dbt (data build tool)

powerful, open source data transformations

Connor McArthur

September 25, 2018

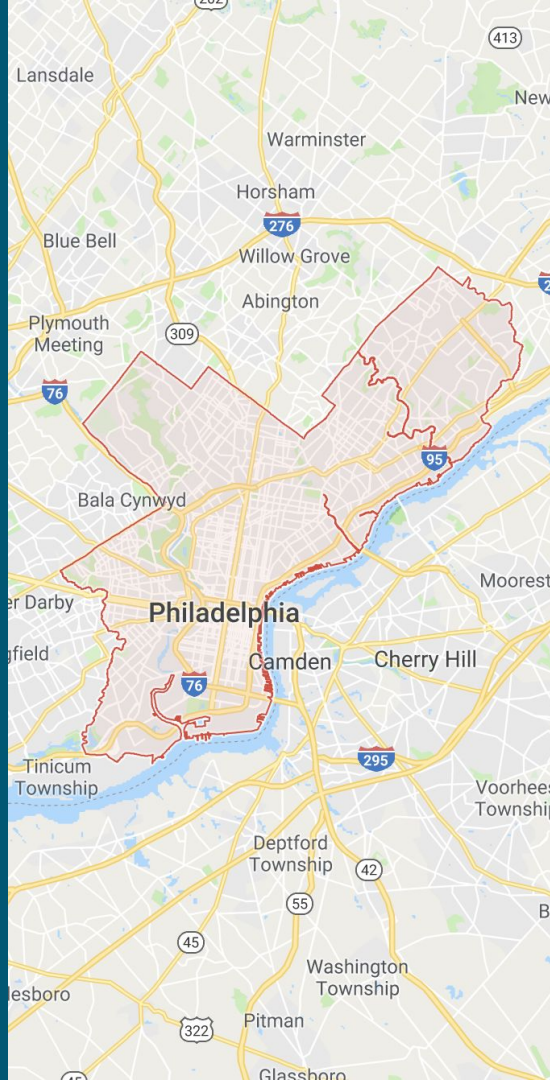
FISHTOWN ANALYTICS

At Fishtown Analytics, our goal is to improve the way analytics work is done.

We run a consultancy, and work with some awesome companies worldwide.

We build a few products, most notably **dbt (data build tool)**.

We're based in **Philadelphia, PA, USA**.



We are practitioners

We do **analytics**, **data science**, and **data engineering** work with our clients.

This talk and dbt itself are both heavily informed by the needs of our employees and users as **practitioners**.

What do **data engineers** do?

What do data engineers do?

Organizationally, they fill two roles:

1. **Shepherds**: we shepherd the organization's data
2. **Librarians**: we understand the data catalog

As engineers do, we automate as much as we can…

...but we often don't do enough!

Case Study

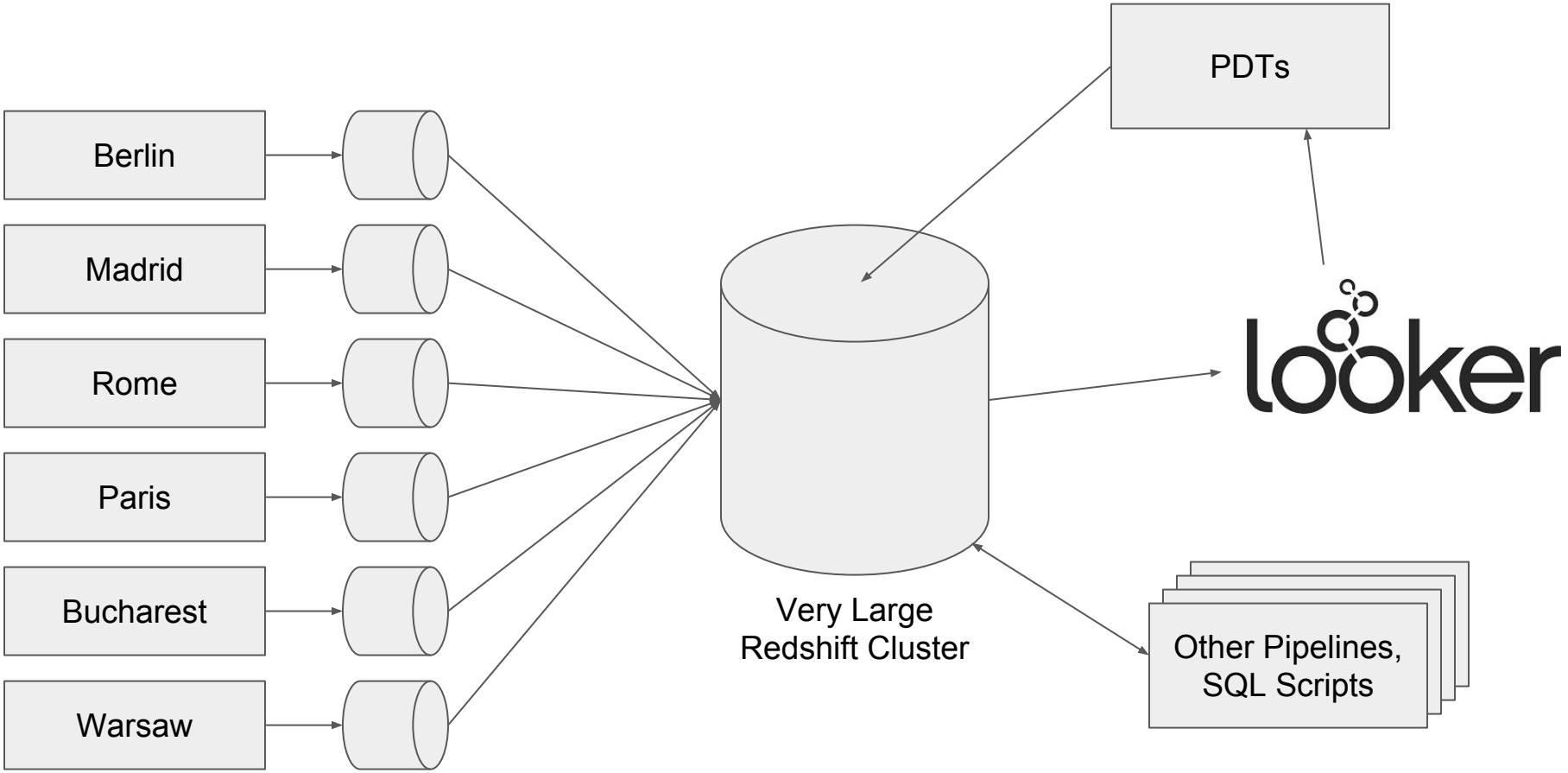
I worked with a medium sized hardware + software tech company for about 6 months.

