Snowflake cloud data-warehouse

“Simplicity is the ultimate sophistication”

Leonardo da Vinci

Day-6

User Defined Functions

UDFs – Let you extend the system to perform operations that are not available via system-defined functions.

Language Supported:

- SQL
- JavaScript
- Java

SQL: Evaluates an arbitrary SQL expression and returns either scalar or tabular results.

JavaScript: Lets you use the JavaScript language to manipulate data and return either scalar or tabular results.

Java: Lets you use the Java language to manipulate data and return scalar results.

Support for Secure UDF (like secure view)

Avoid naming conflict with System defined functions

Overloading

Snowflake User-Defined [Table] Functions


```
CREATE [ OR REPLACE ] [ SECURE ] FUNCTION <name> ( [ <arg_name> <arg_data_type> ] [ , ... ] )
  RETURNS { <result_data_type> | TABLE ( <col_name> <col_data_type> [ , ... ] ) }
  [ [ NOT ] NULL ]
  [ LANGUAGE JAVASCRIPT ]
  [ COMMENT = '<string_literal>' ]
  AS '<function_definition>'
```

Snowflake User-Defined [Table] Functions

UDF-UDTF Demo

Snowflake Programming Objects

Stored Procedure

Allows procedural logic (branching and looping) and error handling, which straight SQL does not support

Enables dynamically creating a SQL statement and executing it

Allows writing code that executes with the privileges of the role that owns the procedure, rather than with the privileges of the role used to run the procedure

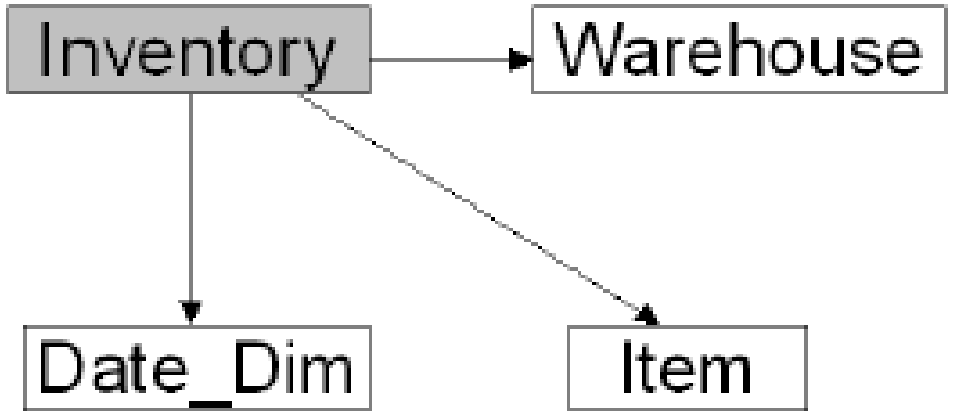
Allows delegating the power to perform specified operations to users who otherwise could not do so

