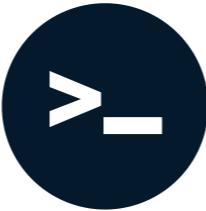


dbt sources

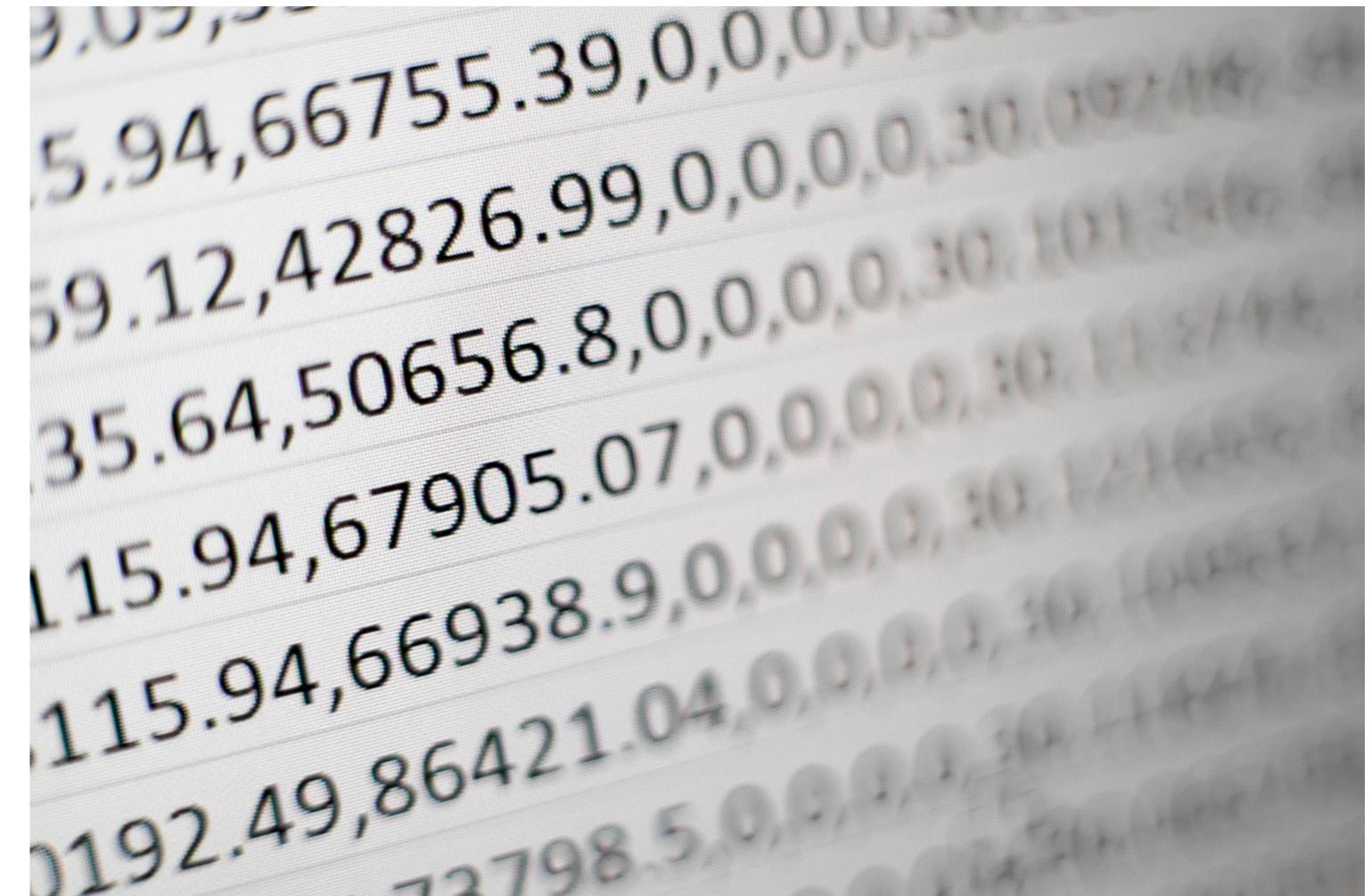
INTRODUCTION TO DBT



Mike Metzger
Data Engineer

What is a dbt source?

