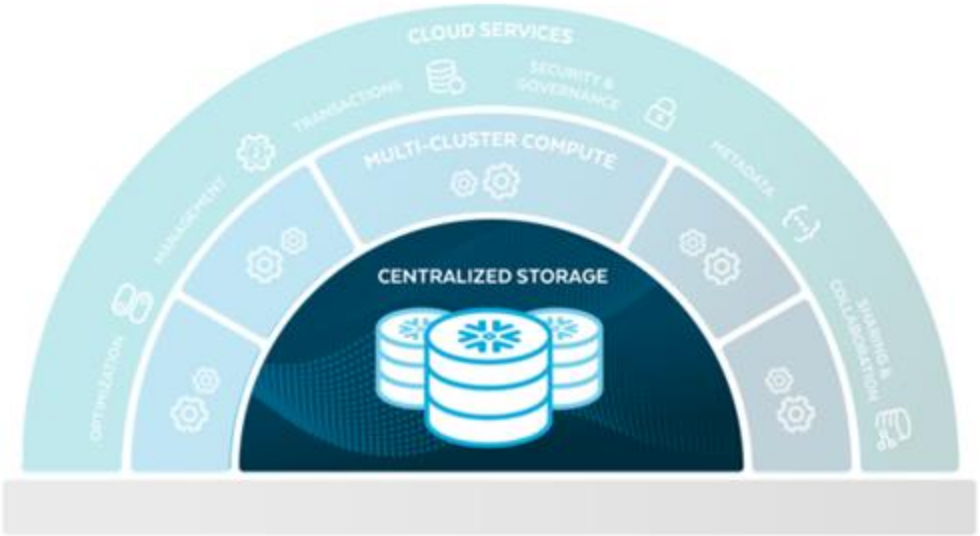
Under the hood



Google Cloud aws Azure

Data Storage

- Centralized storage
- Up to 5x compression
- Native support for semi-structured data
- Data are organised in micro partitions
- Zero-copy cloning
- Time travel
- Storage is small part of the overall bill



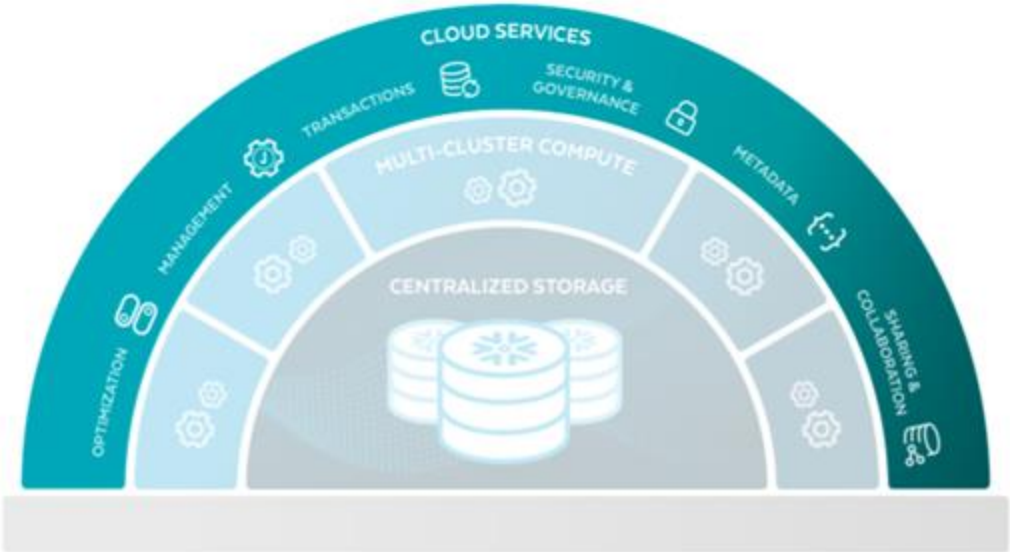
Compute

- On-demand virtual warehouses
  - Elastic (turn on/off instantly)
  - Scaling up - bigger
    - T shirt sizes
  - Scaling out - more
    - multi cluster warehouses

- Suspend
- Per second billing
- Most spendings is for virtual warehouses run time

Cloud services

- Compute that does not run on virtual warehouse
  - Show commands
  - Query result cache
- Serverless features
  - Snowpipe
  - Auto-clustering
  - Maintenance of MV
  - Cross-cloud replication

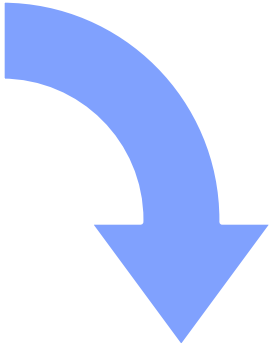


Micropartitions

Storage unit holding the data

- Physical files holding the logical tables data
- Many micro partitions per table
- Max 16 MB compressed data …50 - 500 MB of uncompressed data
- Immutable
    - Updates create a new micropartition
- Snowflake keeps metadata about micro partitions in service layer
    - Helps with query optimization and performance
- Tables divided horizontally in smaller units

ID	NAME
1	Alice
2	Bob
3	Chuck
4	Jane
5	John
6	Mary

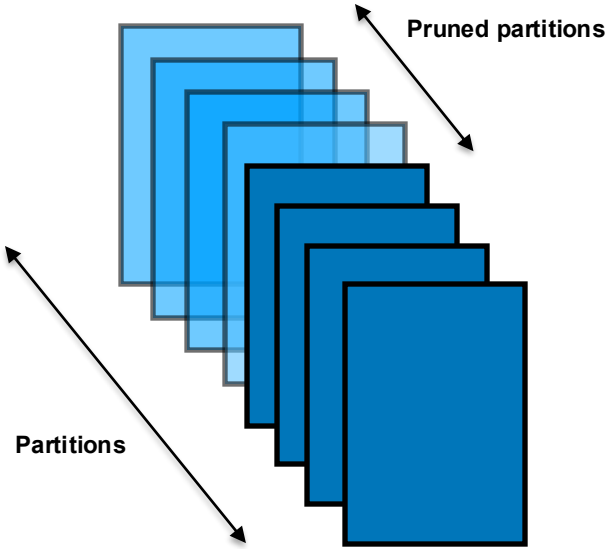


P1	ID	1	2	3
	NAME	Alice	Bob	Chuck

P2	ID	4	5	6
	NAME	Jane	John	Mary

Micropartition advantages

- Metadata used by optimizer for partition pruning
  - Horizontal pruning first - by partition
  - Vertical pruning next - by column
- Unit of management for DML - because of immutability
- Thanks to stored metadata some queries are pure metadata operations
  - No need for running virtual warehouse!
- Data Clustering
  - Effective data loading have impact on query performance



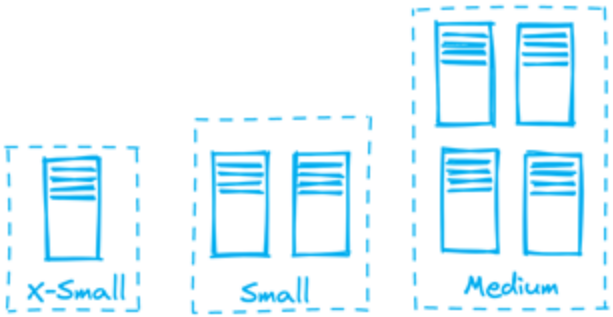
Virtual Warehouses

- A named wrapper around a cluster of servers with CPU, memory, and disk
  - A compute cluster = logical collection of VMs (1+)
- The larger the warehouse, the more servers in the cluster
- Snowflake utilizes T-shirt sizes for Virtual Warehouses
- Each size up doubles in size
  - Number of nodes correspond to configured size of VW
- Jobs requiring compute run on virtual warehouse
- While running, a virtual warehouse consumes Snowflake credits
- You are charged for compute
- Snowflake account object

Scaling up vs. scaling out II.

