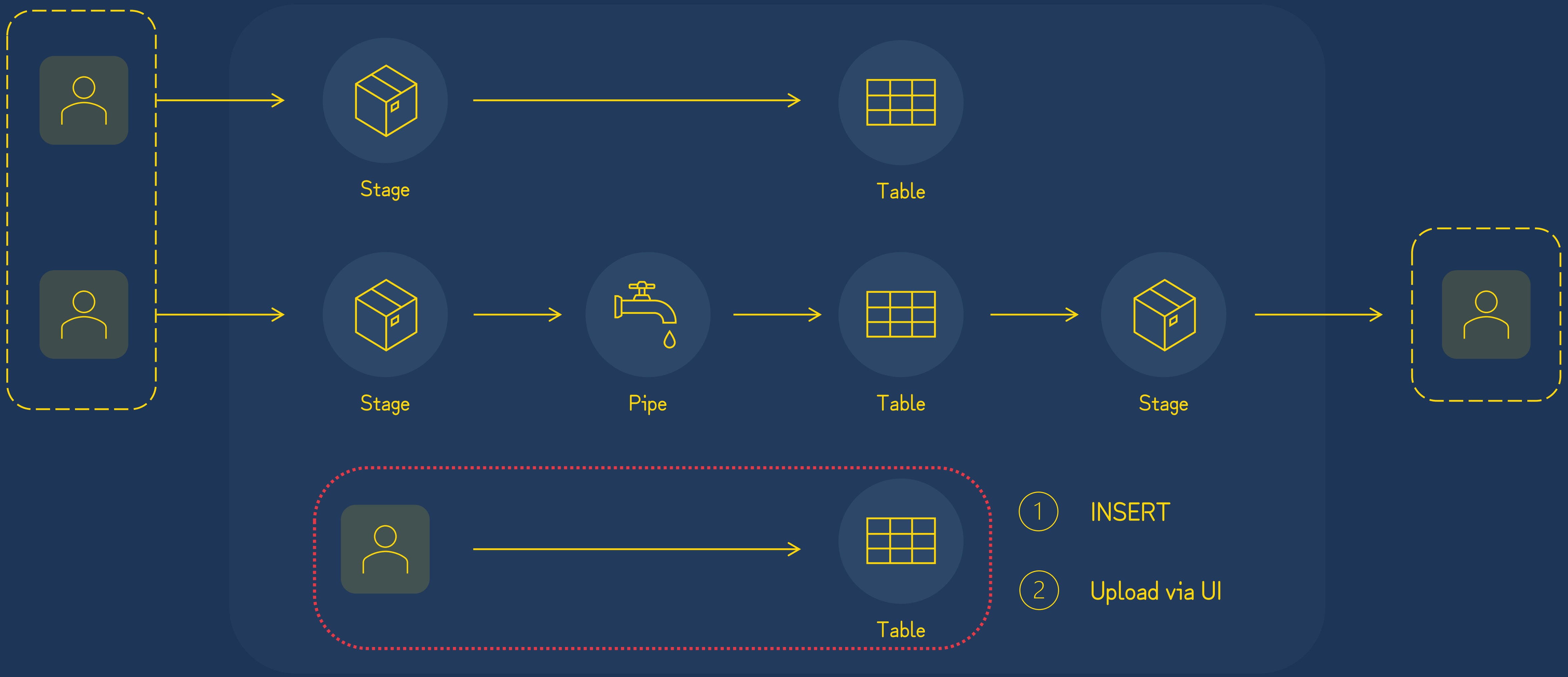
Data Loading Simple Methods

Data Movement



1 INSERT

2 Upload via UI

INSERT

```
INSERT INTO MY_TABLE SELECT ‘001’, ‘John Doughnut’, ‘10/10/1976’;
```

Insert a row into a table from the results of a select query.

```
INSERT INTO MY_TABLE (ID, NAME) SELECT ‘001’, ‘John Doughnut’;
```

To load specific columns, individual columns can be specified

```
INSERT INTO MY_TABLE (ID, NAME, DOB) VALUES
(‘001’, ‘John Doughnut’, ‘10/10/1976’),
(‘002’, ‘Lisa Snowflake’, ‘21/01/1934’),
(‘003’, ‘Oggle Berry’, ‘01/01/2001’);
```

The VALUES keyword can be used to insert multiple rows into a table.

```
INSERT INTO MY_TABLE SELECT * FROM MY_TABLE_2;
```

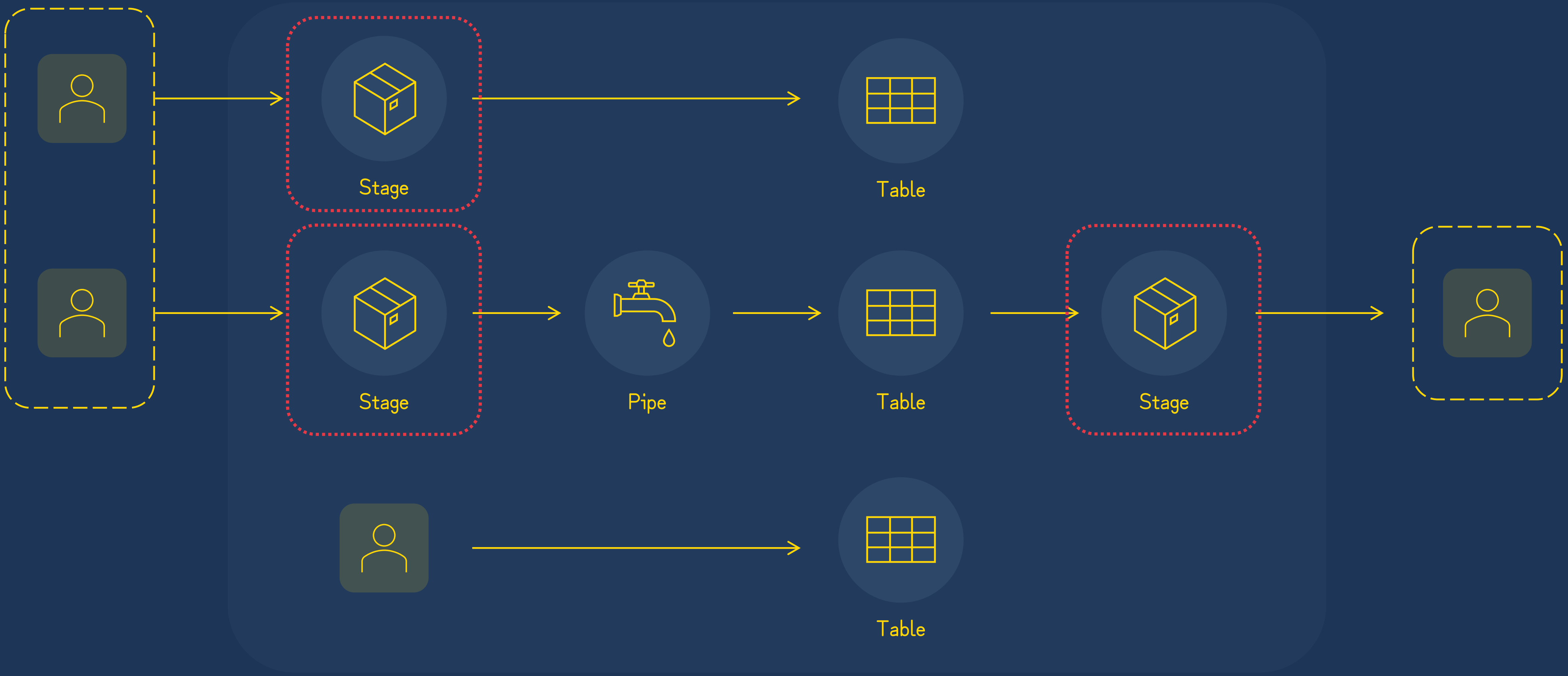
Another table can be used to insert rows into a table.

```
INSERT OVERWRITE INTO MY_TABLE SELECT * FROM MY_TABLE_2;
```

The keyword OVERWRITE will truncate a table before new values are inserted into it.