The data team at that time consisted of:

1 data engineer

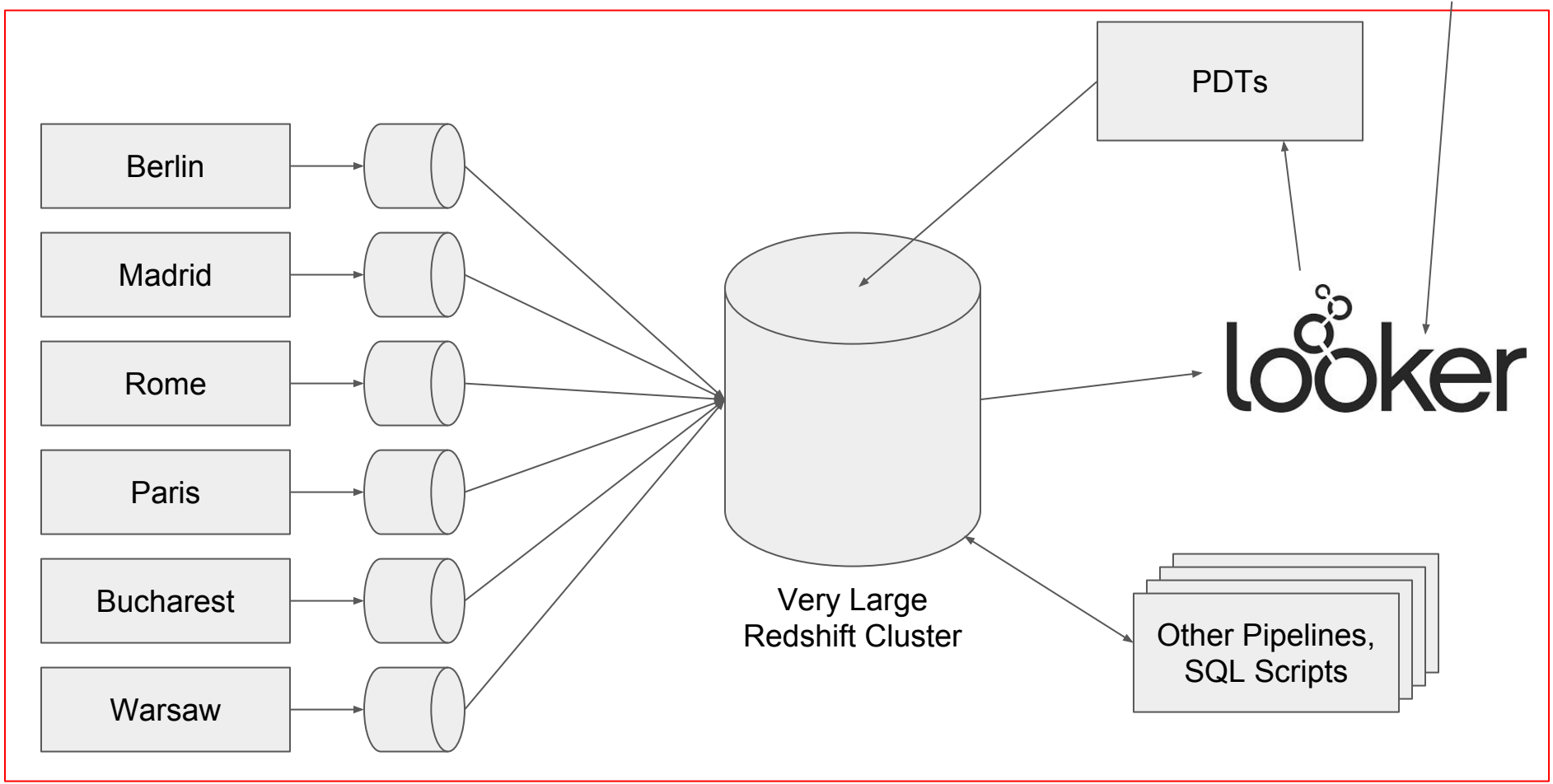
1 data analyst

their manager


Berlin
Madrid
Rome
Paris
Bucharest
Warsaw
PDTs
looker
Very Large
Redshift Cluster
Other Pipelines,
SQL Scripts

Their data engineer owned this part

Their analyst used this part



"Engineers must deploy **platforms**, **services**, **abstractions**, and **frameworks** that allow the data scientists to **conceive of**, **develop**, and **deploy** their ideas with autonomy."

Engineers Shouldn't Write ETL: A Guide to Building a High Functioning Data Science Department, 2016, Jeff Magnusson

https://multithreaded.stitchfix.com/blog/2016/03/16/engineers-shouldnt-write-etl/