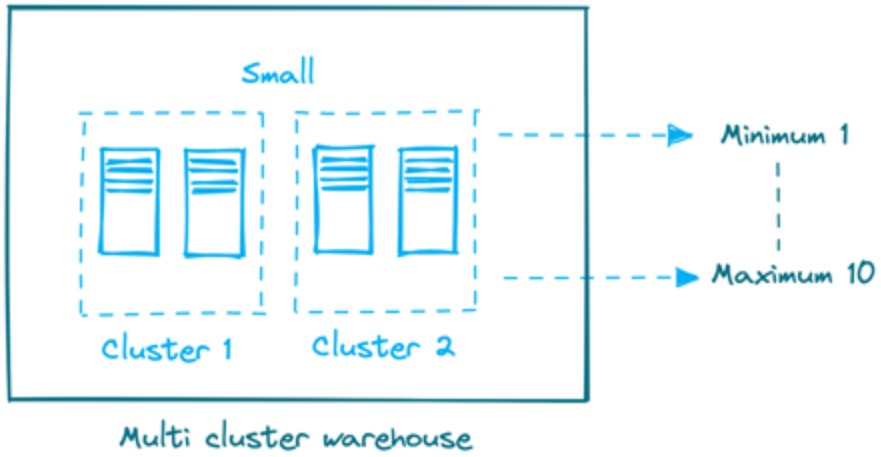
Scaling up

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

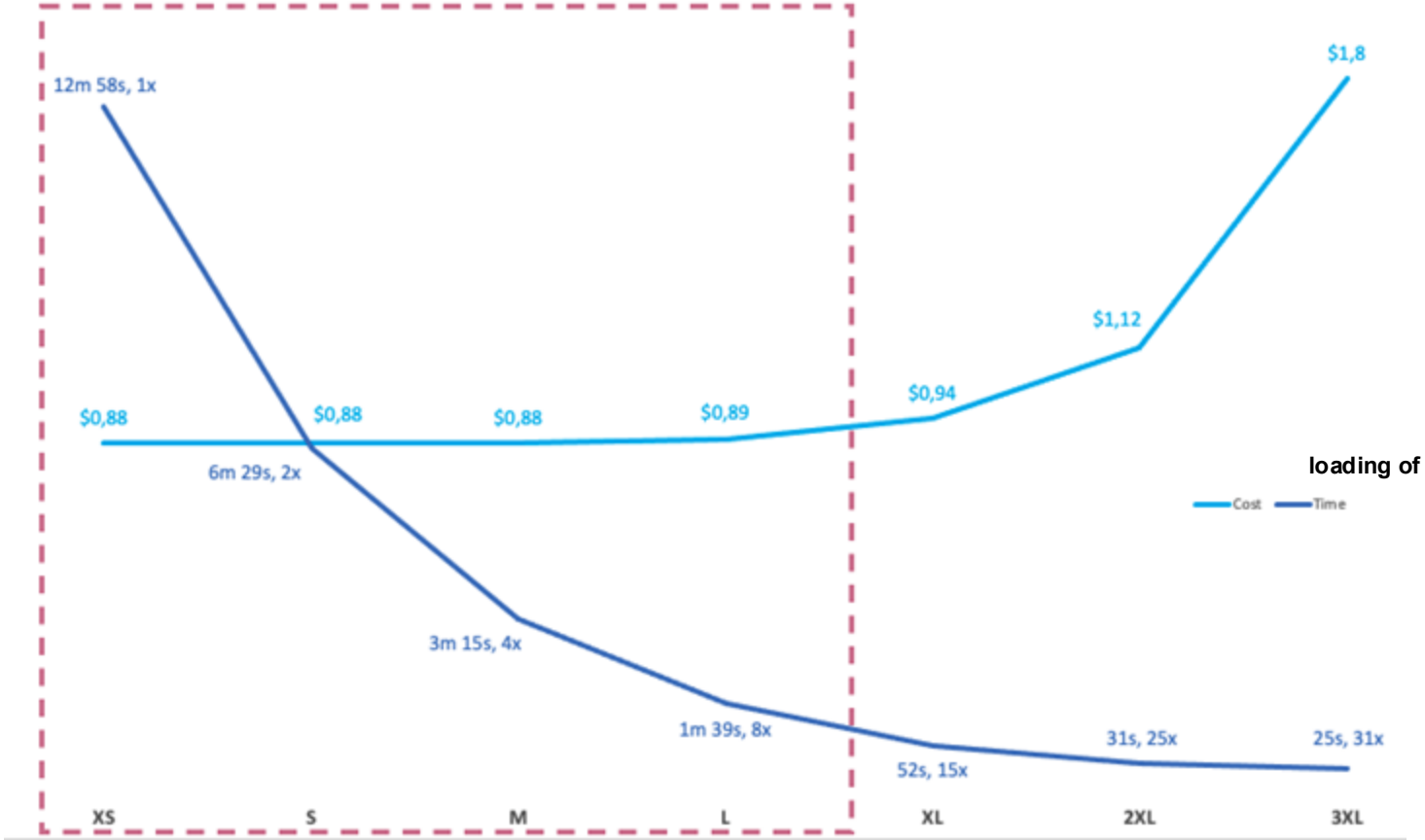


Scaling out

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



Bigger does not mean more expensive



Poll

1. How many layers has Snowflake architecture?

a) 2

b) 3

c) 4

2. Where can you run Snowflake? - Mark all correct options

a) Public cloud
b) Private cloud
c) Hybrid Cloud
d) On prem

3. How do you pay for Snowflake? - Mark all correct options

a) Need to buy a license

b) Usage based pricing model

c) Need to buy servers

8. Micro partitions could be updated

A. Only if there is less than 16 MB of data
B. Never
C. Whenever it is needed

4. What is Snowflake Storage architecture

a) Centralized storage

b) Decentralized Storage

c) Shared storage

5. Does Snowflake store data compressed?

a) True

b) False

6. How many running virtual warehouse can you have?

a) 3

b) 5

c) Almost unlimited

7. Can you change the size of virtual warehouse?

a) True

b) False

O'REILLY®

Data loading strategies

Tomas Sobotik

Different approaches for data loading

Batch processing

Ingestion at given time

Latency in hours or days

Full / delta loads

Source -> cloud storage -> Snowflake

ERP, Billing, HR, Finance systems

Continuous processing

Event driven

Latency in minutes

Delta loads

Source -> cloud storage -> Snowflake

Orders, Customer profiles changes, Consent data

Real time processing

Event driven

Latency in seconds

Delta loads

Source -> event broker -> Snowflake

IoT sensors, logs, credit scoring

Batch processing

Benefits

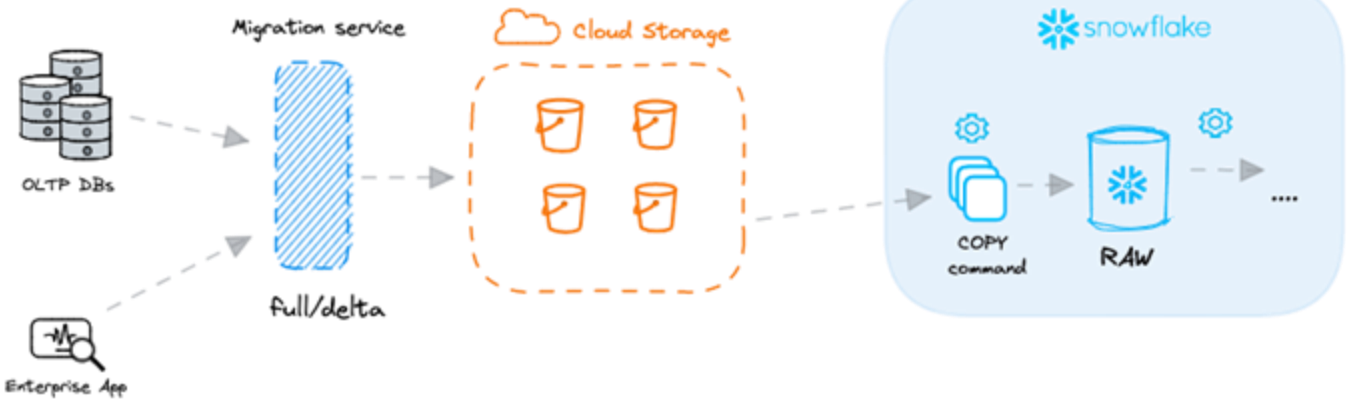
+ stable workload
+ easy to maintain
+ many available tools
+ easy reprocessing

Schedule based

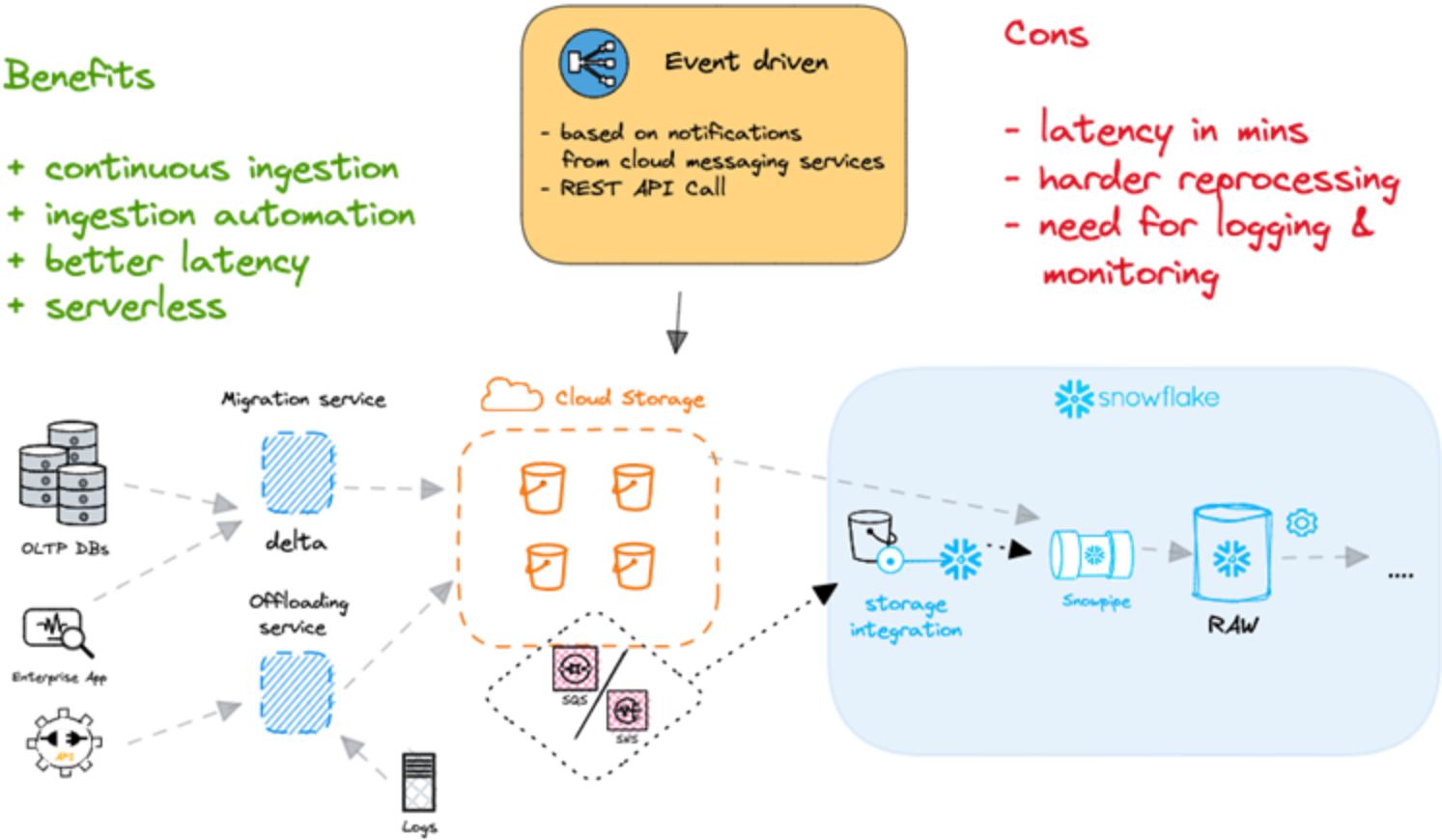
- every day at 03:00
- first Monday in month at 06:00
...

Cons

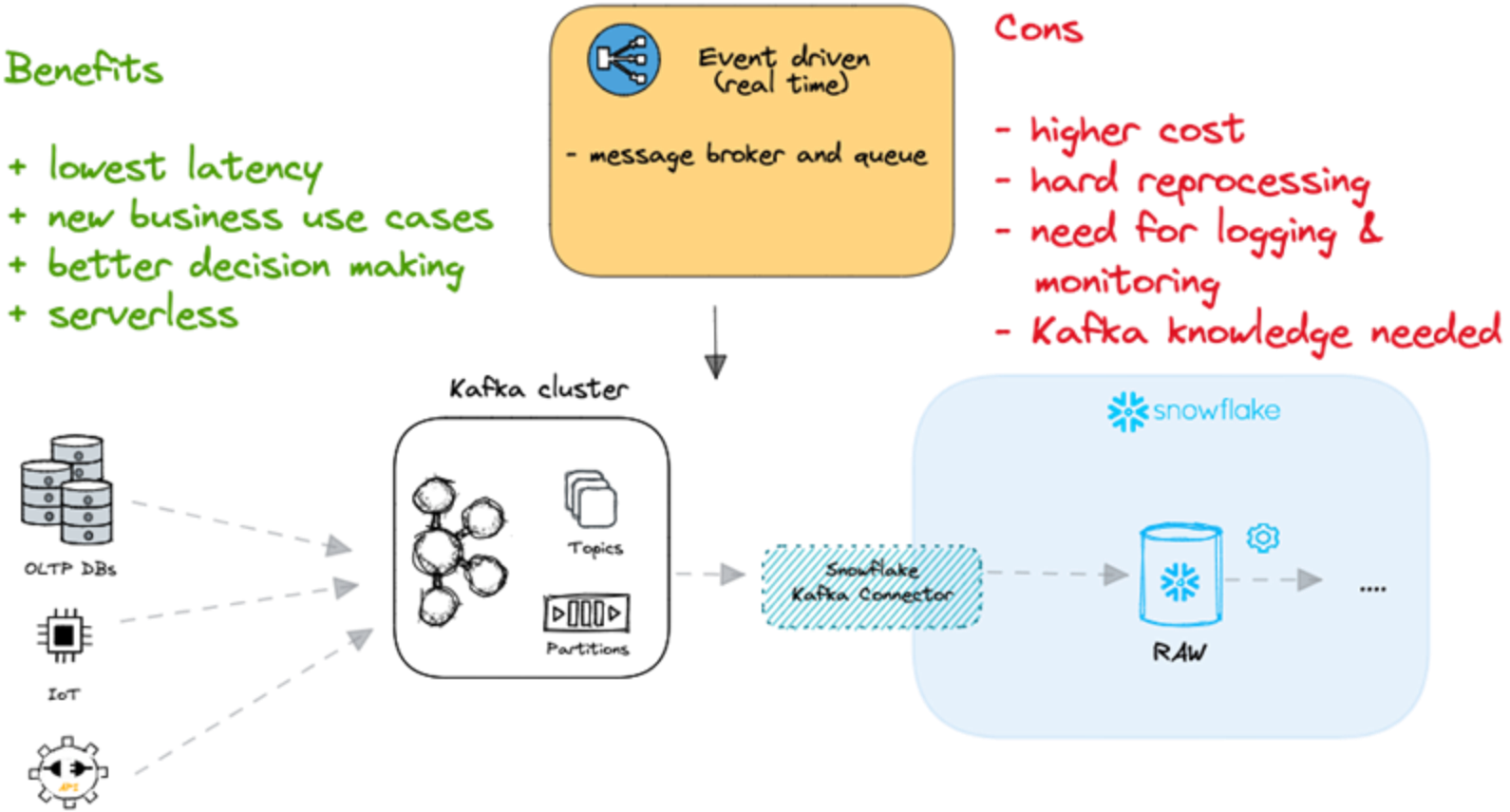
- slow
- hard to implement for some modern data sources



Continuous processing



Realtime processing



How to select the right approach ?

