Jinja

Favorite	☐
Archived	☐
Fleeting	☐
Area/Resource	
Project	
Type	
Review Date	
Image	
URL	
Created	@March 21, 2023 5:37 PM
Updated	@March 21, 2023 6:50 PM
Root Area	
Project Area	
Updated (short)	03/21/2023
Pulls	
Resource Pulls	
Project Archived	
URL Base	
🍳 Recipe Divider	🥗🥗🥗 RECIPE BOOK PROPERTIES 🥗🥗🥗
🍳 Recipe Tags	
📚 Book Divider	📚📚📚 BOOK TRACKER PROPERTIES 📚📚📚
📚 Author	
📚 Date Started	
📚 Date Finished	
📚 Book Status	
📚 Rating	

💡 Jinja is a template engine using python. It is used to generate dynamic markup languages. It allows you to define variables and placeholders within a template which are replaced with real values at run time. It also enables control structures like loops, conditionals and macros to enable more complex template processing.

💡 Jinja is used with DBT to allow more modular data transformation logic. DBT models are defined using SQL but jinja can you allow you to inject SQL based on inputs or parameters.

Jinja syntax

Statements

💡 Statements are used for control flow

```
{% set variable='string' %} -- The curly bracket percentage enables you to use functions in Jinja
```

Expressions

💡 Expressions are used when you want to output a string

```
{{ variable }} -- The double curly brackets allow you to access variables
```

Comments

```
{# This is a comment #}
```

Macros

💡 The point of a macro is to add modularity in SQL and DBT so that we can create DRY code. Macros are a function of Jinja if you want to use more advanced macros concepts look at Jinja docs.

```
{% macro cents_to_dollars(column_name) %}
  {{ arg }} / 100
{% endmacro %}

select
  id,
  order_id,
  {{ cents_to_dollars('amount') as amount -- It is important to pass the column in quotes
from numbers                              -- or else it will look for a variable called amount.
```

```
select * from base

{% if target.name = 'dev' %}
  where time_stamp > 3
{% endif %}

{# THis is logic that can be turned into a macro for reusability #}

{% macro limit_time_in_dev(column_name) %}
  {% if target.name = 'dev' %}
    where {{ column_name }} > 3
  {% endif %}
{% endmacro %}

{# This macro will allow us to limit by 3 days #}

select * from base
{{ limit_time_in_dev('days' }}
```

Advanced macros

⚙️ We can see DBT jinja functions that are built directly into DBT.

Run queries

💡 The run_query built in DBT function allows you to run queries and return the results in a DBT macro. This will return a table object that is similar to a pandas df.

```
{% macro grant_select(schema=target.schema, role=target.role) %}
  {% set query %}
    grant usage on schema {{ schema }} to role {{ role }}
  {% endset %}

  {% do run_query(query) %}

{% endmacro %}
```

- [ ] What is run_operations?

Log

💡 The log function allows you to see under the hood. This is very useful when creating complex macros.

```
{{ log('Granting select on all tables and views in schema ' ~ schema ~ 'to role') }}
{# Since we are already inside a print statement, we do not need to add additional {{}} jinja understands
-- ~ is a concatenation trick #}
```

Execute

💡 execute is jinja variable that returns True when dbt is in ‘execute’ mode. When you execute a dbt compile or dbt run command, dbt at first reads all files and generates a manifest comprised of models, tests and other graph nodes in your project. No SQL is run during this phase so execute == False. The next stage DBT compiles and runs each node. SQL is run during this stage so execute == True.

💡 Without a if on the execute variable the jinja will compile prior to runtime. The error you are getting tells you an `if execute` is required. It occurs when DBT needs to execute some SQL to compile your models; this will generally fail because DBT normally compiles all models before execution, but `if execute` exists to allow you to defer compilation to the point of execution.

```
{% if execute %}
  {% set results = run_query(query).columns[0].values([0]) %}
  {{ log('SQL results' ~ results, info=True) }}

  select
```

```
    {{ results }} as is_real
  from a_real_table
{% endif %}
```

Relations

A relation is an object that is used to interpolate schema and table names into SQL coding with appropriate quoting.