Stream:

Note that a stream itself does ***not*** contain any table data. A stream only stores an offset for the source object and returns CDC records by leveraging the versioning history for the source object.

One example of a consumer of CDC records is a [data pipeline](https://docs.snowflake.com/en/user-guide/data-pipelines.html), in which only the data in staging tables that has changed since the last extraction is transformed and copied into other tables.

This stream type performs a join on inserted and deleted rows in the change set to provide the row level delta. As a net effect, for example, a row that is inserted and then deleted between two transactional points of time in a table is removed in the delta (i.e. is not returned when the stream is queried).

A stream becomes stale when its offset is outside of the data retention period for its source table (or the underlying tables for a source view). When a stream becomes stale, the historical data for the source table is no longer accessible, including any unconsumed change records.

To prevent a stream from becoming stale, consume the stream records within a transaction during the retention period for the table

**CREATE** **STREAM** mystream **ON** **TABLE** mytable **BEFORE** **(TIMESTAMP** **=>** TO\_TIMESTAMP**(**40**\***365**\***86400**));**

**CREATE** **STREAM** mystream **ON** **TABLE** mytable **AT** **(TIMESTAMP** **=>** TO\_TIMESTAMP\_TZ**(**'02/02/2019 01:02:03'**,** 'mm/dd/yyyy hh24:mi:ss'**));**

**Differences Between Standard and Append-only Streams**

The following example shows the differences in behavior between standard (delta) and append-only streams:

-- Create a source table.

**create** **or** **replace** **table** **t(**id **int,** **name** **string);**

-- Create a standard stream on the source table.

**create** **or** **replace** **stream** delta\_s **on** **table** **t;**

-- Create an append-only stream on the source table.

**create** **or** **replace** **stream** append\_only\_s **on** **table** **t** append\_only**=true;**

-- Insert 3 rows into the source table.

**insert** **into** **t** **values** **(**0**,** 'charlie brown'**);**

**insert** **into** **t** **values** **(**1**,** 'lucy'**);**

**insert** **into** **t** **values** **(**2**,** 'linus'**);**

-- Delete 1 of the 3 rows.

**delete** **from** **t** **where** id **=** '0'**;**

-- The standard stream removes the deleted row.

**select** **\*** **from** delta\_s **order** **by** id**;**

**+**----+-------+-----------------+-------------------+------------------------------------------+

| ID | NAME | METADATA$ACTION | METADATA$ISUPDATE | METADATA$ROW\_ID |

|----+-------+-----------------+-------------------+------------------------------------------|

| 1 | lucy | INSERT | False | 7b12c9ee7af9245497a27ac4909e4aa97f126b50 |

| 2 | linus | INSERT | False | 461cd468d8cc2b0bd11e1e3c0d5f1133ac763d39 |

**+**----+-------+-----------------+-------------------+------------------------------------------+

-- The append-only stream does not remove the deleted row.

**select** **\*** **from** append\_only\_s **order** **by** id**;**

**+**----+---------------+-----------------+-------------------+------------------------------------------+

| ID | NAME | METADATA$ACTION | METADATA$ISUPDATE | METADATA$ROW\_ID |

|----+---------------+-----------------+-------------------+------------------------------------------|

| 0 | charlie brown | INSERT | False | e83abf629af50ccf94d1e78c547bfd8079e68d00 |

| 1 | lucy | INSERT | False | 7b12c9ee7af9245497a27ac4909e4aa97f126b50 |

| 2 | linus | INSERT | False | 461cd468d8cc2b0bd11e1e3c0d5f1133ac763d39 |

**+**----+---------------+-----------------+-------------------+------------------------------------------+

-- Create a table to store the change data capture records in each of the streams.

**create** **or** **replace** **table** t2**(**id **int,** **name** **string,** stream\_type **string** **default** **NULL);**

-- Insert the records from the streams into the new table, advancing the offset of each stream.

**insert** **into** t2**(**id**,name,**stream\_type**)** **select** id**,** **name,** 'delta stream' **from** delta\_s**;**

**insert** **into** t2**(**id**,name,**stream\_type**)** **select** id**,** **name,** 'append\_only stream' **from** append\_only\_s**;**

-- Update a row in the source table.

**update** **t** **set** **name** **=** 'sally' **where** **name** **=** 'linus'**;**

-- The standard stream records the update operation.

**select** **\*** **from** delta\_s **order** **by** id**;**

# Introduction to Tasks

A task can execute any one of the following types of SQL code:

* Single SQL statement
* Call to a stored procedure
* Procedural logic using [Snowflake Scripting](https://docs.snowflake.com/en/developer-guide/snowflake-scripting/index.html)

## Compute Resources

Tasks require compute resources to execute SQL code. Either of the following compute models can be chosen for individual tasks:

* Snowflake-managed (i.e. serverless compute model)
* User-managed (i.e. virtual warehouse)

### Serverless Tasks

The serverless compute model for tasks enables you to rely on compute resources managed by Snowflake instead of user-managed virtual warehouses. The compute resources are automatically resized and scaled up or down by Snowflake as required for each workload. Snowflake determines the ideal size of the compute resources for a given run based on a dynamic analysis of statistics for the most recent previous runs of the same task. The maximum size for a serverless task run is equivalent to an XXLARGE warehouse. Multiple workloads in your account share a common set of compute resources.

