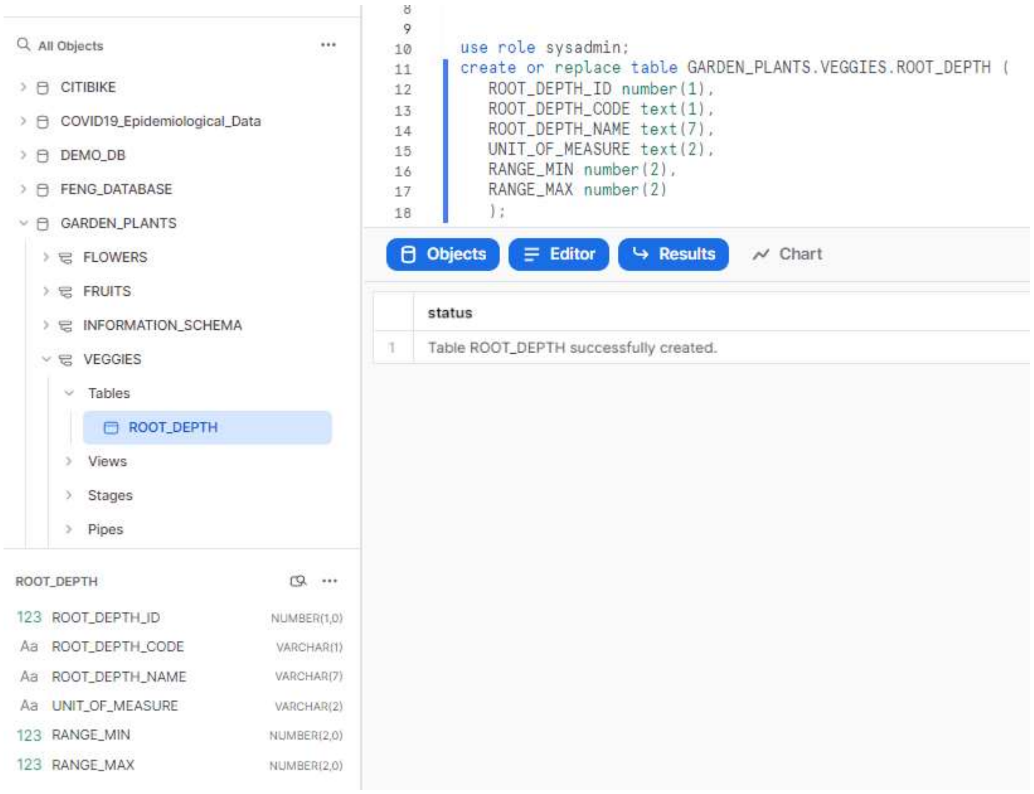
Badge 1: Data Warehousing

Sunday, December 11, 2022 9:38 PM

So for data type, you can use
Number(#)
Text(#)

Snowflake converts text(1) to varchar(1)
Snowflake converts number(1) to number(1,0)

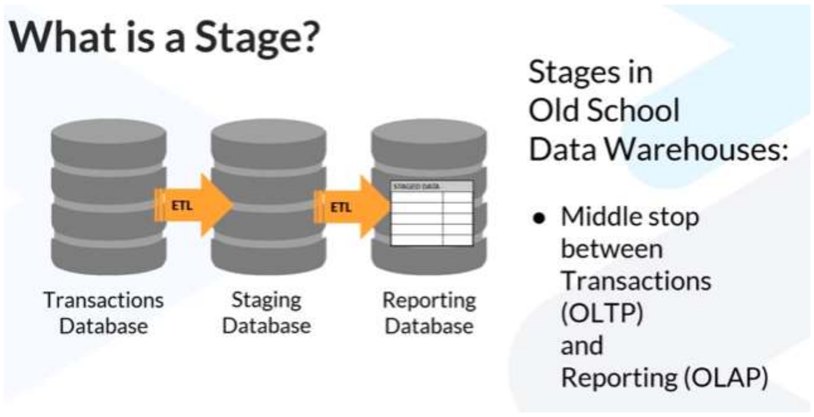


```
use role sysadmin;
create or replace table GARDEN_PLANTS.VEGGIES.ROOT_DEPTH (
    ROOT_DEPTH_ID number(1),
    ROOT_DEPTH_CODE text(1),
    ROOT_DEPTH_NAME text(7),
    UNIT_OF_MEASURE text(2),
    RANGE_MIN number(2),
    RANGE_MAX number(2)
    );
```

	status
1	Table ROOT_DEPTH successfully created.

ROOT_DEPTH

123	ROOT_DEPTH_ID	NUMBER(1,0)
Aa	ROOT_DEPTH_CODE	VARCHAR(1)
Aa	ROOT_DEPTH_NAME	VARCHAR(7)
Aa	UNIT_OF_MEASURE	VARCHAR(2)
123	RANGE_MIN	NUMBER(2,0)
123	RANGE_MAX	NUMBER(2,0)


What is a Stage?
ETL
ETL
STAGED DATA
Transactions Database
Staging Database
Reporting Database
Stages in Old School Data Warehouses:
● Middle stop between Transactions (OLTP) and Reporting (OLAP)

select 'hello';
select 'hello' as "greeting";
select 'hello' as greeting;