The big idea

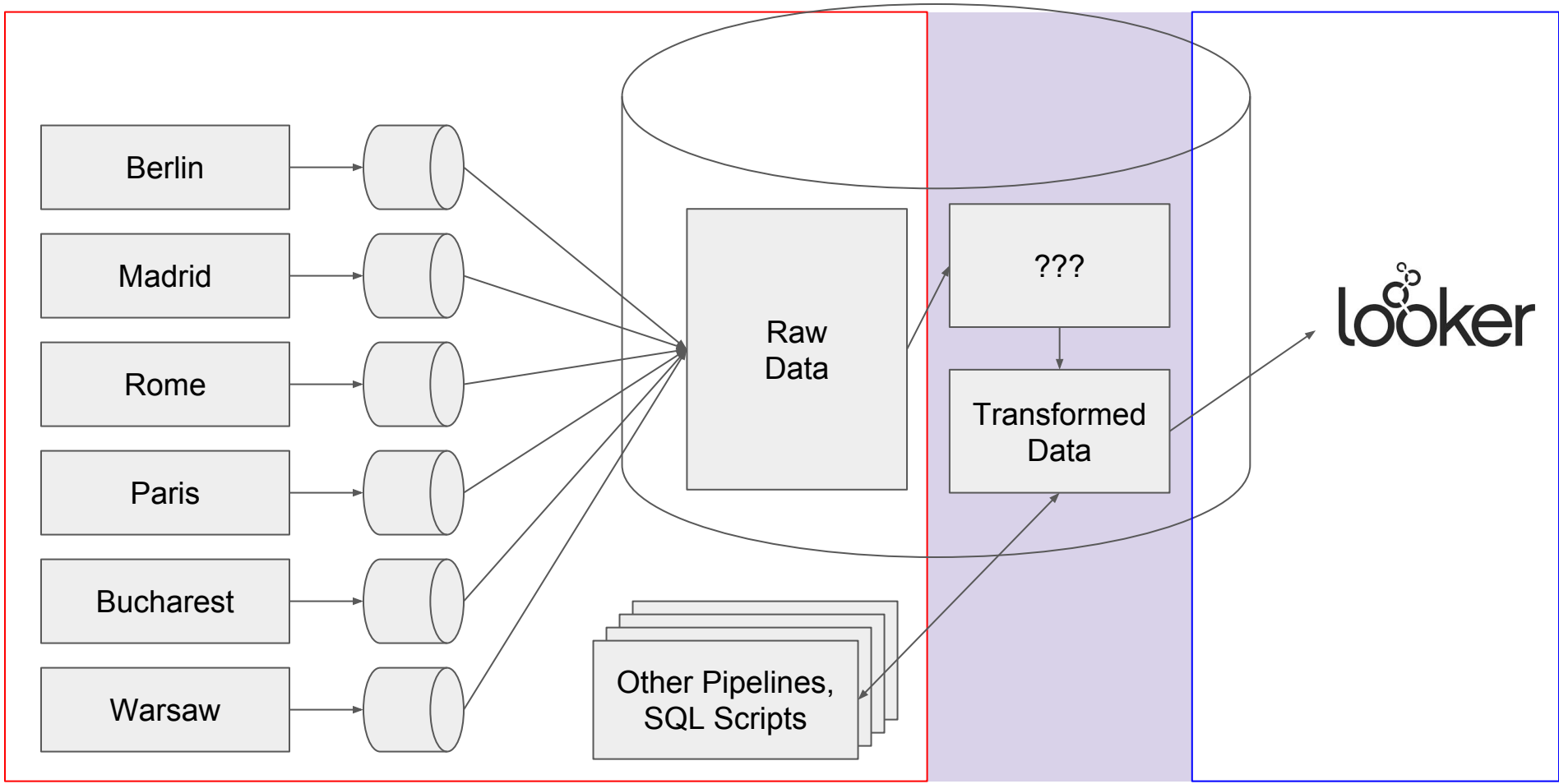
What if we gave our analysts:

1. A **transformation platform** of their own in addition to bespoke pipelines
2. A **data catalog** in place of answering data lineage questions directly

This would save our data engineers time, and would let our analysts contribute materially to the transformations.

The data engineer owns this part

The analysts own this part



dbt_

dbt (data build tool) is an open source, command line based, transformation tool.

Designed for all of the SQL authors at your organization.

~2 years old, >250 weekly active projects.

100% free and open source. You can install with `pip install dbt`, or learn more at https://www.getdbt.com.

Overview

dbt is a **developer tool**. Your team members use a terminal and text editor of their choice. (NOT VIM)

They write SQL transformations, then execute them, then automated test them, then validate & deploy them.

When using dbt, your analysts behave like software developers.

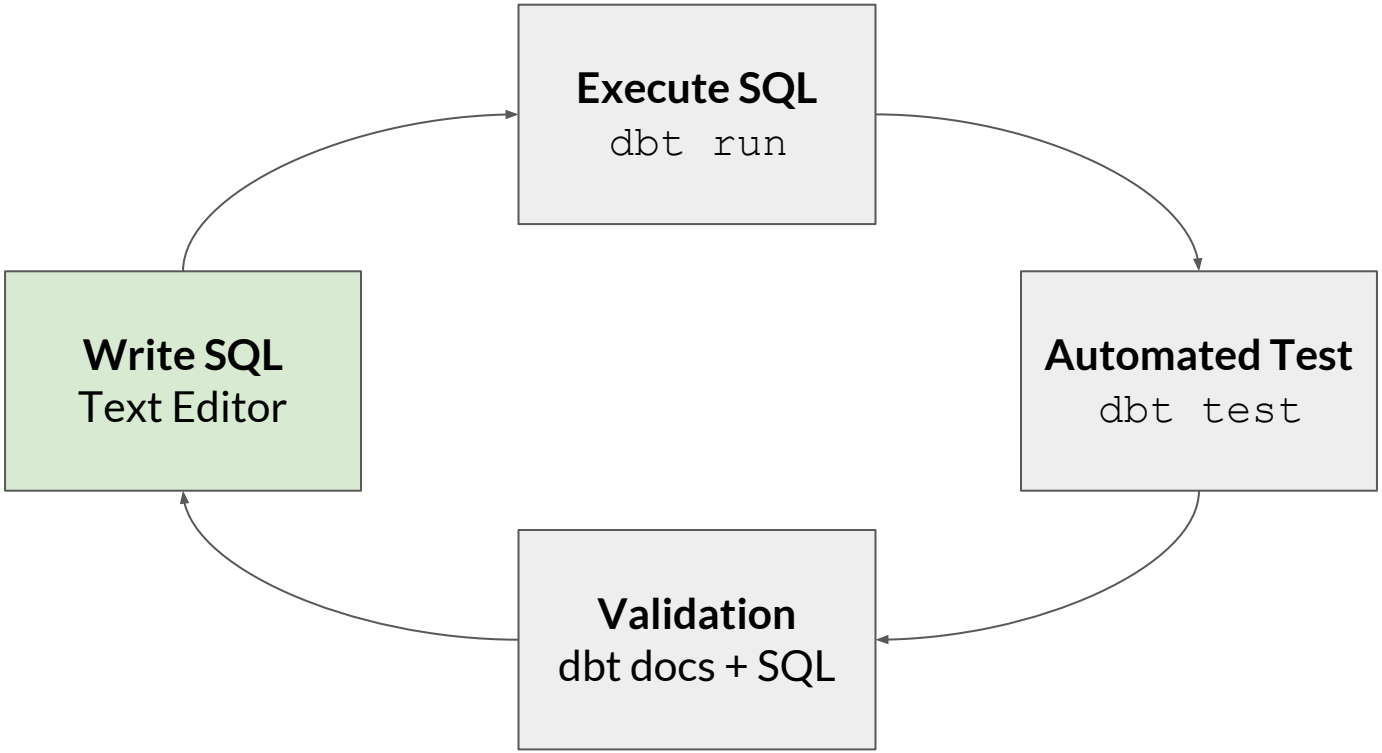
SQL Warehouses

dbt supports **SQL only**. But, modern data warehouses can handle a lot!

It works with:

- **Postgres**: snappy performance for minimal spend
- **Amazon Redshift**: up to and around one billion rows
- **Snowflake** and **Google Bigquery**: many billions of rows

(Wanted: experts in other warehouse tech)


Execute SQL
dbt run
Automated Test
dbt test
Validation
dbt docs + SQL
Write SQL
Text Editor

Transformations

The atomic unit of transformation in dbt is called a *model*.