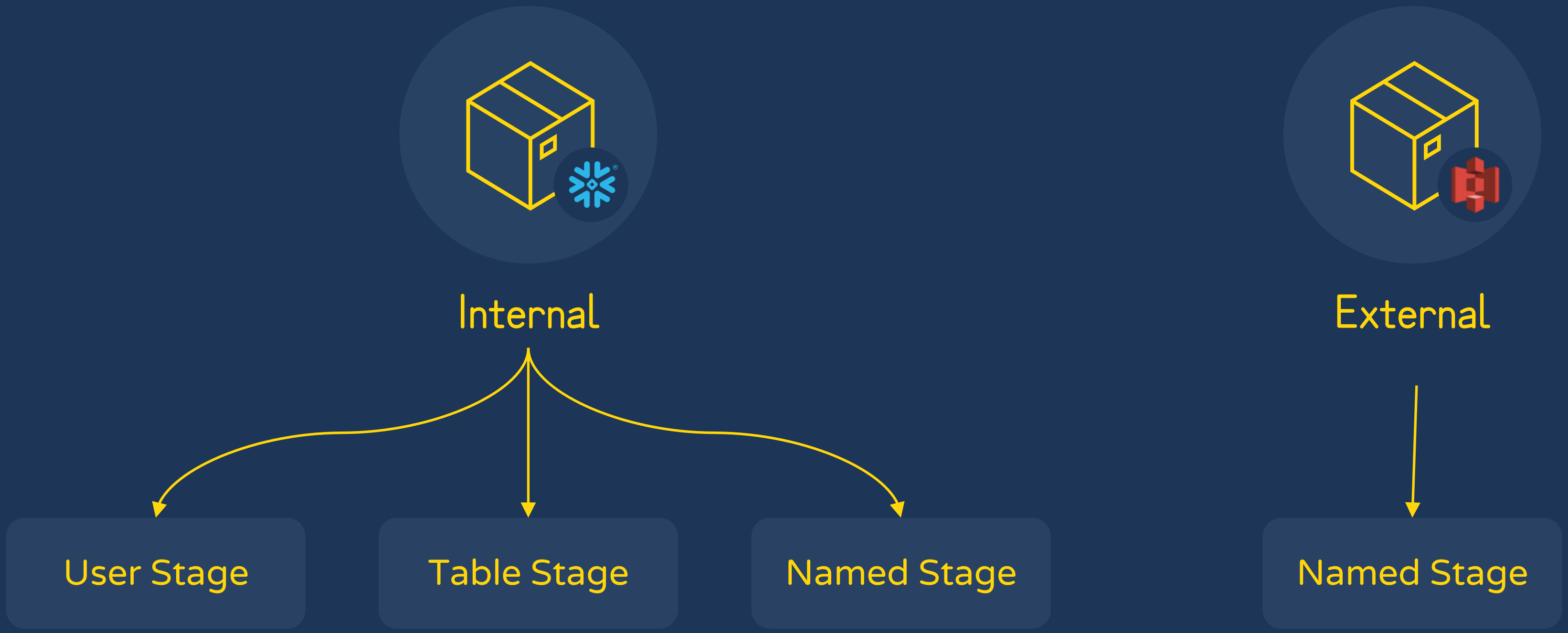
Stages

Stages



Stages

Stages are temporary storage locations for data files used in the data loading and unloading process.



Internal Stages

User Stage	Table Stage	Named Stage
Automatically allocated when a user is created.	Automatically allocated when a table is created.	User created database object.
PUT	PUT	PUT
`ls @~;`	`ls @%MY_TABLE;`	`ls @MY_STAGE;`
Cannot be altered or dropped.	Cannot be altered or dropped.	Securable object.
Not appropriate if multiple users need access to stage.	User must have ownership privileges on table.	Supports copy transformations and applying file formats.

External Stages and Storage Integrations

External stages reference data files stored in a location outside of Snowflake.

External Named Stage

```
CREATE STAGE MY_EXT_STAGE
URL='S3://MY_BUCKET/PATH/'
CREDENTIALS=(AWS_KEY_ID=‘’;AWS_SECRET_KEY=‘’)
STORAGE_INTEGRATION=MY_INT
ENCRYPTION=(MASTER_KEY=‘’)
```

User created database object.

Cloud Utilities

```
ls @MY_STAGE;
```

```
CREATE STORAGE INTEGRATION MY_INT
TYPE=EXTERNAL_STAGE
STORAGE_PROVIDER=S3
STORAGE_AWS_ROLE_ARN=‘ARN:AWS:IAM::98765:ROLE/MY_ROLE’
ENABLED=TRUE
STORAGE_ALLOWED_LOCATIONS=(‘S3://MY_BUCKET/PATH/’);
```

Storage location can be private or public.

Copy options such as ON_ERROR and PURGE can be set on stages.

A storage integration is a reusable and securable Snowflake object which can be applied across stages and is recommended to avoid having to explicitly set sensitive information for each stage definition.

Stage Helper Commands

LIST

```
LIST/ls @MY_STAGE;
LIST/ls @~;
LIST/ls @%MY_TABLE;
```

List the contents of a stage:

- Path of staged file
- Size of staged file
- MD5 Hash of staged file
- Last updated timestamp

Can optionally specify a path for specific folders or files.

Named and internal table stages can optionally include database and schema global pointer.

SELECT

```
SELECT
metadata$filename,
metadata$file_row_number,
$1,
$2
FROM @MY_STAGE
(FILE_FORMAT => 'MY_FORMAT');
```

Query the contents of staged files directly using standard SQL for both internal and external stages.

Useful for inspected files prior to data loading/unloading.

Reference metadata columns such as filename and row numbers for a staged file.

REMOVE

```
REMOVE/rm @MY_STAGE;
REMOVE/rm @~;
REMOVE/rm @%MY_TABLE;
```

Remove files from either an external or internal stage.

Can optionally specify a path for specific folders or files.

Named and internal table stages can optionally include database and schema global pointer.

PUT

The PUT command uploads data files from a local directory on a client machine to any of the three types of internal stage.

PUT cannot be executed from within worksheets.

Duplicate files uploaded to a stage via PUT are ignored.

Uploaded files are automatically encrypted with a 128-bit key with optional support for a 256-bit key.

```
PUT FILE:///FOLDER/MY_DATA.CSV @MY_INT_STAGE;

PUT FILE:///FOLDER/MY_DATA.CSV @~;

PUT FILE:///FOLDER/MY_DATA.CSV @%MY_TABLE;
```

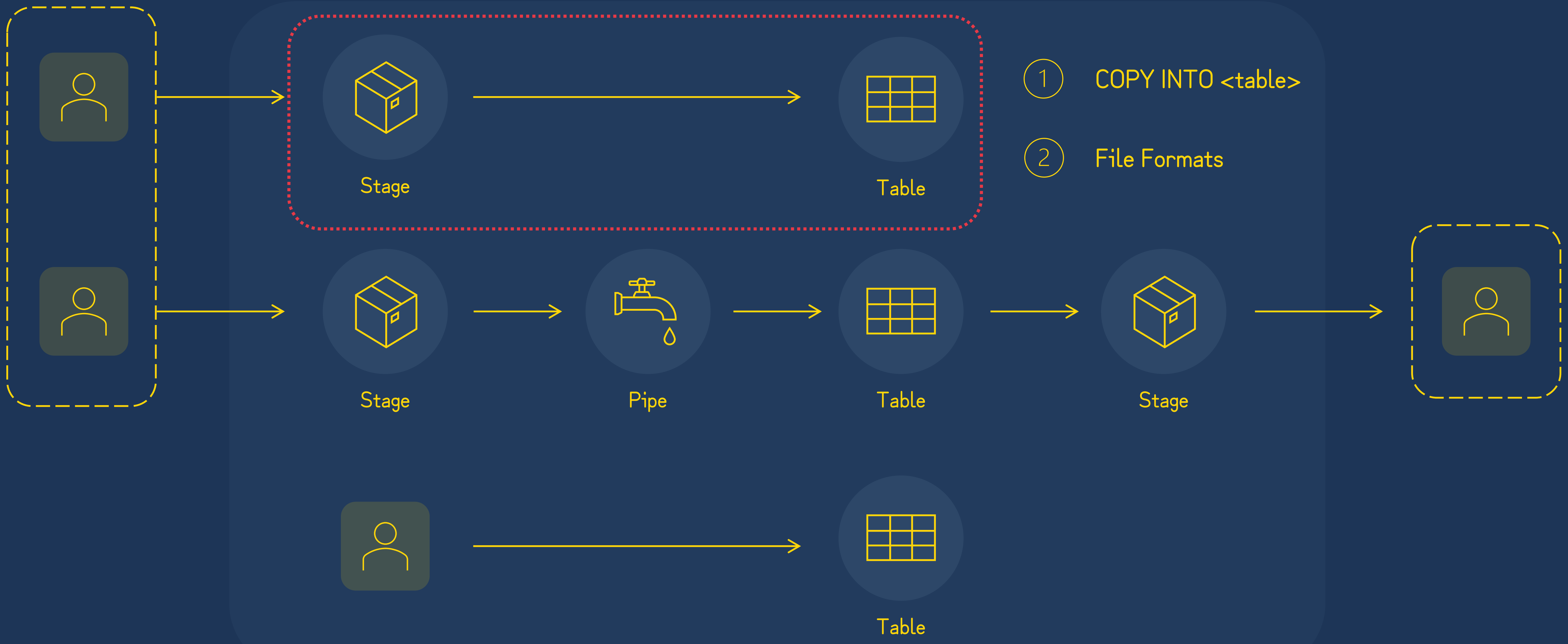
macOS / Linux

```
PUT FILE://c:\\FOLDER\\MY_DATA.CSV @MY_INT_STAGE;
```

Windows

Bulk Loading with COPY INTO <table>

Data Movement



1. COPY INTO <table>
2. File Formats

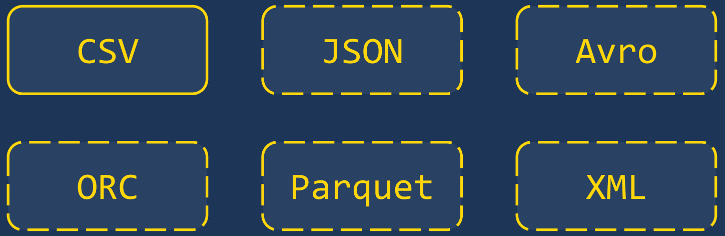
COPY INTO <table>

The COPY INTO <table> statement copies the contents of an internal or external stage or external location directly into a table.

```
COPY INTO MY_TABLE FROM @MY_INT_STAGE;
```

The following file formats can be uploaded to Snowflake:

- Delimited files (CSV, TSC, etc)
- JSON
- Avro
- ORC
- Parquet
- XML



COPY INTO <table> requires a user created virtual warehouse to execute.

Load history is stored in the metadata of the target table for 64 days, which ensures files are not loaded twice.