- Name and description of data loaded by EL process
 - Helps define data lineage
 - Tests
 - Documentation



¹ Photo by Mika Baumeister on Unsplash

Sources

- Provides data lineage information
- Describes the flow of data in warehouse
- Use the Jinja `{{ source() }}` function
 - Similar to the `{{ ref() }}` function
- Simplifies accessing raw data

```
select *  
from  
{{ source('raw', 'orders') }}
```

Defining a source

- In the .yml file
- File can be `models/model_properties.yml`
 - Or any other .yml file in the directory
- Under the `sources:` section
- Name the source with the `- name:` option
- Define each source table with a `- name:` option under `tables:`
- Different options available depending on the data warehouse type, refer to dbt documentation.

```
version: 2
```

```
sources:
```

```
  - name: raw
```

```
    tables:
```

```
      - name: phone_orders
```

```
      - name: web_orders
```

Accessing sources

- Use the `{{ source() }}`
- `{{ source(source_name, table_name) }}`
- Provides the proper name of the table in the compiled query

```
select * from
{{ source('raw', 'phone_orders') }}
UNION
select * from
{{ source('raw', 'web_orders') }}
```

```
-- dbt compiled
select * from
'raw'.'phone_orders'
UNION
select * from
'raw'.'web_orders'
```

Testing sources

- You can apply tests to sources
- Same methods as applying to models
- Defined in the `sources:` section instead of `models:`
- In `.yml` file where sources defined

```
version: 2

sources:
  - name: raw
    tables:
      - name: phone_orders
        columns:
          - name: id
            tests:
              - not_null
              - unique
              - name: web_orders
```

Documentation

- Document using the same tools as models
- Defined in the `sources:` section

```
version: 2
```

```
sources:
```

```
- name: raw
```

```
tables:
```

```
- name: phone_orders
```

```
description: >
```

```
Sales orders by phone, daily
```

```
columns:
```

```
- name: id
```

```
tests:
```

```
- not_null
```

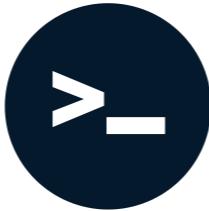
```
- unique
```

Let's practice!

INTRODUCTION TO DBT

dbt seeds

INTRODUCTION TO DBT



Mike Metzger
Data Engineer

What are dbt seeds?

- CSV files to be loaded into data warehouse
- Typically rarely changing sets of data
 - List of countries
 - List of postal codes
- *NOT* meant for raw data



¹ Photo by Maddi Bazzocco on Unsplash

Why?

- Easy to manage
- Easy to use in various scenarios
- Source controllable

How are seeds defined?

- Add CSV file to the `seeds` directory
- Make sure the header is the first row
- Import using the `dbt seed` command

```
zipcode,place,state  
99553,Akutan,Alaska  
99571,Cold Bay,Alaska  
99583,False Pass,Alaska  
bash> dbt seed
```

¹ Postal codes provided via <https://github.com/zauberware/postal-codes-json-xml-csv>

Further configuration

- Several options
 - Which schema?
 - What database?
 - Column quoting
 - Column data types
- Can be applied to the whole project or to individual seeds
- Can be added to `dbt_project.yml` or `seeds/properties.yml`

Defining datatypes

- Available data types depend on data warehouse
- Typical ones are available
 - Integer
 - Varchar
 - etc
- If type is not defined, it is inferred based on the data

```
version: 2
```

```
seeds:
```

```
- name: zipcodes
```

```
config:
```

```
column_types:
```

```
zipcode: varchar(5)
```

Tests & documentation

- Support tests
- Support documentation
- Just like models and sources

```
version: 2
```

```
seeds:
```

```
- name: zipcodes
```

```
description: US zipcodes
```

```
config:
```

```
column_types:
```

```
zipcode: varchar(5)
```

```
columns:
```

```
- name: zipcode
```

```
tests:
```

```
- unique
```

Accessing seeds

- Available via the `{{ ref() }}` command
- Behaves as a model after initial import

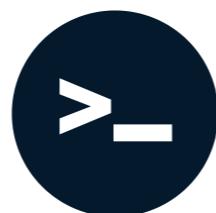
```
select * from  
{{ ref('zipcodes') }}
```

Let's practice!

INTRODUCTION TO DBT

SCD2 with dbt snapshots

INTRODUCTION TO DBT



Mike Metzger

Data Engineer

What is a snapshot?

