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

-- WEEK #1 - SQL REFRESHER

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

-- Use CTRL (OR CMD) + ENTR to execute a select command.

-- Choose the role to use.

-- We will use "sysadmin" to create the warehouse, database, and tables.

USE ROLE sysadmin;

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

-- CREATING WAREHOUSES & DATABASES

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

-- Create a new WAREHOUSE for table creation and data loading.

-- Remember, WAREHOUSES process data. DATABASES store data.

CREATE OR REPLACE WAREHOUSE loading\_wh WITH

WAREHOUSE\_SIZE='X-SMALL'

AUTO\_RESUME = TRUE -- default

AUTO\_SUSPEND = 600 -- default (SECONDS)

INITIALLY\_SUSPENDED = TRUE; -- default

-- Create a new database to store bike trips.

CREATE DATABASE IF NOT EXISTS bike;

-- Set the CONTEXT. (i.e. Information about your environment.)

-- The PUBLIC schema is the default schema that all users have access to.

-- It is often used for development and testing.

USE bike.public;

USE WAREHOUSE loading\_wh;

-- Create the table, where trip data will be stored. (DDL is free.)

CREATE OR REPLACE TABLE trips (

tripduration integer,

starttime timestamp,

stoptime timestamp,

start\_station\_id integer,

start\_station\_name string,

start\_station\_latitude float,

start\_station\_longitude float,

end\_station\_id integer,

end\_station\_name string,

end\_station\_latitude float,

end\_station\_longitude float,

bikeid integer,

membership\_type string,

usertype string,

birth\_year integer,

gender integer

);

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

-- CREATING STAGES

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

-- Create a STAGE that specifies the location of the S3 bucket.

-- STAGES are locations where files are stored for loading or unloading.

-- This is public S3 bucket, so no credentials are required.

CREATE OR REPLACE STAGE bike\_stage

url = 's3://bike-data-files/';

-- List the files in the stage. (It does NOT show delimiters.)

LIST @bike\_stage;

-- Review the properties of the stage. (Delimiters are noted.)

DESCRIBE STAGE bike\_stage;

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

-- CREATING FILE FORMATS

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

-- Create a FILE FORMAT that matches the data structure.

-- It describes what the staged files look like.

-- This statement includes several default values, just as an example.

CREATE OR REPLACE FILE FORMAT bike\_csv

type='csv'

compression = 'auto' -- compression checked automatically (default)

field\_delimiter = ',' -- columns separated by commas (default)

record\_delimiter = '\n' -- rows separated by newlines (default)

skip\_header = 0 -- does not skip lines in the file (default)

trim\_space = false -- does not remove white space from fields (default)

null\_if = ('') -- replace these values with SQL NULL

date\_format = 'auto' -- date checked automatically (default)

timestamp\_format = 'auto' -- time checked automatically (default)

field\_optionally\_enclosed\_by = '\042' -- strings closed by " " (octal)

escape\_unenclosed\_field = '\134' -- backslash escapes the delimiter in unenclosed fields

error\_on\_column\_count\_mismatch = false -- checks if # of columns in file/table match

comment = 'file format for raw trip data'

;

-- Confirm the file format was created.

SHOW FILE FORMATS IN DATABASE bike;

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

-- LOADING DATA

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

-- Use a COPY COMMAND to load staged data into the table.

COPY INTO trips -- the destination table

FROM @bike\_stage -- the source stage

file\_format = bike\_csv -- the file format

PATTERN = '.\*csv.\*'; -- the filenames to match

-- COUNT the number of records in the trips table.

SELECT COUNT(\*) FROM trips;

\_\_\_\_\_

-- Clear the table, so we can experiment with different warehouse sizes.

TRUNCATE TABLE trips;

SELECT \* FROM trips LIMIT 10;

-- Increase warehouse size to evaluate loading impact.

ALTER WAREHOUSE loading\_wh

SET warehouse\_size='LARGE';

-- RELOAD the staged data into the table.

-- Then, review QUERY HISTORY to compare loads.

COPY INTO trips -- the destination table

FROM @bike\_stage -- the source stage

file\_format = bike\_csv -- the file format

PATTERN = '.\*csv.\*'; -- the filenames to match

-- Readjust warehouse size after testing.

ALTER WAREHOUSE loading\_wh

SET warehouse\_size='X-SMALL';

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

-- ANALYZING DATA

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

-- Normally, analytics users have a different role than "sysadmin",

-- Also, queries may be done in other tools, like PowerBI.

-- Create a new warehouse for analysis (queries).

-- This warehouse separation avoids interference and allows for monitoring.

-- For example, "How much are we spending on data loading?"

CREATE OR REPLACE WAREHOUSE query\_wh WITH

WAREHOUSE\_SIZE='X-SMALL'

AUTO\_RESUME = TRUE -- default

AUTO\_SUSPEND = 600 -- default

INITIALLY\_SUSPENDED = TRUE; -- default

-- Preview 10 records of trip data.