Each model lives in one SQL file stored in a git repository.

```
# build "accounts"
$ dbt run --model accounts
```

accounts.sql

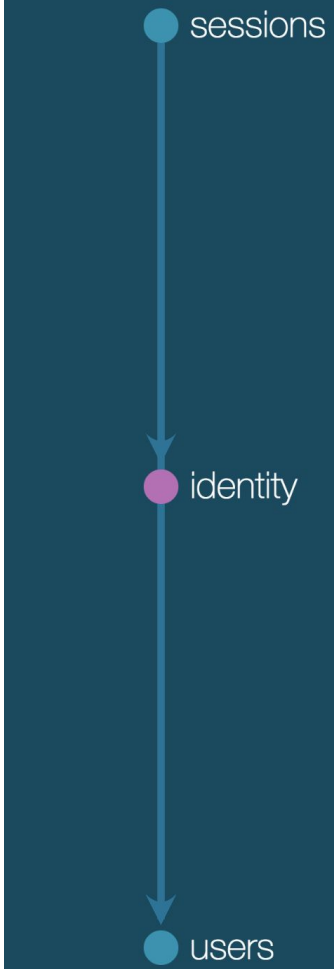
```
1 | select
2 |   id as account_id,
3 |   case when pid = 1
4 |        then 'free'
5 |        ...
6 |   end as plan_name
7 | from accounts
```

Models Build on Models

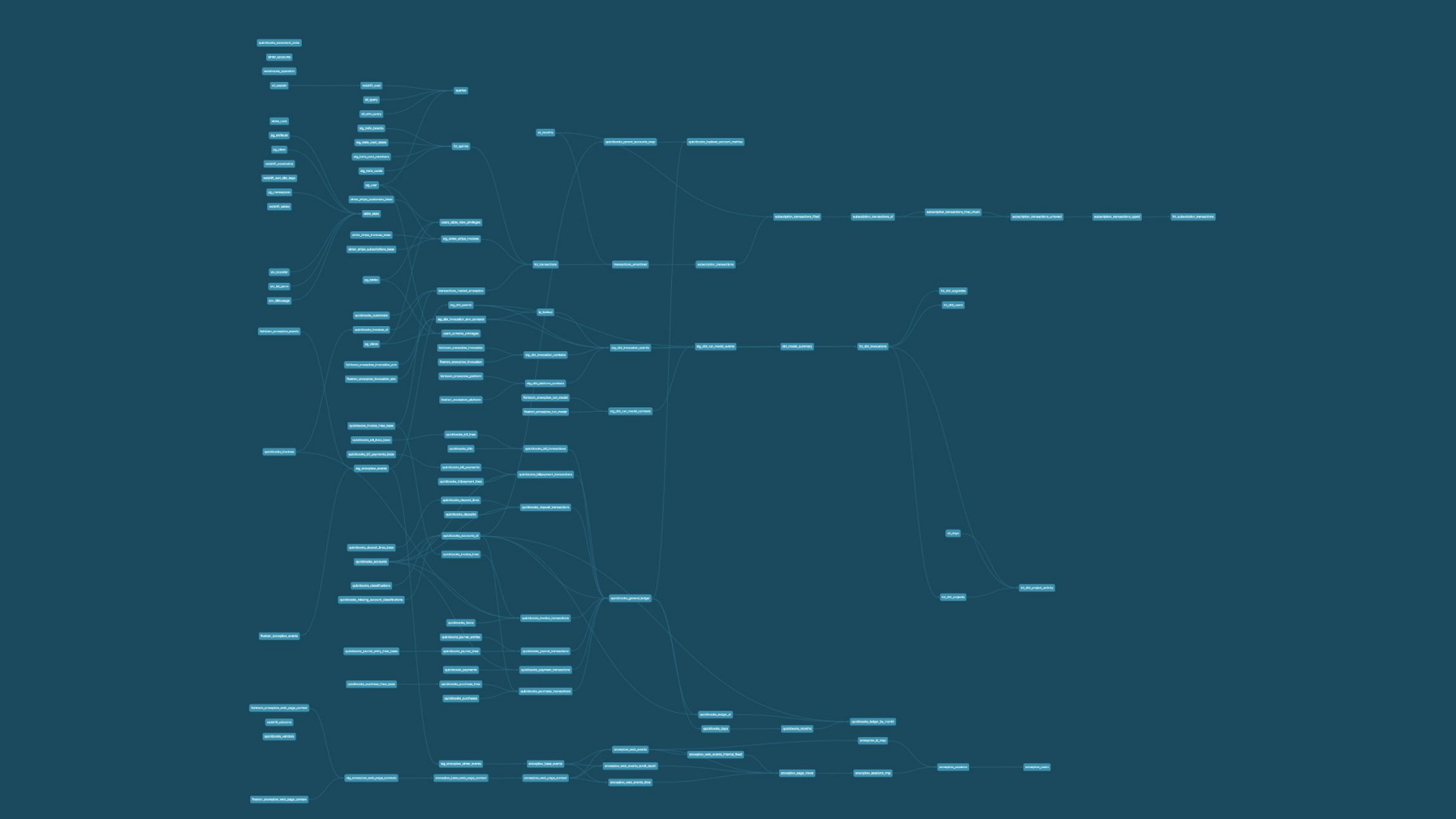
Every dbt model defines all of its inputs using a special Jinja function called `ref()`.