- Driven by business needs
  - Validated by experts (Solution Architects, Data Engineers)

- You should take into consideration
  - Complexity of whole solution
  - Future maintenance
  - Current technology stack
  - Cost

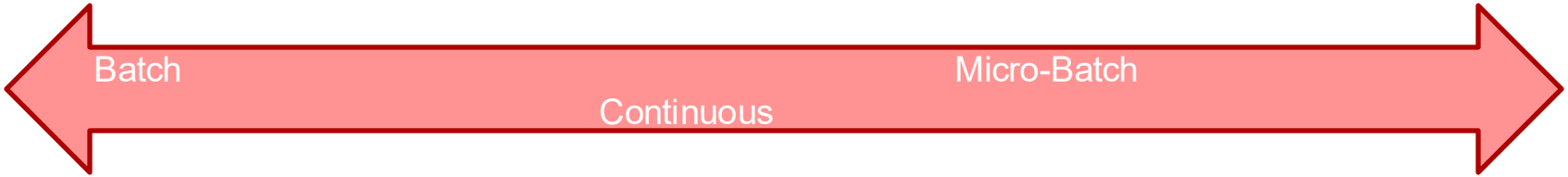
Batch X continuous processing

Batch

- Migration from traditional data sources
- Needs to be scheduled
- Needs a warehouse
- Easy to hit concurrency limits

Continuous

- Ingest from modern data sources
- No scheduling
- No warehouse needed
- Serverless billing model
- Designed for high concurrency



Data files considerations & preparations

Data loading recommendations

Organize files by path

- Create logical paths per required dimension
  - Time
  - Location
  - Department

```
US/east_cost/finance/monthly/2022/02
```

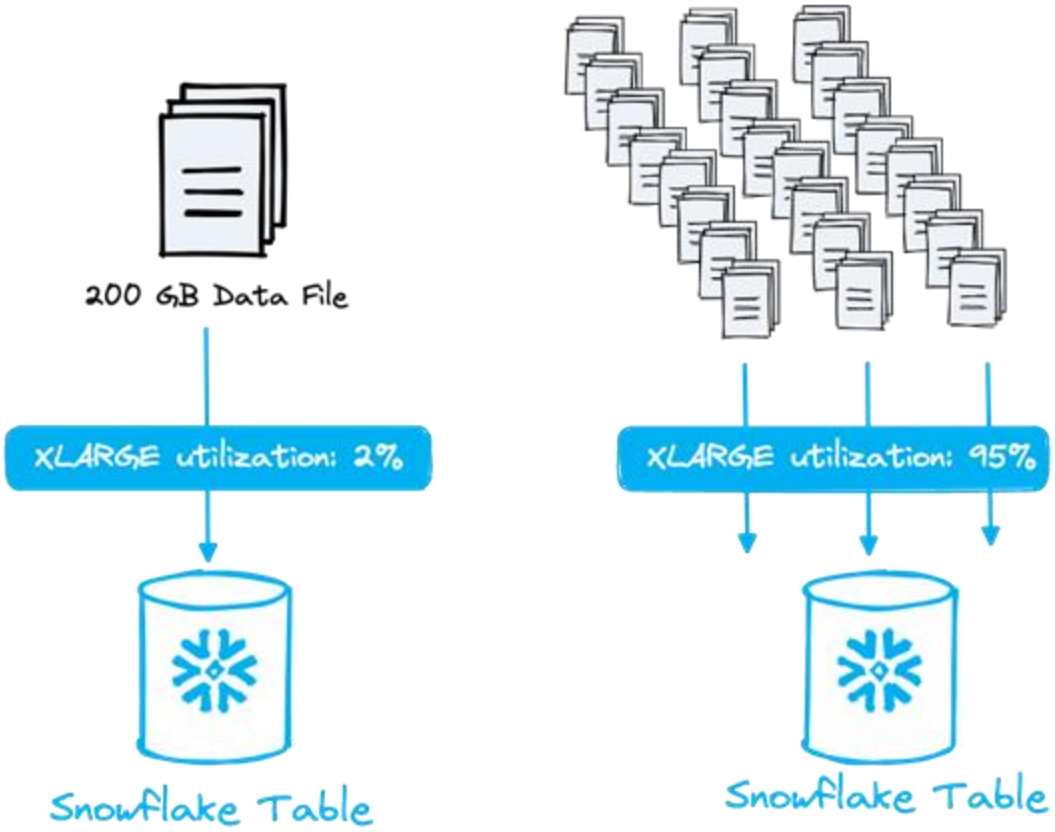
Warehouse size	Number of Threads
XS	8
S	16
M	32
L	64
XL	128

Number of files that can be loaded in parallel

File sizes

- Split large files before loading
- Batch / combine small files before loading
- Recommended size is 100 to 250 MB (compressed)

Serial vs Parallel processing



File locations

All files in single location

Processing is slower as engine needs to scan more files to find the right ones

Organised files in multiple locations

Faster. Multiple directories with same file types allow for scanning fewer files

Keep in mind limitations and recommendations

- Split large files, aggregate smaller files
- 16 MB for single row with VARIANT data type
- Staging files for Snowpipe in 1 min intervals
- Dedicated warehouse for loading operations
- You should take into consideration
    - Complexity of whole solution
    - Future maintenance
    - Current technology stack
    - Cost
- Metadata management

Load Metadata

- Snowflake keeps metadata about each data ingestion load
- Kept for 64 days
- Prevents data duplication
- What is tracked
  - Name of each file
  - File size
  - Number of parsed rows
  - Timestamp of the last load for the file
  - Info about errors

- Be aware of differences between load metadata COPY_HISTORY and staged files metadata

Staged files Metadata

- Metadata about staged files
  - **METADATA$FILENAME**
  - **METADATA$FILE_ROW_NUMBER**
- Could be queried and loaded into table
  - As part of COPY command
- Limitations – not included in following commands (could be only queried by name)
  - SELECT *
  - SHOW
  - DESCRIBE
- Works for both – internal and external stages

```
SELECT
 METADATA$FILENAME,
 METADATA$FILE_ROW_NUMBER,
  t.$1, t.$2
FROM @mystage1
 (file_format => myformat) t;
```