SELECT \* FROM trips LIMIT 10;

-- Identify the earliest and latest date in the data.

--

SELECT MIN(STARTTIME), MAX(STARTTIME) FROM trips;

SELECT ROUND(AVG(tripduration)/60, 1) AS average\_trip\_mins, usertype

FROM trips

GROUP BY usertype

HAVING usertype IS NOT NULL

ORDER BY average\_trip\_mins;

-- Most Popular membership types

SELECT membership\_type,

COUNT(\*) AS num\_trips

FROM trips

GROUP BY membership\_type

ORDER BY num\_trips DESC;

SELECT bikeid,

COUNT(\*) AS num\_rides,

ROUND(SUM(tripduration)/3600) AS

totalhours\_tripduration

FROM trips

GROUP BY bikeid

ORDER BY totalhours\_tripduration DESC, num\_rides DESC

LIMIT 100;

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

-- CLONING TABLES

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

-- Clone the trips table to demonstrate "zero-copy" cloning.

-- The underlying data is not copied.

-- A common use case is coping production for development/testing.

CREATE TABLE \_\_\_\_\_

CLONE \_\_\_\_\_;

-- Count the number of records in the table clone.

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

-- WEEK #2 - ADVANCED TOPICS

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

-- Set the context.

USE ROLE sysadmin;

USE bike.PUBLIC;

USE WAREHOUSE loading\_wh;

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

-- WORKING WITH SEMI-STRUCTURED DATA

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

-- Create a table to store the weather data (JSON file).

CREATE TABLE weather\_json (v variant);

-- Create a stage that specifies the location of the S3 bucket.

-- This is public bucket, so no credentials are required.

CREATE STAGE weather\_stage

url = 's3://weather-files/';

-- List the files in the stage.

-- Remember, our trip data is from mid-2013 to mid-2018.

-- This weather data is from late-2016 to mid-2019.

LIST @weather\_stage;

-- Load the staged data into the table.

COPY INTO weather\_json

FROM @weather\_stage

file\_format = (type = json

strip\_outer\_array = true); --Pull out brackets.

-- Preview 10 records.

SELECT \* FROM weather\_json LIMIT 10;

\_\_\_\_\_

-- Create a view that will structure the semi-structured data.

-- In this case, a VIEW allows the data to be queried like a table.

-- The DOT NOTATION allows us to pull data lower in the hierarchy.

CREATE OR REPLACE VIEW weather\_view AS

SELECT

v:obsTime::timestamp as observation\_time,

v:station::string as station\_id,

v:name::string as city\_name,

v:country::string as country,

v:latitude::float as city\_lat,

v:longitude::float as city\_lon,

v:weatherCondition::string as weather\_conditions,

v:coco::int as weather\_conditions\_code,

v:temp::float as temp,

v:prcp::float as rain,

v:tsun::float as tsun,

v:wdir::float as wind\_dir,

v:wspd::float as wind\_speed,

v:dwpt::float as dew\_point,

v:rhum::float as relative\_humidity,

v:pres::float as pressure

FROM

weather\_json

WHERE

station\_id = '72502';

-- Preview 10 records of the VIEW.

SELECT \* FROM weather\_view LIMIT 10;

-- Examine a month of weather data.

-- This includes hourly measurements during January 2018.

SELECT \*

FROM weather\_view

WHERE DATE\_TRUNC('\_\_\_\_\_',observation\_time) = '2018-01-01';

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

-- USING ADVANCED SQL

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

-- First, we'll switch to the QUERY\_WH as we explore the data.

USE WAREHOUSE query\_wh;

-- Add the queries discussed in class.

--1. Pull row data, create a new field, and provide row # for the trip data (starttime and start\_station)

SELECT starttime, start\_station\_name,

RANK () OVER (ORDER BY starttime) TRIP\_NUMBER

FROM trips

ORDER BY starttime

LIMIT 100;

--2.

SELECT starttime, start\_station\_name, tripduration, AVG(tripduration) AS average\_trip\_duration

FROM trips

ORDER BY average\_trip\_duration

LIMIT 100;

--3

SELECT

EXTRACT (year, starttime) AS year,

EXTRACT (month, starttime) AS month,

COUNT (\*) AS trips,

SUM (trips) OVER (ORDER BY year, month

ROWS UNBOUNDED PRECEDING) AS cumulative\_trips

FROM trips

GROUP BY year, month

ORDER BY year, month

LIMIT 100;

--4

SELECT

EXTRACT (year, starttime) AS year,

EXTRACT (month, starttime) AS month,

COUNT (\*) AS trips,

LAG (trips) OVER (ORDER BY year, month) AS previous\_month,

(trips-previous\_month) AS change\_in\_trips

FROM trips

GROUP BY year, month

ORDER BY year, month

LIMIT 100;

--5. When customers were riding bikes, what weather was it?