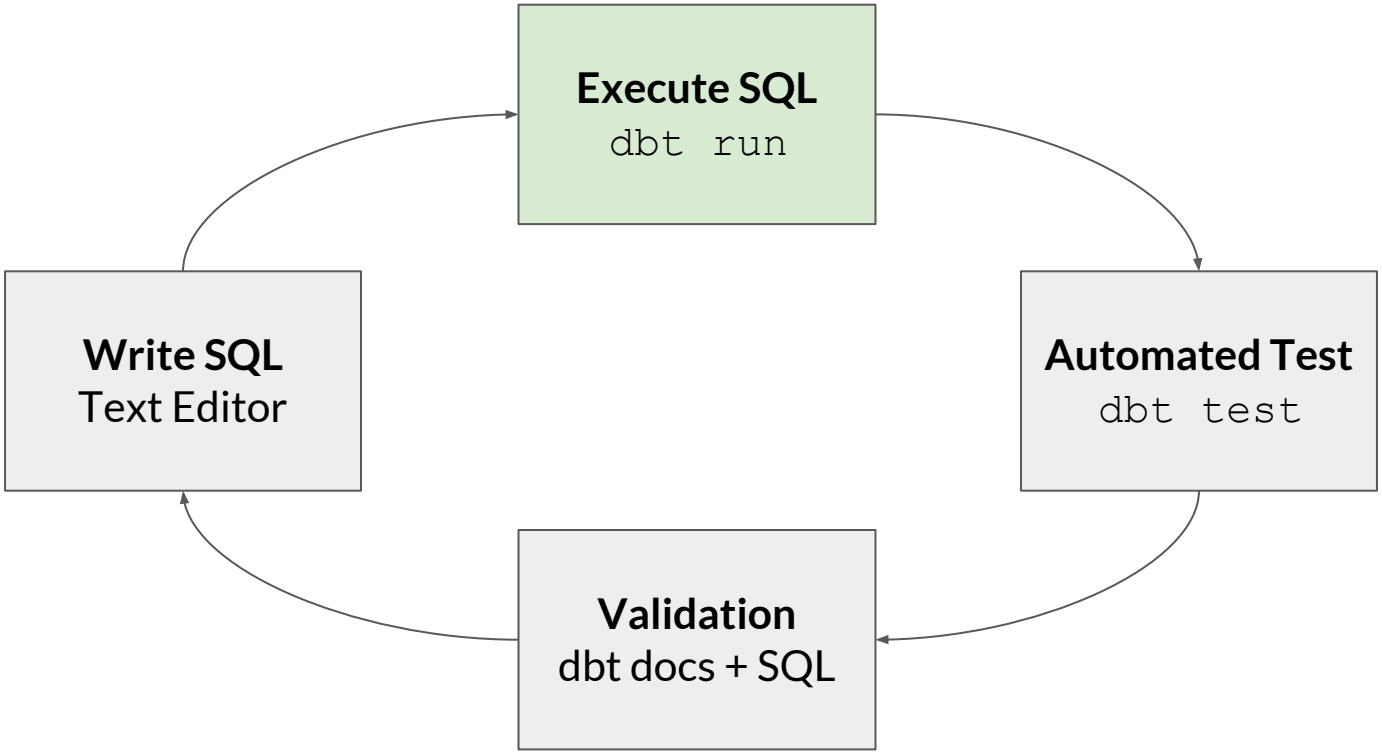
dbt analyzes these to build up a directed acyclic graph (DAG) of your models. Then, **dbt can build any subgraph you'd like in the proper order**.

```
$ dbt run --models +identity+
```




page_view_timing
web_events
page_views
sessions
identity
users


quickbooks_payment_mod
driver_accounts
warehouse_operation
int_payroll
kafka01_cust
int_query
int_alert_query
stripe_core
pg_attribute
pg_class
kafka01_consumers
kafka01_partition_logs
pg_namespace
kafka01_topics
stripe_charges
stripe_customers_base
stripe_disputes
stripe_prices_invoices_base
stripe_prices_subscriptions_base
stg_invoices
stg_refunds
stripe_customers
stripe_invoices_int
stg_disputes
fivetran_prosepilot_events
fivetran_prosepilot_invocation_error
freshpo_prosepilot_invocation_poc
quickbooks_invoice
quickbooks_invoice_f_test_base
quickbooks_bill_lines_base
quickbooks_bill_payments_base
stg_prosepilot_events
fivetran_prosepilot_web_page_context
kafka01_consumers
kafka01_partition_offsets
freshpo_prosepilot_web_page_context
quarter
kafka01_cust
int_query
int_alert_query
int_customers
int_users
int_subscriptions
int_transactions
int_invoices
int_refunds
int_disputes
int_charges
int_payments
int_payment_intents
int_payment_methods
int_payment_method_details
int_payment_intent_details
int_payment_charges
int_payment_refunds
int_payment_disputes
int_payment_balance_transactions
int_payment_events
int_payment_customers
int_payment_invoices
int_payment_subscriptions
int_payment_prices
int_payment_products
int_payment_coupons
int_payment_discounts
int_payment_tax_rates
int_payment_tax_ids
int_payment_refunds_and_disputes
int_payment_payouts
int_payment_payout_balance_transactions
int_payment_payment_intents
int_payment_payment_methods
int_payment_payment_method_details
int_payment_payment_intent_details
int_payment_payment_charges
int_payment_payment_refunds
int_payment_payment_disputes
int_payment_payment_balance_transactions
int_payment_payment_events
int_payment_payment_customers
int_payment_payment_invoices
int_payment_payment_subscriptions
int_payment_payment_prices
int_payment_payment_products
int_payment_payment_coupons
int_payment_payment_discounts
int_payment_payment_tax_rates
int_payment_payment_tax_ids
int_payment_payment_refunds_and_disputes
int_payment_payment_payouts
int_payment_payment_payout


Execute SQL
dbt run
Automated Test
dbt test
Validation
dbt docs + SQL
Write SQL
Text Editor

Command Line Interface

dbt can select different parts of the DAG to execute.

```
# run all the models
$ dbt run

# run one specific model
$ dbt run --models <model-name>

# run a model and those that depend on it
$ dbt run --models <model-name>+
```

This works the same way everywhere.

Materializations

Materializations define *how* to turn a model into a warehouse object. dbt has some built-in:

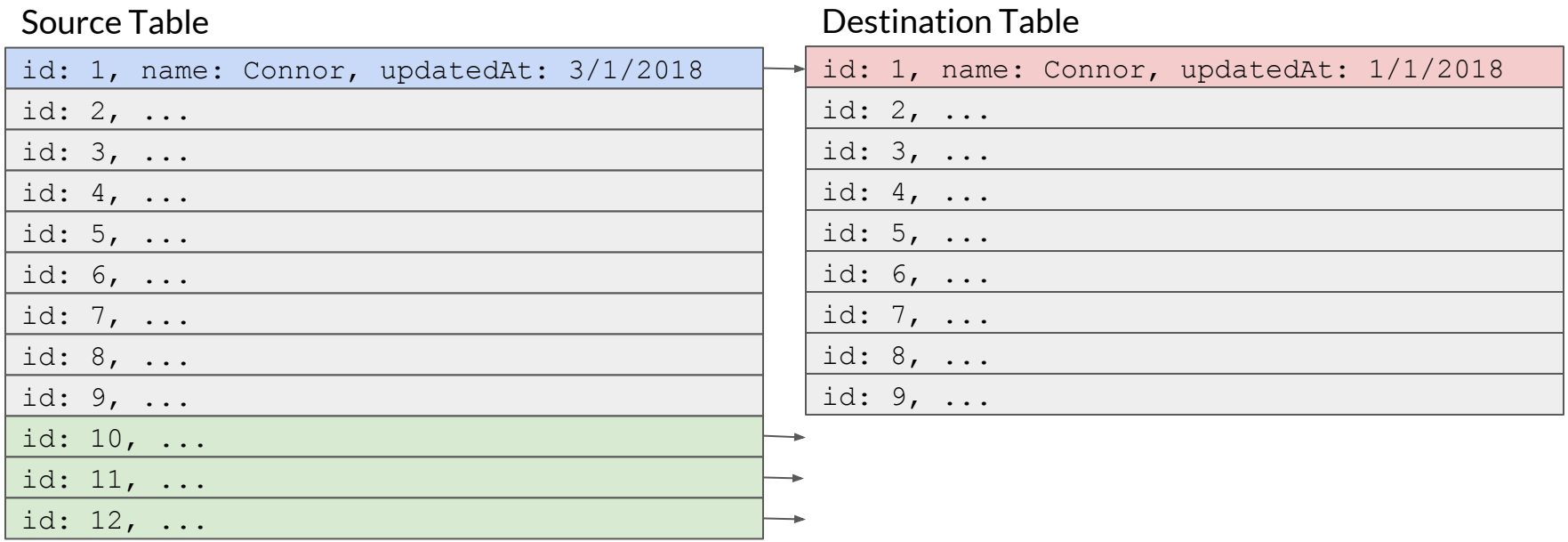
- **view**: `CREATE VIEW AS`
- **table**: `CREATE TABLE AS`
- **incremental**: `DELETE` outdated records, `INSERT` new records