show databases;

```
show schemas;
show schemas in account;


use role sysadmin;
create or replace table GARDEN_PLANTS.VEGGIES.ROOT_DEPTH (
   ROOT_DEPTH_ID number(1),
   ROOT_DEPTH_CODE text(1),
   ROOT_DEPTH_NAME text(7),
   UNIT_OF_MEASURE text(2),
   RANGE_MIN number(2),
   RANGE_MAX number(2)
   );

USE WAREHOUSE COMPUTE_WH;

INSERT INTO ROOT_DEPTH (
        ROOT_DEPTH_ID ,
        ROOT_DEPTH_CODE ,
        ROOT_DEPTH_NAME ,
        UNIT_OF_MEASURE ,
        RANGE_MIN ,
        RANGE_MAX
)

VALUES
(
    1,
    'S',
    'Shallow',
    'cm',
    30,
    45
)
;

insert into root_depth (root_depth_id, root_depth_code
                    , root_depth_name, unit_of_measure
                    , range_min, range_max)
values
 (2,'M','Medium','cm',45,60)
,(3,'D','Deep','cm',60,90)
;
select * from root_depth;

update root_depth set unit_of_measure='cm' where root_depth_id=2;

create table vegetable_details
(
plant_name varchar(25)
, root_depth_code varchar(1)
);

select count(*) from vegetable_details;
```

```
PUT 'file:///fc356bff875005382a8382d5d2a3456f' '@~/uploads/dataloader';
copy into IDENTIFIER('"GARDEN_PLANTS"."VEGGIES"."VEGETABLE_DETAILS"') from
'@~/uploads/dataloader/fc356bff875005382a8382d5d2a3456f' file_format = (TYPE=csv,
FIELD_OPTIONALLY_ENCLOSED_BY='"', ESCAPE_UNENCLOSED_FIELD=None, SKIP_HEADER=1) purge =
true ON_ERROR=CONTINUE;

list @~/uploads/dataloader;

copy into IDENTIFIER('"GARDEN_PLANTS"."VEGGIES"."VEGETABLE_DETAILS"') from
'@~/uploads/dataloader/4a11b251af9407610c5093eb439a2a68' file_format = (TYPE=csv,
FIELD_OPTIONALLY_ENCLOSED_BY='"', ESCAPE_UNENCLOSED_FIELD=None, SKIP_HEADER=1,
FIELD_DELIMITER='|');

//
create file format garden_plants.veggies.PIPECOLSEP_ONEHEADROW
  TYPE = 'CSV'--csv is used for any flat file (tsv, pipe-separated, etc)
  FIELD_DELIMITER = '|' --pipes as column separators
  SKIP_HEADER = 1 --one header row to skip
  ;

create file format garden_plants.veggies.COMMASEP_DBLQUOT_ONEHEADROW
  TYPE = 'CSV'--csv for comma separated files
  SKIP_HEADER = 1 --one header row
  FIELD_OPTIONALLY_ENCLOSED_BY = '"' --this means that some values will be wrapped in double-
quotes bc they have commas in them
  ;


select * from vegetable_details;

delete from vegetable_details where plant_name='Spinach' and root_depth_code='D';


show schemas in account;

// Dora
use role accountadmin;
create or replace api integration dora_api_integration
api_provider = aws_api_gateway
api_aws_role_arn = 'arn:aws:iam::321463406630:role/snowflakeLearnerAssumedRole'
enabled = true
api_allowed_prefixes = ('https://awy6hshxy4.execute-api.us-west-2.amazonaws.com/dev/edu_dora');

use role accountadmin;
create or replace external function demo_db.public.grader(
     step varchar
  , passed boolean
  , actual integer
  , expected integer
  , description varchar)
returns variant
api_integration = dora_api_integration
context_headers = (current_timestamp,current_account, current_statement)
as 'https://awy6hshxy4.execute-api.us-west-2.amazonaws.com/dev/edu_dora/grader'
;
```

```
use role accountadmin;
use database demo_db; --change this to a different database if you prefer
use schema public; --change this to a different schema if you prefer

select grader(step, (actual = expected), actual, expected, description) as graded_results from
(SELECT
 'DORA_IS_WORKING' as step
 ,(select 123) as actual
 ,123 as expected
 ,'Dora is working!' as description
);

//
SELECT *
FROM GARDEN_PLANTS.INFORMATION_SCHEMA.SCHEMATA;

SELECT *
FROM GARDEN_PLANTS.INFORMATION_SCHEMA.SCHEMATA
where schema_name in ('FLOWERS','FRUITS','VEGGIES');

SELECT count(*) as SCHEMAS_FOUND, '3' as SCHEMAS_EXPECTED
FROM GARDEN_PLANTS.INFORMATION_SCHEMA.SCHEMATA
where schema_name in ('FLOWERS','FRUITS','VEGGIES');


--You may set these manually or you may edit the code on the next 3 lines
use database DEMO_DB;
use schema PUBLIC;
use role ACCOUNTADMIN;

--Do NOT EDIT ANYTHING BELOW THIS LINE
select GRADER(step, (actual = expected), actual, expected, description) as graded_results from (
 SELECT
 'DWW01' as step
 ,(select count(*)
   from GARDEN_PLANTS.INFORMATION_SCHEMA.SCHEMATA
   where schema_name in ('FLOWERS','VEGGIES','FRUITS')) as actual
  ,3 as expected
  ,'Created 3 Garden Plant schemas' as description
);

--You may set these manually or you may edit the code on the next 3 lines
use database DEMO_DB;
use schema PUBLIC;
use role ACCOUNTADMIN;

--Do NOT EDIT ANYTHING BELOW THIS LINE
select GRADER(step, (actual = expected), actual, expected, description) as graded_results from (
 SELECT 'DWW02' as step
 ,(select count(*)
   from GARDEN_PLANTS.INFORMATION_SCHEMA.SCHEMATA
   where schema_name = 'PUBLIC') as actual
 , 0 as expected
 ,'Deleted PUBLIC schema.' as description
);
```

```
-- Set your worksheet role to ACCOUNTADMIN
use role accountadmin;
-- Set your worksheet database to DEMO_DB
use database DEMO_DB;
-- Set your worksheet schema to PUBLIC
use schema PUBLIC;

--Do NOT EDIT ANYTHING BELOW THIS LINE
select GRADER(step, (actual = expected), actual, expected, description) as graded_results from (
 SELECT 'DWW03' as step
 ,(select count(*)
   from GARDEN_PLANTS.INFORMATION_SCHEMA.TABLES
   where table_name = 'ROOT_DEPTH') as actual
 , 1 as expected
 ,'ROOT_DEPTH Table Exists' as description
);

--Set your worksheet drop list role to ACCOUNTADMIN
--Set your worksheet drop list database and schema to the location of your GRADER function

-- DO NOT EDIT ANYTHING BELOW THIS LINE. THE CODE MUST BE RUN EXACTLY AS IT IS WRITTEN
select GRADER(step, (actual = expected), actual, expected, description) as graded_results from (
 SELECT 'DWW04' as step
 ,(select count(*) as SCHEMAS_FOUND
   from UTIL_DB.INFORMATION_SCHEMA.SCHEMATA) as actual
 , 2 as expected
 , 'UTIL_DB Schemas' as description
);


--Set your worksheet drop list role to ACCOUNTADMIN
--Set your worksheet drop list database and schema to the location of your GRADER function

-- DO NOT EDIT ANYTHING BELOW THIS LINE. THE CODE MUST BE RUN EXACTLY AS IT IS WRITTEN
select GRADER(step, (actual = expected), actual, expected, description) as graded_results from (
 SELECT 'DWW05' as step
 ,(select count(*)
   from GARDEN_PLANTS.INFORMATION_SCHEMA.TABLES
   where table_name = 'VEGETABLE_DETAILS') as actual
 , 1 as expected
 ,'VEGETABLE_DETAILS Table' as description
);



--Set your worksheet drop list role to ACCOUNTADMIN
--Set your worksheet drop list database and schema to the location of your GRADER function

-- DO NOT EDIT ANYTHING BELOW THIS LINE. THE CODE MUST BE RUN EXACTLY AS IT IS WRITTEN
select GRADER(step, (actual = expected), actual, expected, description) as graded_results from (
 SELECT 'DWW06' as step
,(select row_count
  from GARDEN_PLANTS.INFORMATION_SCHEMA.TABLES
  where table_name = 'ROOT_DEPTH') as actual
, 3 as expected
```

```
,'ROOT_DEPTH row count' as description
);


--Set your worksheet drop list role to ACCOUNTADMIN
--Set your worksheet drop list database and schema to the location of your GRADER function

-- DO NOT EDIT ANYTHING BELOW THIS LINE. THE CODE MUST BE RUN EXACTLY AS IT IS WRITTEN
select GRADER(step, (actual = expected), actual, expected, description) as graded_results from (
 SELECT 'DWW07' as step
 ,(select row_count
   from GARDEN_PLANTS.INFORMATION_SCHEMA.TABLES
   where table_name = 'VEGETABLE_DETAILS') as actual
 , 41 as expected
 , 'VEG_DETAILS row count' as description
);


--Set your worksheet drop list role to ACCOUNTADMIN
--Set your worksheet drop list database and schema to the location of your GRADER function

-- DO NOT EDIT ANYTHING BELOW THIS LINE. THE CODE MUST BE RUN EXACTLY AS IT IS WRITTEN