Poll

1. Which processing requires scheduling

a) Batch

b) Continuous

c) Real time

2. Which processing has latency in minutes

a) Batch

b) Continuous

c) Real time

3. Real-time processing has the best latency

a) True

b) False

4. How should be files organized in stage area

a) Randomly

b) By Size

c) By Path

5. What is recommend file size

a) 16 MB

b) 100 – 250 MB

c) 2 GB

6. METADATA$FILE_LOCATION is one of the Snowflake metadata

a) True

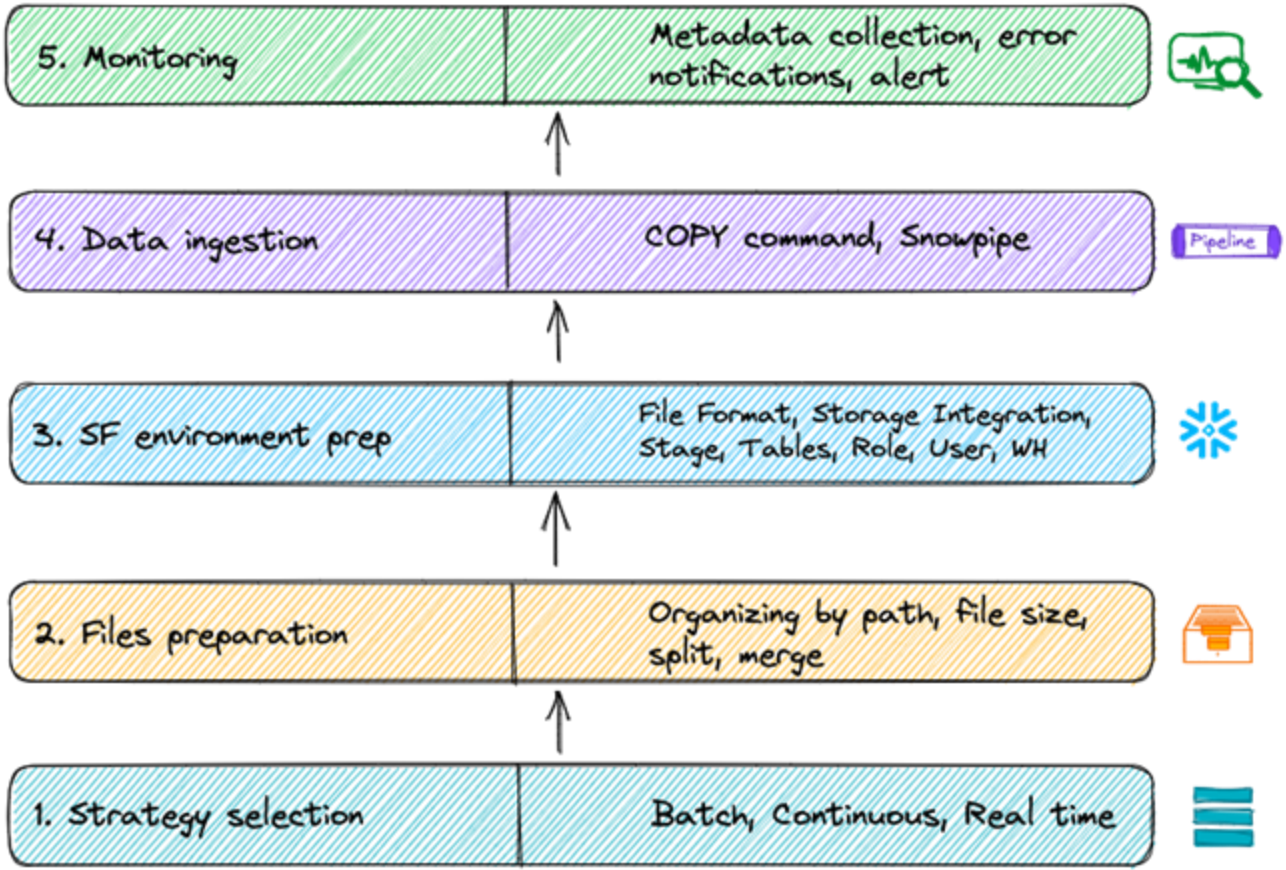
b) False

O'REILLY®

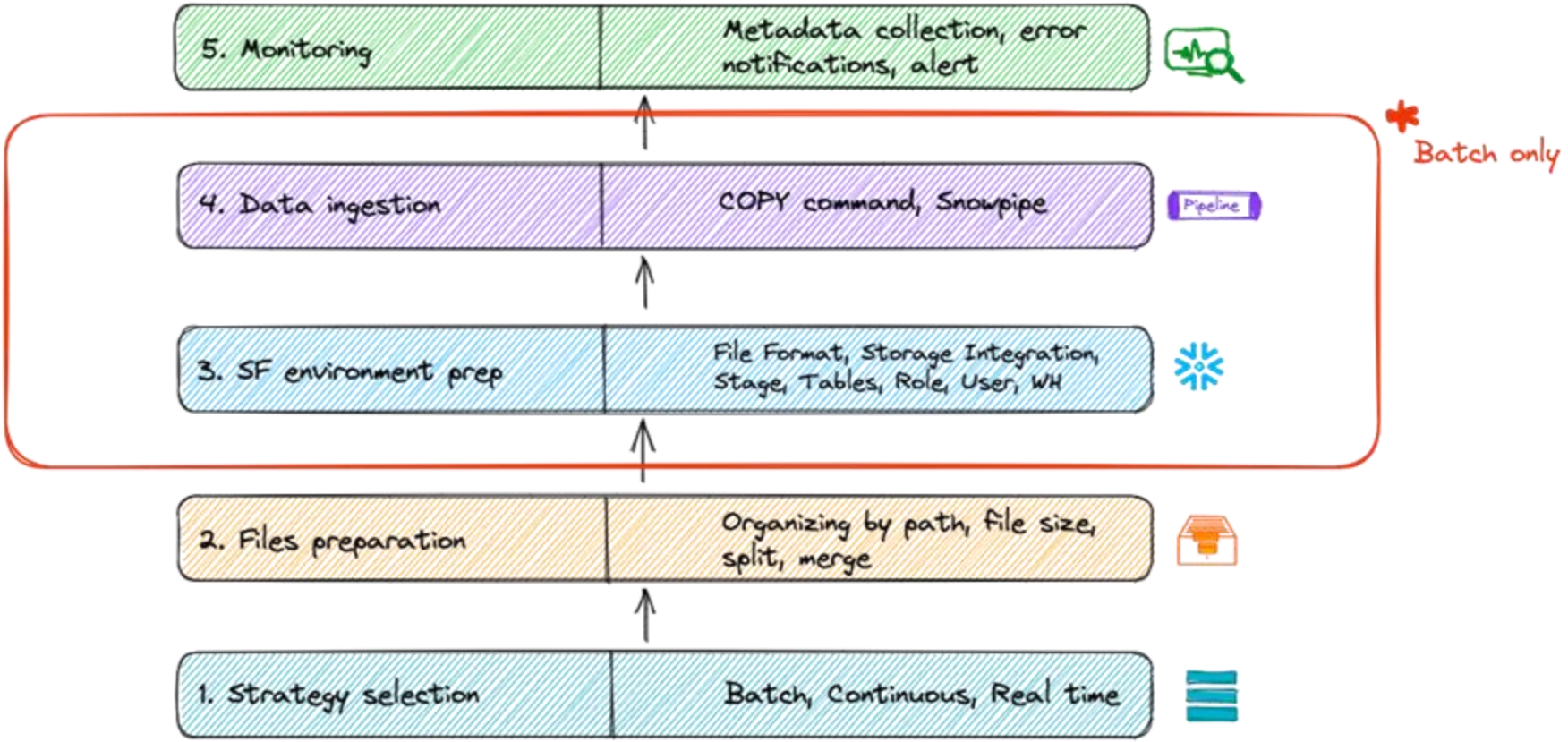
Data Ingestion Workflow

Tomas Sobotik

Ingestion workflow



Ingestion workflow



FILE FORMAT

File structure description

CSV

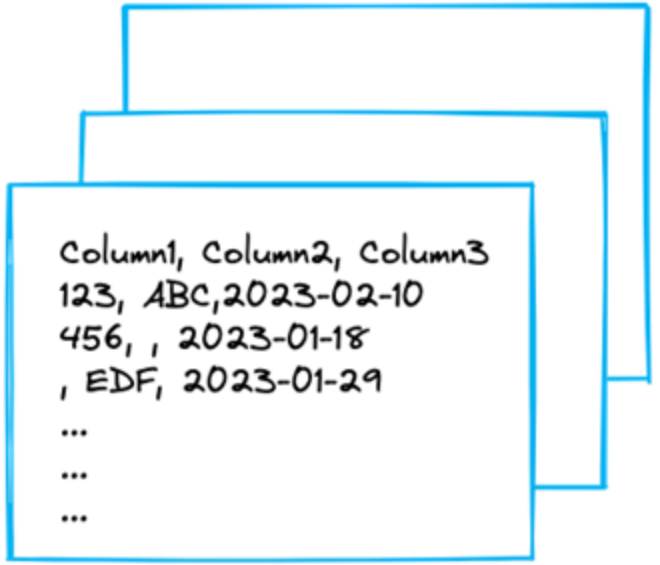
- Record and field delimiters
- Escape characters
- NULL handling and other parameters

JSON

- Strip null values
- Date and time format
- Trim spaces and other parameters

Can be defined on different levels

- STAGE object
- COPY command
- Named object




```
CREATE OR REPLACE FILE FORMAT my_csv
  TYPE = csv
  FIELD_DELIMITER = ','
  SKIP_HEADER = 1
  NULL_IF = ''
```

FILE FORMAT

File structure description

CSV

- Record and field delimiters
- Escape characters
- NULL handling and other parameters

JSON

- Strip null values
- Date and time format
- Trim spaces and other parameters

Can be defined on different levels

- STAGE object
- COPY command
- Named object

Use name objects!

- Reusability
- Segregation of duties
- Strong governance model
- Speed up the development process
- Easy to change it – just one place

STAGE

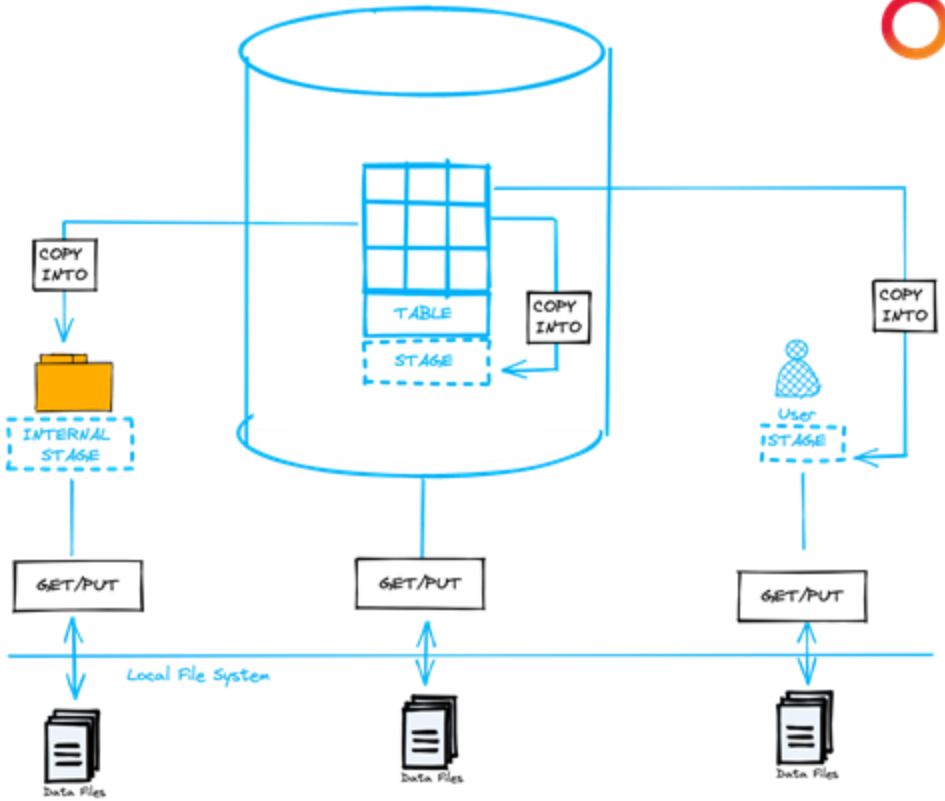
File location description

Internal or External

- Files could be loaded from local machine or existing landing zone in the cloud (S3, Blob storage, Cloud Storage)
- PUT command for loading local files into internal stage

Stage types

- Table Stage: Created automatically for each table: **@%[Table_Name]**
- User stage: Created automatically for each user: **@~**
- Named stage: created manually with SQL - internal or external
- You can't set file format for user and table stages



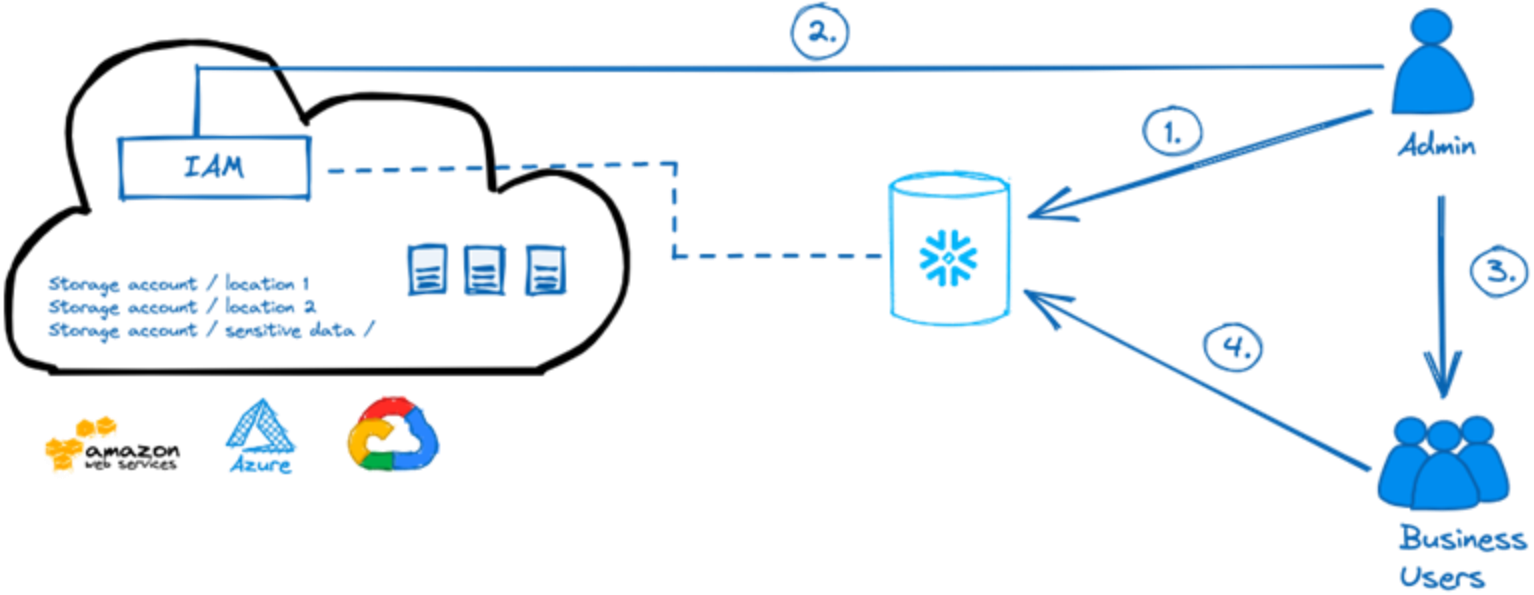
Stage security parameters

```
create stage my_ext_stage1
  url='s3://load/files/'
  credentials=(aws_key_id='1a2b3c' aws_secret_key='4x5y6z');
```

- Access keys are part of Stage definition
  - Disadvantages:
    - Security information needs to be shared with developers
    - Key rotation process
    - Needs to be defined for each stage
    - Security information is stored in Snowflake

Credential-less stages

Storage integration



1. Admin creates storage integration
2. Admin create trust policy between cloud provider and Snowflake
3. Admin grant storage integration usage to dev or business roles
4. Business users can create external stage with this storage integration

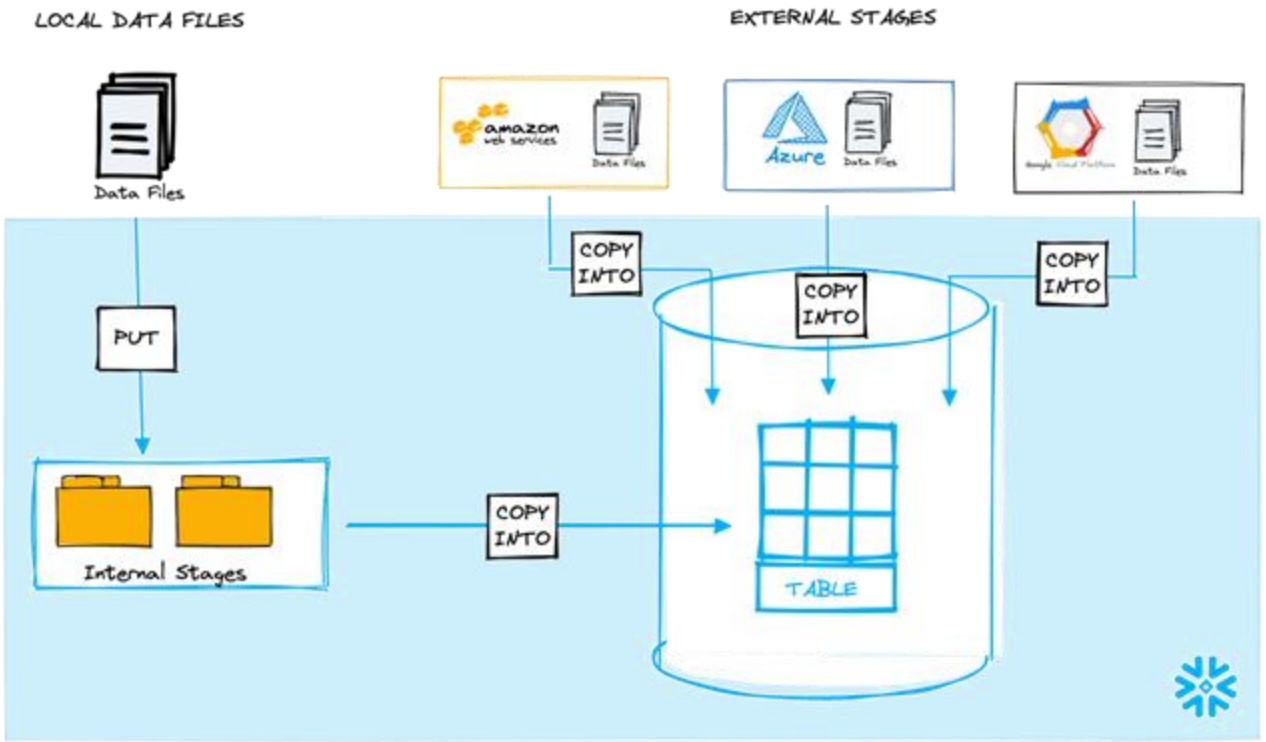
Storage integration benefits

- Credentials are neither shared nor stored in Snowflake
- Reusability
- No need for recurring keys rotation
- Developers do not need to have the credentials in order to access files in external storages from Snowflake
- Segregation of duties

```
CREATE STORAGE INTEGRATION s3_int
TYPE = EXTERNAL_STAGE
STORAGE_PROVIDER = 'S3'
ENABLED = TRUE
STORAGE_AWS_ROLE_ARN = 'arn:aws:iam::001234567890:role/myrole'
STORAGE_ALLOWED_LOCATIONS = ('s3://mybucket1/mypath1/', 's3://mybucket2/mypath2/')
STORAGE_BLOCKED_LOCATIONS = ('s3://mybucket1/mypath1/sensitivedata/');
```

```
CREATE STAGE my_s3_stage
STORAGE_INTEGRATION = s3_int
URL = 's3://bucket1/path1/'
FILE_FORMAT = my_csv_format;
```

