**Loading Semi Structured Data**

In Snowflake, semi-structured data like JSON, Avro, ORC, XML or Parquet can be loaded directly into a VARIANT column using the COPY INTO command. I usually define a file format (JSON/Parquet), create a stage pointing to cloud storage, and then load the files into Snowflake tables. After loading, I query the data using dot notation (column:field) or flatten functions for arrays. For example, in one project, we ingested JSON logs from Azure Blob into Snowflake, stored them in VARIANT columns, and extracted fields like customer ID and transaction amount for reporting.

-- Plain text data

select 'book\_name','Snowflake Fundamentals','pages',100,'author','John' ;

select 'Data Science Basics' AS DSB,'Snowflake Fundamentals','Oracle Database Development','Basics of machine learning' ;

-- constructs

select object\_construct ('book\_name','Snowflake Fundamentals','pages',100,'author','John','Price',1200) as object\_book ;

select array\_construct ('Data Science Basics','Snowflake Fundamentals','Oracle Database Development','Basics of machine learning') as array\_books ;

-- create tables

create table book (book\_info) as select object\_construct ('book\_name','Snowflake Fundamentals','pages',100,'author','John') as object\_book;

drop table book ;

select \* from book ;

select book\_info:author as book\_author, book\_info:pages as no\_of\_pages from book ;

select book\_info:author::string as book\_author, book\_info:pages as no\_of\_pages from book ;

select cast(book\_info:author as string) as book\_author from book;

-- Array

create table books (book\_arr) as select array\_construct ('Data Science Basics','Snowflake Fundamentals','Oracle Database Development','Basics of machine learning') as array\_books ;

select \* from books ;

select book\_arr[0],book\_arr[1],book\_arr[2],book\_arr[3] from books ;

select book\_arr[0]::string as book1,book\_arr[1]::string as book2,

book\_arr[2]::string as book3,book\_arr[3]::string as book4

from books ;

--

-- Entire JSON/AVRO/ORC/PARQUET files are stored in VARIANT

-- clean up

drop table book ;

drop table books ;

----------Load the JSON Data

-- Create table to load the JSON data

create or replace table raw\_books (src variant);

-- Check the data

select \* from raw\_books ;

-- Insert data using parse\_json

insert into raw\_books

select parse\_json(column1) as src from values

('{

"name": "Snowflake Fundamentals",

"author\_first\_name":"John",

"author\_last\_name":"Doe",

"pages":250,

"publication\_years":[2012, 2014, 2015, 2018],

"languages":["English","French","German","Hindi"],

"publisher":

{"name":"Highland Publication Limited",

"street":"2434 Senora Lane",

"city":"Richmond",

"state":"Virginia",

"country":"US"

},

"contents":[

{"section":"Snowflake Architecture", "chapters":["Introduction","Architecture","Performance management"]},

{"section":"Access Management and Snowflake Objects", "chapters":["Roles","Hierarchy","Tables and Views","Stored Procedures"]},

{"section":"Data Movement 1", "chapters":["Data Loading","Data Unloading"]},

{"section":"Data Movement 2", "chapters":["Semi-Structured Data","SnowPipe"]},

{"section":"Dynamic Data Masking", "chapters":["Introduction","Best Practices"]},

{"section":"Data Sharing and Data Protection", "chapters":["Zero Copy Cloning","Tasks","Streams"]}

]

}

') ;

-- Check the data

select \* from raw\_books ;

select src:name::string as name,

src:author\_first\_name::string as author\_first\_name

from raw\_books;

----------------------------------------------------------------------

-- Check the data

select \* from raw\_books ;

-- truncate table and insert by COPY command

truncate table raw\_books ;

-- Create integration object

create or replace storage integration int\_aws\_s3 -- s3://snowflakes3datafiles/csvfiles/

type = external\_stage

storage\_provider = s3

enabled = true

storage\_aws\_role\_arn = 'arn:aws:iam::755743318453:role/snowflake-s3-full-access-role'

storage\_allowed\_locations = ('s3://snowflakes3datafiles/jsonfiles/')

--storage\_blocked\_locations = ('<>')

comment = 'This is the integration object for loading the files from AWS S3 to Snowflake' ;

-- Describe

desc integration int\_aws\_s3 ;

-- create file format - since it will be used in external stage and copy command. so creating is better

create or replace file format file\_format\_json type = JSON ;

--describe file formats

describe file format file\_format\_json ;

-- create external stage for loading data

create or replace stage ext\_stage\_ld

url = 's3://snowflakes3datafiles/jsonfiles/'

storage\_integration = int\_aws\_s3

file\_format = file\_format\_json ;

-- Show stages

show stages ;

-- list

list @ext\_stage\_ld ;

-- Copy command

copy into raw\_books from @ext\_stage\_ld

files = ('book.json');

-- select

select \* from raw\_books ;