COPY INTO <table>

```
COPY INTO MY_TABLE FROM @MY_INT_STAGE;

COPY INTO MY_TABLE FROM @MY_INT_STAGE/folder1;

COPY INTO MY_TABLE FROM @MY_INT_STAGE/folder1/file1.csv;

COPY INTO MY_TABLE FROM @MY_INT_STAGE
FILE=('folder1/file1.csv', 'folder2/file2.csv');

COPY INTO MY_TABLE FROM @MY_INT_STAGE
PATTERN=('people/.*[.]csv');
```

Copy all the contents of a stage into a table.

Copy contents of a stage from a specific folder/file path.

COPY INTO <table> has an option to provide a list of one or more files to copy.

COPY INTO <table> has an option to provide a regular expression to extract files to load.

COPY INTO <table> Load Transformations

Snowflake allows users to perform simple transformations on data as it's loaded into a table.

Load transformations allows the user to perform:

- Column reordering.
- Column omission.
- Casting.
- Truncate test string that exceed target length.

Users can specify a set of fields to load from the staged data files using a standard SQL query.

```
COPY INTO MY_TABLE FROM (
SELECT
TO_DOUBLE(T.$1),
T.$2,
T.$3,
TO_TIMESTAMP(T.$4)
FROM @MY_INT_STAGE T);
```

COPY External Stage/Location

```
COPY INTO MY_TABLE FROM @MY_EXTERNAL_STAGE;
```

```
COPY INTO MY_TABLE FROM S3://MY_BUCKET/
STORAGE_INTEGRATION=MY_INTEGRATION
ENCRYPTION=(MASTER_KEY=‘’);
```

Files can be loaded from external stages in the same way as internal stages.

Data transfer billing charges may apply when loading data from files in a cloud storage service in a different region or cloud platform from your Snowflake account.

Files can be copied directly from a cloud storage service location.

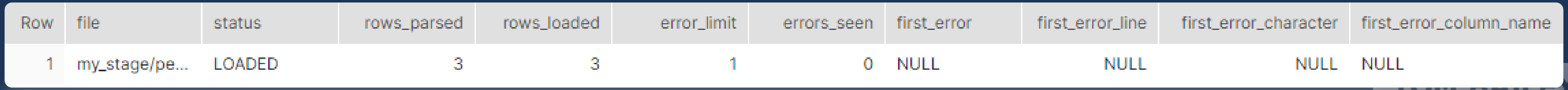
Snowflake recommend encapsulating cloud storage service in an external stage.

Copy Options

Copy Option	Definition	Default Value
ON_ERROR	Value that specifies the error handling for the load operation: • CONTINUE • SKIP_FILE • SKIP_FILE_<num> • SKIP_FILE_<num>% • ABORT_STATEMENT	'ABORT_STATEMENT'
SIZE_LIMIT	Number that specifies the maximum size of data loaded by a COPY statement.	null (no size limit)
PURGE	Boolean that specifies whether to remove the data files from the stage automatically after the data is loaded successfully.	FALSE
RETURN_FAILED_ONLY	Boolean that specifies whether to return only files that have failed to load in the statement result.	FALSE
MATCH_BY_COLUMN_NAME	String that specifies whether to load semi-structured data into columns in the target table that match corresponding columns represented in the data.	NONE
ENFORCE_LENGTH	Boolean that specifies whether to truncate text strings that exceed the target column length.	TRUE
TRUNCATECOLUMNS	Boolean that specifies whether to truncate text strings that exceed the target column length.	FALSE
FORCE	Boolean that specifies to load all files, regardless of whether they've been loaded previously and have not changed since they were loaded.	FALSE
LOAD_UNCERTAIN_FILES	Boolean that specifies to load files for which the load status is unknown. The COPY command skips these files by default.	FALSE

COPY INTO <table> Output

Column Name	Data Type	Description
FILE	TEXT	Name of source file and relative path to the file.
STATUS	TEXT	Status: loaded, load failed or partially loaded.
ROWS_PARSED	NUMBER	Number of rows parsed from the source file.
ROWS_LOADED	NUMBER	Number of rows loaded from the source file.
ERROR_LIMIT	NUMBER	If the number of errors reaches this limit, then abort.
ERRORS_SEEN	NUMBER	Number of error rows in the source file.
FIRST_ERROR	TEXT	First error of the source file.
FIRST_ERROR_LINE	NUMBER	Line number of the first error.
FIRST_ERROR_CHARACTER	NUMBER	Position of the first error character.
FIRST_ERROR_COLUMN_NAME	TEXT	Column name of the first error.



Row	file	status	rows_parsed	rows_loaded	error_limit	errors_seen	first_error	first_error_line	first_error_character	first_error_column_name
1	my_stage/pe...	LOADED	3	3	1	0	NULL	NULL	NULL	NULL

COPY INTO <table> Validation

VALIDATION_MODE

Optional parameter allows you to perform a dry-run of load process to expose errors.

- RETURN_N_ROWS
- RETURN_ERRORS
- RETURN_ALL_ERRORS

```
COPY INTO MY_TABLE
FROM @MY_INT_STAGE;
VALIDATION_MODE = ‘RETURN_ERRORS’;
```

VALIDATE

Validate is a table function to view all errors encountered during a previous COPY INTO execution.

Validate accepts a job id of a previous query or the last load operation executed.

```
SELECT * FROM TABLE(VALIDATE(MY_TABLE,
JOB_ID=>'5415FA1E-59C9-4DDA-B652-533DE02FDCF1'));
```

File Formats

File Formats

File format options can be set on a named stage or COPY INTO statement.

```
CREATE STAGE MY_STAGE
FILE_FORMAT=(TYPE='CSV' SKIP_HEADER=1);
```

Explicitly declared file format options can all be rolled up into independent File Format Snowflake objects.

```
CREATE FILE FORMAT MY_CSV_FF
TYPE='CSV’
SKIP_HEADER=1;
```

File Formats can be applied to both named stages and COPY INTO statements. If set on both COPY INTO will take precedence.

```
CREATE OR REPLACE STAGE MY_STAGE
FILE_FORMAT=MY_CSV_FF;
```

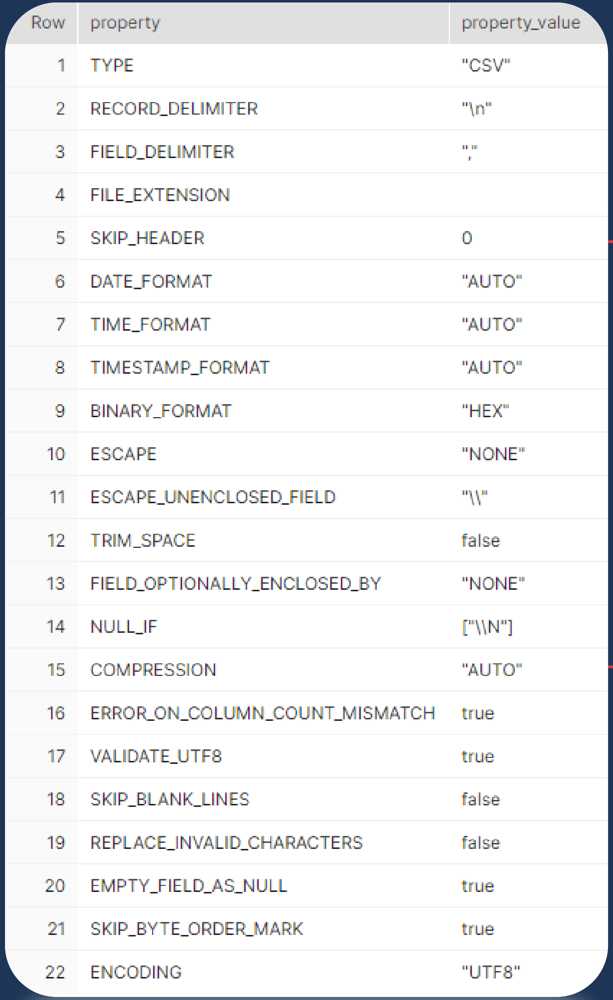
File Formats

In the File Format object the file format you're expecting to load is set via the 'type' property with one of the following values: CSV , JSON, AVRO, ORC, PARQUET or XML.

Each 'type' has it's own set of properties related to parsing that specific file format.

```
CREATE FILE FORMAT MY_CSV_FF
TYPE='CSV';
```

If a File Format object or options are not provided to either the stage or COPY statement, the default behaviour will be to try and interpret the contents of a stage as a CSV with UTF-8 encoding.



Row	property	property_value
1	TYPE	"CSV"
2	RECORD_DELIMITER	"\n"
3	FIELD_DELIMITER	","
4	FILE_EXTENSION	
5	SKIP_HEADER	0
6	DATE_FORMAT	"AUTO"
7	TIME_FORMAT	"AUTO"
8	TIMESTAMP_FORMAT	"AUTO"
9	BINARY_FORMAT	"HEX"
10	ESCAPE	"NONE"
11	ESCAPE_UNENCLOSED_FIELD	"\\"
12	TRIM_SPACE	false
13	FIELD_OPTIONALLY_ENCLOSED_BY	"NONE"
14	NULL_IF	["\\N"]
15	COMPRESSION	"AUTO"
16	ERROR_ON_COLUMN_COUNT_MISMATCH	true
17	VALIDATE_UTF8	true
18	SKIP_BLANK_LINES	false
19	REPLACE_INVALID_CHARACTERS	false
20	EMPTY_FIELD_AS_NULL	true
21	SKIP_BYTE_ORDER_MARK	true
22	ENCODING	"UTF8"

Number of lines at the start of the file to skip.