Snowflake Stored Procedure

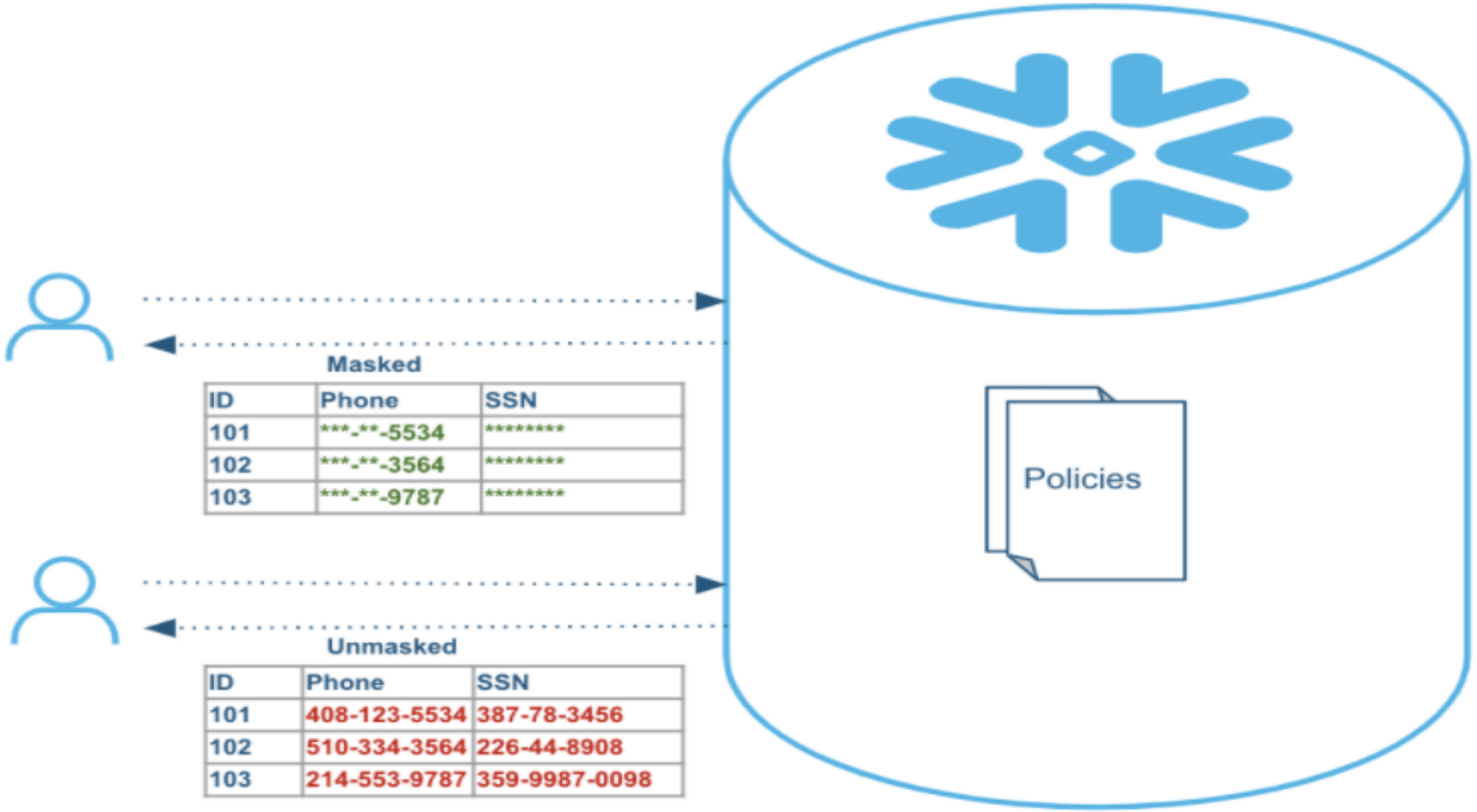

```
CREATE [ OR REPLACE ] PROCEDURE <name> ( [ <arg_name> <arg_data_type> ] [ , ... ] )
 RETURNS <result_data_type> [ NOT NULL ]
 LANGUAGE JAVASCRIPT
 [ { CALLED ON NULL INPUT | { RETURNS NULL ON NULL INPUT | STRICT } } ]
 [ COMMENT = '<string_literal>' ]
 [ EXECUTE AS { CALLER | OWNER } ]
 AS '<procedure_definition>'
```

