1. Setup of the project

* Created a transform role and a user with that role in Snowflake for dbt to use
* Passwords to connections are stored in ~/.dbt/profiles.yml so that they don’t get pushed to git
* When setting up target schema in dbt we are indicating a place where all the output is created
* dbt power user – extension for VS code to work better with dbt

1. Data flow

* Raw layer – raw data that we have in Snowflake
* Staging Layer – basic checks, renaming/aliasing columns
* Core Layer – where big transformation/cleansing of data happens
* Mart Layer – data ready for BI Tools

1. Snowflake Pro Tips

* When clicking on a table we can copy it’s reference to paste in Worksheet SQL. “Place Name in SQL”

1. Terms

* model – an sql file that contains sql statements that perform some transformations
* materializations:
* view – select statement used for lightweight representation, don’t use it when you read from a view model several times (it’s a select statement so it needs to execute – sloooow)
* table – actual physical table that needs to be recreated every time model is used, it’s useful when you read repeatedly from the model that generates a table
* incremental (table appends) – records are appended, as data comes in it is appended to the table. It wouldn’t be useful if you wanted to edit historical records
* ephermal (CTE) – intermediate step model, for example when you want to simply add alias to colums. You just create a CTE model that renames the models and destroys materialization at the end
* referring to model – we use dbt template when refering to other models, eg.

SELECT \* FROM {{ ref('src\_listings') }}

* source – data that is already in the warehouse (ingested by Fivetram, Stich from external systems)
* seed – local files that are uploaded to warehouse
* slowly changing dimensions – dimensions where changes happen very rarely. Updating email address
* snapshots – feature that enables retaining original record before change happens (usually with slowly changing dimensions)
* singular tests – sql query that are expected to return 0 rows, otherwise test fails
* built-in generic tests:
* unique – if column has unique values
* not\_null – if column has not null values
* accepted\_values – if column contains only specified values
* relationships – if column has valid references to other tables
* custom tests – test that you write yourself or download from dbt package
* macros – pieces of code using jinja that can be used for repeatable actions ( for loops are very common)
* dbt\_utils – dbt package that has useful functions (surrogate key for example, it enables to create a unique key)
* documentation – you can add documentation with dbt that will generate an html file. Some basic documentation is added with “description:” tag in schema.yml
* markdown-based documentation – we can create docs.md file in models folder to create more sophisticated documentation
* overview in documentation – you can change default appearance of the overview page in served documentation
* assets – you can use your own images for documentation(overview), you simply need to include “assets” folder and configure it in dbt\_project.yml
* DAG – directed acyclic graph shows dependencies of models to each other in form of a linage graph
* analyses – folder where you can store ad-hoc queries that are used for analysis
* hooks – analysis SQL queries that are executed on predefined times (on-run-start, on-run-end etc) also you can execute granting role with post hooks for every model
* exposures – configs that lead to external assets such as reports, dashboards that will be integrated and compiled in documentation

1. Commands/parameters/info in dbt

* [+materialized: view] # “+” means that we want to use builtin dbt parameter, not a reference to some variable “materialized”
* [dbt run --full-refresh] #rebuild all incremental tables (useful when schema changes)
* [target-path: "target"] # directory which will store compiled SQL files, they will have hidden ephermal CTE, tests etc.
* [sources.yml] # we can assign aliases to snowflake tables and then use in models
* [dbt compile] # we can compile code before running it to check if it compiles
* [sources.yml -> freshness] # we can setup a warning and error if data is stale based on some column
* [dbt docs generate] – generates documentation files
* [dbt docs serve] – opens you docs on a server with nice interface