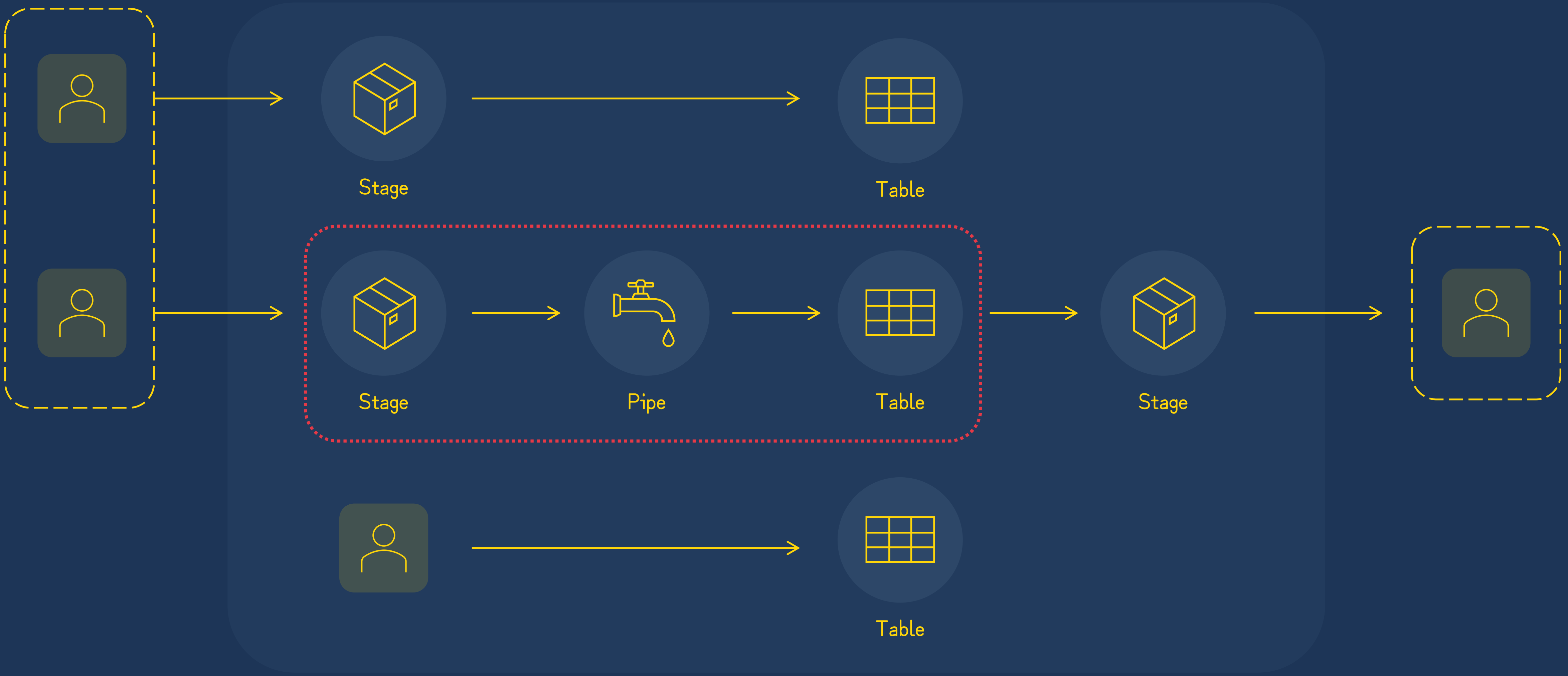
Specifies the current compression algorithm for the data file.

TOM BAILEY COURSES

tombaileycourses.com

Snowpipe and Loading Best Practises

Snowpipe



Snowpipe

```
CREATE PIPE MY_PIPE
AUTO_INGEST=TRUE
AS
COPY INTO MY_TABLE
FROM @MY_STAGE
FILE_FORMAT = (TYPE = 'CSV');
```

There are two methods for detecting when a new file has been uploaded to a stage:

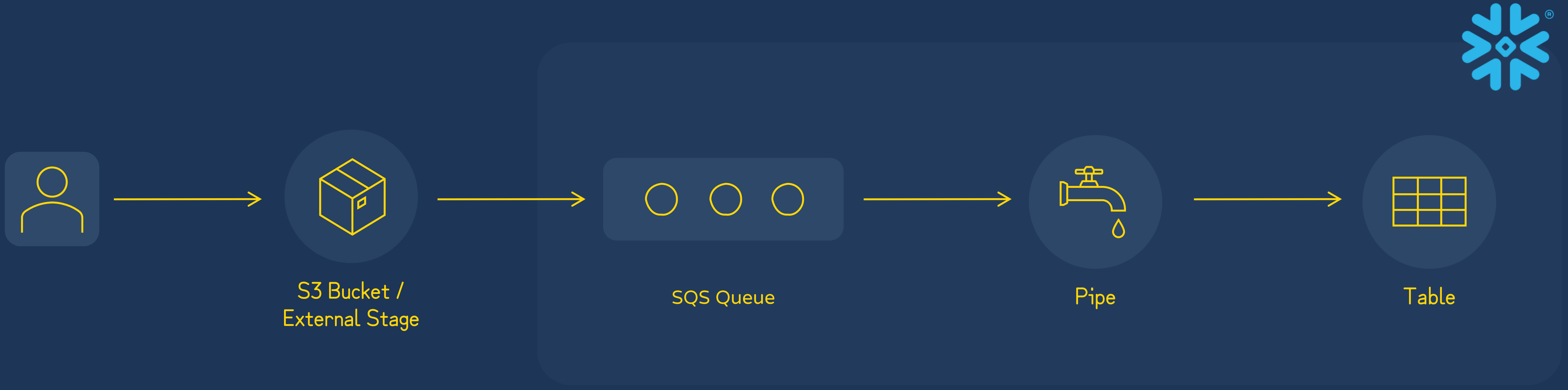
- Automating Snowpipe using cloud messaging

  (external stages only)
- Calling Snowpipe REST endpoints

  (internal and external stages)

The Pipe object defines a COPY INTO <table> statement that will execute in response to a file being uploaded to a stage.

Snowpipe: Cloud Messaging

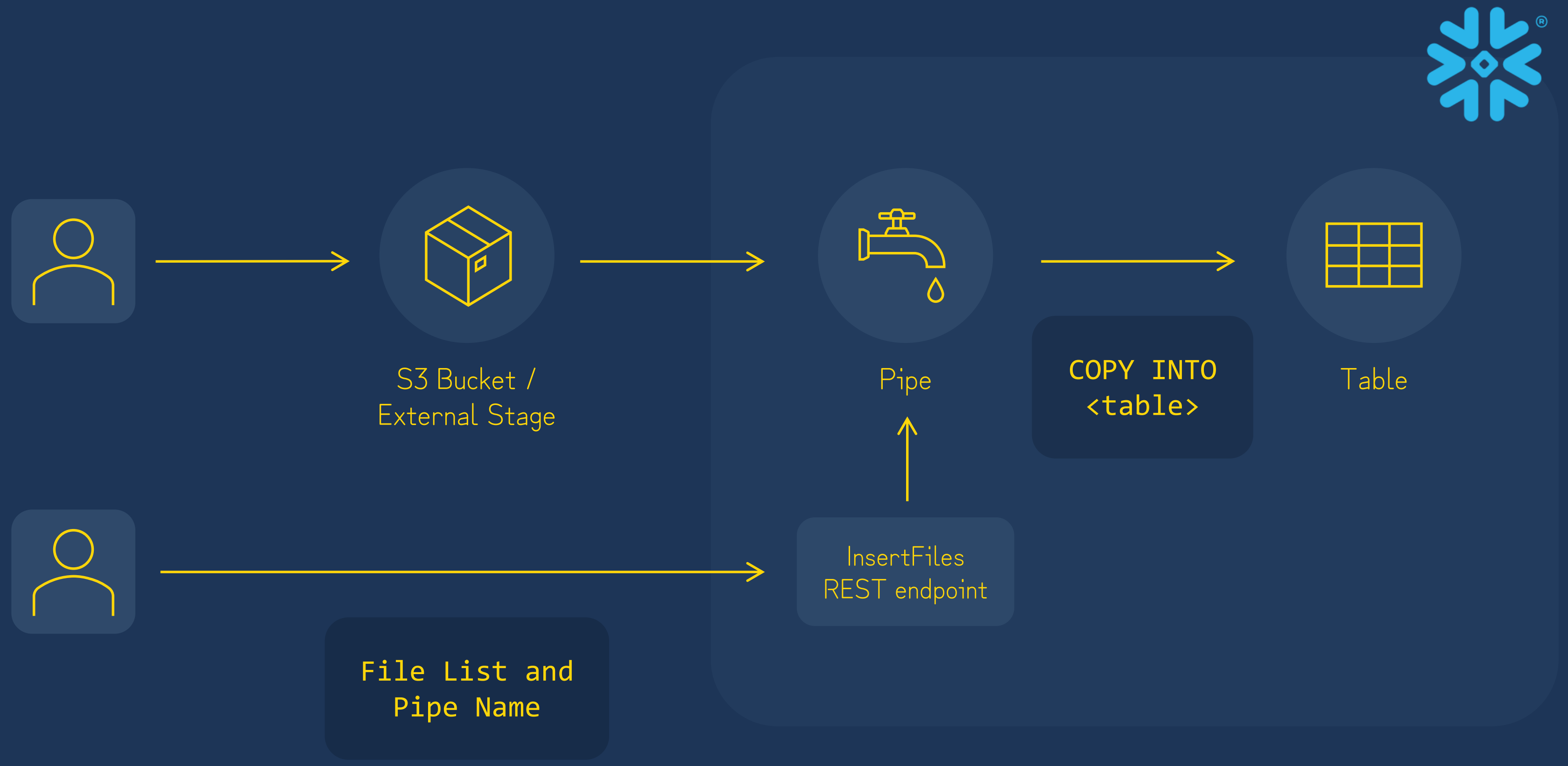


`Cloud Utils`

`PutObject`

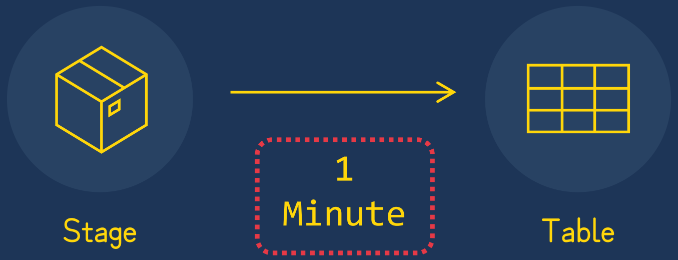
`COPY INTO <table>`

Snowpipe: REST Endpoint

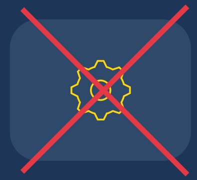


Snowpipe

Snowpipe is designed to load new data typically within a minute after a file notification is sent.