Loading data into Snowflake



Batch data loading

- Running virtual warehouse is required
- Local files
- External cloud landing zones

COPY command – basics

- Migration from traditional data sources
- Batch processing
- Independent scaling of compute resources based on your needs and different workloads
- For both data loading and unloading
- Using Stage object as data source/target
- Need to define user managed warehouses for processing
- Basics transformations could be part of command
- A lot of copy options
- Support for plenty of formats
  - Flat files (CSV, TSV)
  - Semi structured JSON, Parquet, AVRO

Basic syntax

```
COPY INTO <table_name>
FROM <internal_stage> |
     <external_stage>
FILE_FORMAT = ( … )
```

Most usable COPY options

- ON_ERROR
- FORCE
- PURGE
- MATCH_BY_COLUMN_NAME
  - Semi structured formats
- RETURN_FAILED_ONLY
- ENFORCE_LENGTH

Command output

Column name	Description
FILE	Name of source file and relative path to the file
STATUS	Loaded, failed …
ROWS_PARSED	# parsed rows from source file
ROWS_LOADED	# loaded rows from source file
ERROR_LIMIT	If the number of errors reaches this limit, then abort
FIRST_ERROR	First error in source file
FIRST_ERROR_LINE	Line number of the first error
FIRST_ERROR_CHARACTER	Position of the first error character
FISRT_ERROR_COLUMN_NAME	Column name of the first error

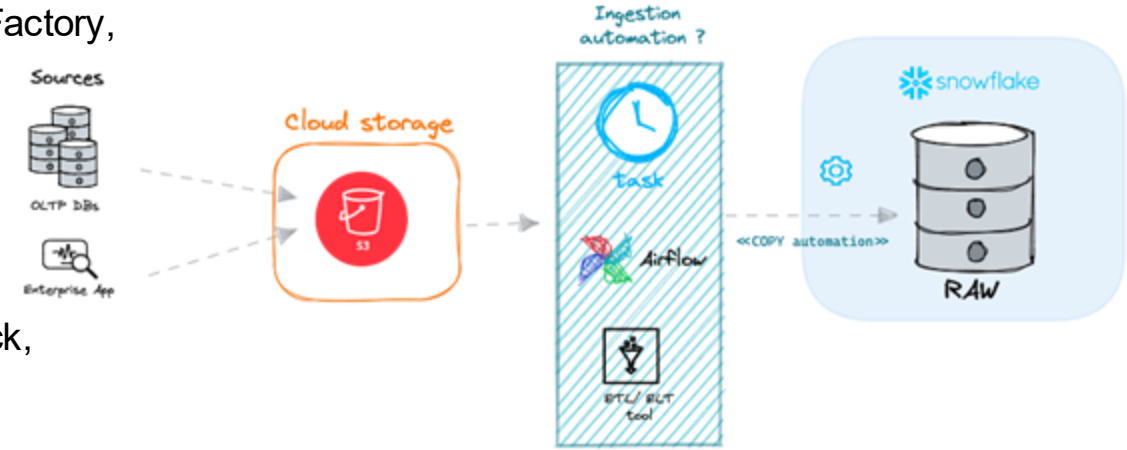
Transformations in COPY command

- Avoid the use of temporary tables to store pre-transformed data
- Column ordering, omission and cast
- Done through the **SELECT** statement with limitations. Not supported features
  - WHERE, ORDER BY, LIMIT, FETCH, TOP, JOIN, GROUP BY
- What can be done
  - Using CURRENT_TIMESTAMP()
  - Using SEQUENCE, AUTOINCREMENT or IDENTITY columns
  - FLATTEN semi-structured data into individual columns

```
copy into home_sales(city, zip, sale_date, price)
from (select t.$1, t.$2, t.$6, t.$7 from @mystage/sales.csv.gz t)
FILE_FORMAT = (FORMAT_NAME = mycsvformat);
```

INGESTION AUTOMATION

- External tools
  - Airflow
  - ELT/ETL tools (Azure Data Factory, Fivetran, Airbyte, etc.)
- Internal features
  - Tasks
  - Snowpipes
- Depends on your technology stack, customer requirements



Best practise / recommendation?

Keep it simple!

Load metadata

available for each file

COPY_HISTORY

Table function
Information schema
14 days retention

Status of the load
row count
error count
target table
- ..

LOAD_HISTORY

view
Information schema
14 days retention

Status of the load
row count
error count
target table
- ..

LOAD_HISTORY

view
ACCOUNT_USAGE
365 days retention
90 mins latency

Status of the load
row count
error count
target table
..

Exercise

This exercise will be dedicated to practising data ingestion into Snowflake. We need to create several Snowflake objects in order to start with ingestion.

Please find detailed instructions on GitHub:

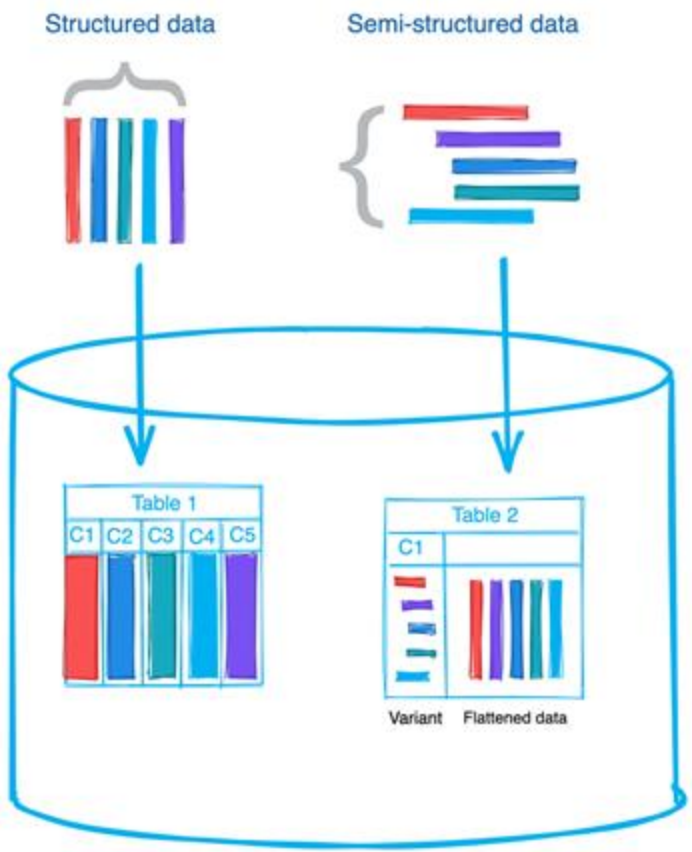
- week1/week1_exercise_desc.pdf
- week1/week1_exercises.sql

O'REILLY®

Semi Structured Data

Tomas Sobotik

Overview



- Native support for JSON, AVRO, Parquet, XML, ORC
- Snowflake has a VARIANT data type for storing semi structured data
- Stores original document as is
- Traditional DBs requires complex transforms to get data into column structure
- SQL syntax is simple dot notation
- When ingesting SSD, data could be columnized and metadata collected
- Native support for all SQL operations

Semi structured data challenges

Only for imports/exports

Parquet, Avro, ORC

Popular formats for Data Lakes

Not readable by humans

Need to know the schema before ingestion

Table creation and COPY command automation could be automated nowadays

Import/export or storing

Easily readable by humans

Popular formats for APIs

No defined structure

Could contains many nested levels with objects, arrays or key – value pairs

Store it as is or flatten?

Parquet ingestion challenges

- You need to define schema when you read from Parquet file
  - You need to create a landing table with file schema
  - You need to use the schema in COPY command
  - You need to Cast columns in COPY command - defining target data type
- All parquet data are stored in Single column
- We use SELECT statement as part of COPY command

A lot of manual effort is needed

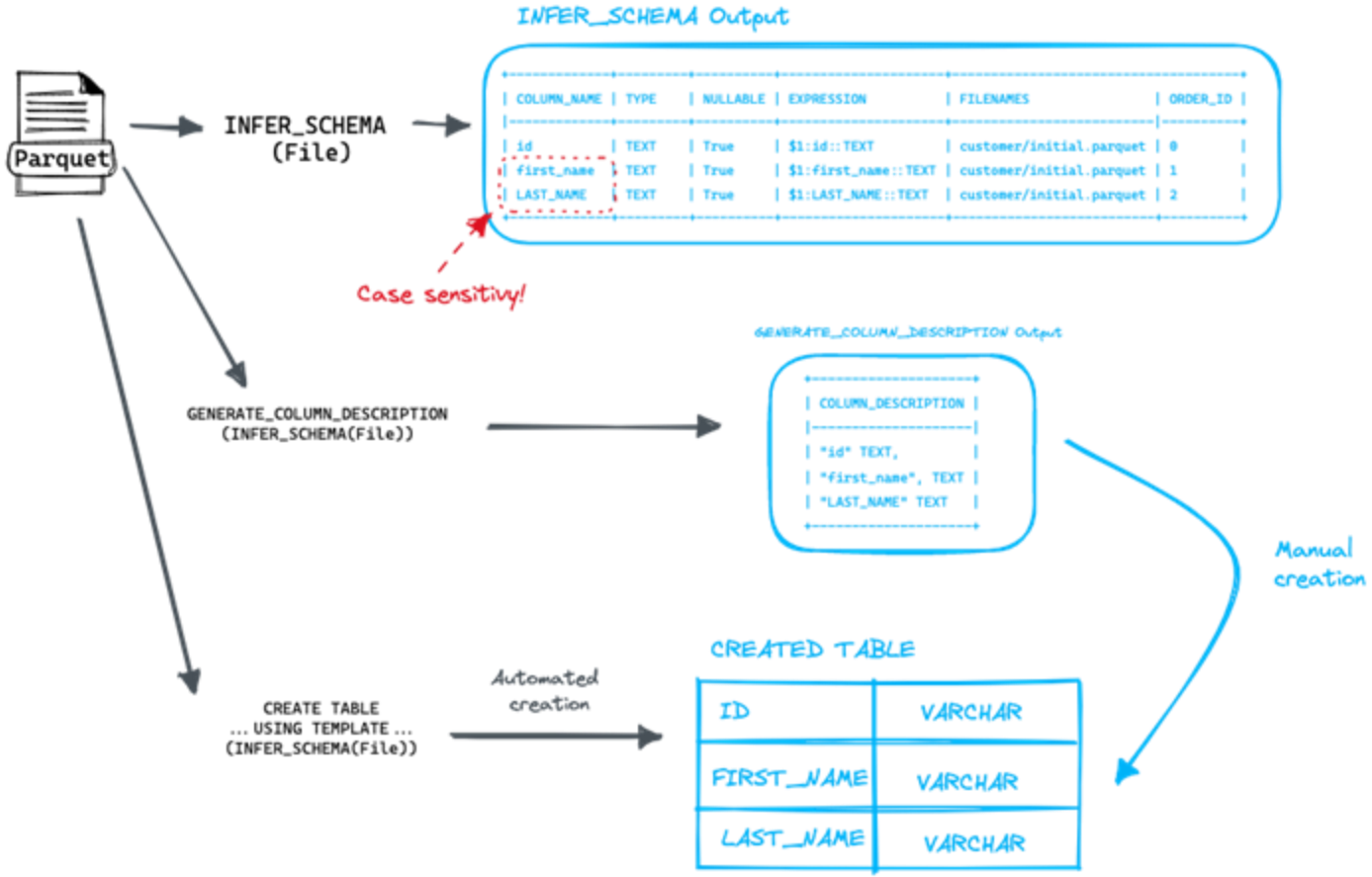
Syntax example:

```
COPY INTO <table_name>
   FROM ( SELECT
      $1:column1::<target_data_type>
      $1:column2::<target_data_type>
      $1:column3::<target_data_type>
      FROM
<my_stage>.<my_file.parquet>
);
```