select GRADER(step, (actual = expected), actual, expected, description) as graded_results from (
   SELECT 'DWW08' as step
   ,(select count(*)
     from GARDEN_PLANTS.INFORMATION_SCHEMA.FILE_FORMATS
     where FIELD_DELIMITER =','
     and FIELD_OPTIONALLY_ENCLOSED_BY ='"') as actual
   , 1 as expected
   , 'File Format 1 Exists' as description
);


--Set your worksheet drop list role to ACCOUNTADMIN
--Set your worksheet drop list database and schema to the location of your GRADER function

-- DO NOT EDIT ANYTHING BELOW THIS LINE. THE CODE MUST BE RUN EXACTLY AS IT IS WRITTEN
select GRADER(step, (actual = expected), actual, expected, description) as graded_results from (
 SELECT 'DWW09' as step
 ,(select count(*)
   from GARDEN_PLANTS.INFORMATION_SCHEMA.FILE_FORMATS
   where FIELD_DELIMITER ='|'
   ) as actual
 , 1 as expected
 ,'File Format 2 Exists' as description
);

use role sysadmin;
create stage garden_plants.veggies.like_a_window_into_an_s3_bucket
 url = 's3://uni-lab-files';

 --Set your worksheet drop list role to ACCOUNTADMIN
--Set your worksheet drop list database and schema to the location of your GRADER function

-- DO NOT EDIT ANYTHING BELOW THIS LINE. THE CODE MUST BE RUN EXACTLY AS IT IS WRITTEN
select GRADER(step, (actual = expected), actual, expected, description) as graded_results from (
 SELECT 'DWW10' as step
  ,(select count(*)
    from GARDEN_PLANTS.INFORMATION_SCHEMA.stages
    where stage_url='s3://uni-lab-files'
    and stage_type='External Named') as actual
  , 1 as expected
  , 'External stage created' as description
 );

 create or replace table vegetable_details_soil_type
( plant_name varchar(25)
 ,soil_type number(1,0)
);

copy into vegetable_details_soil_type
from @like_a_window_into_an_s3_bucket
files = ( 'VEG_NAME_TO_SOIL_TYPE_PIPE.txt')
file_format = ( format_name=PIPECOLSEP_ONEHEADROW );


--Set your worksheet drop list role to ACCOUNTADMIN
--Set your worksheet drop list database and schema to the location of your GRADER function

-- DO NOT EDIT ANYTHING BELOW THIS LINE. THE CODE MUST BE RUN EXACTLY AS IT IS WRITTEN
select GRADER(step, (actual = expected), actual, expected, description) as graded_results from (
  SELECT 'DWW11' as step
  ,(select row_count
    from GARDEN_PLANTS.INFORMATION_SCHEMA.TABLES
    where table_name = 'VEGETABLE_DETAILS_SOIL_TYPE') as actual
  , 42 as expected
  , 'Veg Det Soil Type Count' as description
 );


create or replace table LU_SOIL_TYPE(
SOIL_TYPE_ID number,
SOIL_TYPE varchar(15),
SOIL_DESCRIPTION varchar(75)
 );

-- create file format:  L8_CHALLENGE_FF
create or replace file format garden_plants.veggies.L8_CHALLENGE_FF
    TYPE = 'CSV'--csv is used for any flat file (tsv, pipe-separated, etc)
    FIELD_DELIMITER = '\t' --pipes as column separators
    SKIP_HEADER = 1 --one header row to skip
    TRIM_SPACE = true
    ;

--create file format garden_plants.veggies.COMMASEP_DBLQUOT_ONEHEADROW
--    TYPE = 'CSV'--csv for comma separated files
--    SKIP_HEADER = 1 --one header row
--    FIELD_OPTIONALLY_ENCLOSED_BY = '"' --this means that some values will be wrapped in double-
quotes bc they have commas in them
--    ;

copy into LU_SOIL_TYPE
```

```
from @like_a_window_into_an_s3_bucket
files = ( 'LU_SOIL_TYPE.tsv')
file_format = ( format_name=L8_CHALLENGE_FF);


--
create or replace table VEGETABLE_DETAILS_PLANT_HEIGHT (
   plant_name text(32),
   UOM text(1),
   Low_End_of_Range number(2),
   High_End_of_Range number(2)
);
copy into VEGETABLE_DETAILS_PLANT_HEIGHT
from @like_a_window_into_an_s3_bucket
files = ( 'veg_plant_height.csv')
file_format = ( format_name=COMMASEP_DBLQUOT_ONEHEADROW);


--Set your worksheet drop list role to ACCOUNTADMIN
--Set your worksheet drop list database and schema to the location of your GRADER function

-- DO NOT EDIT ANYTHING BELOW THIS LINE. THE CODE MUST BE RUN EXACTLY AS IT IS WRITTEN
select GRADER(step, (actual = expected), actual, expected, description) as graded_results from (
     SELECT 'DWW12' as step
     ,(select row_count
       from GARDEN_PLANTS.INFORMATION_SCHEMA.TABLES
       where table_name = 'VEGETABLE_DETAILS_PLANT_HEIGHT') as actual
     , 41 as expected
     , 'Veg Detail Plant Height Count' as description
);


--Set your worksheet drop list role to ACCOUNTADMIN
--Set your worksheet drop list database and schema to the location of your GRADER function

-- DO NOT EDIT ANYTHING BELOW THIS LINE. THE CODE MUST BE RUN EXACTLY AS IT IS WRITTEN
select GRADER(step, (actual = expected), actual, expected, description) as graded_results from (
    SELECT 'DWW13' as step
    ,(select row_count
      from GARDEN_PLANTS.INFORMATION_SCHEMA.TABLES
      where table_name = 'LU_SOIL_TYPE') as actual
    , 8 as expected
    ,'Soil Type Look Up Table' as description
);


-- Set your worksheet drop lists
-- DO NOT EDIT THE CODE
select GRADER(step, (actual = expected), actual, expected, description) as graded_results from (
    SELECT 'DWW14' as step
    ,(select count(*)
      from GARDEN_PLANTS.INFORMATION_SCHEMA.FILE_FORMATS
      where FILE_FORMAT_NAME='L8_CHALLENGE_FF'
      and FIELD_DELIMITER = '\t') as actual
    , 1 as expected
    ,'Challenge File Format Created' as description
);

--
use role sysadmin;

// Create a new database and set the context to use the new database
CREATE DATABASE LIBRARY_CARD_CATALOG COMMENT = 'DWW Lesson 9 ';
USE DATABASE LIBRARY_CARD_CATALOG;

// Create and Author table
CREATE OR REPLACE TABLE AUTHOR (
   AUTHOR_UID NUMBER
  ,FIRST_NAME VARCHAR(50)
  ,MIDDLE_NAME VARCHAR(50)
  ,LAST_NAME VARCHAR(50)
);

// Insert the first two authors into the Author table
INSERT INTO AUTHOR(AUTHOR_UID,FIRST_NAME,MIDDLE_NAME, LAST_NAME)
Values
(1, 'Fiona', '','Macdonald')
,(2, 'Gian','Paulo','Faleschini');

// Look at your table with it's new rows
SELECT *
FROM AUTHOR;

create sequence SEQ_AUTHOR_UID
    start = 1
    increment = 1
    comment = 'use this to fill in AUTHOR_UID';

use role sysadmin;

//See how the nextval function works
SELECT SEQ_AUTHOR_UID.nextval;
show sequences;


use role sysadmin;

//Drop and recreate the counter (sequence) so that it starts at 3
// then we'll add the other author records to our author table
CREATE OR REPLACE SEQUENCE "LIBRARY_CARD_CATALOG"."PUBLIC"."SEQ_AUTHOR_UID"
START 3
INCREMENT 1
COMMENT = 'Use this to fill in the AUTHOR_UID every time you add a row';

//Add the remaining author records and use the nextval function instead
//of putting in the numbers
INSERT INTO AUTHOR(AUTHOR_UID,FIRST_NAME,MIDDLE_NAME, LAST_NAME)
Values
(SEQ_AUTHOR_UID.nextval, 'Laura', 'K','Egendorf')
,(SEQ_AUTHOR_UID.nextval, 'Jan', '','Grover')
,(SEQ_AUTHOR_UID.nextval, 'Jennifer', '','Clapp')
,(SEQ_AUTHOR_UID.nextval, 'Kathleen', '','Petelinsek');
```

```
--
USE DATABASE LIBRARY_CARD_CATALOG;

// Create a new sequence, this one will be a counter for the book table
CREATE OR REPLACE SEQUENCE "LIBRARY_CARD_CATALOG"."PUBLIC"."SEQ_BOOK_UID"
START 1
INCREMENT 1
COMMENT = 'Use this to fill in the BOOK_UID everytime you add a row';

// Create the book table and use the NEXTVAL as the
// default value each time a row is added to the table
CREATE OR REPLACE TABLE BOOK
( BOOK_UID NUMBER DEFAULT SEQ_BOOK_UID.nextval
 ,TITLE VARCHAR(50)
 ,YEAR_PUBLISHED NUMBER(4,0)
);

// Insert records into the book table
// You don't have to list anything for the
// BOOK_UID field because the default setting
// will take care of it for you
INSERT INTO BOOK(TITLE,YEAR_PUBLISHED)
VALUES
 ('Food',2001)
,('Food',2006)
,('Food',2008)
,('Food',2016)
,('Food',2015);

// Create the relationships table