Snowpipe is serverless feature, using Snowflake managed compute resources to load data files not a user managed Virtual Warehouse.



Snowpipe load history is stored in the metadata of the pipe for 14 days, used to prevent reloading the same files in a table.

14 Days

When a pipe is paused, event messages received for the pipe enter a limited retention period. The period is 14 days by default.

14 Days

Bulk Loading vs. Snowpipe

Feature	Bulk Loading	Snowpipe
Authentication	Relies on the security options supported by the client for authenticating and initiating a user session.	When calling the REST endpoints: Requires key pair authentication with JSON Web Token (JWT). JWTs are signed using a public/private key pair with RSA encryption.
Load History	Stored in the metadata of the target table for **64 days.**	Stored in the metadata of the pipe for **14 days.**
Compute Resources	Requires a user-specified warehouse to execute COPY statements.	Uses Snowflake-supplied compute resources.
Billing	Billed for the amount of time each virtual warehouse is active.	Snowflake tracks the resource consumption of loads for all pipes in an account, with per-second/per-core granularity, as Snowpipe actively queues and processes data files. In addition to resource consumption, an overhead is included in the utilization costs charged for Snowpipe: **0.06 credits per 1000 files** notified or listed via event notifications or REST API calls.

Data Loading Best Practises

100-250 MB Compressed

- 2022/07/10/05/
- 2022/06/01/11/

Organize Data By Path

Load

Query

Pre-sort data

1 Minute

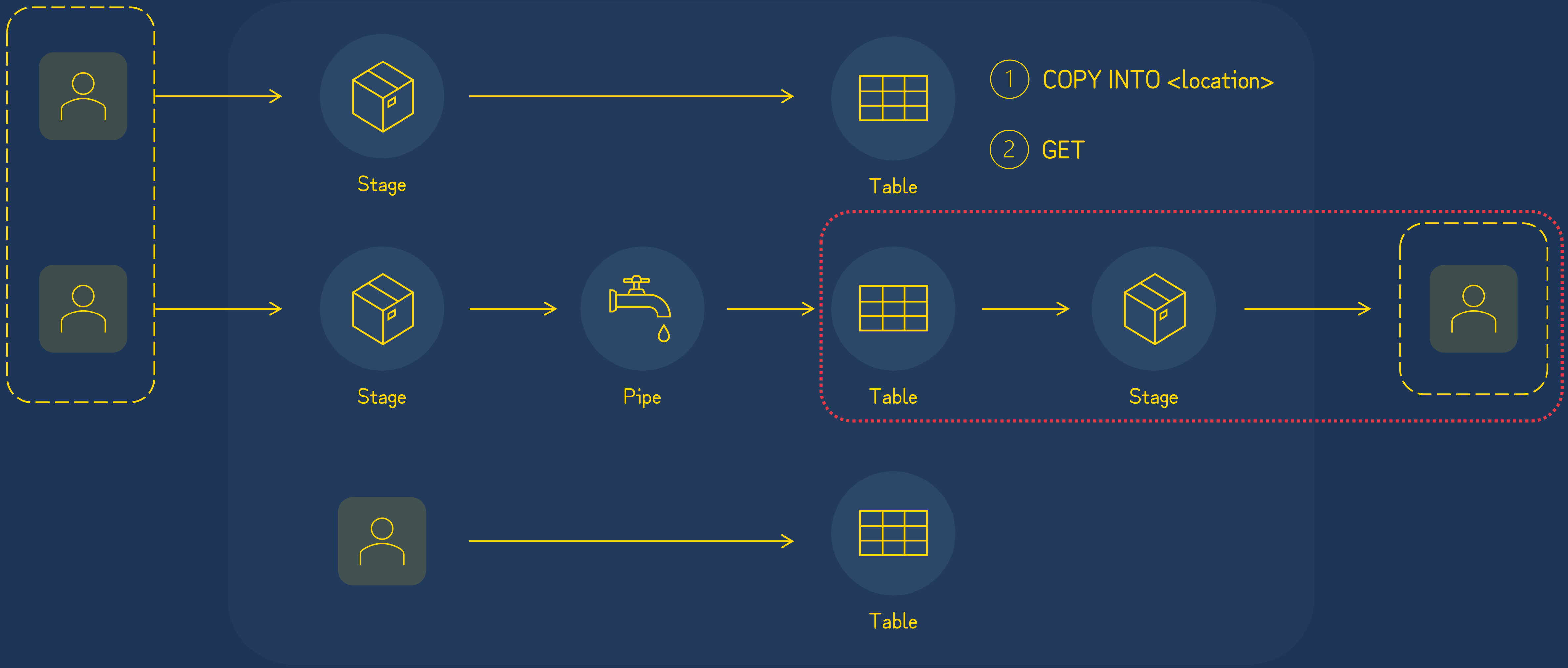
Once per minute

TOM BAILEY COURSES

tombaileycourses.com

Data Unloading Overview

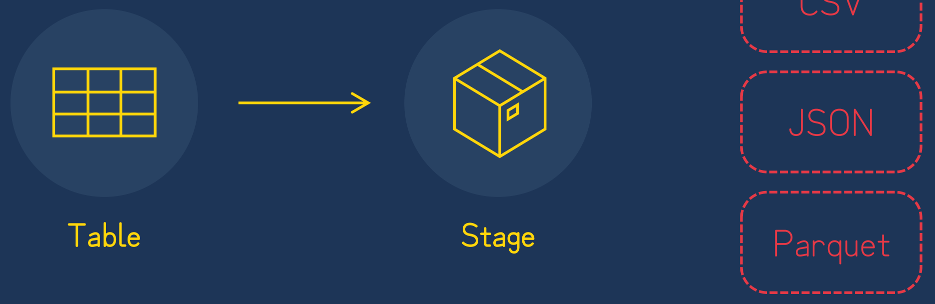
Data Unloading



Data Unloading

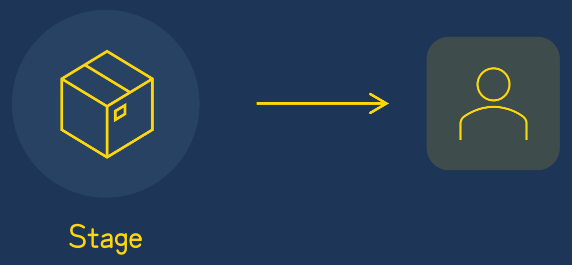
Table data can be unloaded to a stage via the
COPY INTO <location> command.

```
COPY INTO @MY_STAGE
FROM MY_TABLE;
```



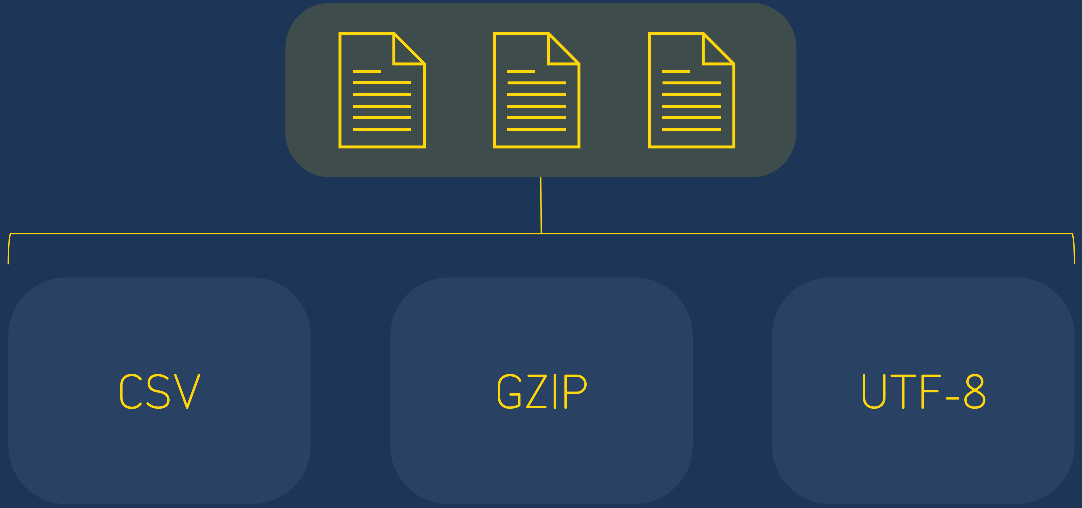
The GET command is used to download a staged
file to the local file system.

```
GET @MY_STAGE
file:///folder/files/;
```



Data Unloading

By default results unloaded to a stage using COPY INTO <location> command are split in to multiple files:



All data files unloaded to internal stages are automatically encrypted using 128-bit keys.