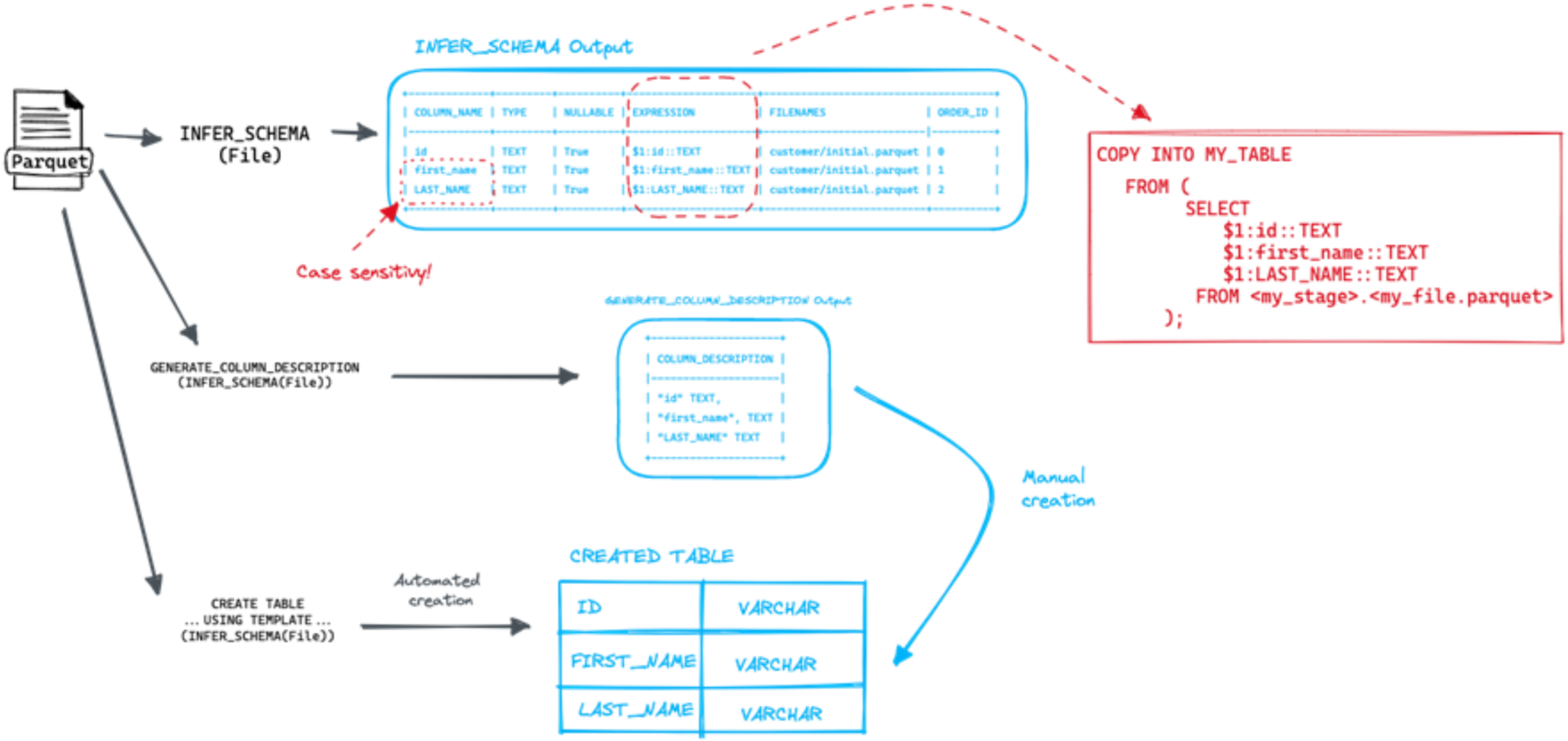
Binary format ingestion automation

- Built-in functions help you with schema detection, COPY command automation and landing table creation
- Faster and more reliable development by eliminating manual tasks which are very error prone
- It works for Parquet, Avro, and ORC
- We use SELECT statement as part of COPY command
  - **INFER_SCHEMA** - retrieves and returns the schema from staged files
  - **GENERATE_COLUMN_DESCRIPTION** - returns list of columns necessary to create a table / view
  - **CREATE TABLE … USING TEMPLATE** - automatically Creates table based on detected schema

Ingestion Automation schema



Ingestion Automation schema



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

How to store JSON data

Single 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

Accessing data in JSONs

Dot notation

```
Src:person.address.street
```

If key does not conform SQL identifier rules -> double quotes

```
"last name"
```

Bracket notation

```
Src:['person'].['address'].
['street']
```

Keys are always in quotes

Expanding the arrays

```
Src:person:phone[0]
```

Item position in brackets

Starts with 0

FLATTENING

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

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

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

```
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

```
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”]

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

Exercise

We are going to practice ingestion of the semi structured data into Snowflake and then using the features which can simplify the whole process significantly. All the instructions are available on GitHub where you can find:

- Exercise description: week1/week1_exercise_desc.pdf
- Source files: week1/source_files
- Solution: week1/week1_exercises.sql

O'REILLY®

SnowSQL Intro into CLI

Tomas Sobotik

Snow SQL basics

- Command line interface (CLI) to interact with your Snowflake account from your terminal
- Almost everything that can be done via UI can be done in CLI
    - Execute SQL queries
    - Manage snowflake objects
    - Loading data
- There are operations which could be done only in CLI (managing files in internal stages)
- Support for all major OS (Linux, Mac, Windows)

How to install SnowSQL

- Depends on OS version
- SF provides installer executable for each OS
  - Available on https://developers.snowflake.com/snowsql/
  - Just download and run
  - Homebrew support for Mac OS

```
Brew install –cask –snowflake-snowsql
```

  - Docs have guides for each OS

- Once installed it is immediately available for use in command line

```
snowsql –a <account_identifier> -u <user>
```

- Support for private-key pair, SSO and MFA authentication via connection parameters

Customization of SnowSQL

- Many customization options
  - Named connection profiles
  - Auto completion
  - Paging
  - Customizing the prompts
- `Everything is stored inside config file`
  - Mac and Linux: `~/.snowsql/`
  - Windows: `%USERPROFILE%\.snowsql\`

DEMO

SnowSQL vs SnowCLI

- SnowSQL – official tool
- SnowCLI – Snowflake developer CLI, newly also official tool to support developers
  - Originally was created under Snowflake Labs – open source initiative under Snowflake
  - Newly (April 2024) in preview
  - You can contribute to the development by submitting ideas
  - Some patterns might be incorporated into SnowSQL in the future
  - Support for Snowpark Python – UDFs, SP, Streamlit apps, GIT stages
  - Automatically uploads deployment artifacts into Snowflake stages
  - Only Python support for now
  - Installation via homebrew, pip
  - It requires Python >= 3.8

https://github.com/snowflakedb/snowflake-cli

Exercise

We are going to test the SNOW SQL. Detailed instructions are available on Github repo as always. This time you will find all the commands in:

`week1/week1_exercises.sql`

O'REILLY

Views

Different types of views in Snowflake

Tomas Sobotik

Basics around views

- View – named SQL query
- Result could be accessed like it would be a table
- Result are created at query run time – not stored anywhere
- Could be used everywhere where table could be used
- Disadvantage – slower processing in case of complex transformations
- Use cases
  - Data Combination
  - Data segregation
  - Data protection

Materialized views

- Named and **stored** SQL query
- Query result is pre-computed
- It behaves more like table
- Requires some storage and maintenance -> additional costs
- Could be used everywhere where table could be used
- Advantage – improving query performance for common & repeated queries
- Disadvantage – some limitations in Snowflake
- Use cases
  - Heavy aggregations
  - Calculations which are often accessed
  - Huge, rather static data as base table

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

Query is on external table

Cost

Serverless feature

Additional storage

Maintenance cost

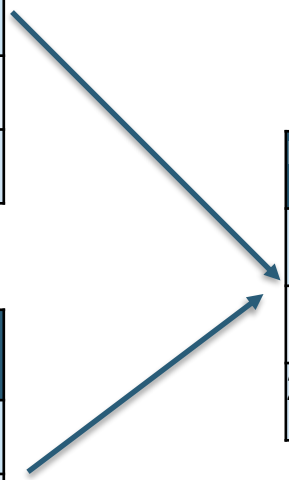
Materialized views limitations

Col 1	Col 2	Col N
	T	
	F	
	F	

Partition 14

Col 1	Col 2	Col N
	F	
	T	
	T	

Partition 28

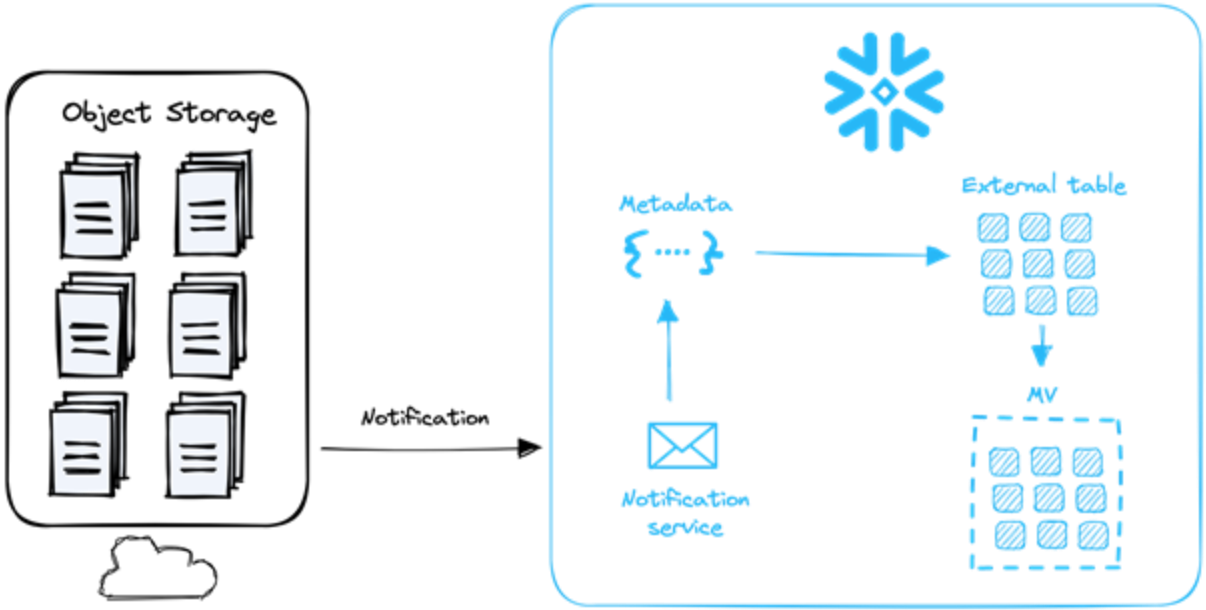


Materialized view

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

MV on external Table



+ faster queries
+ lower cost
+ fewer compute credits

Secure views

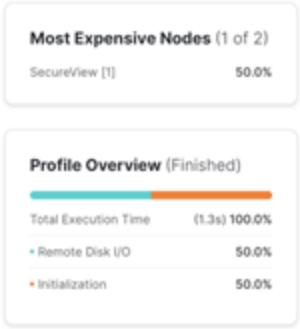
- Special type available for both views and MVs
- Protect underlying data from unintentional exposing
  - Some Internal optimization for views require access to the underlying data in the base tables
  - Secure views omit those optimization
- Standard view DDL is visible to user in various commands or interfaces -> not for secure views
  - Because of security or privacy reasons you might not wish to expose it

- If view/MV needs to be shared it must be secured
- It should be used only in case of security concern, not like a replacement of ordinary view

Secure views

- How to find out if view is secure?
  - INFORMATION_SCHEMA/ACCOUNT_USAGE views
  - SHOW VIEWS

- Secure views and query profile
  - Internals are not exposed
  - That applies also for the views owner



Unintentional data exposure by a non-secure view?

```
SELECT *
    FROM widgets_view
    WHERE 1/iff(color = 'Purple', 0, 1) = 1;
```

View or Materialized view ?