// this is sometimes called a "Many-to-Many table"
CREATE TABLE BOOK_TO_AUTHOR
(  BOOK_UID NUMBER
  ,AUTHOR_UID NUMBER
);

//Insert rows of the known relationships
INSERT INTO BOOK_TO_AUTHOR(BOOK_UID,AUTHOR_UID)
VALUES
 (1,1) // This row links the 2001 book to Fiona Macdonald
,(1,2) // This row links the 2001 book to Gian Paulo Faleschini
,(2,3) // Links 2006 book to Laura K Egendorf
,(3,4) // Links 2008 book to Jan Grover
,(4,5) // Links 2016 book to Jennifer Clapp
,(5,6);// Links 2015 book to Kathleen Petelinsek


//Check your work by joining the 3 tables together
//You should get 1 row for every author
select *
from book_to_author ba
join author a
on ba.author_uid = a.author_uid
```

```
join book b
on b.book_uid=ba.book_uid;

-- Set your worksheet drop lists
-- DO NOT EDIT THE CODE
select GRADER(step, (actual = expected), actual, expected, description) as graded_results from (
    SELECT 'DWW15' as step
    ,(select count(*)
     from LIBRARY_CARD_CATALOG.PUBLIC.Book_to_Author ba
     join LIBRARY_CARD_CATALOG.PUBLIC.author a
     on ba.author_uid = a.author_uid
     join LIBRARY_CARD_CATALOG.PUBLIC.book b
     on b.book_uid=ba.book_uid) as actual
    , 6 as expected
    , '3NF DB was Created.' as description
);



// Create an Ingestion Table for XML Data
CREATE TABLE LIBRARY_CARD_CATALOG.PUBLIC.AUTHOR_INGEST_XML
(
  "RAW_AUTHOR" VARIANT
);
//Create File Format for XML Data
CREATE FILE FORMAT LIBRARY_CARD_CATALOG.PUBLIC.XML_FILE_FORMAT
TYPE = 'XML'
STRIP_OUTER_ELEMENT = FALSE
;

list @GARDEN_PLANTS.VEGGIES.LIKE_A_WINDOW_INTO_AN_S3_BUCKET;


copy into AUTHOR_INGEST_XML
from @GARDEN_PLANTS.VEGGIES.like_a_window_into_an_s3_bucket
files = ( 'author_with_header.xml')
file_format = ( format_name=LIBRARY_CARD_CATALOG.PUBLIC.XML_FILE_FORMAT);
select * from AUTHOR_INGEST_XML;

copy into AUTHOR_INGEST_XML
from @GARDEN_PLANTS.VEGGIES.like_a_window_into_an_s3_bucket
files = ( 'author_no_header.xml')
file_format = ( format_name=LIBRARY_CARD_CATALOG.PUBLIC.XML_FILE_FORMAT);
select * from AUTHOR_INGEST_XML;

CREATE OR REPLACE FILE FORMAT LIBRARY_CARD_CATALOG.PUBLIC.XML_FILE_FORMAT
TYPE = 'XML'
COMPRESSION = 'AUTO'
PRESERVE_SPACE = FALSE
STRIP_OUTER_ELEMENT = TRUE
DISABLE_SNOWFLAKE_DATA = FALSE
DISABLE_AUTO_CONVERT = FALSE
IGNORE_UTF8_ERRORS = FALSE;

truncate table AUTHOR_INGEST_XML;
select * from AUTHOR_INGEST_XML;
```

```
copy into AUTHOR_INGEST_XML
from @GARDEN_PLANTS.VEGGIES.like_a_window_into_an_s3_bucket
files = ( 'author_with_header.xml')
file_format = ( format_name=LIBRARY_CARD_CATALOG.PUBLIC.XML_FILE_FORMAT);
select * from AUTHOR_INGEST_XML;

//Returns entire record
SELECT raw_author
FROM author_ingest_xml;

// Presents a kind of meta-data view of the data
SELECT raw_author:"$"
FROM author_ingest_xml;

//shows the root or top-level object name of each row
SELECT raw_author:"@"
FROM author_ingest_xml;

//returns AUTHOR_UID value from top-level object's attribute
SELECT raw_author:"@AUTHOR_UID"
FROM author_ingest_xml;

//returns value of NESTED OBJECT called FIRST_NAME
SELECT XMLGET(raw_author, 'FIRST_NAME'):"$"
FROM author_ingest_xml;

//returns the data in a way that makes it look like a normalized table
SELECT
raw_author:"@AUTHOR_UID" as AUTHOR_ID
,XMLGET(raw_author, 'FIRST_NAME'):"$" as FIRST_NAME
,XMLGET(raw_author, 'MIDDLE_NAME'):"$" as MIDDLE_NAME
,XMLGET(raw_author, 'LAST_NAME'):"$" as LAST_NAME
FROM AUTHOR_INGEST_XML;

//add ::STRING to cast the values into strings and get rid of the quotes
SELECT
raw_author:"@AUTHOR_UID" as AUTHOR_ID
,XMLGET(raw_author, 'FIRST_NAME'):"$"::STRING as FIRST_NAME
,XMLGET(raw_author, 'MIDDLE_NAME'):"$"::STRING as MIDDLE_NAME
,XMLGET(raw_author, 'LAST_NAME'):"$"::STRING as LAST_NAME
FROM AUTHOR_INGEST_XML;


// JSON DDL Scripts
USE LIBRARY_CARD_CATALOG;

// Create an Ingestion Table for JSON Data
CREATE or replace TABLE "LIBRARY_CARD_CATALOG"."PUBLIC"."AUTHOR_INGEST_JSON"
(
  "RAW_AUTHOR" variant
);
//Create File Format for JSON Data
CREATE or replace FILE FORMAT LIBRARY_CARD_CATALOG.PUBLIC.JSON_FILE_FORMAT
TYPE = 'JSON'
COMPRESSION = 'AUTO'
```

```
ENABLE_OCTAL = FALSE
ALLOW_DUPLICATE = FALSE
STRIP_OUTER_ARRAY = TRUE
STRIP_NULL_VALUES = FALSE
IGNORE_UTF8_ERRORS = FALSE;

list @GARDEN_PLANTS.VEGGIES.LIKE_A_WINDOW_INTO_AN_S3_BUCKET;

truncate table AUTHOR_INGEST_JSON;
copy into AUTHOR_INGEST_JSON
from @GARDEN_PLANTS.VEGGIES.like_a_window_into_an_s3_bucket
files = ( 'author_with_header.json')
file_format = ( format_name=LIBRARY_CARD_CATALOG.PUBLIC.JSON_FILE_FORMAT);
select * from AUTHOR_INGEST_JSON;


//returns AUTHOR_UID value from top-level object's attribute
select raw_author:AUTHOR_UID
from author_ingest_json;

//returns the data in a way that makes it look like a normalized table
SELECT
 raw_author:AUTHOR_UID
,raw_author:FIRST_NAME::STRING as FIRST_NAME
,raw_author:MIDDLE_NAME::STRING as MIDDLE_NAME
,raw_author:LAST_NAME::STRING as LAST_NAME
FROM AUTHOR_INGEST_JSON;


-- Set your worksheet drop lists. DO NOT EDIT THE DORA CODE.
select GRADER(step, (actual = expected), actual, expected, description) as graded_results from
(
  SELECT 'DWW16' as step
  ,(select row_count
    from LIBRARY_CARD_CATALOG.INFORMATION_SCHEMA.TABLES
    where table_name = 'AUTHOR_INGEST_JSON') as actual
  ,6 as expected
  ,'Check number of rows' as description
 );


 //
 // Create an Ingestion Table for the NESTED JSON Data
CREATE OR REPLACE TABLE LIBRARY_CARD_CATALOG.PUBLIC.NESTED_INGEST_JSON
(
  "RAW_NESTED_BOOK" VARIANT
);
list @GARDEN_PLANTS.VEGGIES.LIKE_A_WINDOW_INTO_AN_S3_BUCKET;

copy into NESTED_INGEST_JSON
from @GARDEN_PLANTS.VEGGIES.like_a_window_into_an_s3_bucket
files = ('json_book_author_nested.json')
file_format = ( format_name=LIBRARY_CARD_CATALOG.PUBLIC.JSON_FILE_FORMAT);
select * from NESTED_INGEST_JSON;
```

```
//a few simple queries
SELECT RAW_NESTED_BOOK
FROM NESTED_INGEST_JSON;

SELECT RAW_NESTED_BOOK:year_published
FROM NESTED_INGEST_JSON;

SELECT RAW_NESTED_BOOK:authors
FROM NESTED_INGEST_JSON;

//try changing the number in the bracketsd to return authors from a different row
SELECT RAW_NESTED_BOOK:authors[0].first_name
FROM NESTED_INGEST_JSON;

//Use these example flatten commands to explore flattening the nested book and author data
SELECT RAW_NESTED_BOOK:authors    // there are 5 rows. first row has two elements in an array [ {...},
{...}]
FROM NESTED_INGEST_JSON;
SELECT value                  // flatten removes the outer array. and put each element to separate rows.
FROM NESTED_INGEST_JSON
,LATERAL FLATTEN(input => RAW_NESTED_BOOK:authors);

SELECT value:first_name
FROM NESTED_INGEST_JSON
,LATERAL FLATTEN(input => RAW_NESTED_BOOK:authors);

SELECT value:first_name
FROM NESTED_INGEST_JSON
,table(flatten(RAW_NESTED_BOOK:authors));

//Add a CAST command to the fields returned
SELECT value:first_name::VARCHAR, value:last_name::VARCHAR
FROM NESTED_INGEST_JSON
,LATERAL FLATTEN(input => RAW_NESTED_BOOK:authors);

//Assign new column  names to the columns using "AS"
SELECT value:first_name::VARCHAR AS FIRST_NM
, value:last_name::VARCHAR AS LAST_NM
FROM NESTED_INGEST_JSON
,LATERAL FLATTEN(input => RAW_NESTED_BOOK:authors);


select GRADER(step, (actual = expected), actual, expected, description) as graded_results from (
   SELECT 'DWW17' as step
    ,(select row_count
      from LIBRARY_CARD_CATALOG.INFORMATION_SCHEMA.TABLES
      where table_name = 'NESTED_INGEST_JSON') as actual
    , 5 as expected
    ,'Check number of rows' as description
);

//
CREATE DATABASE SOCIAL_MEDIA_FLOODGATES
COMMENT = 'There\'s so much data from social media - flood warning';

USE DATABASE SOCIAL_MEDIA_FLOODGATES;
```