SELECT w.weather\_conditions, t.starttime

FROM trips t

INNER JOIN weather\_view w

ON DATE\_TRUNC('hour', t.starttime) = DATE\_TRUNC('hour', w.observation\_time)

ORDER BY t.starttime

LIMIT 200;

--6. SELECT w.weather\_conditions, t.starttime

SELECT COUNT (\*) t.trips AS trips\_by\_weather, w.weather\_conditions, t.starttime

FROM trips t

INNER JOIN weather\_view w

ON DATE\_TRUNC('hour', t.starttime) = DATE\_TRUNC('hour', w.observation\_time)

GROUP BY weather\_conditions

ORDER BY t.starttime

LIMIT 200;

--7

WITH monthly\_rides AS (

SELECT

DATE\_TRUNC('month', starttime) AS month,

COUNT(\*) AS num\_rides

FROM trips

GROUP BY month

ORDER BY month

),

monthly\_temp AS (

SELECT

DATE\_TRUNC('month', observation\_time) AS month,

ROUND(AVG(w.temp), 1) AS avg\_temp

FROM weather\_view w

GROUP BY month

ORDER BY month

)

SELECT

EXTRACT(year, r.month) AS year,

EXTRACT(month, r.month) AS month,

r.num\_rides,

t.avg\_temp

FROM monthly\_rides r

INNER JOIN monthly\_temp t

ON r.month = t.month

ORDER BY year, month;

--8. What is our monthly grow rate in retars. Is there seasonality?

SELECT DATE\_TRUNC('MONTH', starttime) AS month,

COUNT(\*) AS current\_rentals,

LAG(current\_rentals) OVER (ORDER BY month) AS previous\_rentals,

ROUND((current\_rentals-previous\_rentals)/previous\_rentals,2) AS growth\_rate

FROM trips

GROUP BY month

ORDER BY month;

--9. For each bike for each year what is the usage by year, annual bike hours and using a window total bike time

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

-- TIME TRAVEL

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

-- Enables historical data access.

-- The default is 24 hours.

-- Common uses include restoring or backing up data.

-- DROP & UNDROP TABLES

DROP TABLE weather\_json;

SELECT \* FROM weather\_json LIMIT 10;

UNDROP TABLE weather\_json;

SELECT \* FROM weather\_json LIMIT 10;

-- ROLLBACK TABLES

-- Count rides by starting station.

SELECT start\_station\_name, COUNT(\*) FROM trips GROUP BY start\_station\_name;

-- Set all starting stations to "oops".

UPDATE trips SET start\_station\_name = 'oops';

-- Recount rides by starting station.

SELECT start\_station\_name, COUNT(\*) FROM trips GROUP BY start\_station\_name;

-- Option #1: Find the query ID in the console.

-- Option #2: Find the query ID using SQL.

SELECT \* FROM table(information\_schema.query\_history\_by\_session (result\_limit=>10));

-- Restore the table using the query ID.

CREATE OR REPLACE TABLE trips AS

(SELECT \* FROM trips before (statement => '01bacee6-0002-82e5-0007-2e3e0008a092'));

-- RECOUNT rides by starting station. (It should be restored.)

SELECT start\_station\_name, COUNT(\*) FROM trips GROUP BY start\_station\_name;

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

-- ACCESS CONTROL

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

-- Dictates what a user can access and to what extent.

-- For our purposes, let's assume a new analyst joined the team.

-- Switch to the SECURITYADMIN role to create a new role.

USE ROLE \_\_\_\_\_;

-- Create a new role and assign a user to it (yourself).

CREATE ROLE \_\_\_\_\_;

GRANT ROLE \_\_\_\_\_ TO USER \_\_\_\_\_;

-- Switch roles to see access (nothing) then switch back to grant.

USE ROLE junior\_analyst;

SELECT \* FROM trips LIMIT 10;

USE ROLE sysadmin;

SELECT \* FROM trips LIMIT 10;

-- Provide access to the appropriate warehouse and databases.

-- We will not provide "GRANT ALL" to this user.

USE ROLE securityadmin;

GRANT USAGE ON DATABASE \_\_\_\_\_ TO ROLE junior\_analyst;

GRANT USAGE ON SCHEMA \_\_\_\_\_ TO ROLE junior\_analyst;

GRANT SELECT ON ALL TABLES IN SCHEMA \_\_\_\_\_ to ROLE junior\_analyst;

GRANT OPERATE ON WAREHOUSE \_\_\_\_\_ TO ROLE junior\_analyst;

-- Recheck access levels, then switch back.

USE ROLE junior\_analyst;

SELECT \* FROM trips LIMIT 10;

USE ROLE sysadmin;

SELECT \* FROM trips LIMIT 10;

-- If you need to try again:

USE ROLE accountadmin;

DROP ROLE junior\_analyst;

USE ROLE sysadmin;

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

-- RESETTING THE ENVIRONMENT

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

-- Reset your environment.

