CS 329E Project 6, due Thursday, 04/17.

## Objectives

Implement an end-to-end data pipeline in dbt from your previously developed SQL and python notebooks. This will entail crafting dbt models, executing them, and testing the generated tables for data integrity.

# Methodology

Go through your staging, intermediate, and mart notebooks and identify the SQL and Python blocks which contain one or more mutations. Apply the appropriate conversion strategy detailed below based to the type of mutation you are evaluating:

- For create-table-as-select, insert-table-as-select, and pandas Dataframe writes, convert each one to a dbt model such that SQL statements are translated into dbt SQL models and Python blocks are translated into dbt Python models:
  - SQL models should have one or more common table expressions, followed by a select statement for dbt to evaluate and persist as a table.
  - Python models should contain data manipulations that can't easily be expressed in SQL, namely complex LLM interactions and conditional logic. They must return a dataset formatted as a <u>PySpark Dataframe</u>. Note: Python models get executed on a Dataproc cluster on GCP which uses a PySpark runtime, hence the need to convert from Pandas to PySpark Dataframe.
  - All models that source from the raw layer should call the <u>source()</u> function to read the raw tables.
  - All models that source from the staging layer or above should call the <u>ref()</u> function to read the staging or intermediate tables, whether they are the final table or just a temp table.
- For DML statements, copy them into <u>post-hooks</u> or rewrite them as select statements.
- For uniqueness, not null tests, and referential integrity tests, convert them to yaml and add them to schema.yml using dbt's <u>unique</u> and <u>not null</u> and <u>relationship</u> constructs. Note that in order to test for the uniqueness of a combination of fields, an additional macro is required, which you can copy from <u>this example</u>. For more details on macros see the Hints section.
- Drop table statements can be translated into <u>post-hooks</u> and attached to the final model so that they get executed only after the final table has been created. For example, if tmp\_airports is a temp table that persists some intermediate results and is used to create the final table called Airport, add the drop table statement into a post-hook in the Airport.sql file.

#### **Implementation Plan**

- After running the dbt init command, go to the models folder and delete the examples subfolder. Create the following directory structure in its place:
  - o models/stg
  - o models/int
  - o models/mrt
- Edit your dbt\_project.yml file based on <u>this example</u>. This file should specify your stg, int, and mrt folder names, dataset names, as well as materialization type. Use table materialization instead of view. Follow this naming convention: dbt\_[your-domain]\_stg, dbt\_[your-domain]\_int, and dbt\_[your-domain]\_mrt for your BQ datasets.
- Create a macro in the dbt macros folder called generate\_schema\_name.sql and copy the contents from this file into it. This macro is needed to avoid datasets being named like this: dbt\_air\_travel\_stg\_dbt\_air\_travel\_stg.
- To specify the data source for the staging tables, create a schema.yml file in your models/stg directory by following this example. This will allow us to refer to the raw tables using dbt's source() function. Note: we are not creating a new raw dataset for dbt, we will source the dbt generated staging tables from the existing raw dataset ([your-domain]\_raw).
- Create your model files under each model subfolder, one layer at a time, starting with the stg models. Look at the snippets repo to see how this was done for the air travel example.
- Run the <u>dbt run</u> command to compile your models. This should be run iteratively so that you can more easily debug errors. My suggestion is to port one model at a time. Make sure that the model compiles and it generates the expected output in BQ before continuing. Once you have the staging tables built, proceed with the int layer, etc. Note: the intermediate layer will take you longer to port because sections of the Python code will need to be rewritten for dbt.
- Once all of your intermediate models compile, prepare a schema.yml file with your primary key and foreign key constraints by following this example. Primary key constraints are specified as uniqueness and not null checks, while foreign key constraints are specified as relationship checks.

All of your final int models should be represented in one or more schema.yml files. If all of your int models are located in the same folder, you'll end up with a single schema.yml.If your int models are spread out across multiple subfolders, you'll end up

with multiple schema.yml files. Remember that we're only verifying the referential integrity of the final tables in the int layer, so there is no need to create tests for the staging and mart layers.