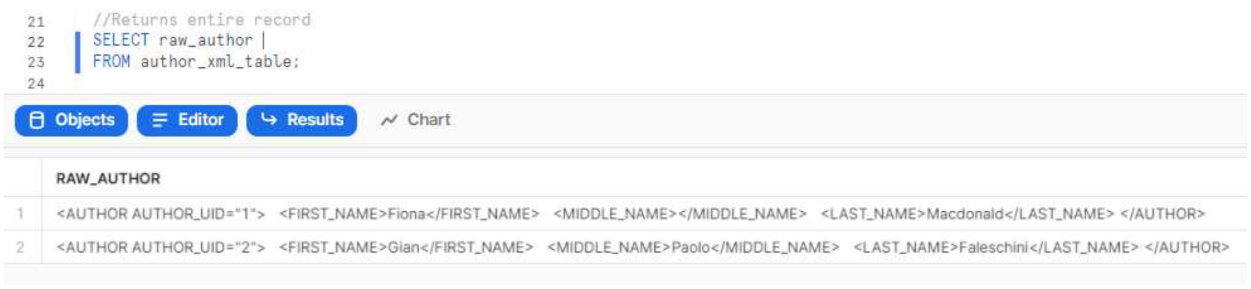
```
//Create a table in the new database
CREATE TABLE SOCIAL_MEDIA_FLOODGATES.PUBLIC.TWEET_INGEST
("RAW_STATUS" VARIANT)
COMMENT = 'Bring in tweets, one row per tweet or status entity';

//Create a JSON file format in the new database
CREATE FILE FORMAT SOCIAL_MEDIA_FLOODGATES.PUBLIC.JSON_FILE_FORMAT
TYPE = 'JSON'
COMPRESSION = 'AUTO'
ENABLE_OCTAL = FALSE
ALLOW_DUPLICATE = FALSE
STRIP_OUTER_ARRAY = TRUE
STRIP_NULL_VALUES = FALSE
IGNORE_UTF8_ERRORS = FALSE;

list @GARDEN_PLANTS.VEGGIES.like_a_window_into_an_s3_bucket;
copy into SOCIAL_MEDIA_FLOODGATES.PUBLIC.TWEET_INGEST
from @GARDEN_PLANTS.VEGGIES.like_a_window_into_an_s3_bucket
files = ('nutrition_tweets.json')
file_format = ( format_name=SOCIAL_MEDIA_FLOODGATES.PUBLIC.JSON_FILE_FORMAT);
select * from SOCIAL_MEDIA_FLOODGATES.PUBLIC.TWEET_INGEST;


//select statements as seen in the video
SELECT RAW_STATUS
FROM TWEET_INGEST;

SELECT RAW_STATUS:entities
FROM TWEET_INGEST;

SELECT RAW_STATUS:entities:hashtags
FROM TWEET_INGEST;

//Explore looking at specific hashtags by adding bracketed numbers
//This query returns just the first hashtag in each tweet
SELECT RAW_STATUS:entities:hashtags[0].text
FROM TWEET_INGEST;

//This version adds a WHERE clause to get rid of any tweet that
//doesn't include any hashtags
SELECT RAW_STATUS:entities:hashtags[0].text
FROM TWEET_INGEST
WHERE RAW_STATUS:entities:hashtags[0].text is not null;

//Perform a simple CAST on the created_at key
//Add an ORDER BY clause to sort by the tweet's creation date
SELECT RAW_STATUS:created_at::DATE
FROM TWEET_INGEST
ORDER BY RAW_STATUS:created_at::DATE;

//Flatten statements that return the whole hashtag entity
SELECT value
FROM TWEET_INGEST
,LATERAL FLATTEN
(input => RAW_STATUS:entities:hashtags);
```

```
SELECT value
FROM TWEET_INGEST
,TABLE(FLATTEN(RAW_STATUS:entities:hashtags)); // table function flatten.
SELECT *
FROM TWEET_INGEST,TABLE(FLATTEN(TWEET_INGEST.RAW_STATUS:entities:hashtags));

//Flatten statement that restricts the value to just the TEXT of the hashtag
SELECT value:text
FROM TWEET_INGEST
,LATERAL FLATTEN
(input => RAW_STATUS:entities:hashtags);


//Flatten and return just the hashtag text, CAST the text as VARCHAR
SELECT value:text::VARCHAR
FROM TWEET_INGEST
,LATERAL FLATTEN
(input => RAW_STATUS:entities:hashtags);

//Flatten and return just the hashtag text, CAST the text as VARCHAR
// Use the AS command to name the column
SELECT value:text::VARCHAR AS THE_HASHTAG
FROM TWEET_INGEST
,LATERAL FLATTEN
(input => RAW_STATUS:entities:hashtags);

//Add the Tweet ID and User ID to the returned table
SELECT RAW_STATUS:user:id AS USER_ID
,RAW_STATUS:id AS TWEET_ID
,value:text::VARCHAR AS HASHTAG_TEXT
FROM TWEET_INGEST
,LATERAL FLATTEN
(input => RAW_STATUS:entities:hashtags);



-- Set your worksheet drop lists. DO NOT EDIT THE DORA CODE.
select GRADER(step, (actual = expected), actual, expected, description) as graded_results from
(
   SELECT 'DWW18' as step
  ,(select row_count
    from SOCIAL_MEDIA_FLOODGATES.INFORMATION_SCHEMA.TABLES
    where table_name = 'TWEET_INGEST') as actual
  , 9 as expected
  ,'Check number of rows' as description
 );



create or replace view SOCIAL_MEDIA_FLOODGATES.PUBLIC.HASHTAGS_NORMALIZED as
(SELECT RAW_STATUS:user:id AS USER_ID
,RAW_STATUS:id AS TWEET_ID
,value:text::VARCHAR AS HASHTAG_TEXT
FROM TWEET_INGEST
,LATERAL FLATTEN
(input => RAW_STATUS:entities:hashtags)
);


select GRADER(step, (actual = expected), actual, expected, description) as graded_results from
(
 SELECT 'DWW19' as step
 ,(select count(*)
   from SOCIAL_MEDIA_FLOODGATES.INFORMATION_SCHEMA.VIEWS
   where table_name = 'HASHTAGS_NORMALIZED') as actual
 , 1 as expected
 ,'Check number of rows' as description
 );
```