Snowflake Stored Procedure

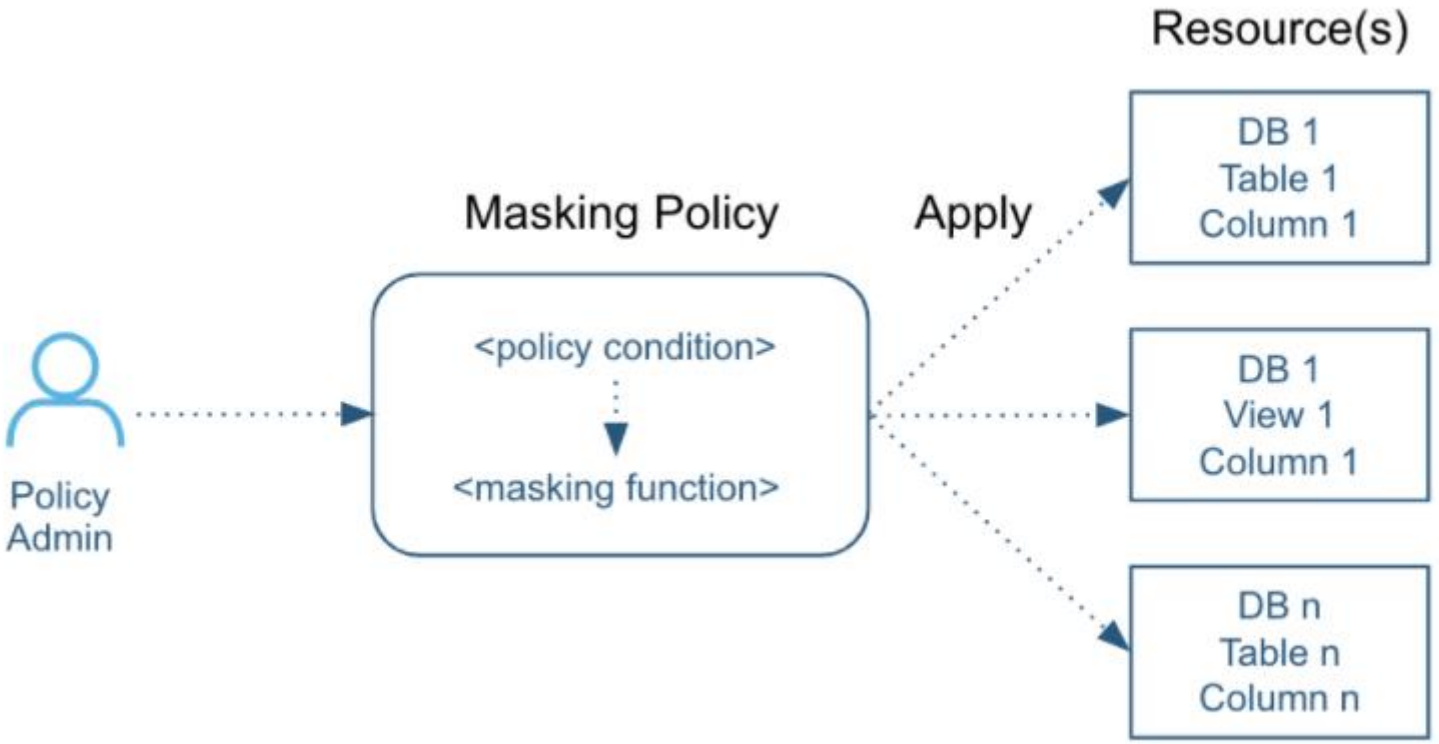
Stored Procedure Demo



Snowflake Dynamic Data Masking



Snowflake Dynamic Data Masking



Context Functions

current_role

invoker_role

Snowflake Dynamic Data Masking

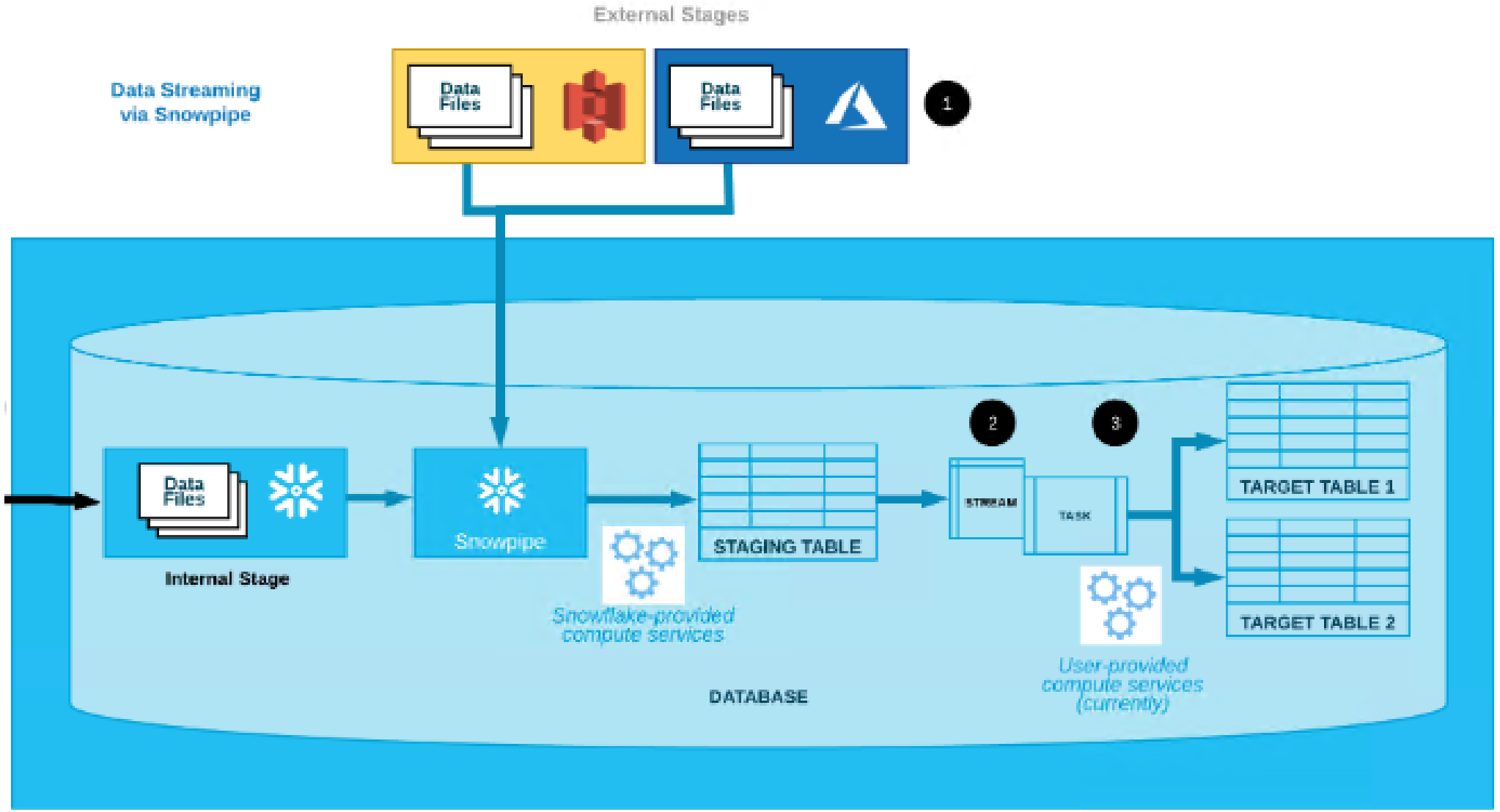
Points to remember

- ❑ Available on Enterprise Edition or higher
- ❑ Selectively hide, mask or encrypt sensitive data
- ❑ Policy declaration is reusable
- ❑ Masking policies are schema-level objects
- ❑ Applicable on column level
- ❑ Can be applied on semi-structured data
- ❑ Can not be applied on Virtual Columns
- ❑ Can not create materialized view from masked table columns
- ❑ Restriction on result cache re-use

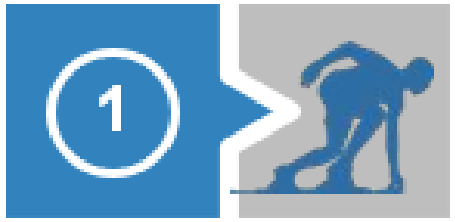
Snowflake Dynamic Data Masking

Dynamic Data Masking Demo

Continuous Data Pipeline



Continuous Data Pipeline



Prepare your files

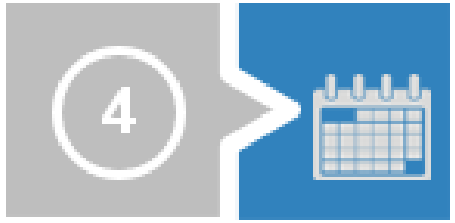
If required pre-process the files into a format that is optimal for loading