COPY INTO <location> Examples

```
COPY INTO @MY_STAGE/RESULT/DATA_
FROM (SELECT * FROM T1)
FILE_FORMAT = MY_CSV_FILE_FORMAT;
```

Output files can be prefixed by specifying a string at the end of a stage path.

```
COPY INTO @%T1
FROM T1
PARTITION BY ('DATE=' || TO_VARCHAR(DT))
FILE_FORMAT=MY_CSV_FILE_FORMAT;
```

COPY INTO <location> includes a PARTITION BY copy option to partition unloaded data into a directory structure.

```
COPY INTO 'S3://MYBUCKET/UNLOAD/'
FROM T1
STORAGE_INTEGRATION = MY_INT
FILE_FORMAT=MY_CSV_FILE_FORMAT;
```

COPY INTO <location> can copy table records directly to external cloud provider's blob storage.

COPY INTO <location> Copy Options

Copy Option	Definition	Default Value
OVERWRITE	Boolean that specifies whether the COPY command overwrites existing files with matching names, if any, in the location where files are stored.	'ABORT_STATEMENT'
SINGLE	Boolean that specifies whether to generate a single file or multiple files.	FALSE
MAX_FILE_SIZE	Number (> 0) that specifies the upper size limit (in bytes) of each file to be generated in parallel per thread.	FALSE
INCLUDE_QUERY_ID	Boolean that specifies whether to uniquely identify unloaded files by including a universally unique identifier (UUID) in the filenames of unloaded data files.	FALSE

GET

GET is the reverse of PUT. It allows users to specify a source stage and a target local directory to download files to.

```
GET @MY_STAGE FILE:///TMP/DATA/;
```

GET cannot be used for external stages.

GET cannot be executed from within worksheets.

Downloaded files are automatically decrypted.

Parallel optional parameter specifies the number of threads to use for downloading files. Increasing this number can improve parallelisation with downloading large files.

```
GET @MY_STAGE FILE:///TMP/DATA/
PARALLEL=99;
```

Pattern optional parameter specifies a regular expression pattern for filtering files to download.

```
GET @MY_STAGE FILE:///TMP/DATA/
PATTERN=‘*\\.(csv)’;
```

Semi-structured Overview

Semi-structured Data Types

VARIANT

- VARIANT is universal semi-structured data type of Snowflake for loading data in semi-structured data formats.
- VARIANT are used to represent arbitrary data structures.
- Snowflake stores the VARIANT type internally in an efficient compressed columnar binary representation.
- Snowflake extracts as much of the data as possible to a columnar form, based on certain rules.
- VARIANT data type can hold up to 16MB compressed data per row.
- VARIANT column can contain SQL NULLs and VARIANT NULL which are stored as a string containing the word “null”.

Semi-structured Data Overview

```
{"widget": {
    "debug": "on",
    "window": {
        "title": "Sample Konfabulator Widget",
        "name": "main_window",
        "width": 500,
        "height": 500
    },
    "image": {
        "src": "Images/Sun.png",
        "name": "sun1",
        "hOffset": [250, 300, 850],
        "alignment": "center"
    },
    "text": {
        "data": "Click Here",
        "size": 36,
        "style": "bold",
        "name": "text1",
        "hOffset": 250,
        "vOffset": 100,
        "alignment": "center",
        "onMouseUp": "sun1.opacity = 90;"
    }}
```

JSON

```
<widget>
    <debug>on</debug>
    <window title="Sample Konfabulator Widget">
        <name>main_window</name>
        <width>500</width>
        <height>500</height>
    </window>
    <image src="Images/Sun.png" name="sun1">
        <hOffset>250</hOffset>
        <hOffset>300</hOffset>
        <hOffset>850</hOffset>
        <alignment>center</alignment>
    </image>
    <text data="Click Here" size="36" style="bold">
        <name>text1</name>
        <hOffset>250</hOffset>
        <vOffset>100</vOffset>
        <alignment>center</alignment>
        <onMouseUp>
            sun1.opacity = 90;
        </onMouseUp>
    </text>
</widget>
```

XML

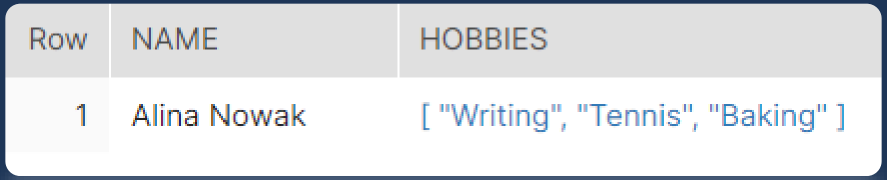
Semi-structured Data Types

ARRAY

Contains 0 or more elements of data. Each element is accessed by its position in the array.

```
CREATE TABLE MY_ARRAY_TABLE (
NAME VARCHAR,
HOBBIES ARRAY
);
```

```
INSERT INTO MY_ARRAY_TABLE
SELECT 'Alina Nowak’, ARRAY_CONSTRUCT('Writing', 'Tennis', 'Baking');
```



Row	NAME	HOBBIES
1	Alina Nowak	["Writing", "Tennis", "Baking"]

TOM BAILEY COURSES

tombaileycourses.com

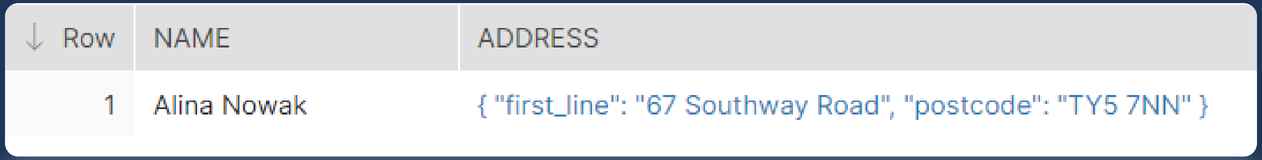
Semi-structured Data Types

OBJECT

Represent collections of key-value pairs.

```
CREATE TABLE MY_OBJECT_TABLE (
NAME VARCHAR,
ADDRESS OBJECT
);
```

```
INSERT INTO MY_OBJECT_TABLE
SELECT 'Alina Nowak', OBJECT_CONSTRUCT('postcode', 'TY5 7NN', 'first_line', '67 Southway Road');
```



↓ Row	NAME	ADDRESS
1	Alina Nowak	{ "first_line": "67 Southway Road", "postcode": "TY5 7NN" }

Semi-structured Data Types

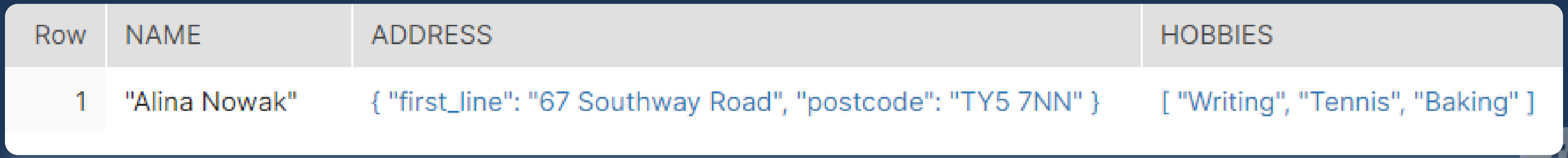
VARIANT

Universal Semi-structured data type used to represent arbitrary data structures.

VARIANT data type can hold up to 16MB compressed data per row

```
CREATE TABLE MY_VARIANT_TABLE (
NAME VARIANT,
ADDRESS VARIANT,
HOBBIES VARIANT
);
```

```
INSERT INTO MY_VARIANT_TABLE
SELECT
'Alina Nowak'::VARIANT,
OBJECT_CONSTRUCT('postcode', 'TY5 7NN', 'first_line', '67 Southway Road'),
ARRAY_CONSTRUCT('Writing', 'Tennis', 'Baking’);
```



Row	NAME	ADDRESS	HOBBIES
1	"Alina Nowak"	{ "first_line": "67 Southway Road", "postcode": "TY5 7NN" }	["Writing", "Tennis", "Baking"]

TOM BAILEY COURSES

tombaileycourses.com

Semi-structured Data Formats

JSON

Plain-text data-interchange format based on a subset of the JavaScript programming language.

Load Unload

AVRO

Binary row-based storage format originally developed for use with Apache Hadoop.

Load

ORC

Highly efficient binary format used to store Hive data.

Load

PARQUET

Binary format designed for projects in the Hadoop ecosystem.