View	Secure view	Materialized View
The result of the view change often	Data protection/security concern	The result of the view **don't** change often
The result are **not** used often (materialization of the pipeline)	Data should be shared via data sharing	The result are used often
Not resource intensive query		Query consume a lot of resource
It is not costly to rerun it		Speed up processing of data stored in external table

Poll

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

2. How is Materialized view refreshed - mark all possible answers
   a) By REFRESH_MV() function
   b) IN UI - refresh button
   c) Automatically

4. What can be used in MV definition. Select all correct answers
   a) JOIN
   b) GROUP BY
   c) SUM
   d) ORDER BY
   e) HAVING
   f) Another view

3. Data engineer has been asked to create a view on top of EMPLOYEE base table and combine it with data from other tables like DEPARTMENTS and FINANCE. What view should he create
   a) view
   b) secure view
   c) materialized view

Exercise

This part is dedicated to practising working with views and their differences. We will try to create standard view, secure view and materializied view. Thanks to this exercise we will be able to describe their limitations. Please follow the code available in `week1/week1_exercises.sql`

O'REILLY

Continuous Data Pipelines

Tomas Sobotik

Batch x Continuous

Batch processing

Ingestion at given time

Latency in hours or days

User specified WH

Always single transaction

Load Metadata stored for 64 days

Continuous processing

Ingestion triggered by event

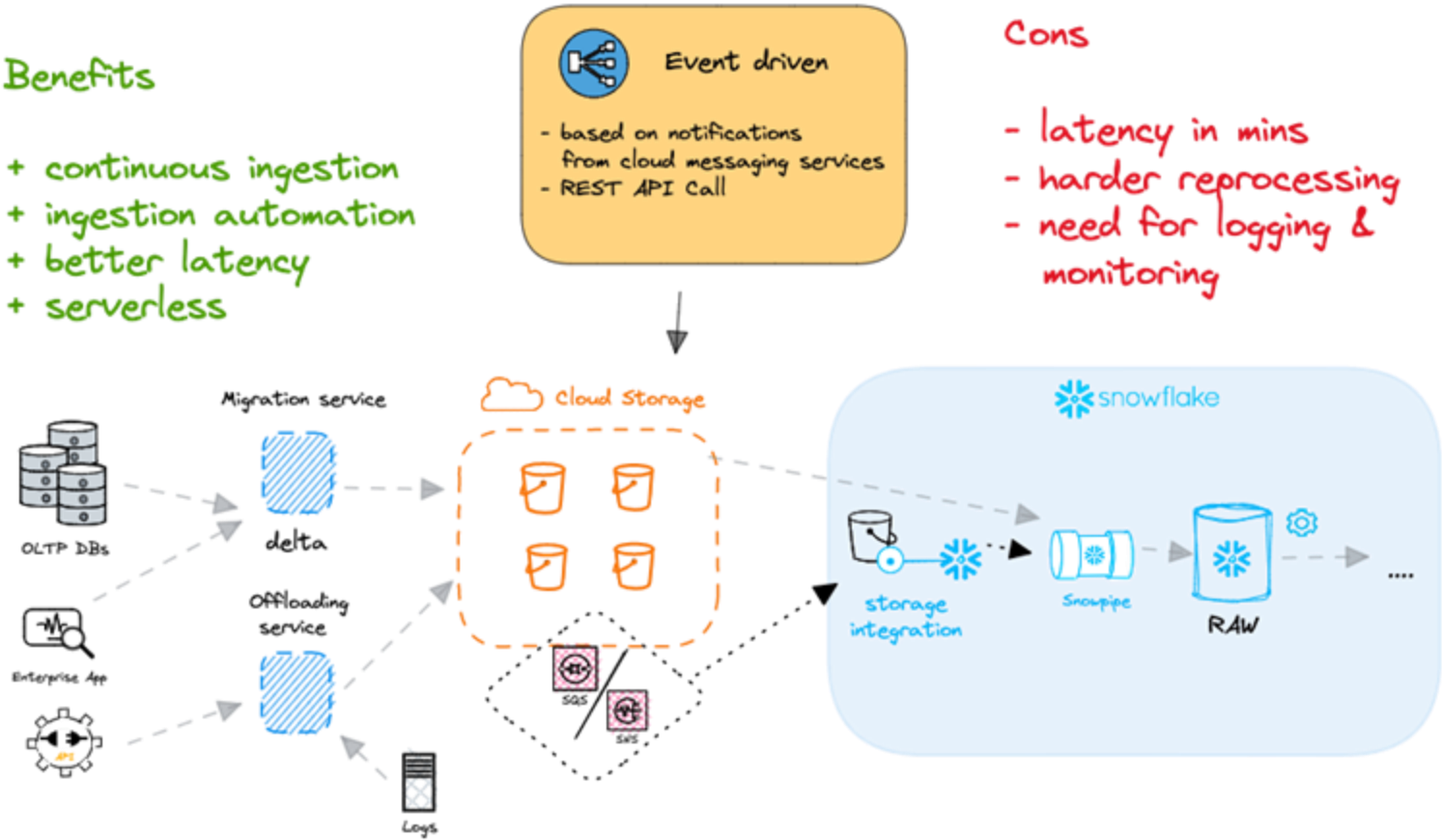
Latency in minutes

Serverless – Snowflake provide WH

Single or multiple transactions based on the number and size of the rows

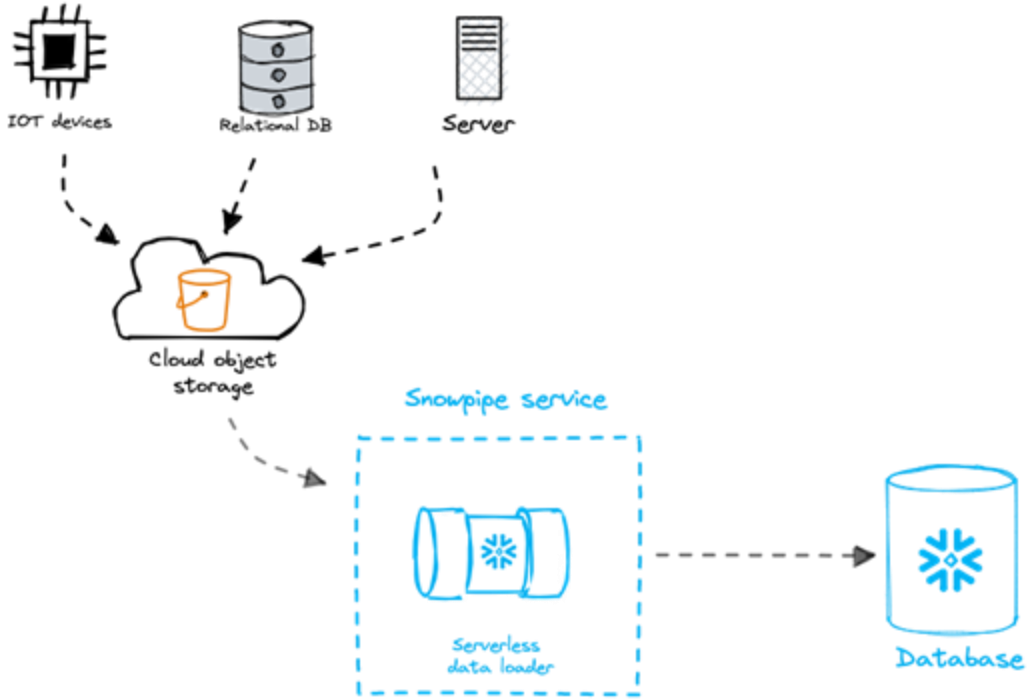
Load Metadata stored for 14 days

Continuous processing

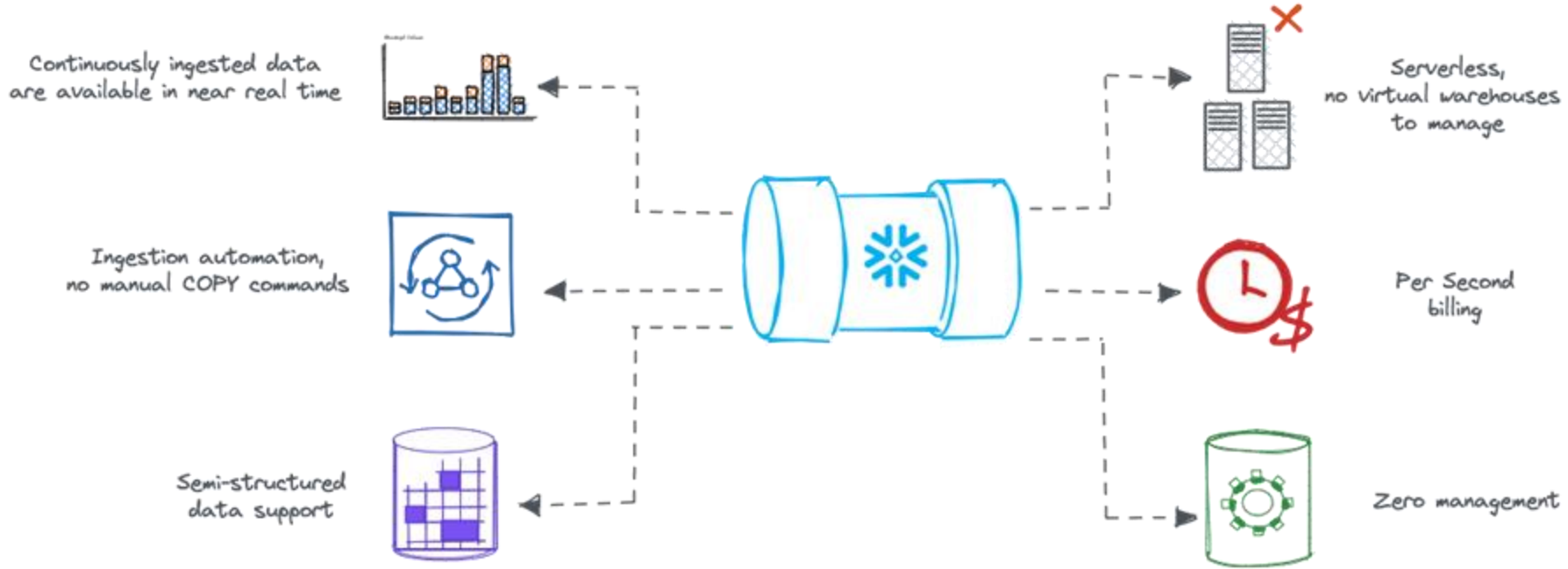


Snowpipes

- 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
- Use cases
  - Automated data ingestion
  - IOT data stream import
  - Data Lake files import



Faster data ingestion with Snowpipes



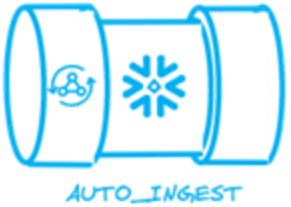
Snowpipes - how it works

- Pre-defined object that specifies the COPY command for data load
- Contains instructions how and where to load and convert data into records in the target table
- Supports all data types (including semistructured)