Stage the data

Make snowflake aware of the data. Internal or external staging area

Create PIPE and execute COPY command

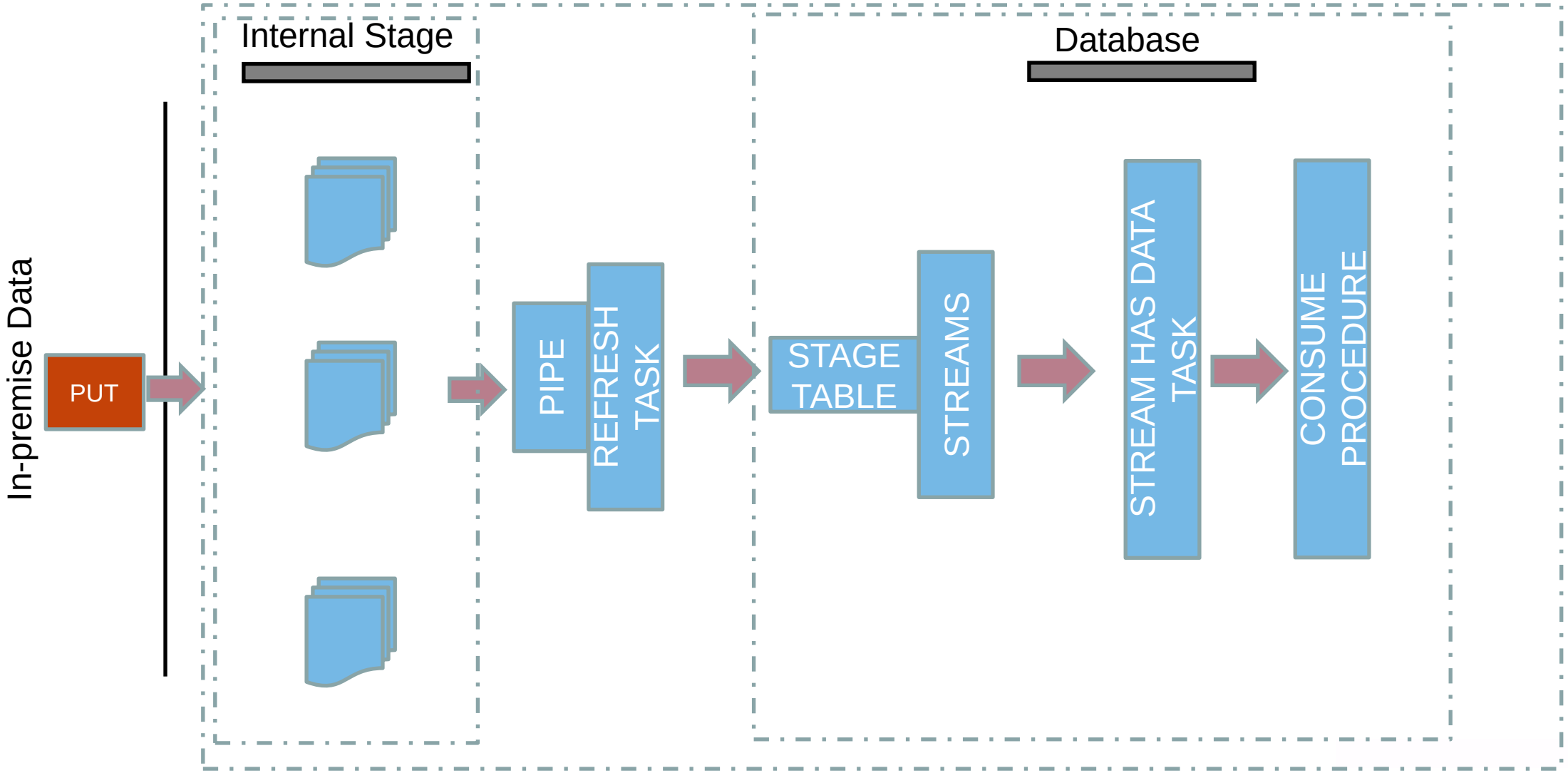
COPY the data into the table



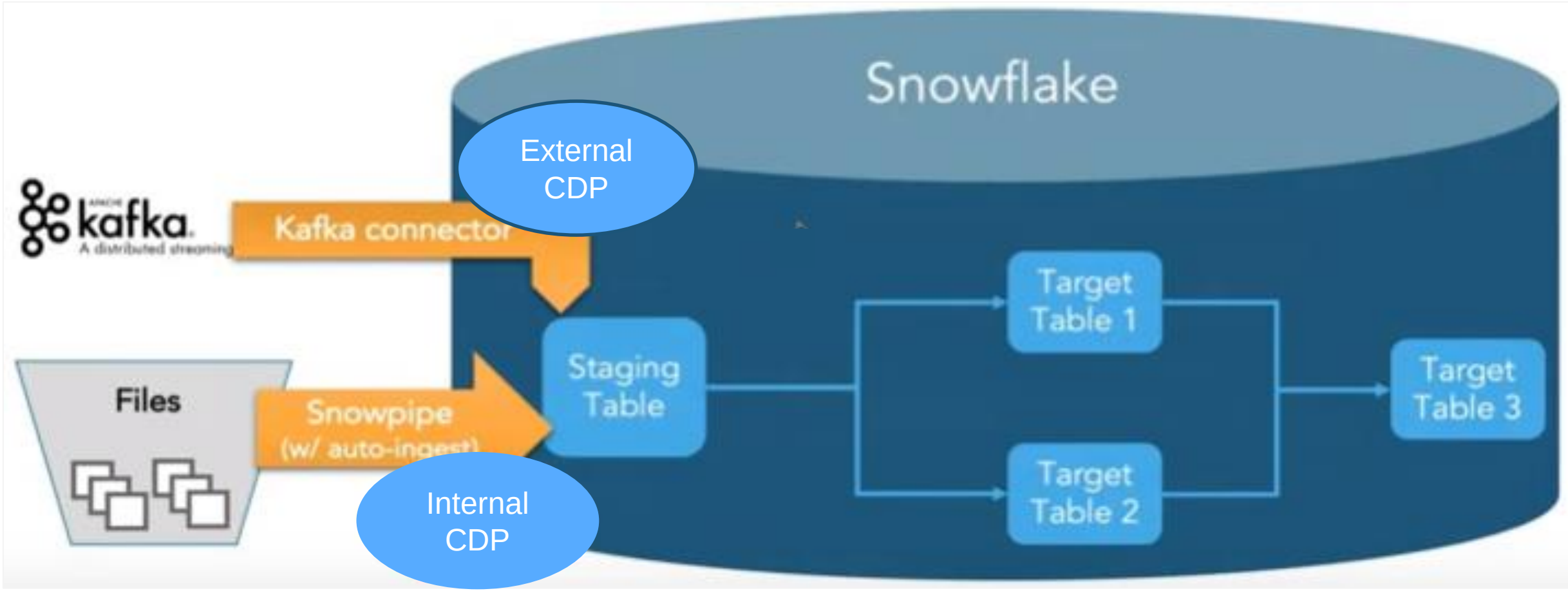
Managing regular loads by triggering continuous load