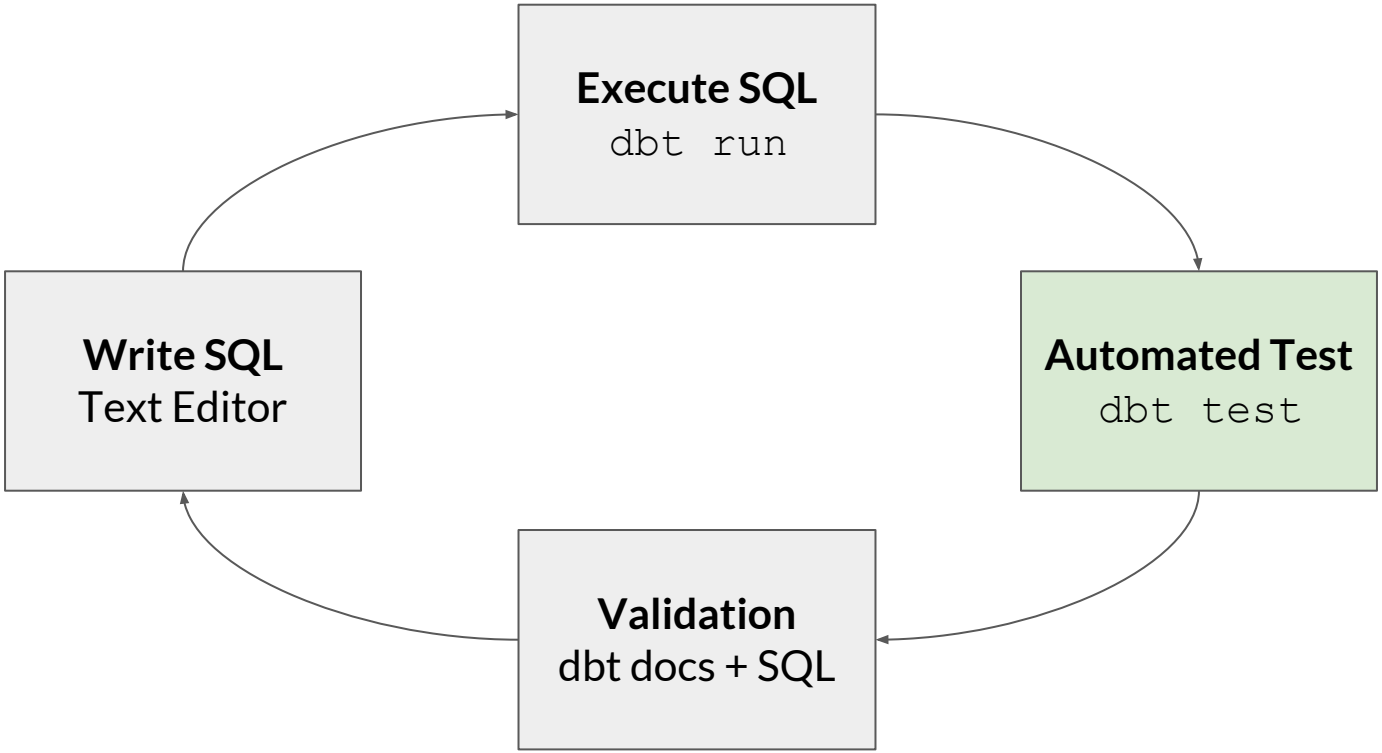
```
{{config(
    materialized='table'
)}}

  select
    ...
  from accounts
```

is rendered to:

```
  CREATE TABLE ... AS (
    select
      ...
    from accounts
  );
```

Incremental Models




Execute SQL
dbt run
Automated Test
dbt test
Validation
dbt docs + SQL
Write SQL
Text Editor

Testing

In software development, we want automated tests to prevent bugs from hitting production.

In a warehouse context, automated tests can catch things like

- bad calculations
- bad joins / fanouts
- schema mismatches

dbt provides this functionality via tests of two kinds.

SQL Tests

dbt tests are `SELECT` statements that grab failing records.

test.sql

```
select *
from {{ref('accounts')}}
where id is not null
```



```
with test_query as (
  select *
  from accounts
  where id is not null
)

select count(*)
from test_query
```

Schema Tests

Replace constraints in a traditional RDBMS.

Out of the box, you can test **Uniqueness**, **Not Null**, and **Foreign Key** constraints.

You can also define custom schema tests.