USE ROLE accountadmin;

DROP SHARE IF EXISTS crunchbase\_company\_data;

DROP ROLE IF EXISTS junior\_analyst;

DROP DATABASE IF EXISTS bike;

DROP DATABASE IF EXISTS crunchbase\_company\_data;

-- Delete Crunchbase worksheet.

USE ROLE sysadmin;

USE WAREHOUSE loading\_wh;

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

-- CREATE THE DATABASE & TABLES

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

-- Create a new database.

CREATE DATABASE IF NOT EXISTS tickets;

-- Create the tables.

CREATE OR REPLACE TABLE listings (

listid integer not null,

sellerid integer not null,

eventid integer not null,

dateid smallint not null,

numtickets smallint not null,

priceperticket decimal(8,2),

totalprice decimal(8,2),

listtime timestamp

);

CREATE OR REPLACE TABLE sales (

salesid integer not null,

listid integer not null,

sellerid integer not null,

buyerid integer not null,

eventid integer not null,

dateid smallint not null,

qtysold smallint not null,

pricepaid decimal(8,2),

commission decimal(8,2),

saletime timestamp

);

CREATE OR REPLACE TABLE users (

userid integer not null,

username char(8),

firstname varchar(30),

lastname varchar(30),

city varchar(30),

state char(2),

email varchar(100),

phone char(14),

likesports boolean,

liketheatre boolean,

likeconcerts boolean,

likejazz boolean,

likeclassical boolean,

likeopera boolean,

likerock boolean,

likevegas boolean,

likebroadway boolean,

likemusicals boolean

);

CREATE OR REPLACE TABLE venues (

venueid smallint not null,

venuename varchar(100),

venuecity varchar(30),

venuestate char(2),

venueseats integer

);

CREATE OR REPLACE TABLE categories (

catid smallint not null,

catgroup varchar(10),

catname varchar(10),

catdesc varchar(50)

);

CREATE OR REPLACE TABLE dates (

dateid smallint not null,

caldate date not null,

day character(3) not null,

week smallint not null,

month character(5) not null,

qtr character(5) not null,

year smallint not null,

holiday boolean default('N')

);

CREATE OR REPLACE TABLE events (

eventid integer not null,

venueid smallint not null,

catid smallint not null,

dateid smallint not null,

eventname varchar(200),

starttime timestamp

);

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

-- REVIEW THE ERD

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

-- See the detailed ERD in the slides.

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

-- LOAD ONE OF THE TABLES

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

-- Create a STAGE that specifies the location of the S3 bucket.

-- STAGES are locations where files are stored for loading or unloading.

-- This is public S3 bucket, so no credentials are required.

CREATE OR REPLACE STAGE ticket\_files

url = 's3://746-tickets';

-- List the files in the stage. (It does NOT show delimiters.)

LIST @ticket\_files;

-- Review the properties of the stage. (Delimiters are noted.)

DESCRIBE STAGE ticket\_files;

-- Query some of the files in the staging area.

SELECT $1 FROM @ticket\_files/venues\_pipe.txt LIMIT 20;

SELECT $1 FROM @ticket\_files/sales\_tab.txt LIMIT 20;

-- Create a FILE FORMAT that matches the data structure.

-- It describes what the staged files look like.

CREATE OR REPLACE FILE FORMAT tickets\_tab

type='csv'

field\_delimiter = '\t' -- columns separated by commas (default)

record\_delimiter = '\n' -- rows separated by newlines (default)

skip\_header = 0 -- does not skip lines in the file (default)

trim\_space = true -- does not remove white space from fields (default)

null\_if = ('')

date\_format = 'auto'

timestamp\_format = 'auto'

error\_on\_column\_count\_mismatch = true

;

-- Load data into the sales table.

COPY INTO sales -- destination table

FROM @ticket\_files/sales\_tab -- source file(s)

file\_format = tickets\_tab;

SELECT \* FROM sales LIMIT 10;

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

-- ANSWER BUSINESS QUESTIONS

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

-- Switch to the query warehouse.

USE WAREHOUSE query\_wh;

-- ADD OTHER QUERIES FROM CLASS.

--What time period does this data cover?

SELECT

MIN(saletime) AS earliest\_sale\_date,

MAX(saletime) AS latest\_sale\_date

FROM sales;

--How many tickets were sold? How much revenue was generated?

SELECT

SUM(qtysold) AS "Total Ticker Sold",

SUM(commission) AS "Total Commission Earned"

FROM sales;

--What was the average price paid per ticket?

SELECT ROUND(SUM(pricepaid)/SUM(qtysold),2) AS "Price Paid Per Ticket"

FROM sales;

--How many buyers?

SELECT COUNT (DISTINCT buyerid) AS "# of Buyers"

FROM sales;

--How many sellers also bought tickets?