Organize files & schedule your loads

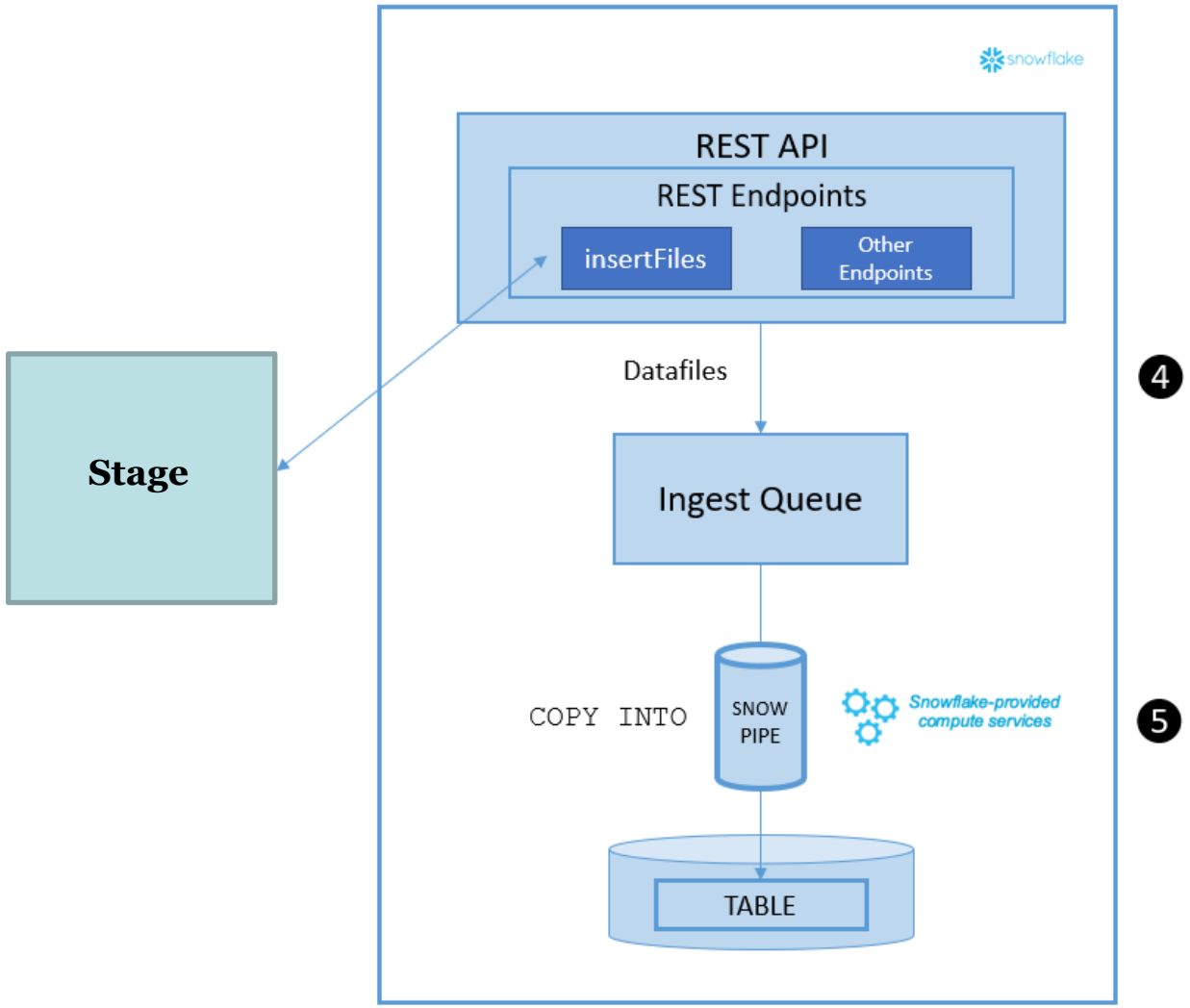
Continuous Data Pipeline



Snowflake Data Pipeline



Snowflake Data Pipeline



Points to consider while using Snowpipe

- Uses Snowflake-supplied compute resources.
- Billed according to the compute resources used in the Snowpipe warehouse
- Snowpipe is designed to load new data typically within a minute after a file notification
- Extra management charges for internal load queue is included in the utilization costs
- Recommend size of data files roughly **100 MB to 250 MB** in size compressed
- Does not guarantee that files are loaded in the same order they are staged
- Information Schema's COPY History metadata retained for 14 days
- COPY statement for column reordering, column omission, and casts is supported
- Filtering using a WHERE clause is not supported
- PURGE parameter not supported in COPY Statement