-- Load the books.json

truncate table raw\_books ;

copy into raw\_books from @ext\_stage\_ld files = ('books.json');

-- Check the data

select \* from raw\_books ;

-- Parsing the JSON data

-- select specific columns

select src:name as name, src:author\_first\_name as author\_first\_name, src:author\_last\_name as author\_last\_name, src:pages as pages from raw\_books ;

-- type case as string and int

select src:name::string as name, src:author\_first\_name::string as author\_first\_name,

src:author\_last\_name::string as author\_last\_name, src:pages::int as pages

from raw\_books ;

-- Handling Array - selecting the string array and number array

select src:name::string as name, src:publication\_years as publication\_years, src:languages as languages from raw\_books ;

-- select first element of Array

select src:name::string as name, src:publication\_years[0] as publication\_years, src:languages[0]::string as languages from raw\_books ;

-- ARRAY\_SIZE - to check the length of array and do operations traversing in loop etc

select array\_size(src:publication\_years) as pub\_year\_arr\_size, array\_size(src:languages) as language\_arr\_size from raw\_books ;

select src:publication\_years as publication\_years,

src:publication\_years[0] as pub\_year,

src:publication\_years[1] as pub\_year,

src:publication\_years[2] as pub\_year,

src:publication\_years[3] as pub\_year from raw\_books ;

select \* from raw\_books;

select

rb.src:publisher.name::string as publisher\_name,

rb.src:publisher.street::string as publisher\_street,

rb.src:publisher.city::string as publisher\_city,

rb.src:publisher.state::string as publisher\_state,

rb.src:publisher.country::string as publisher\_country

from raw\_books rb ;

------------------- -- FLATTEN (explodes compound into multiple)

-- table function which takes variant, object and array column and produces a lateral view

-- used for converting semi-structured to relational

-- input is expr -> object, array or variant. This will be converted to rows and will be a mandatory parameter

-- output is - (seq, key, path, index,value,this)

-- (generated number, key of exploded expression, path of the element, index if array, value, element )

--

select src:name::string as name, src:publication\_years as publication\_years, src:languages as languages from raw\_books ;

select \* from raw\_books;

-- Lets see the example of publication\_years

select py.value as publication\_years from raw\_books rb,

lateral flatten(input=> rb.src:publication\_years) py

where src:name = 'Snowflake Fundamentals' ;

-- object

select \* from raw\_books rb,

lateral flatten(input=> rb.src:publisher) py

where src:name = 'Snowflake Fundamentals' ;

-- variant

select \* from raw\_books rb, lateral flatten(input=> rb.src) py where src:name = 'Snowflake Fundamentals' ;

--------------------------------------------------------------------------

-- Lets start on the data

-- rb.src:name::string as name,

select rb.src:name::string as name, src:publication\_years as publication\_years, py.index, py.value

from raw\_books rb, lateral flatten(input=> rb.src:publication\_years) py ;

-- You can use as table as well

select rb.src:name::string as name, src:publication\_years as publication\_years, py.index, py.value

from raw\_books rb, table(flatten(rb.src:publication\_years)) py ;

---

-- Lets see the example of languages

select \* from raw\_books rb, lateral flatten(input=> rb.src:languages) ;

-- rb.src:name::string as name,

select rb.src:name::string as name, src:publication\_years as publication\_years, lng.index, lng.value::string as value

from raw\_books rb, lateral flatten(input=> rb.src:languages) lng ;

-- You can use as table as well

select rb.src:name::string as name, src:publication\_years as publication\_years, lng.index, lng.value::string as value

from raw\_books rb, lateral flatten(input=> rb.src:languages)) lng ;

--Final Query

select --element

distinct

rb.src:name::string as name,

rb.src:author\_first\_name::string as author\_first\_name,

rb.src:author\_last\_name::string as author\_last\_name,

rb.src:pages as pages,

--Array

pyr.value::string as publication\_year,

--Array

lng.value::string as language,

--nested object

rb.src:publisher.name::string as publisher\_name,

rb.src:publisher.street::string as publisher\_street,

rb.src:publisher.city::string as publisher\_city,

rb.src:publisher.state::string as publisher\_state,

rb.src:publisher.country::string as publisher\_country,

-- nested array of objects. Object also has an array -- more complicated version

cnt.value:section::string as section,

chp.value::string as chapters

from raw\_books rb,

table(flatten(rb.src:publication\_years)) pyr,

table(flatten(rb.src:languages)) lng,

table(flatten(rb.src:contents)) cnt,

table(flatten(cnt.value:chapters)) chp

where src:name = 'Snowflake Fundamentals';

-- where src:name = 'Snowflake Advanced - Architect' ;

select \* from raw\_books;