SELECT COUNT (DISTINCT sellerid) AS "# of Buyers"

FROM sales

WHERE sellerid IN (SELECT buyerid

FROM sales);

--What was the % change of quarter vs. quarter sales?

SELECT

EXTRACT (YEAR FROM saletime) AS year,

EXTRACT(QUARTER FROM saletime) AS quarter,

ROUND(sum(pricepaid) / 1000000,2) AS current\_sales\_mil,

LAG(current\_sales\_mil) OVER (ORDER BY year, quarter) AS previous\_sales\_mil,

ROUND((current\_sales\_mil - previous\_sales\_mil)/previous\_sales\_mil)\*100,2 AS pct\_change

FROM sales

GROUP BY year, quarter;

---Calculate cumulative commission revenue by month in millions of dollars. What was cumulative revenue at the end of September?

CREATE OR REPLACE TABLE sales (

salesid integer not null,

listid integer not null,

sellerid integer not null,

buyerid integer not null,

eventid integer not null,

dateid smallint not null,

qtysold smallint not null,

pricepaid decimal(8,2),

commission decimal(8,2),

saletime timestamp

);

SELECT

DATE\_TRUNC('month', saletime) AS month,

SUM(commission) / 1000000 AS monthly\_commission,

SUM(SUM(commission)) OVER (ORDER BY DATE\_TRUNC('month',saletime))

FROM sales

WHERE DATE\_TRUNC('month', saletime) <= '2024-09-30'

GROUP BY DATE\_TRUNC ('month', saletime)

ORDER BY month;

-- Calculate a 7-day rolling average of tickets sold. On June 19th, what was the average?

select date\_trunc(day,saletime) AS day,

SUM(qtysold) AS tickets\_sold,

AVG(tickets\_sold) OVER (ORDER BY day

ROWS BETWEEN 6 PRECEDING

AND CURRENT ROW)

AS seven\_day\_avg

--Calculate buyer spend by event. dditionally, in two other columns, show the buyer's total spend and the % of their total spend on the listed event. For our largest buyer (#4303), what % of their spend was on event #7851? (They spent the most money on that event. The concert was for a band called "Live".)

SELECT

buyerid,

eventid,

SUM(pricepaid) AS buyer\_event\_spend,

SUM(buyer\_event\_spend) OVER (PARTITION BY buyerid) AS buyer\_spend,

ROUND(buyer\_event\_spend/buyer\_spend,3)\*100

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

-- EVALUTE QUERY PERFORMANCE

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

-- QUERY PROFILE

-- Provides execution details for a query.

-- Includes graphs/stats to understand query components.

-- Temporarily turn off cached results. (For experimentation.)

ALTER SESSION SET USED\_CACHED\_RESULT = FALSE;

-- Execute a query, which we'll review in the QUERY PROFILE.

-- For each event, it shows tickets sold and averge ticket price.

SELECT

eventid,

SUM(qtysold) AS tickets\_sold,

ROUND(AVG(pricepaid/qtysold)) AS avg\_price\_per\_ticket

FROM sales

GROUP BY eventid

HAVING SUM(qtysold) >= 25

ORDER BY SUM(qtysold) DESC;

-- ANALYZE THE QUERY PROFILE

-- In the query details, use the ellipse to view the profile.

-- QUERY DETAILS TAB - shows summary information

-- QUERY PROFILE TAB - see the SLIDES for more detail

-- Use EXPLAIN as another way to breakdown the query.

\_\_\_\_\_

\_\_\_\_\_;

-- METADATA CACHE

-- Snowflake stores COUNT, MAX, MIN, etc. in metadata cache.

-- When needed, it can avoid table scans and calculations.

-- It simply pulls those metrics from metadata cache.

SELECT MAX(pricepaid), MIN(pricepaid) FROM sales;

-- Then, view the QUERY PROFILE.

-- Turn cache reuse back on.

ALTER SESSION SET USE\_CASH\_RESULT = TRUE;

-- Rerun our previous query (twice if needed) to show results cache.

SELECT

eventid,

SUM(qtysold) AS tickets\_sold,

ROUND(AVG(pricepaid/qtysold)) AS avg\_price\_per\_ticket

FROM sales

GROUP BY eventid

HAVING SUM(qtysold) >= 25

ORDER BY SUM(qtysold) DESC;

USE tickets.public;

USE WAREHOUSE loading\_wh;

LIST @ticket\_files;

-- Create a format that matches our pipe files.

CREATE OR REPLACE FILE FORMAT tickets\_pipe

type='csv'

field\_delimiter = '|'

record\_delimiter = '\n'

skip\_header = 0

null\_if = ('')

;

-----------

-- LISTINGS

-----------

-- Preview the listings file(s) on staging.

SELECT $1 FROM @ticket\_files/listings LIMIT 100;

-- Load data into the listings table.