Snowflake Data Pipeline


```
CREATE [ OR REPLACE ] PIPE [ IF NOT EXISTS ] <name>
  [ AUTO_INGEST = [ TRUE | FALSE ] ]
  [ AWS_SNS_TOPIC = <string> ]
  [ INTEGRATION = '<string>' ]
  [ COMMENT = '<string_literal>' ]
  AS <copy_statement>
```

<copy_statement>

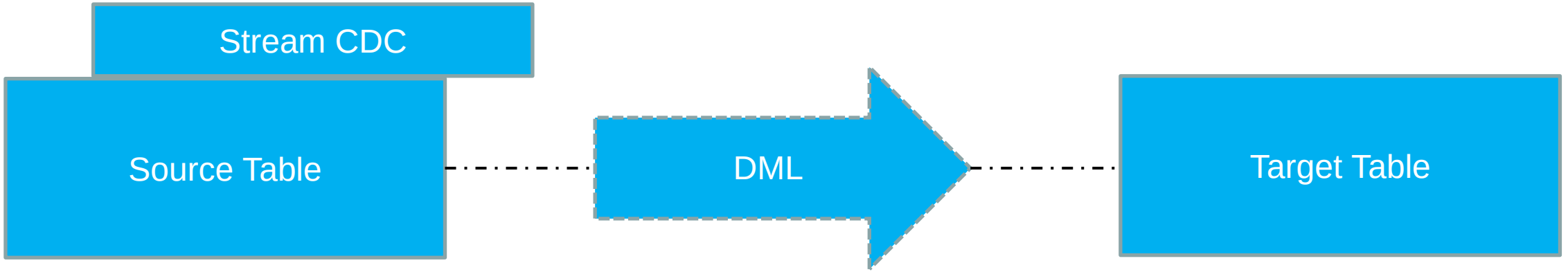
Following parameters not supported

```
FILES = ( 'file_name1' [ , 'file_name2', ... ] )
ON_ERROR = ABORT_STATEMENT
SIZE_LIMIT = num
PURGE = TRUE | FALSE
FORCE = TRUE | FALSE
VALIDATION_MODE = <RETURN ... ERRORS>
```

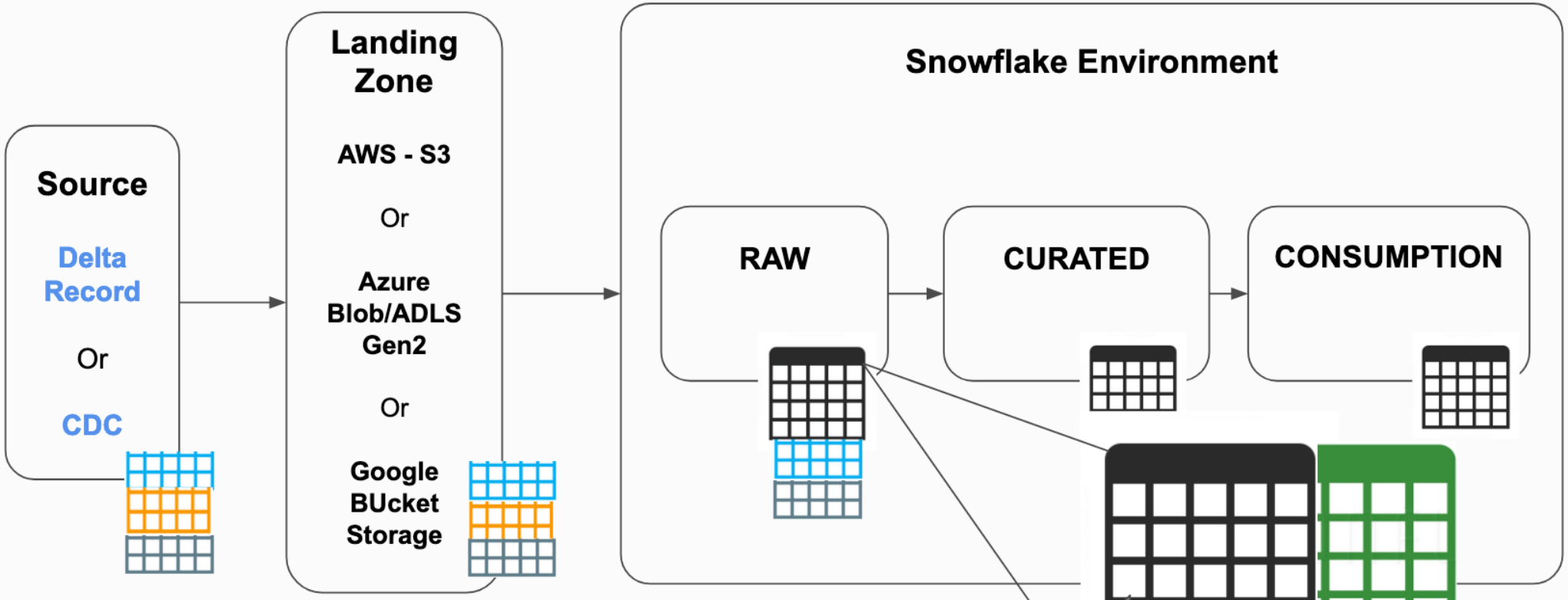
Snowflake Data Pipeline

SNOWPIPE Demo

Snowflake Change Data Capture (CDC)