- A look into the changes of a dataset over time
- Illustrate the various states of an object, such as
 - Order status
 - Production state
 - Shipping status



¹ Photo by micheile henderson on Unsplash

SCD2

- Slowly changing dimension
 - Type 2
 - SCD2
 - Kimball-style data warehousing
 - *Introduction to Data Warehousing*
- Tracks changes over time
- dbt implements SCD2 with snapshots



¹ Photo by Luke Chesser on Unsplash

SCD2 example

- Order status
- Available states:
 - Received
 - Packed
 - Shipped

id	order_status	last_updated
1	Shipped	2023-07-01 11:30

id	order_status	last_updated
1	Received	2023-07-01 10:45
1	Packed	2023-07-01 11:15
1	Shipped	2023-07-01 11:30

SCD2 in dbt

- dbt uses snapshots to implement SCD2
- Can track the changes automatically
- Adds extra columns to the output
 - `dbt_valid_from`
 - `dbt_valid_to`

<code>id</code>	<code>order_status</code>	<code>last_updated</code>	<code>dbt_valid_from</code>	<code>dbt_valid_to</code>
1	Received	2023-07-01 10:45	2023-07-01 10:45	2023-07-01 11:15
1	Packed	2023-07-01 11:15	2023-07-01 11:15	2023-07-01 11:30
1	Shipped	2023-07-01 11:30	2023-07-01 11:30	null

Implementing dbt snapshots

- SQL file, `snapshots/snapshot_name.sql`

```
{% snapshot snapshot_orders %}

  {{ config(
    target_schema='snapshots',
    strategy='timestamp',
    unique_key='id',
    updated_at='last_updated'
  )}

}

  select * from {{ source('raw', 'orders') }}

{% endsnapshot %}
```

dbt snapshot

- Run `dbt snapshot`
- Create new model using the `ref()` command to query snapshot
 - `select * from {{ ref('snapshot_orders') }}`
- Run `dbt snapshot` frequently to see potentially changed data
 - Schedule for automatic updates

Let's practice!

INTRODUCTION TO DBT

Automating with dbt build

INTRODUCTION TO DBT



Mike Metzger
Data Engineer

Review

- `sources` and `seeds` feed initial data to dbt
- `models` handle the transformation of data (usually from `sources` / `seeds`) for downstream users
- `snapshots` track changes in datasets
- `tests` can validate `sources`, `seeds`, `models`, and even `snapshots`
 - Built-in (`unique`, `not_null`, `relation`, `accepted_values`)
 - Singular
 - Generic / Reusable
- `dbt build` performs all these tasks, usually in production

dbt build

dbt build :

- Combination of multiple tasks
- Runs models (dbt run)
- Run validations via tests (dbt test)
- Create snapshots (dbt snapshot)
- Process any seeds (dbt seed)

Remember the commands can be run individually if required



¹ Photo by Randy Fath on Unsplash

Why?

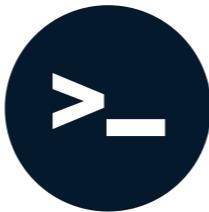
- Individual subcommands work well, but don't handle all potential issues
 - `dbt run` doesn't validate first (ie, no tests are run)
 - `dbt snapshot` isn't run to maintain any potential changes
 - `dbt seed` may not be complete for certain queries
- `dbt build` will determine dependencies and run all tests prior to production changes
- `dbt build` may be overkill if only testing or small incremental changes are made
- Steps can be run manually instead if required

Let's practice!

INTRODUCTION TO DBT

Course review

INTRODUCTION TO DBT



Mike Metzger
Data Engineer

What we've learned

- dbt commands
 - `dbt run` , `dbt test` , `dbt -h`
- Projects in dbt
 - General folder structure
 - `dbt_project.yml`
- Creating dbt models with SQL
 - Defining the model in SQL files
 - Modifying the configurations in YAML
- Building on top of existing models
 - Using Jinja `{{ ref() }}` function to build lineage
- Validating our data through testing
 - Creating and applying various tests
 - Built-in
 - Singular tests in SQL
 - Generic tests

What we've learned (continued)

- Documentation
 - Ability to automatically document dbt objects via YAML
 - Generating and serving documentation via dbt docs
- Creating dbt sources
 - Defining lineage
 - Adding testing and documentation
- Loading reference data with dbt seeds
 - Creating new seeds via YAML configuration
 - Loading CSV files into data warehouse with dbt seed
- Creating SCD2-style tracking with dbt snapshot
 - Adding and tracking changes to a dataset via dbt

What we've learned (final)

Troubleshooting

- Models
- Tests
- Snapshots
- Projects

Production considerations

- Using `dbt build`
- Fixing errors with unfamiliar projects

Potential next topics

- Incremental models
 - Loading partial data changes into warehouse without re-loading all data
- More advanced SQL
 - CTEs
- Jinja commands and macros
 - `{{ ENV() }}`, `{% for %}`
- Building models with Python
- Documentation blocks
- Production & automation
 - Adding hooks to run tasks automatically
 - Integrating with orchestrators (such as Airflow)

References

- dbt Documentation
 - <https://docs.getdbt.com>
- dbt Slack channel
- Countless blogs / newsletters
- Future DataCamp courses

Thank you!

INTRODUCTION TO DBT