Load Unload

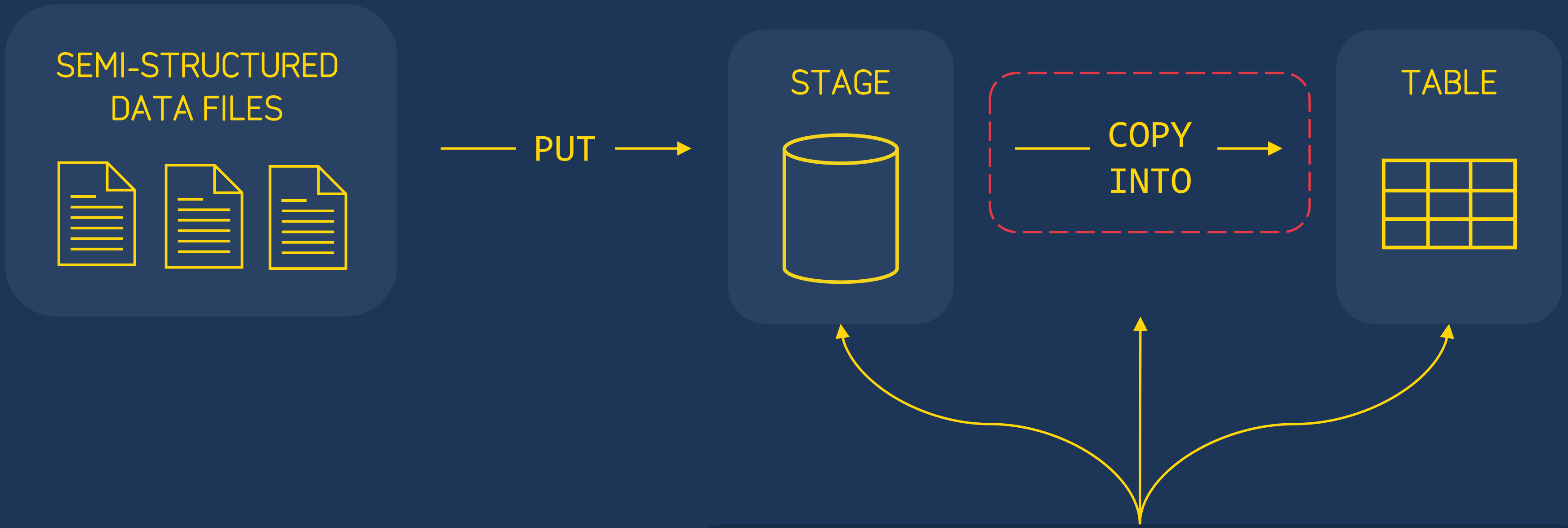
XML

Consists primarily of tags <> and elements.

Load

Loading and Unloading Semi-structured Data

Loading Semi-structured Data

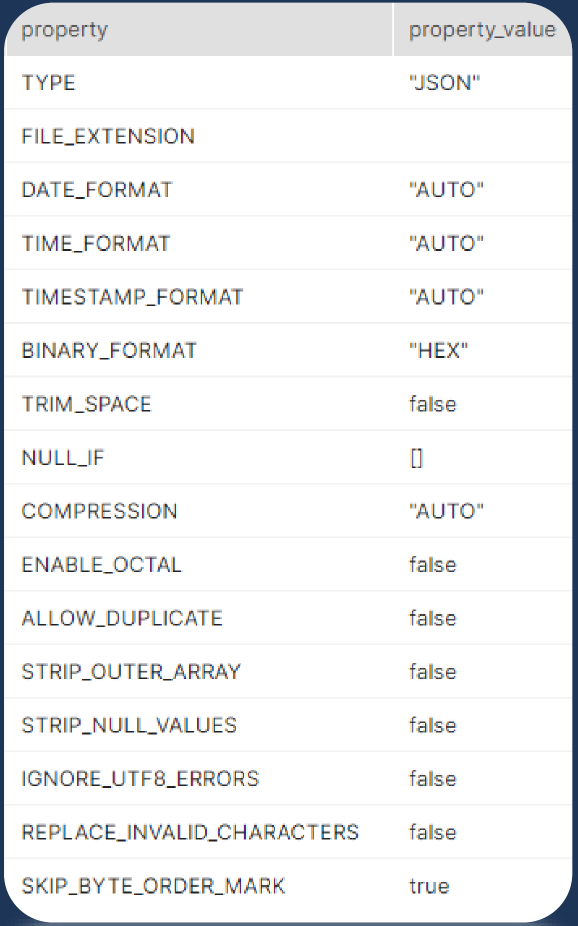


```
CREATE FILE FORMAT <name>
TYPE = { JSON | AVRO | ORC | PARQUET | XML }
[<FORMAT OPTIONS>];
```

JSON File Format options

Option	Details
DATE_FORMAT	Used only for loading JSON data into separate columns. Defines the format of date string values in the data files.
TIME FORMAT	Used only for loading JSON data into separate columns. Defines the format on time string values in the data files.
COMPRESSION	Supported algorithms: GZIP, BZ2, BROTLI, ZSTD, DEFLATE, RAW_DEFLATE, NONE. If BROTLI, cannot use AUTO.
ALLOW DUPLICATE	Only used for loading. If TRUE, allows duplicate object field names (only the last one will be preserved)
STRIP OUTER ARRAY	Only used for loading. If TRUE, JSON parser will remove outer brackets []
STRIP NULL VALUES	Only used for loading. If TRUE, JSON parser will remove object fields or array elements containing NULL

```
DESC FILE FORMAT FF_JSON;
```



property	property_value
TYPE	"JSON"
FILE_EXTENSION	
DATE_FORMAT	"AUTO"
TIME_FORMAT	"AUTO"
TIMESTAMP_FORMAT	"AUTO"
BINARY_FORMAT	"HEX"
TRIM_SPACE	false
NULL_IF	[]
COMPRESSION	"AUTO"
ENABLE_OCTAL	false
ALLOW_DUPLICATE	false
STRIP_OUTER_ARRAY	false
STRIP_NULL_VALUES	false
IGNORE_UTF8_ERRORS	false
REPLACE_INVALID_CHARACTERS	false
SKIP_BYTE_ORDER_MARK	true

Semi-structured Data Loading Approaches

ELT

1

```
CREATE TABLE MY_TABLE (
V VARIANT
);
```

2

```
COPY INTO MY_TABLE
FROM @MY_STAGE/FILE1.JSON
FILE_FORMAT = FF_JSON;
```

ETL

1

```
CREATE TABLE MY_TABLE (
NAME STRING,
AGE NUMBER,
DOB DATE
);
```

2

```
COPY INTO MY_TABLE
FROM ( SELECT
V:name,
V:age,
V:dob
FROM @MY_STAGE/FILE1.JSON)
FILE_FORMAT = FF_JSON;
```

Automatic Schema Detection

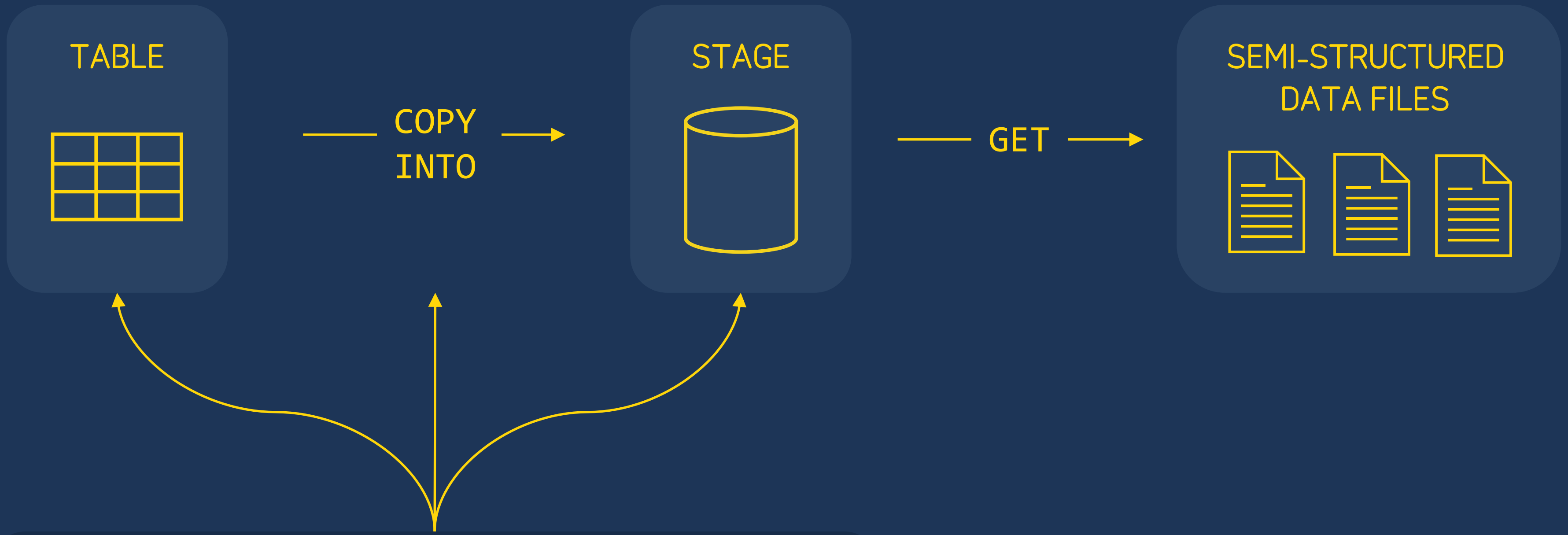
INFER_SCHEMA

```
CREATE TABLE MY_TABLE
  USING TEMPLATE (
    SELECT ARRAY_AGG(OBJECT_CONSTRUCT(*))
      FROM TABLE(
        INFER_SCHEMA(
          LOCATION=>'@MYSTAGE',
          FILE_FORMAT=>'FF_PARQUET'
        )
      ));
```

MATCH_BY_COLUMN_NAME

```
COPY INTO MY_TABLE
FROM @MY_STAGE/FILE1.JSON
FILE_FORMAT = (TYPE = ‘JSON’)
MATCH_BY_COLUMN_NAME = CASE_SENSITIVE;
```

Unloading Semi-structured Data



```
CREATE FILE FORMAT <name>
TYPE = { JSON | AVRO | ORC | PARQUET | XML }
[<FORMAT OPTIONS>];
```