Snowflake Change Data Capture (CDC)



When a **stream** is created, the existing table gets 3 additional columns named as

1. METADATA$ACTION
2. METADATA$ISUDATE
3. METADATA$ROWID

The stream object only contains the offset value

Snowflake Change Data Capture (CDC)

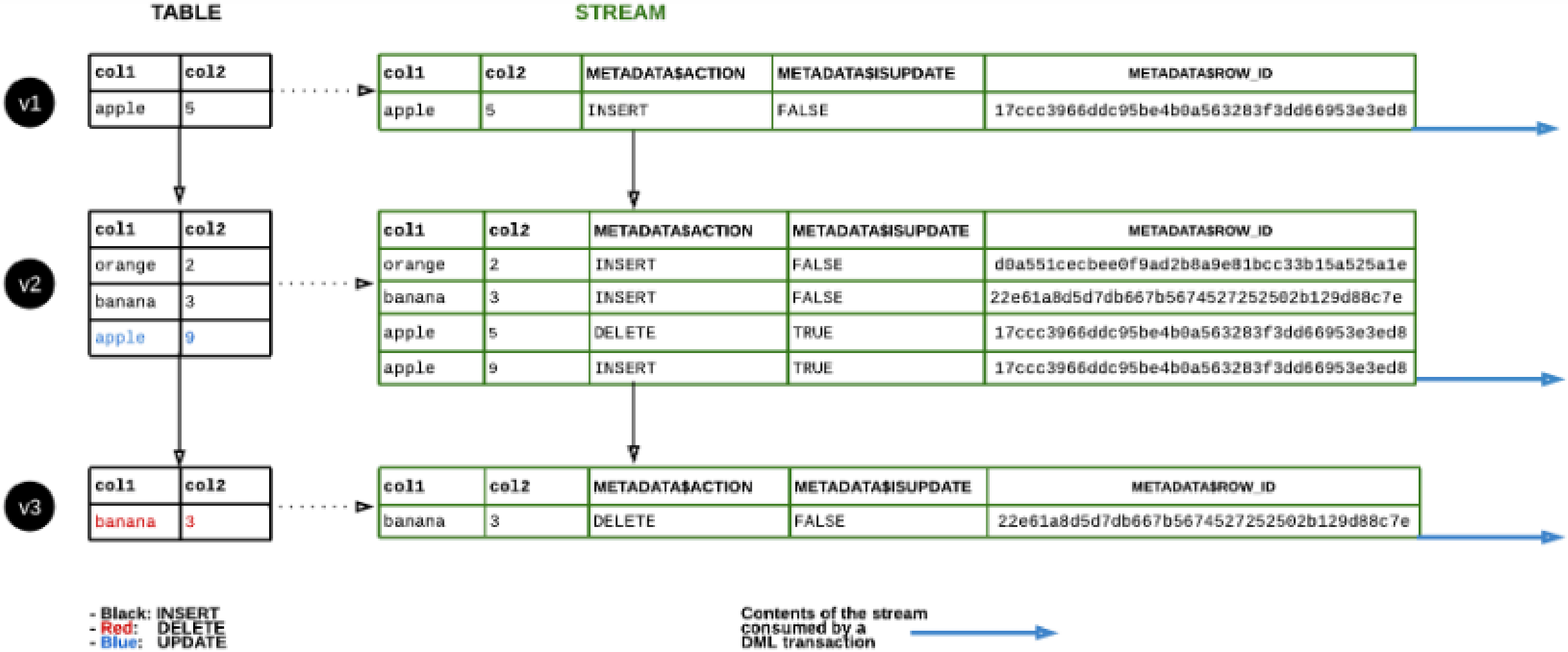


Table Streams

- Provide a set of changes made to the underlying table since last time
- Consumed in a DML statement- Designed for transforms (“T” in ELT)
- Consuming advances on commit like a forward cursor
- Multiple streams possible on a given tables
- Table stream does not hold any data
- Streams cannot track changes in materialized views
- Types of Streams:
  - Standard
  - Append-only
  - Insert-only- External Tables
- Change retention determined 14 days or MAX_DATA_EXTENSION_TIME_IN_DAYS

Snowflake Change Data Capture (CDC)

Streams Demo