schema.yml

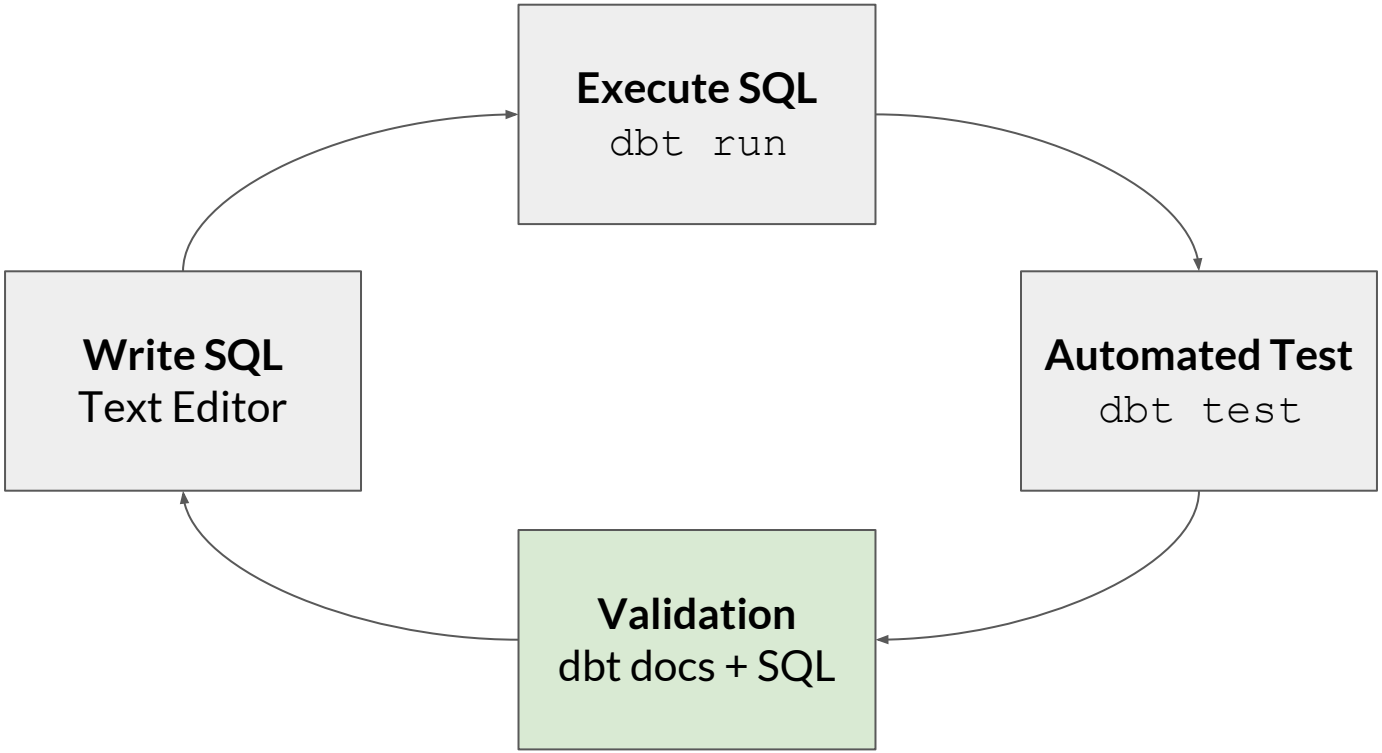
```
models:
  - name: accounts
    columns:
      - name: id
        tests:
          - unique
          - not_null
```

Schema Tests

schema.yml

```
models:
  - name: accounts
    columns:
      - name: id
        tests:
          - unique
          - not_null
```

```
$ dbt test --models accounts

Concurrency: 1 threads
1/2 unique_accounts_id [RUN]
1/2 unique_accounts_id [PASS
in 0.02s]

...
```


Execute SQL
dbt run
Automated Test
dbt test
Validation
dbt docs + SQL
Write SQL
Text Editor

Auto-documentation

dbt ships with a documentation generator. It inspects:

- The contents of your project
- The metadata provided in schema.yml files
- The relations in your warehouse

It uses this data to create a fully interactive catalog for your warehouse.

dbt_

Search for models...

Overview

Project | Database

- dbt_cmcarthur
  - identity
  - page_view_timing
  - page_views
  - sessions
  - users
  - web_events

sessions table

Details | Description | Columns | SQL

Details

DETAILS	OWNER	TYPE	PACKAGE	RELATION
	root	table	snowplow	dbt_cmcarthur.sessions

Description

This table contains one record per session, with the `session_id` field being the primary key. Page views are rolled up to sessions, where 30 minutes of inactivity will close the existing session. The next time a user visits the site, a new session will be generated.

Domains:

DOMAIN	TRACKING ENABLED?
somedomain.com	yes
anotherdomain.com	no

- Sources: Snowplow
- Owner: Drew Banin

Columns

COLUMN	TYPE	DESCRIPTION	TESTS	MORE?
session_id	text	A unique identifi...	U N	›

dbt_

Search for models...

Overview

Project | Database

- snowplow
  - models
    - identity
      - identity
    - page_views
    - sessions
      - sessions
    - users

sessions table

Details | Description | Columns | SQL

Details

DETAILS	OWNER	TYPE	PACKAGE	RELATION
	root	table	snowplow	dbt_cmcarthur.sessions

Description

This table contains one record per session, with the `session_id` field being the primary key. Page views are rolled up to sessions, where 30 minutes of inactivity will close the existing session. The next time a user visits the site, a new session will be generated.

Domains:

DOMAIN	TRACKING ENABLED?
somedomain.com	yes
anotherdomain.com	no

- Sources: Snowplow
- Owner: Drew Banin

Columns

COLUMN	TYPE	DESCRIPTION	TESTS	MORE?
session_id	text			

dbt_

Search for models...

Overview

Project | Database

- dbt_cmcarthur
  - identity
  - page_view_timing
  - page_views
  - sessions
  - users
  - web_events

sessions table

Details | Description | Columns | SQL

SQL

Source | Compiled | copy to clipboard

```
{{ config(materialized='table') }}

with page_views as (

    select * from {{ ref('page_views') }}

),

sessionized as (

    select
        session_id,
        inferred_user_id,

        -- time
        min(page_view_start) as session_start,
        max(page_view_end) as session_end,

        -- engagement
        count(*) as page_views,
        sum(time_engaged_in_s) as time_engaged_in_s,

        max(case when page_view_in_session_index = 1 then page_url else null end) as landing_pa

    from page_views
    group by 1, 2
```

dbt_

Search for models...

Overview

Project | Database

- dbt_cmcarthur
  - identity
  - page_view_timing
  - page_views
  - sessions
  - users
  - web_events

sessions table

Details | Description | Columns | SQL

SQL

Source | Compiled

copy to clipboard

```
with page_views as (

    select * from "dbt_cmcarthur"."page_views"

),

sessionized as (

    select
        session_id,
        inferred_user_id,

        -- time
        min(page_view_start) as session_start,
        max(page_view_end) as session_end,

        -- engagement
        count(*) as page_views,
        sum(time_engaged_in_s) as time_engaged_in_s,

        max(case when page_view_in_session_index = 1 then page_url else null end) as landing_pa

    from page_views
    group by 1, 2
```

dbt_

Search for models...

Overview

Project | Database

- dbt_cmcarthur
  - identity
  - page_view_timing
  - page_views
  - sessions
  - users
  - web_events

sessions table

Details | Description | Columns | SQL

Details

DETAILS	OWNER	TYPE	PACKAGE	RELATION
	root	table	snowplow	dbt_cmcarthur.sessions

Description

This table contains one record per session, with the `session_id` field being the pr
up to sessions, where 30 minutes of inactivity will close the existing session. The n
new session will be generated.