COPY INTO listings

FROM @ticket\_files/listings

file\_format = tickets\_pipe

;

-- Preview the listings table.

SELECT \* FROM listings LIMIT 100;

-----------

-- DATES

-----------

-- Preview the date file(s) on staging.

SELECT $1 FROM @ticket\_files/dates LIMIT 100;

-- Load data into the dates table.

COPY INTO dates

FROM @ticket\_files/dates

file\_format = tickets\_pipe;

-- Preview the dates table.

SELECT \* FROM dates LIMIT 100;

ALTER TABLE dates

ADD COLUMN season string;

-----------

-- EVENTS

-----------

-- Preview the events file(s) on staging.

SELECT $1 FROM @ticket\_files/events LIMIT 100;

-- Load data into the events table.

COPY INTO events

FROM @ticket\_files/events

file\_format = tickets\_pipe

ON\_ERROR = 'CONTINUE';

-- Preview the events table.

SELECT \* FROM events LIMIT 100;

-----------

-- USERS

-----------

-- Preview the user file(s) on staging.

SELECT $1 FROM @ticket\_files/users LIMIT 100;

-- Load data into the users table.

COPY INTO users

FROM @ticket\_files/users

file\_format = (field\_delimiter = '|'

trim\_space = TRUE);

-- Preview the users table.

SELECT \* FROM users LIMIT 100;

-----------

-- CATEGORIES

-----------

-- Query the category files(s) on staging.

SELECT $1 FROM @ticket\_files/categories LIMIT 100; --This is a JSON file

-- Create a table to store the categories data (JSON).

CREATE OR REPLACE TABLE categories\_json (v variant);

-- Load data into the categories table.

COPY INTO categories\_json

FROM @ticket\_files/categories

file\_format = (type = 'json'

strip\_outer\_array = TRUE)

;

-- Preview the categories data (JSON).

SELECT \* FROM categories\_json;

-- Create a view to structure the data.

CREATE OR REPLACE VIEW categories\_view AS

SELECT

v:catid::integer AS catid,

v:catgroup::string AS catgroup,

v:catname::string AS catname,

v:catdesc::string AS catdesc

FROM categories\_json;

-- Preview data in the categories view.

SELECT \* FROM categories\_view;

-----------

-- VENUES

-----------

-- Query the venues files(s) on staging.

SELECT $1 FROM @ticket\_files/venues LIMIT 200;

-- Load data into the venues table.

-- This will generate an ERROR that needs fixing.

COPY INTO venues

FROM @ticket\_files/venues

FILE\_FORMAT = (field\_delimiter = ':', null\_if = ('NULL'))

;

-- Preview data in the venues table.

SELECT \* FROM venues LIMIT 100;

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

-- UNLOADING DATA

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

-- Instead of loading data into a table...

-- We will UNLOAD DATA to a FILE, and put that on a STAGE.

-- This is different than just exporting query results.

-- Oftentimes, you will be exporting to an external stage like S3.

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

-- UNLOAD TO AN INTERNAL, TABLE STAGE

-- Copy the sales table into a file within its table stage.

COPY INTO @%sales FROM sales;

-- Examine the files in the TABLE STAGE.

LIST @%sales;

-- Verify there is data in the file.

SELECT \* FROM @%sales/data\_0\_0\_0.csv LIMIT 10;

-- Download the file locally. (ONLY WORKS IN SNOWSQL.)

GET @%sales 'file:///C:/Users/\_\_\_\_\_\_/Downloads/'; -- For Windows

GET @%sales 'file:///tmp/data/'; -- For Linux or Mac

-- Remove files from the TABLE STAGE; then confirm the deletion.

REMOVE @%sales;

LIST @%sales;

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

-- UNLOAD TO AN INTERNAL, NAMED STAGE

-- Create a named stage.

CREATE OR REPLACE STAGE tickets\_named\_stage;

-- Copy the results from a query to the stage.

COPY INTO @tickets\_named\_stage FROM

(SELECT \* FROM dates);

-- Examine the files in the NAMED STAGE.

LIST @tickets\_named\_stage;

-- Verify there is data in the file.

SELECT $1, $2, $3 FROM @\_\_\_\_\_/data\_0\_0\_0.csv.gz LIMIT 10;

-- Remove the file from staging, and drop the stage.

REMOVE @tickets\_named\_stage/data\_0\_0\_0.csv.gz;

LIST @tickets\_named\_stage;

DROP STAGE tickets\_named\_stage;

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

-- UNLOAD TO AN EXTERNAL, NAMED STAGE (AWS S3)

-- Create the stage, which points to S3.

-- This won't work as we haven't setup an integration with S3.

CREATE OR REPLACE STAGE stage\_tickets\_s3

URL = 's3://746-tickets'

STORAGE\_INTEGRATION = s3\_int;

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

-- ANSWERING BUSINESS QUESTIONS

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

-- Switch to the query warehouse.