################################

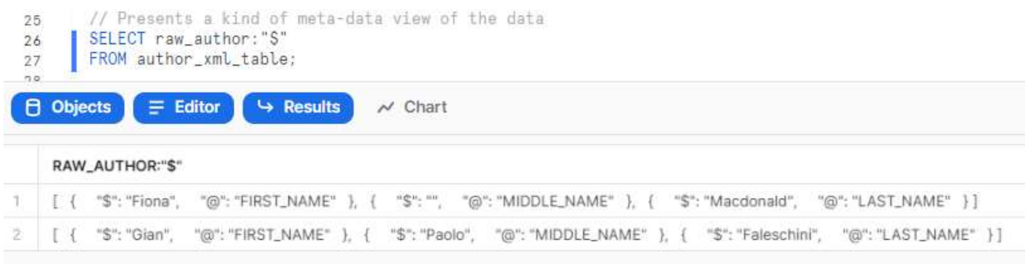
```
<?xml version='1.0' encoding='UTF-8'?>

<dataset>

 <AUTHOR AUTHOR_UID = 1>
   <FIRST_NAME>Fiona</FIRST_NAME>
   <MIDDLE_NAME/>
   <LAST_NAME>Macdonald</LAST_NAME>
 </AUTHOR>
 <AUTHOR AUTHOR_UID = 2>
   <FIRST_NAME>Gian</FIRST_NAME>
   <MIDDLE_NAME>Paolo</MIDDLE_NAME>
   <LAST_NAME>Faleschini</LAST_NAME>
 </AUTHOR>
</dataset>
```

$: element's value
https://community.snowflake.com/s/article/HOW-TO-QUERY-NESTED-XML-DATA-IN-SNOWFLAKE



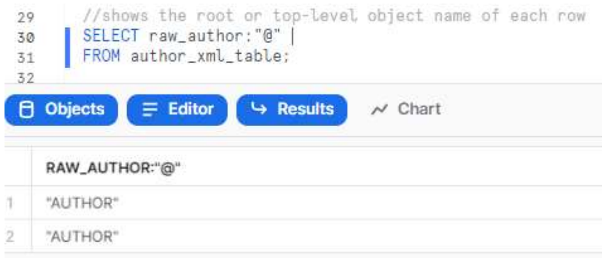
```
//Returns entire record
SELECT raw_author
FROM author_xml_table;
```

	RAW_AUTHOR
1	<AUTHOR AUTHOR_UID="1"> <FIRST_NAME>Fiona</FIRST_NAME> <MIDDLE_NAME></MIDDLE_NAME> <LAST_NAME>Macdonald</LAST_NAME> </AUTHOR>
2	<AUTHOR AUTHOR_UID="2"> <FIRST_NAME>Gian</FIRST_NAME> <MIDDLE_NAME>Paolo</MIDDLE_NAME> <LAST_NAME>Faleschini</LAST_NAME> </AUTHOR>



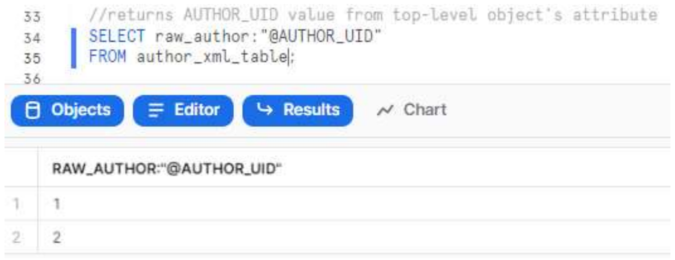
```
// Presents a kind of meta-data view of the data
SELECT raw_author:"$"
FROM author_xml_table;
```

	RAW_AUTHOR:"$"
1	[{ "$": "Fiona", "@": "FIRST_NAME" }, { "$": "", "@": "MIDDLE_NAME" }, { "$": "Macdonald", "@": "LAST_NAME" }]
2	[{ "$": "Gian", "@": "FIRST_NAME" }, { "$": "Paolo", "@": "MIDDLE_NAME" }, { "$": "Faleschini", "@": "LAST_NAME" }]



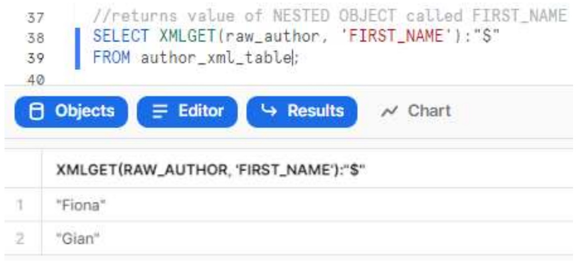
```
//shows the root or top-level object name of each row
SELECT raw_author:"@" |
FROM author_xml_table;
```