--- Create the final table from JSON file

create or replace table parsed\_books as

select --element

rb.src:name::string as name,

rb.src:author\_first\_name::string as author\_first\_name,

rb.src:author\_last\_name::string as author\_last\_name,

rb.src:pages as pages,

--Array

pyr.value::string as publication\_year,

--Array

lng.value::string as language,

--nested object

rb.src:publisher.name::string as publisher\_name,

rb.src:publisher.street::string as publisher\_street,

rb.src:publisher.city::string as publisher\_city,

rb.src:publisher.state::string as publisher\_state,

rb.src:publisher.country::string as publisher\_country,

-- nested array of objects. Object also has an array -- more complicated version

cnt.value:section::string as section,

chp.value::string as chapters

from raw\_books rb,

table(flatten(rb.src:publication\_years)) pyr,

table(flatten(rb.src:languages)) lng,

table(flatten(rb.src:contents)) cnt,

table(flatten(cnt.value:chapters)) chp;

-- select the relational form of JSON data

select \* from parsed\_books where name = 'Snowflake Fundamentals' and language = 'German' and publication\_year = 2018;

select \* from parsed\_books where name = 'Snowflake Advanced - Administrator' and language = 'English' and publication\_year = 2020;

select \* from parsed\_books where name = 'Snowflake Advanced - Architect' and language = 'English' and publication\_year = 2020;

select \* from parsed\_books where name = 'Snowflake Advanced - Data Engineer' and language = 'English' and publication\_year = 2021;

select \* from parsed\_books where name = 'Snowflake Advanced - Data Scientist' and language = 'English' and publication\_year = 2022;

-- cleanup

drop table raw\_books ;

drop table parsed\_books ;

drop stage ext\_stage\_ld ;

drop storage integration int\_gcp ;

drop file format file\_format\_json ;

-- Data - sample one record

-- {

-- "continent": "Europe",

-- "country": {

-- "city": [

-- "Paris",

-- "Nice",

-- "Marseilles",

-- "Cannes"

-- ],

-- "name": "France"

-- }

-- }

--create table to load the above parquet file

create or replace table raw\_cities (src variant);

-- Check the data

select \* from raw\_cities ;

-- Create integration object

create or replace storage integration int\_aws\_s3 -- s3://snowflakes3datafiles/csvfiles/

type = external\_stage

storage\_provider = s3

enabled = true

storage\_aws\_role\_arn = 'arn:aws:iam::755743318453:role/snowflake-s3-full-access-role'

storage\_allowed\_locations = ('s3://snowflakes3datafiles/parquetfiles/')

--storage\_blocked\_locations = ('<>')

comment = 'This is the integration object for loading the files from AWS S3 to Snowflake' ;

-- Describe

desc integration int\_aws\_s3 ;

-- create file format - since it will be used in external stage and copy command. so creating is better

create or replace file format file\_format\_parquet type = PARQUET ;

--describe file formats

describe file format file\_format\_parquet ;

-- create external stage for loading data

create or replace stage ext\_stage\_ld

url = 's3://snowflakes3datafiles/parquetfiles/'

storage\_integration = int\_aws\_s3

file\_format = file\_format\_parquet ;

-- Show stages

show stages ;

-- list

list @ext\_stage\_ld ;

-- check stage

select \* from @ext\_stage\_ld ;

--

-- Copy command

copy into raw\_cities from @ext\_stage\_ld

files = ('cities.parquet');

-- Parsing the CITIES data

--PARQUET DATA PARSING

select \* from raw\_cities ;

-- Handling Array - selecting the arrays

select src:continent::string, src:country:name::string as country\_name, src:country:city as cities from raw\_cities ;

-- FLATTEN (explodes compound into multiple)

-- table function which takes variant, object and arrat column and produces a lateral view

-- used for converting semi-structured to relational

-- input is expr -> object, array or variant. This will be converted to rows and will be a mandatory parameter

-- output is - (seq, key, path, index,value,this)

-- (generated number, key of exploded expression, path of the element, index if array, value, element )

-- Lets see the example

select rct.src:continent::string, rct.src:country:name::string as country\_name, r.value::string as cities

from raw\_cities rct, lateral flatten(input=> rct.src:country:city) r;

--- Create the final table from PARQUET file

create or replace table parsed\_cities as

select rct.src:continent::string as continent, rct.src:country:name::string as country\_name, r.value::string as cities

from raw\_cities rct, lateral flatten(input=> rct.src:country:city) r;

-- select the relational form of PARQUET data

select \* from parsed\_cities where continent = 'Europe' and country\_name = 'Greece' ;

select \* from parsed\_cities where continent = 'Europe' and country\_name = 'France' ;

select \* from parsed\_cities where continent = 'North America' and country\_name = 'Canada' ;

-- cleanup

drop table raw\_cities ;

drop table parsed\_cities ;

drop stage ext\_stage\_ld ;

drop storage integration int\_gcp ;

drop file format file\_format\_parquet ;