**+**----+-------+-----------------+-------------------+------------------------------------------+

| ID | NAME | METADATA$ACTION | METADATA$ISUPDATE | METADATA$ROW\_ID |

|----+-------+-----------------+-------------------+------------------------------------------|

| 2 | sally | INSERT | True | 461cd468d8cc2b0bd11e1e3c0d5f1133ac763d39 |

| 2 | linus | DELETE | True | 461cd468d8cc2b0bd11e1e3c0d5f1133ac763d3\

**+**----+-------+-----------------+-------------------+------------------------------------------+

-- The append-only stream does not record the update operation.

**select** **\*** **from** append\_only\_s **order** **by** id**;**

**+**----+------+-----------------+-------------------+-----------------+

| ID | NAME | METADATA$ACTION | METADATA$ISUPDATE | METADATA$ROW\_ID |

|----+------+-----------------+-------------------+-----------------|

**+**----+------+-----------------+-------------------+-----------------+

A task can execute any one of the following types of SQL code:

* Single SQL statement
* Call to a stored procedure
* Procedural logic using [Snowflake Scripting](https://docs.snowflake.com/en/developer-guide/snowflake-scripting/index.html)

Computer resources:

* Snowflake-managed (i.e. serverless compute model)
* User-managed (i.e. virtual warehouse)

2. choosing a warehouse

Choosing a compute model

Task scheduling

Serveeless : Recommended when you cannot fully utilize a warehouse because too few tasks run concurrently or they run to completion quickly (in less than 1 minute).

**User-managed Tasks :** Recommended when you can fully utilize a single warehouse by scheduling multiple concurrent tasks to take advantage of available compute resources.

Also recommended for spiky or unpredictable loads on compute resources. [Multi-cluster warehouses](https://docs.snowflake.com/en/user-guide/warehouses-multicluster.html) with [auto-suspend and auto-resume](https://docs.snowflake.com/en/user-guide/warehouses-overview.html#label-auto-suspension-and-auto-resumption) enabled could help moderate your credit consumption.

If a run of a standalone task or scheduled DAG exceeds nearly all of this interval, Snowflake increases the size of the compute resources (to a maximum of the equivalent of a 2X-Large warehouse).

* Standalone tasks are automatically suspended after the specified number of consecutive task runs either fail or time out.

Example:

**CREATE** **TASK** t1

**SCHEDULE** **=** '60 MINUTE'

TIMESTAMP\_INPUT\_FORMAT **=** 'YYYY-MM-DD HH24'

USER\_TASK\_MANAGED\_INITIAL\_WAREHOUSE\_SIZE **=** 'XSMALL'

**AS**

**INSERT** **INTO** mytable**(ts)** **VALUES(**CURRENT\_TIMESTAMP**);**

Create a task that inserts the current timestamp into a table every 5 minutes:

**CREATE** **TASK** mytask\_minute

**WAREHOUSE** **=** mywh

**SCHEDULE** **=** '5 MINUTE'

**AS**

**INSERT** **INTO** mytable**(ts)** **VALUES(**CURRENT\_TIMESTAMP**);**

**CREATE** **TASK** mytask1

**WAREHOUSE** **=** mywh

**SCHEDULE** **=** '5 minute'

**WHEN**

SYSTEM$STREAM\_HAS\_DATA**(**'MYSTREAM'**)**

**AS**

**INSERT** **INTO** mytable1**(**id**,name)** **SELECT** id**,** **name** **FROM** mystream **WHERE** METADATA$ACTION **=** 'INSERT'**;**

This example adds child task task5 to the DAG and specifies task2, task3, and task4 as predecessor tasks:

-- Create task5 and specify task2, task3, task4 as predecessors tasks.

-- The new task is a serverless task that inserts the current timestamp into a table column.

**CREATE** **TASK** task5

**AFTER** task2**,** task3**,** task4

**AS**

**INSERT** **INTO** t1**(ts)** **VALUES(**CURRENT\_TIMESTAMP**);**

-- Create a stored procedure that unloads data from a table

-- The COPY statement in the stored procedure unloads data to files in a path identified by epoch time (using the Date.now() method)

**create** **or** **replace** **procedure** my\_unload\_sp**()**

**returns** **string** **not** **null**

**language** **javascript**

**as**

**$$**

var my\_sql\_command **=** ""

var my\_sql\_command **=** my\_sql\_command**.**concat**(**"copy into @mystage"**,**"/"**,Date.**now**(),**"/"**,**" from mytable overwrite=true;"**);**

var statement1 **=** snowflake**.**createStatement**(** **{**sqlText: my\_sql\_command**}** **);**

var result\_set1 **=** statement1**.execute();**

**return** my\_sql\_command**;** **//** **Statement** returned **for** **info/**debug purposes

**$$;**

-- Create a task that calls the stored procedure every hour

**create** **task** my\_copy\_task

**warehouse** **=** mywh

**schedule** **=** '60 minute'

**as**

**call** my\_unload\_sp**();**