USE WAREHOUSE query\_wh;

-- Revisit what the tables look like.

SELECT \* FROM listings LIMIT 10;

SELECT \* FROM sales LIMIT 10;

SELECT \* FROM venues LIMIT 10;

SELECT \* FROM events LIMIT 10;

SELECT \* FROM categories\_view LIMIT 10;

SELECT \* FROM dates LIMIT 10;

SELECT \* FROM users LIMIT 10;

-- Answer the business questions from class.

---What was the highest-selling individual event?

SELECT e.eventid, e.eventname, e.starttime, ROUND(SUM(S.pricepaid)) AS total\_sales\_usd

FROM sales s

LEFT JOIN events e

ON s.eventid = e.eventid

GROUP BY e.eventid, e.eventname, e.starttime

ORDER BY total\_sales\_usd DESC

LIMIT 10;

---Who bought the most tickets?

SELECT s.buyerid, u.firstname, u.lastname, SUM(s.qtysold)

FROM users u

LEFT JOIN sales s

ON u.

---Re-create the query

SELECT

d.caldate AS day,

d.season,

SUM(s.qtysold) AS ticket\_sold,

SUM(ticket\_sold) OVER (ORDER BY d.caldate) AS cumulative\_tickets,

SUM(ticket\_sold) OVER (PARTITION BY d.season) AS tickets\_season

FROM sales s

INNER JOIN dates d

ON s.dateid = d.dateid

GROUP BY d.caldate, d.season

ORDER BY d.caldate;

---Re-crate query two

SELECT c.catname, SUM(s.qtysold) AS listed\_tickets\_total

FROM category c

INNER JOIN sales s

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

-- RESETTING THE ENVIRONMENT

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

-- Reset your environment.

\_\_\_\_\_;

\_\_\_\_\_;

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

-- WEEK #5 - DATA TRANSFORMATION

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

-- Set the context.

USE ROLE \_\_\_\_\_;

USE WAREHOUSE LOADING\_WH;

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

-- CREATE DATABASE OBJECTS

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

-- Create the database.

CREATE OR REPLACE DATABASE tacos;

-- Create the orders table.

CREATE OR REPLACE TABLE orders

(

order\_id integer,

truck\_id integer,

timestamp timestamp\_ntz(9),

amount number(38,2)

);

-- Create the (raw) trucks table.

CREATE OR REPLACE TABLE trucks\_raw

(

truck\_id integer,

city string,

region string,

country string,

franchise\_flag integer,

year integer,

make string,

model string,

opening\_date date

);

-- Create a stage that points to the S3 bucket.

CREATE OR REPLACE STAGE taco\_files

url = 's3://746-tacos';

-- List files in the staging environment.

LIST @taco\_files;

-- Create a file format that matches the files.

CREATE OR REPLACE FILE FORMAT taco\_csv

type='csv'

skip\_header = 1

null\_if = 'NULL';

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

-- LOAD ORDERS (no transformations needed)

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

-- Preview the orders file on staging.

SELECT $1, $2, $3, $4

FROM @taco\_files/orders.csv;

-- Load the orders data.

COPY INTO orders

FROM @taco\_files/orders.csv

FILE\_FORMAT = taco\_csv;

-- Preview the orders table.

SELECT \* FROM orders LIMIT 20;

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

-- LOAD TRUCKS (transform before loading)

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

-- Preview the trucks file on staging.

-- Review the transformations from the handout.

SELECT $1, $3, $4, $5, $6, $7, $8, TRIM($9), $2

FROM @taco\_files/trucks.csv (file\_format => taco\_csv);

-- Query the file again, but reorder columns and trim the "model".

-- Load the data with the transformations.

COPY INTO trucks\_raw

FROM (

SELECT $1, $3, $4, $5, $6, $7, $8, TRIM($9), $2

FROM @taco\_files/trucks.csv (file\_format => taco\_csv)

);

-- Preview the (raw) trucks table.

SELECT \* FROM trucks\_raw LIMIT 20;

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

-- CONTINUE TRANSFORMATIONS AFTER LOADING (part 1) (CTAS)

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

-- Build a query that transforms the data.

-- Copy the transformed data into a new table.

CREATE OR REPLACE TABLE trucks AS

SELECT DISTINCT

truck\_id,

InitCap(city) AS city,

region,

CASE

WHEN country = 15 THEN 'United States'

WHEN country = 3 THEN 'Canada'

ELSE 'Unknown'

END AS country,

franchise\_flag,

year,

COALESCE(make,'Custom') AS make,

model,

opening\_date

FROM trucks\_raw

;

-- Preview the (clean) trucks table.

SELECT \* FROM trucks LIMIT 20;

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

-- CONTINUE TRANSFORMATIONS AFTER LOADING (part 2) (UPDATE)

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

-- Query the truck "make" to explore the "Ford" issue.