Domains:

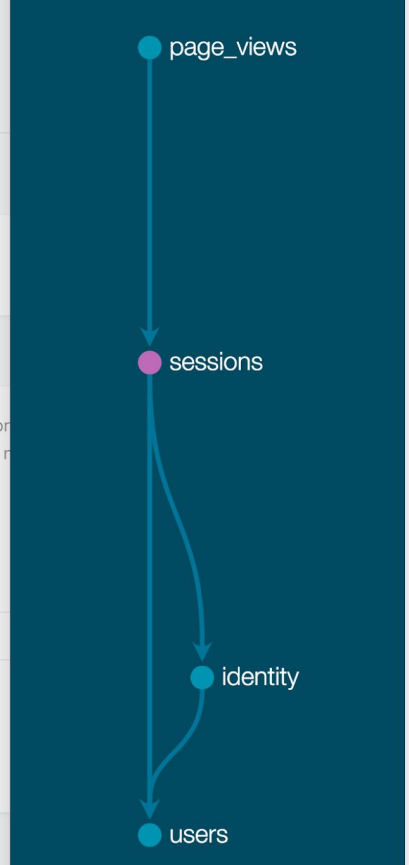
DOMAIN	TRACKING ENABLED?
somedomain.com	yes
anotherdomain.com	no

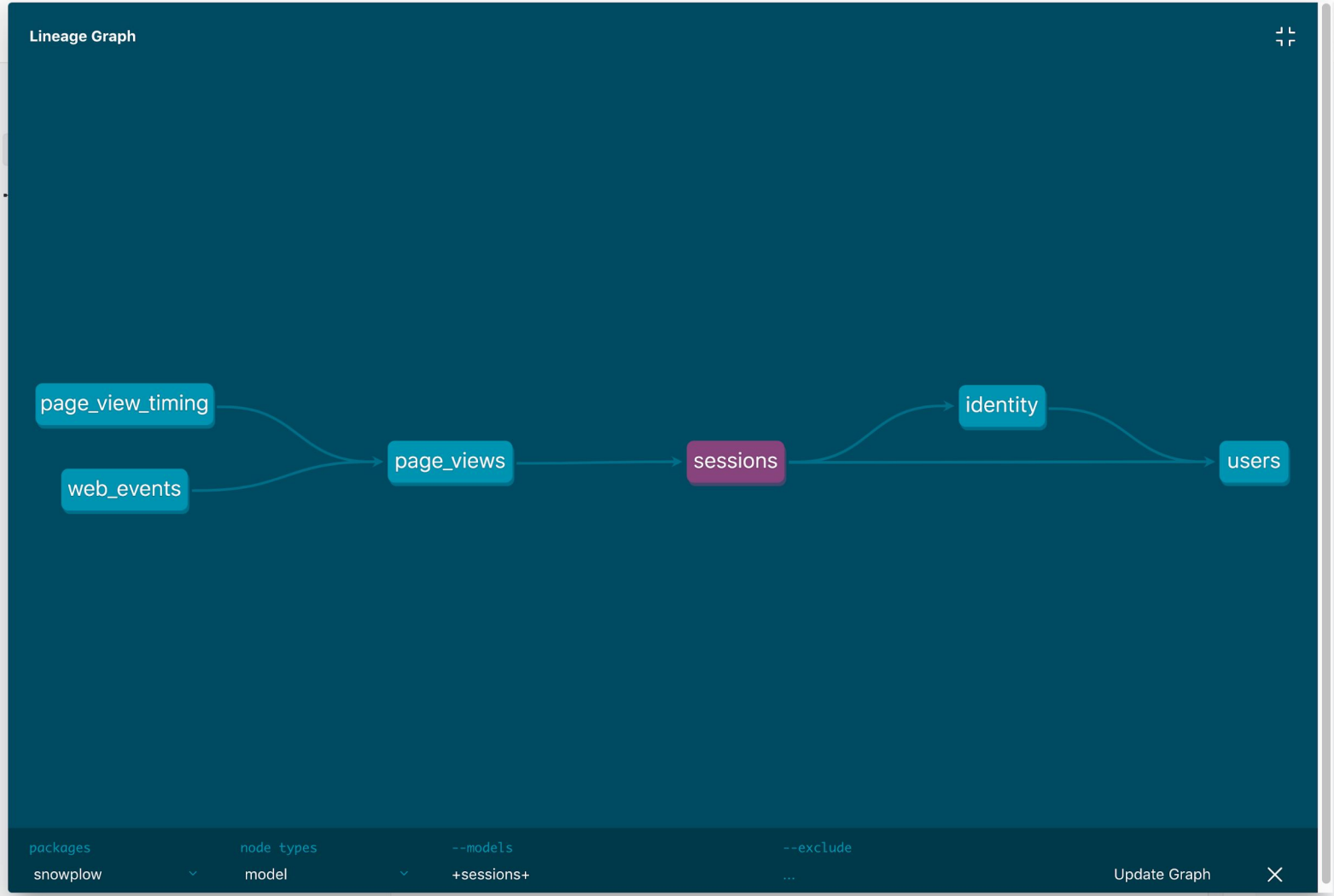
- Sources: Snowplow
- Owner: Drew Banin

Columns

COLUMN	TYPE	DESCRIPT
session_id	text	A unique identifi... U N

Lineage Graph




Lineage Graph
page_view_timing
web_events
page_views
sessions
identity
users
packages
snowplow
node types
model
--models
+sessions+
--exclude
...
Update Graph

Deployment

Repeatability

Repeatability a key feature of dbt -- running dbt twice on the same set of inputs should always yield the same result.

This means each user can:

- Build feature branches in development,
- Then run **continuous integration processes**,
- Then **deploy to production**.

Just as you would with any other software project.

Continuous Integration & Deployment

Our users have deployed dbt with:

- Sinter, a SaaS backend for dbt that we developed
- Airflow
- Kubernetes
- Cron
- Jenkins
- Gitlab CI
- CircleCI
- Travis CI
- … and more

Demo

dbt_

Thanks for listening!

Find out more at:

www.getdbt.com

Image Credits

Slide 2: Screenshot taken from Google Maps https://maps.google.com/

Slide 8, 9, 12: Looker logo property of Looker Data Sciences, Inc.® https://looker.com/

Slide 13, 42: dbt logo property of Fishtown Analytics https://fishtownanalytics.com

Slide 18, 19, 20, 32, 33, 34, 35, 36, 37: screenshots of dbt docs property of Fishtown Analytics https://fishtownanalytics.com