- Execute the <u>dbt test</u> command to run your data integrity tests on your final tables in the int layer. Resolve any errors from those tests and ensure that your uniqueness, not null, and relationship checks all pass. To resolve the errors, you will need to modify your models, re-run them, and then re-run the failing tests until they pass. Don't worry if the primary key and foreign key constraints are not visible from BQ Studio. There is a bug in the dbt bigquery connector that prevents those constraints from getting created correctly. The key outcome is for all of your dbt tests to pass.
- Generate one or more <u>lineage graphs</u> that capture the dependencies between the models. To do that, you'll first need to compile the <u>documentation</u> for your warehouse and then bring up the UI, just like we did during the setup. Take a screenshot of your full lineage graph. If your graph has > 30 nodes and is difficult to visualize in its entirety, take additional screenshots of each layer (staging, intermediate, and mart).
- To submit your work, clone your repo onto the VM and make a new project6 folder. Copy your top-level dbt project folder into this project6 one. For the lineage graphs, make a lineage subfolder in your project6 folder. Place your screenshots there.
- Once you have committed your code changes, create the usual submission.json file and upload it to Canvas by the deadline. Only one person per group needs to do this step.

### Hints

- To get started, run the dbt init [your-domain] command to generate a new dbt project for your warehouse. You'll need to modify .dbt/profiles.yml just like we did during the setup. From there, you should create the proper directory structure under the models folder and then modify the default dbt\_project.yml. You should also create the two macros and the models/stg/schema.yml file.
- I suggest keeping around your temp tables in BigQuery until you have verified the correctness of the final models. Otherwise, you will be recompiling them several times, which slows down development.
- To speed up the dbt run command, you can increase the number of threads from 1 to 3 in your .dbt/profiles.yml. This will allow dbt to execute up to 3 models concurrently without violating dependencies.
- To save time, you can compile a specific model, or models that match a specific path, using the --select option. More details <u>here</u>.

- If you are renaming a model or changing the schema of an existing model, use the dbt run --full-refresh option. More details here.
- For the primary key, foreign key, uniqueness, not null and relationship tests, I would suggest copying one of the schema.yml files from the snippets repo (such as this one) and using it as your starting point.
- If you get an "Insufficient quota" error, whether it be CPU or Persistent Disk, go to the <u>Quotas</u> page and request an increase for the resource in question. For CPU, I would recommend requesting 48 CPUs in the us-central1 region, which would be double from the current allocation. For storage, I would recommend also doubling the volume of available disk to 8 TB. These requests should get auto-approved within minutes. If they don't, please contact me as soon as possible.

Compute	CPUs	Quota	region : us-central1	24	16.67%	
Engine API						

• To create the lineage graphs, run dbt docs generate followed by dbt docs serve --port 8181 on the machine where you are running dbt. If you are running dbt on a VM, you'll also need to bring up a local terminal, run gcloud init, and then set up the ssh tunnel to connect to the VM. You can use this command to create the tunnel: gcloud compute ssh dbt --project [GCP-PROJECT\_ID] --zone us-central1-c ---NL 8181:localhost:8181

Be sure to replace [GCP-PROJECT\_ID] with your own project. Then, open a browser on your local machine and navigate to http://localhost:8181. To view the lineage graph,

click on this icon: <sup>1</sup> on the bottom-right corner of the home page. The graph should look similar to the image below:



# CS 329E Project 6 Rubric **Due Date: 04/17/25**

<ul> <li>dbt project folder is thorough and meets all requirements</li> <li>-3 for each missing table in stg, int, and mrt datasets. Tables must be generated by dbt models. Number of final tables in each layer should match previously generated layers from Colab. For example, the mrt dataset should have a minimum of 10 tables.</li> <li>-3 for each empty table in stg, int or mrt datasets</li> <li>-2 for each missing test, needs to have a minimum of 20 tests</li> <li>-2 for each failing uniqueness, not null or relationship test</li> <li>-2 for each model not using source () or ref()</li> <li>-3 for not following our dataset naming convention</li> <li>-90 missing dbt project folder under project6</li> </ul>				
<ul> <li>Lineage subfolder contains one or more screenshots of the model dependency graph</li> <li>-1 for each missing model or dependency that doesn't correspond to the model files in the repo</li> <li>-2 model names are not legible in the provided screenshot(s)</li> <li>-10 missing lineage folder under project6</li> </ul>	10			
submission.json submitted into Canvas. Your project will not be graded without this submission. The file should have the following schema:				
<pre>{     "commit-id": "your most recent commit ID from Github",     "project-id": "your project ID from GCP" }</pre>				
Example:				
<pre>{     "commit-id": "dab96492ac7d906368ac9c7a17cb0dbd670923d9",     "project-id": "some-project-id" }</pre>				
Total Credit:	100			