	RAW_AUTHOR:"@"
1	"AUTHOR"
2	"AUTHOR"



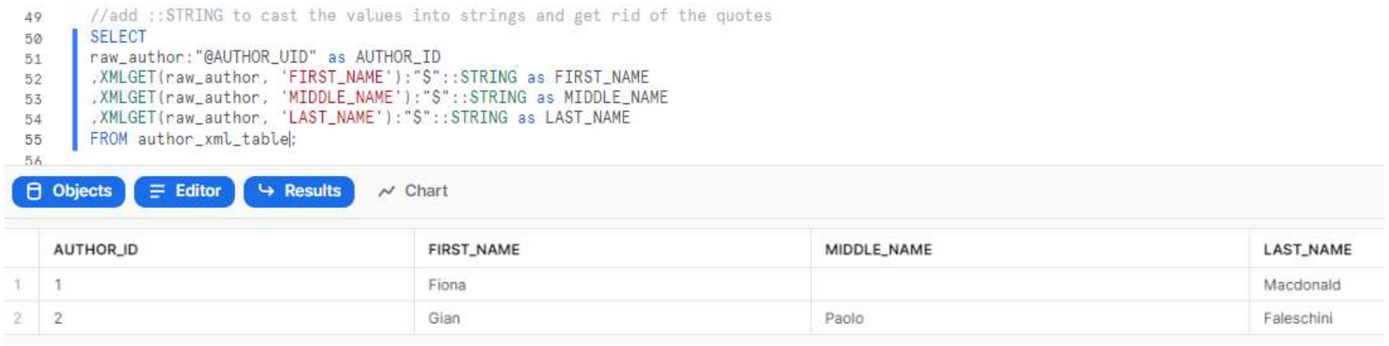
```
//returns AUTHOR_UID value from top-level object's attribute
SELECT raw_author:"@AUTHOR_UID"
FROM author_xml_table;
```

	RAW_AUTHOR:"@AUTHOR_UID"
1	1
2	2



```
//returns value of NESTED OBJECT called FIRST_NAME
SELECT XMLGET(raw_author, 'FIRST_NAME'):"$"
FROM author_xml_table;
```

	XMLGET(RAW_AUTHOR, 'FIRST_NAME'):"$"
1	"Fiona"
2	"Gian"



```
//add ::STRING to cast the values into strings and get rid of the quotes
SELECT
raw_author:"@AUTHOR_UID" as AUTHOR_ID
,XMLGET(raw_author, 'FIRST_NAME'):"$"::STRING as FIRST_NAME
,XMLGET(raw_author, 'MIDDLE_NAME'):"$"::STRING as MIDDLE_NAME
,XMLGET(raw_author, 'LAST_NAME'):"$"::STRING as LAST_NAME
FROM author_xml_table;
```

	AUTHOR_ID	FIRST_NAME	MIDDLE_NAME	LAST_NAME
1	1	Fiona		Macdonald
2	2	Gian	Paolo	Faleschini

```
-- XML
create or replace stage xml_json_stage;
list @xml_json_stage;

create or replace table author_xml_table
(
  raw_author variant
);

create or replace file format xml_file_format
    type='xml'
    strip_outer_element = true;

copy into author_xml_table
    from @xml_json_stage
    files = ('author.xml.gz')
    file_format=(format_name = 'xml_file_format');

select * from author_xml_table;
```

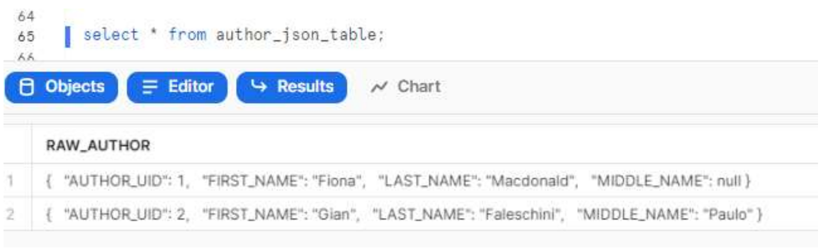
```
//Returns entire record
SELECT raw_author
FROM author_xml_table;

// Presents a kind of meta-data view of the data
SELECT raw_author:"$"
FROM author_xml_table;

//shows the root or top-level object name of each row
SELECT raw_author:"@"
FROM author_xml_table;

//returns AUTHOR_UID value from top-level object's attribute
SELECT raw_author:"@AUTHOR_UID"
FROM author_xml_table;

//returns value of NESTED OBJECT called FIRST_NAME
SELECT XMLGET(raw_author, 'FIRST_NAME'):"$"
FROM author_xml_table;

//add ::STRING to cast the values into strings and get rid of the quotes
SELECT
raw_author:"@AUTHOR_UID" as AUTHOR_ID
,XMLGET(raw_author, 'FIRST_NAME'):"$"::STRING as FIRST_NAME
,XMLGET(raw_author, 'MIDDLE_NAME'):"$"::STRING as MIDDLE_NAME
,XMLGET(raw_author, 'LAST_NAME'):"$"::STRING as LAST_NAME
FROM author_xml_table;
```

############## JSON

```
[{
"AUTHOR_UID":1,
"FIRST_NAME":"Fiona",
"MIDDLE_NAME":null,
"LAST_NAME":"Macdonald"
},
{
"AUTHOR_UID":2,
"FIRST_NAME":"Gian",
"MIDDLE_NAME":"Paulo",
"LAST_NAME":"Faleschini"
}]
```

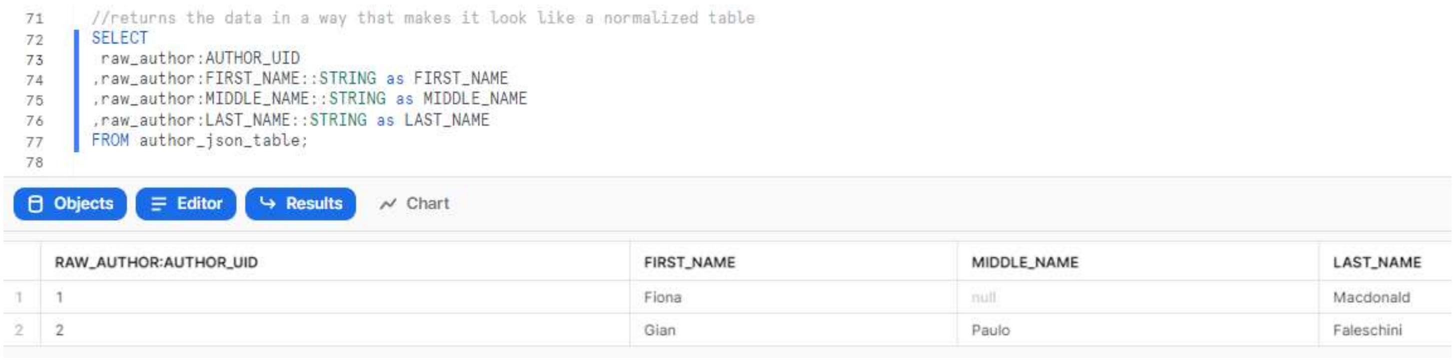
```
Type SQL statements or !help
forrestli2011#(no warehouse)@(no database).(no schema)>put file://author.xml
@feng_database.feng_schema.xml_json_stage;
+------------+---------------+-------------+-------------+--------------------+--------------------+----------+---------+
| source     | target        | source_size | target_size | source_compression | target_compression | status
| message |
|------------+---------------+-------------+-------------+--------------------+--------------------+----------+---------|
| author.xml | author.xml.gz |         348 |         208 | NONE               | GZIP               | UPLOADED |         |
+------------+---------------+-------------+-------------+--------------------+--------------------+----------+---------+
1 Row(s) produced. Time Elapsed: 1.051s
forrestli2011#(no warehouse)@(no database).(no schema)>put file://author_full.json
@feng_database.feng_schema.xml_json_stage;
+------------------+---------------------+-------------+-------------+--------------------+--------------------+----------+---------+
| source           | target              | source_size | target_size | source_compression | target_compression |
status   | message |
|------------------+---------------------+-------------+-------------+--------------------+--------------------+----------+---------|
| author_full.json | author_full.json.gz |         554 |         240 | NONE               | GZIP               | UPLOADED
|         |
+------------------+---------------------+-------------+-------------+--------------------+--------------------+----------+---------+
1 Row(s) produced. Time Elapsed: 1.304s
forrestli2011#(no warehouse)@(no database).(no schema)>put file://author.json
@feng_database.feng_schema.xml_json_stage;
+-------------+----------------+-------------+-------------+--------------------+--------------------+----------+---------+
| source      | target         | source_size | target_size | source_compression | target_compression | status
| message |
|-------------+----------------+-------------+-------------+--------------------+--------------------+----------+---------|
| author.json | author.json.gz |         188 |         144 | NONE               | GZIP               | UPLOADED |         |
+-------------+----------------+-------------+-------------+--------------------+--------------------+----------+---------+
1 Row(s) produced. Time Elapsed: 0.994s
forrestli2011#(no warehouse)@(no database).(no schema)>
```

```
create or replace table author_json_table
(
 raw_author variant
);

create or replace file format json_file_format
   type='json'
   strip_outer_array = true;

copy into author_json_table
   from @xml_json_stage
   files = ('author.json.gz')
   file_format=(format_name = 'json_file_format');
```



```
select * from author_json_table;
```