Manually call Snowpipe REST API endpoint

Pass a list of stage files in the stage location

Works with internal and external stages



Receive notifications from cloud provider when files arrive

Notification trigger the processing

Only external stages are supported

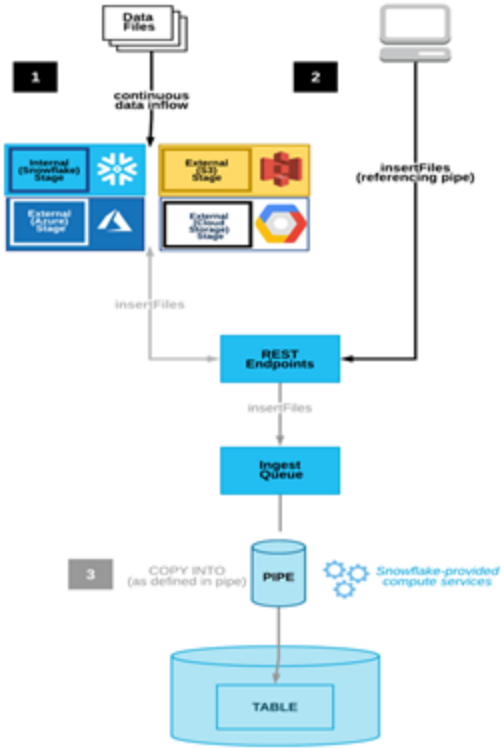
Snowpipe cross cloud support

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

Snowpipe REST endpoints to load data

- Uses key-based authentication when calling REST endpoints
- Data apps approach
  - 1. Data files copied to an internal or external stage
  - 2. Client calls `insertFiles` endpoint with list of files to ingest and a defined pipe
  - 3. Snowflake provided WH loads files to table

- Use client app / AWS Lambda to call the REST Endpoint



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}`

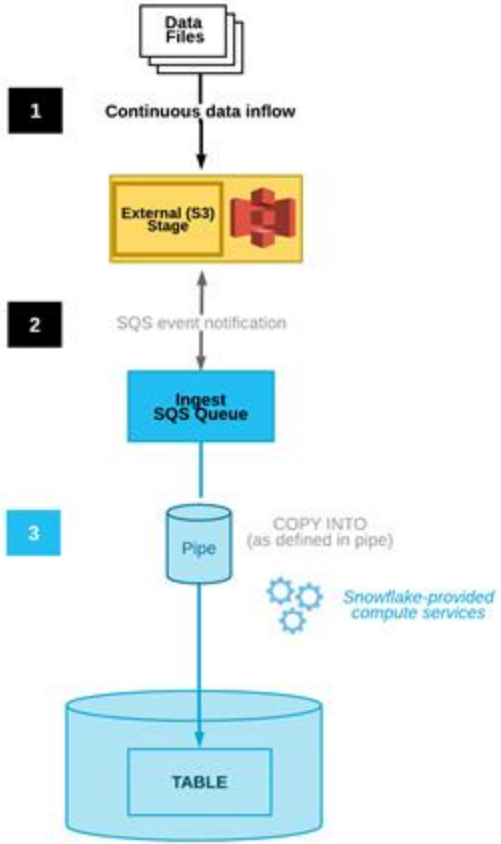
- `insertReport`
  - `GET`
  - Retrieves a report of files submitted via `insertFiles`
  - The 10000 most recent events
  - Events are retained for max 10 mins

- `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

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

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



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.

Configure Access permissions for the S3 bucket

- You need to grant following permissions to Snowflake
  - s3:GetBucketLocation
  - s3:GetObject
  - s3:GetObjectVersion
  - s3:ListBucket
- Best practise: Create an IAM policy for Snowflake access to the S3 Bucket
  - Assign the policy to IAM role

Storage integration object and integration between Snowflake and AWS

1. Create Storage Integration object
    - Use IAM role created in previous step
2. Retrieve AWS IAM user for your Snowflake account
    - `DESC Integration my_integration`
    - Record the STORAGE_AWS_IAM_USER_ARN and STORAGE_AWS_EXTERNAL_ID
3. Grant the IAM user Permissions to Access Bucket Objects
    - Configure a Trust relationship in IAM role we created earlier

Create a S3 Event Notification to Automate Snowpipe

Let’s use SQS for that

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

Poll

1. How big warehouse is recommended to assign to snowpipe

a) MEDIUM
b) X-LARGE
c) no warehouse should be assigned

2. What are snowpipe types? Select all of them

A) Notification based
B) REST endpoint
C) Manually triggered
D) AUTO INGEST

3. What billing model does snowpipe use?

a) Per file billing
b) Per load billing
c) Per second billing

4. What objects are used by snowpipe? Select all of them

a) Virtual warehouse
b) COPY command
c) FILE_FORMAT
d) Stream
e) Stage

Snowpipe demo & exercise

Exercise

We are going to practise the continuous data ingestion into Snowflake in this exercise. We will use snowpipe feature for this. It again requires configuration on the Cloud provider which is similar to settings which we have done for the batch data ingestion. There are required the bucket access permission and IAM role in order to access Snowflake. Now we also need some kind of notification to be sent to snowpipe when a new file will be landed in cloud storage.

We will do all the needed setup on both ends (Snowflake & AWS) in this exercise and then we will try to ingest some data through snowpipe.

In the next session we will be talking more how to automate it further and automatically process newly loaded data via snowpipe.

Detailed description of this exercise is available on github:

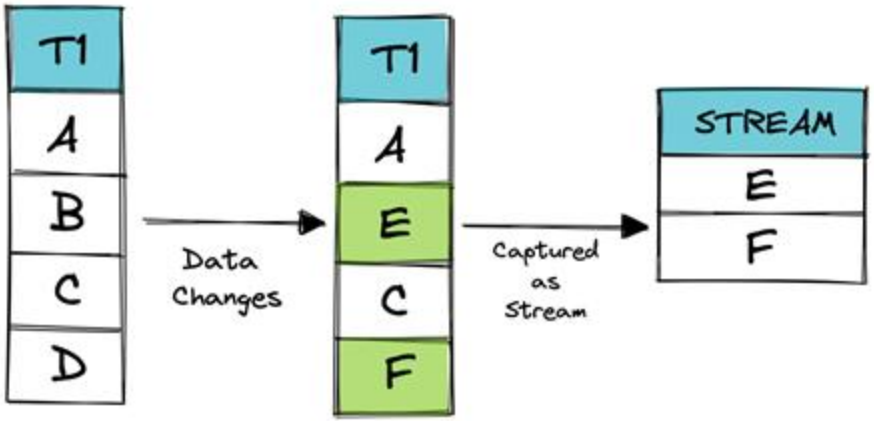
sf_data_engineering_bootcamp/week1/exercise_desc.pdf

O'REILLY

Streams and Tasks

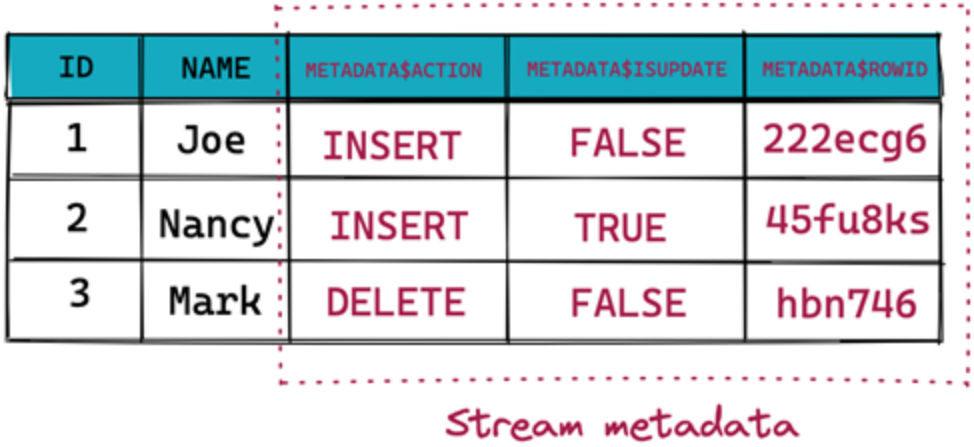
Tomas Sobotik

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)

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
- Multiple consumers? -> multiple streams

Staleness

- Retention period for table < 14 days -> Snowflake temporarily extend the period up to a maximum of 14 days by default
- Maximum number of extension days is controlled by MAX_DATA_EXTENSION_TIME_IN_DAYS

DATA_RETENTION_TIME_IN_DAYS	MAX_DATA_EXTENSION_TIME_IN_DAYS	Consume Stream in X Days
14	0	14
1	14	14
0	90	90

Streams on Views

- Views, secure views but not MVs
- Views in Data Sharing!
- Requirements
  - All of underlaying tables must be native tables
  - Not yet supported operations: GROUP BY, QUALIFY, LIMIT, correlated subqueries, subqueries not in FROM clause
  - Only native scalar functions
  - Enabled change tracking on underlaying tables
- Be careful about Join Behavior
  - Changes that have occurred on the left table since the stream offset are being joined with the right table, changes on the right table since the stream offset are being joined with the left table, and changes on both tables since the stream offset are being joined with each other.
  - https://docs.snowflake.com/en/user-guide/streams-intro#join-results-behavior

CHANGES – alternative to Streams

- On table, views – not for directory and external tables
- Enables querying the change tracking metadata without having to create a stream with explicit transactional offset.
- Multiple queries can retrieve the change tracking metadata between different start and endpoints
- You must enable change tracking on source object or have as stream
  - `ALTER TABLE t1 SET CHANGE_TRACKING = TRUE`

- Parameters
  - `INFORMATION => DEFAULT | APPEND_ONLY`
  - `OFFSET =>` difference in seconds from the current time
  - `STATEMENT => <id>` – query ID as a reference point for Time Travel
    - DML, SELECT, BEGIN, COMMIT transaction
  - `STREAM => <stream name>`, stream offset is used as the AT point in time

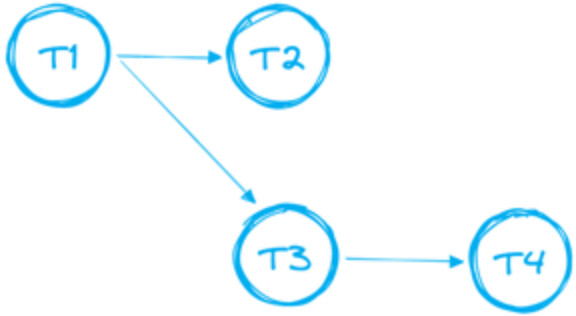
Exercise 1 - Practise the streams

Let's follow-up on previous session dedicated to snowpipes. Now we try to automatically process the data which have been ingested by snowpipe. We need two additional database objects for it - streams and task.

In this first exercise we will create stream object on top of our landing table from previous session dedicated to snowpipe and we will be slowly building up the complete integration for continuous integration. Detailed instructions are again available on github.

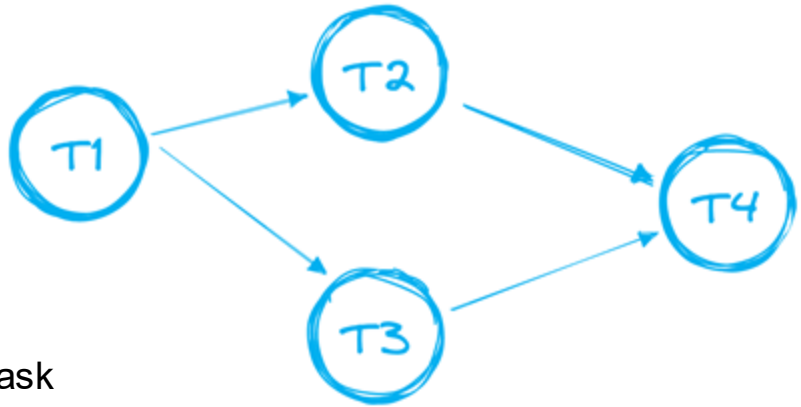
Tasks

- 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



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
  - New UI

Task creation

- Key parameters
  - WAREHOUSE
  - SCHEDULE
  - WHEN
  - AFTER
  - SUSPEND_TASK_AFTER_NUM_FAILURES

```
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');
```

- Newly created task is suspended
  - Needs to be resumed by ALTER command

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');
```

Serverless Task creation

- Key parameters
  - ~~WAREHOUSE~~
  - SCHEDULE
  - WHEN
  - AFTER
  - SUSPEND_TASK_AFTER_NUM_FAILURES

```
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');
```

- Newly created task is suspended
  - Needs to be resumed by ALTER command
- Simpler and more cost efficient
- Existing task can be modified to be serverless
- It uses last few executions to decide WH size

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');
```

Exercise 2 - task creation

And last part for our continuous data pipeline. We need to somehow process the loaded data which are now available in the stream. For that we need task which will be running only when the stream has some fresh data. Thanks to that we will achieve complete integration, starting by uploading fresh data into external stage. Such integration will be incremental, we will be always processing only the freshly loaded data.