Accessing Semi-structured Data

Accessing Semi-structured Data

```
{
  “employee”:{
    “name”:“Aiko Tanaka”,
    “_id”:“UNX789544”,
    “age”:42
    },
  “joined_on”:“2019-01-01”,
  “skills”:[“Java”, “Kotlin”, “Android”],
  “is_manager”:true,
  “base_location”:null,
}
```

$\xrightarrow{\hspace{3em}}$

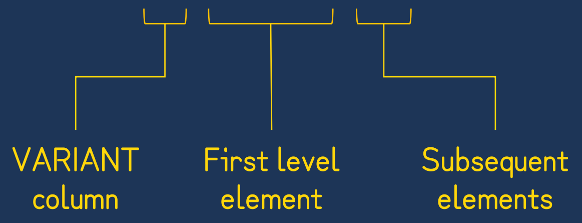
COPY INTO

```
CREATE TABLE EMPLOYEE (
src VARIANT
);
```

Accessing Semi-structured Data

Dot Notation

```
SELECT src:employee.name FROM EMPLOYEES;
```



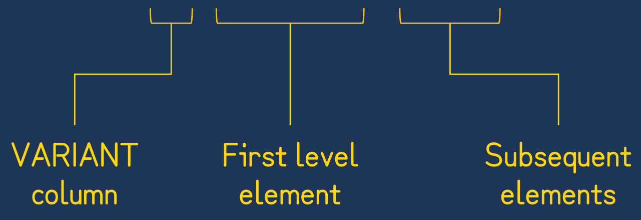
Row	SRC:EMPLOYEE.NAME
1	"Aiko Tanaka"

```
SELECT SRC:Employee.name FROM EMPLOYEES;
```

Column name is case insensitive but key names are case insensitive so the above query would result in an error.

Bracket Notation

```
SELECT SRC[‘employee’][‘name’] FROM EMPLOYEES;
```



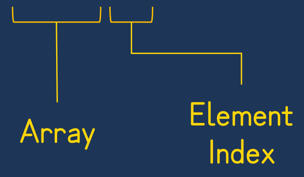
Row	SRC:EMPLOYEE.NAME
1	"Aiko Tanaka"

```
SELECT SRC[‘Employee’][‘name’] FROM EMPLOYEES;
```

Column name is case insensitive but key names are case insensitive so the above query would result in an error.

Repeating Element

```
SELECT SRC:skills[0] FROM EMPLOYEES;
```



Row	SRC:SKILLS[0]
1	"Java"

```
SELECT GET(SRC, ‘employee’)
FROM EMPLOYEE;
```

```
SELECT GET(SRC, ‘skills’)[0]
FROM EMPLOYEE;
```

Casting Semi-structured Data

```
SELECT src:employee.name, src:joined_on, src:employee.age, src:is_manager, src:base_location FROM EMPLOYEE;
```

Row	SRC:EMPLOYEE.NAME	SRC:JOINED_ON	SRC:EMPLOYEE.AGE	SRC:IS_MANAGER	SRC:BASE_LOCATION
1	"Aiko Tanaka"	"2019-01-01"	42	true	null

Double colon

```
SELECT src:employee.joined_on::DATE
FROM EMPLOYEE;
```

Row	SRC:JOINED_ON::DATE
1	2019-01-01

TO_<datatype>()

```
SELECT TO_DATE(src:employee.joined_on)
FROM EMPLOYEE;
```

Row	TO_DATE(SRC:JOINED_ON)
1	2019-01-01

AS_<datatype>()

```
SELECT AS_VARCHAR(src:employee.name)
FROM EMPLOYEE;
```

Row	AS_VARCHAR(SRC:EMPLOYEE.NAME)
1	Aiko Tanaka

Semi-structured Functions

Semi-structured Functions

JSON and XML Parsing	Array/Object Creation and Manipulation	Extraction	Conversion/Casting	Type Predicates
CHECK_JSON	ARRAY_AGG	FLATTEN	AS_<object>	IS_<object>
CHECK_XML	ARRAY_APPEND	GET	AS_ARRAY	IS_ARRAY
JSON_EXTRACT_PATH_TEXT	ARRAY_CAT	GET_IGNORE_CASE	AS_BINARY	IS_BOOLEAN
PARSE_JSON	ARRAY_COMPACT	GET_PATH	AS_CHAR , AS_VARCHAR	IS_BINARY
PARSE_XML	ARRAY_CONSTRUCT	OBJECT_KEYS	AS_DATE	IS_CHAR ,
STRIP_NULL_VALUE	ARRAY_CONSTRUCT_COMPACT	XMLGET	AS_DECIMAL , AS_NUMBER	IS_VARCHAR
	ARRAY_CONTAINS		AS_DOUBLE , AS_REAL	IS_DATE ,
	ARRAY_INSERT		AS_INTEGER	IS_DATE_VALUE
	ARRAY_INTERSECTION		AS_OBJECT	IS_DECIMAL
	ARRAY_POSITION		AS_TIME	IS_DOUBLE ,
	ARRAY_PREPEND		AS_TIMESTAMP_*	IS_REAL
	ARRAY_SIZE		STRTOK_TO_ARRAY	IS_INTEGER
	ARRAY_SLICE		TO_ARRAY	IS_NULL_VALUE
	ARRAY_TO_STRING		TO_JSON	IS_OBJECT
	ARRAYS_OVERLAP		TO_OBJECT	IS_TIME
	OBJECT_AGG		TO_VARIANT	IS_TIMESTAMP_*
	OBJECT_CONSTRUCT		TO_XML	TYPEOF
	OBJECT_CONSTRUCT_KEEP_NULL			
	OBJECT_DELETE			
	OBJECT_INSERT			
	OBJECT_PICK			

FLATTEN Table Function

Flatten is a table function that accepts compound values (VARIANT, OBJECT & ARRAY) and produces a row for each item.

```
SELECT VALUE FROM TABLE(FLATTEN(INPUT => SELECT src:skills FROM EMPLOYEE));
```

Row	VALUE::VARCHAR
1	Java
2	Kotlin
3	Android

Path

```
SELECT VALUE FROM TABLE(FLATTEN(
INPUT => PARSE_JSON('{"A":1, "B":[77,88]}’),
PATH => 'B'));
```

Specify a path inside object to flatten.

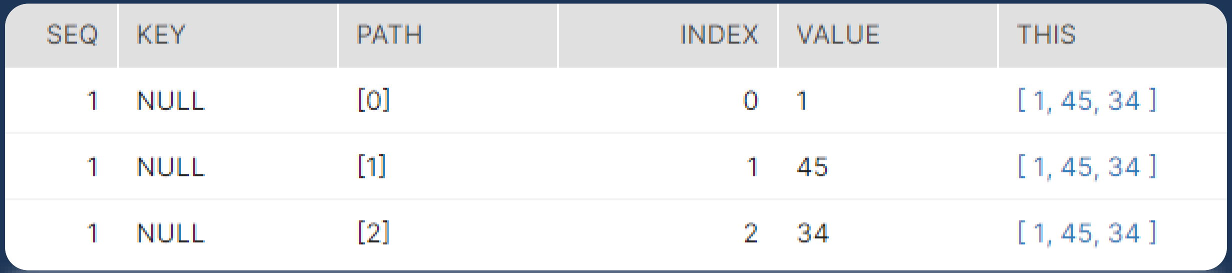
Recursive

```
SELECT VALUE FROM TABLE(FLATTEN(
INPUT => PARSE_JSON('{"A":1, "B":[77,88]}’),
RECURSIVE => true));
```

Flattening is performed for all sub-elements recursively.

FLATTEN Output

```
SELECT * FROM TABLE(FLATTEN(INPUT => (ARRAY_CONSTRUCT(1,45,34))));
```



SEQ	KEY	PATH	INDEX	VALUE	THIS
1	NULL	[0]	0	1	[1, 45, 34]
1	NULL	[1]	1	45	[1, 45, 34]
1	NULL	[2]	2	34	[1, 45, 34]

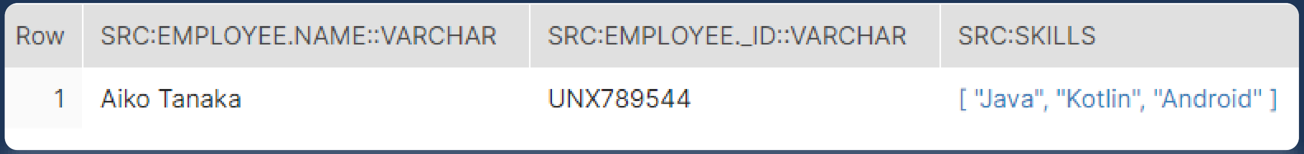
SEQ	KEY	PATH	INDEX	VALUE	THIS
A unique sequence number associated with the input record.	For maps or objects, this column contains the key to the exploded value.	The path to the element within a data structure which needs to be flattened.	The index of the element, if it is an array; otherwise NULL.	The value of the element of the flattened array/object.	The element being flattened (useful in recursive flattening).

TOM BAILEY COURSES

tombaileycourses.com

LATERAL FLATTEN

```sql
SELECT src:employee.name::varchar, src:employee._id::varchar, src:skills FROM EMPLOYEE;
```



Row	SRC:EMPLOYEE.NAME::VARCHAR	SRC:EMPLOYEE._ID::VARCHAR	SRC:SKILLS
1	Aiko Tanaka	UNX789544	["Java", "Kotlin", "Android"]

```sql
SELECT
src:employee.name,
src:employee._id,
f.value
FROM EMPLOYEE e,
LATERAL FLATTEN(INPUT => e.src:skills) f;
```

```
Aiko Tanaka, UNX789544, [“Java”, “Kotlin” , “Android” ]

Aiko Tanaka, UNX789544,

Aiko Tanaka, UNX789544,
```