	RAW_AUTHOR
1	{ "AUTHOR_UID": 1, "FIRST_NAME": "Fiona", "LAST_NAME": "Macdonald", "MIDDLE_NAME": null }
2	{ "AUTHOR_UID": 2, "FIRST_NAME": "Gian", "LAST_NAME": "Faleschini", "MIDDLE_NAME": "Paulo" }



```
//returns the data in a way that makes it look like a normalized table
SELECT
 raw_author:AUTHOR_UID
,raw_author:FIRST_NAME::STRING as FIRST_NAME
,raw_author:MIDDLE_NAME::STRING as MIDDLE_NAME
,raw_author:LAST_NAME::STRING as LAST_NAME
FROM author_json_table;
```

	RAW_AUTHOR:AUTHOR_UID	FIRST_NAME	MIDDLE_NAME	LAST_NAME
1	1	Fiona	null	Macdonald
2	2	Gian	Paulo	Faleschini

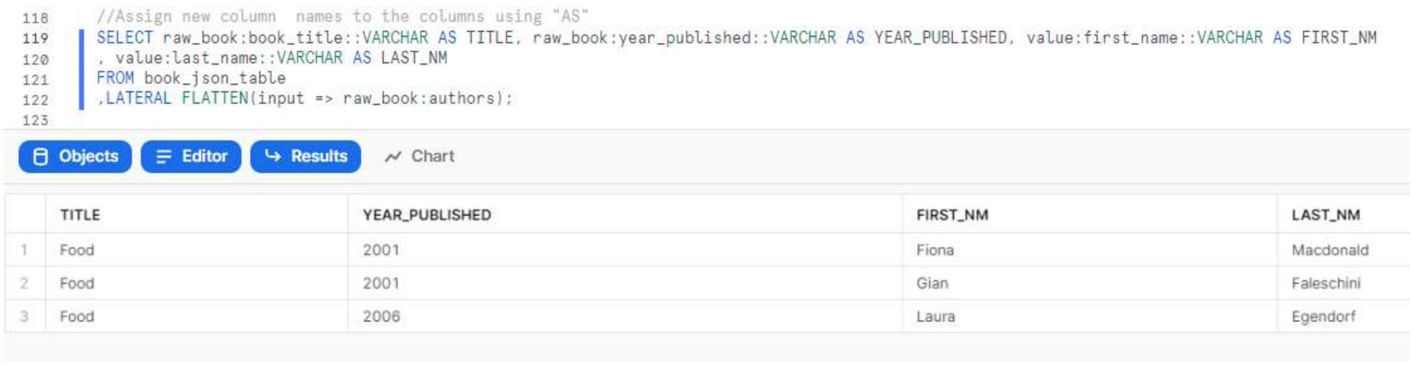
############ book

```
[{
  "book_title":"Food",
  "year_published":2001,
  "authors": [
          {
            "first_name":"Fiona",
            "middle_name":null,
            "last_name":"Macdonald"
            },
            {
            "first_name":"Gian",
            "middle_name":"Paulo",
            "last_name":"Faleschini"
            }
          ]
 },
{
  "book_title":"Food",
  "year_published":2006,
  "authors":
          [{
            "first_name":"Laura",
            "middle_name":"K",
            "last_name":"Egendorf"
            }
          ]
 }]
```

```
forrestli2011#(no warehouse)@(no database).(no schema)>put file://book.json
@feng_database.feng_schema.xml_json_stage;
+-----------+--------------+-------------+-------------+--------------------+--------------------+----------+---------+
| source    | target       | source_size | target_size | source_compression | target_compression | status
| message |
|-----------+--------------+-------------+-------------+--------------------+--------------------+----------+---------|
| book.json | book.json.gz |         440 |         224 | NONE               | GZIP               | UPLOADED |         |
+-----------+--------------+-------------+-------------+--------------------+--------------------+----------+---------+
1 Row(s) produced. Time Elapsed: 1.204s
forrestli2011#(no warehouse)@(no database).(no schema)>
```

----- nested book ---

```
create or replace table book_json_table
(
  raw_book variant
);

copy into book_json_table
    from @xml_json_stage
    files = ('book.json.gz')
    file_format=(format_name = 'json_file_format');

select * from author_json_table;

//a few simple queries
SELECT raw_book
FROM book_json_table;

SELECT raw_book:year_published
FROM book_json_table;

SELECT raw_book:authors
FROM book_json_table;

//try changing the number in the bracketsd to return authors from a different row
SELECT raw_book:authors[0].first_name
FROM book_json_table;

//Use these example flatten commands to explore flattening the nested book and author data
SELECT value:first_name
FROM book_json_table
,LATERAL FLATTEN(input => raw_book:authors);

SELECT value:first_name
FROM book_json_table
,table(flatten(raw_book:authors));

//Assign new column  names to the columns using "AS"
SELECT raw_book:book_title::VARCHAR AS TITLE, raw_book:year_published::VARCHAR AS
YEAR_PUBLISHED, value:first_name::VARCHAR AS FIRST_NM
, value:last_name::VARCHAR AS LAST_NM
FROM book_json_table
,LATERAL FLATTEN(input => raw_book:authors);
```



```
//Assign new column  names to the columns using "AS"
SELECT raw_book:book_title::VARCHAR AS TITLE, raw_book:year_published::VARCHAR AS YEAR_PUBLISHED, value:first_name::VARCHAR AS FIRST_NM
, value:last_name::VARCHAR AS LAST_NM
FROM book_json_table
,LATERAL FLATTEN(input => raw_book:authors);
```

	TITLE	YEAR_PUBLISHED	FIRST_NM	LAST_NM
1	Food	2001	Fiona	Macdonald
2	Food	2001	Gian	Faleschini
3	Food	2006	Laura	Egendorf