Refactoring SQL for Modularity

https://courses.getdbt.com/courses/refactoring-sql-for-modularity

Step 1: Migrate Legacy Code 1:1

- Transfer your legacy code to your dbt project as is as a .sql file in the models folder.
- Ensure that it can run and build in your data warehouse by running dbt run.
- Depending on the systems you are migrating between, you may need to adjust the flavor of SQL in your existing code to successfully build the model.

Step 2: Implement Sources / Translate Hard Coded Table References

- For each of the raw tables referenced in the new model, configure a source to map to those tables.
- Replace all the explicit table references in your query using the source macro.

Step 3: Choosing a Refactoring Strategy

Decide on your refactoring strategy...

1. **Refactor on top of the existing model** - create a new branch and refactor directly on the model that you created in the steps above.
2. **Refactor alongside the existing model** - rename the existing model by prepending it with legacy. Then copy the code into a new file with the original file name.

The second option plays better with the auditing in step 6.

Step 4: CTE Groupings and Cosmetic Cleanups

- Create one CTE for each source referenced at the top of your model.
- Reimplement subqueries as CTEs beneath the source CTEs.
- Update code to follow your style guide (at dbt Labs, we use all lowercase keywords, leverage whitespace for readability)
- **Resource:** dbt Labs, dbt style guide

Step 5: Centralizing Transformations & Splitting up Models

Structure your SQL into layers of modeling via staging models, intermediate models and final models.

- **Staging models**
  - Light transformations on source data should be captured in staging models
  - e.g. renaming columns, concatenating fields, converting data types
  - Update aliases with purposeful names
  - Scan for redundant transformations in the code and migrate into staging models.
  - Build dependencies between the existing model and the newly created staging models.
- **CTEs or intermediate models**
  - Inspect the grain of the transformations in latest version of the model, look for opportunities to move filters and aggregations into earlier CTEs
  - If the model code is lengthy or could be reusable in another case, break those CTEs into intermediate models.
- **Final models**
  - For the remaining logic, look for opportunities to simplify aggregations and joins.
  - It can also be helpful to update naming of CTEs for readability in the future.

Step 6: Auditing

- Audit your new model against your old query to ensure that none of the changes you implemented changed the results of the modeling.
- The goal is for both the original code and your final model to produce the same results.
- The audit_helper package can be particularly helpful here (audit_helper on hub.getdbt.com)