SELECT make FROM trucks LIMIT 20;

-- Create a development table to avoid impacting production.

CREATE OR REPLACE TABLE trucks\_dev

CLONE trucks;

-- Preview the development table

SELECT \* FROM trucks\_dev;

-- Correct "Ford\_" in the development table.

UPDATE trucks\_dev

SET make = 'Ford'

WHERE make = 'Ford\_';

-- Explore the "Ford" change in both tables.

SELECT make FROM trucks\_dev WHERE make LIKE 'Ford%' LIMIT 20;

SELECT make FROM trucks WHERE make LIKE 'Ford%'LIMIT 20;

-- Add a column to store “truck\_age”.

ALTER TABLE trucks\_dev

ADD COLUMN truck\_age integer;

-- Calculate the “truck\_age”.

UPDATE trucks\_dev

SET truck\_age = YEAR(CURRENT\_DATE()) - YEAR;

-- Confirm the age calculation is correct.

SELECT truck\_id, year, truck\_age FROM trucks\_dev;

---Change country names

UPDATE trucks\_dev

SET country = CASE city

WHEN 'Sydney' THEN 'Australia'

WHEN 'Melbourne' THEN 'Australia'

WHEN 'Sao Paulo' THEN 'Brazil'

WHEN 'Rio De Janeiro' THEN 'Brazil'

WHEN 'Cairo' THEN 'Egypt'

WHEN 'Manchester' THEN 'England'

WHEN 'London' THEN 'England'

WHEN 'Paris' THEN 'France'

WHEN 'Nice' THEN 'France'

WHEN 'Berlin' THEN 'Germany'

WHEN 'Hamburg' THEN 'Germany'

WHEN 'Munich' THEN 'Germany'

WHEN 'Delhi' THEN 'India'

WHEN 'Mumbai' THEN 'India'

WHEN 'Tokyo' THEN 'Japan'

WHEN 'Warsaw' THEN 'Poland'

WHEN 'Krakow' THEN 'Poland'

WHEN 'Barcelona' THEN 'Spain'

WHEN 'Madrid' THEN 'Spain'

WHEN 'Stockholm' THEN 'Sweden'

WHEN 'Seoul' THEN 'South Korea'

WHEN 'Cape Town' THEN 'South Africa'

ELSE country

END;

SELECT country FROM trucks\_dev;

-- Move the development table to production.

ALTER TABLE trucks\_dev

SWAP WITH trucks;

-- Validate the production table.

SELECT truck\_id, make, year, truck\_age, country

FROM trucks

WHERE make LIKE 'Ford%';

-- Drop the development table (no longer needed).

DROP TABLE trucks\_dev;

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

-- ANSWER BUSINESS QUESTIONS

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

-- Switch to our warehouse for querying data.

USE WAREHOUSE query\_wh;

-- How many food trucks do we have?

SELECT COUNT(\*) AS food\_trucks

FROM trucks

;

-- What types of trucks are in our fleet?

SELECT make, model, COUNT(\*) AS accurate\_trucks

FROM trucks

GROUP BY make, model

;

-- How many trucks are company-owned vs franchise-owned?

SELECT

CASE

WHEN franchise\_flag = 1 THEN 'Franchise Owned'

ELSE 'franchise\_owned' END AS truck\_ownership,

COUNT(\*) AS num\_trucks

FROM trucks

GROUP BY truck\_ownership

;

--- What are our five oldest company-owned vehicles?

SELECT make, truck\_age

FROM trucks

WHERE franchise\_flag = 0

ORDER BY truck\_age DESC

LIMIT 5;

-- Recreate Table

SELECT o.truck\_id, o.order\_id, DATE(o.timestamp) AS order\_date, o.amount,

SUM(amount) OVER (PARTITION BY truck\_id) AS total\_truck\_rev,

FROM orders o

ORDER BY total\_truck\_rev DESC, timestamp

;

-- Sum by total truck revenue

SELECT o.truck\_id, o.order\_id, DATE(o.timestamp) AS order\_date, o.amount,

SUM(amount) OVER (PARTITION BY truck\_id) AS total\_truck\_rev,

SUM(amount) OVER (PARTITION BY truck\_id, DATE(timestamp)) AS daily\_truck\_rev

FROM orders o

ORDER BY total\_truck\_rev DESC, timestamp

;

-- Running total of Truck Revenue

SELECT o.truck\_id, o.order\_id, DATE(o.timestamp) AS order\_date, o.amount,

SUM(amount) OVER (PARTITION BY truck\_id) AS total\_truck\_rev,

SUM(amount) OVER (PARTITION BY truck\_id, DATE(timestamp)) AS daily\_truck\_rev,

SUM(amount) OVER (PARTITION BY truck\_id ORDER BY timestamp

ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)

AS cumulative\_truck\_rev,

FROM orders o

ORDER BY total\_truck\_rev DESC, timestamp

;