What are we going to build?

Our Silver & Gold Tables

We're going to build 2 tables, test them, and document them:

- *Note: Our Bronze (raw data) has been loaded by Fivetran*
- **Silver:**
  - dim_customers: a dimension table holding attributes about our customer, plus a unique_id we will generate
- **Gold:**
  - top_customers: a summary table of our top 100 customers by spend for this business

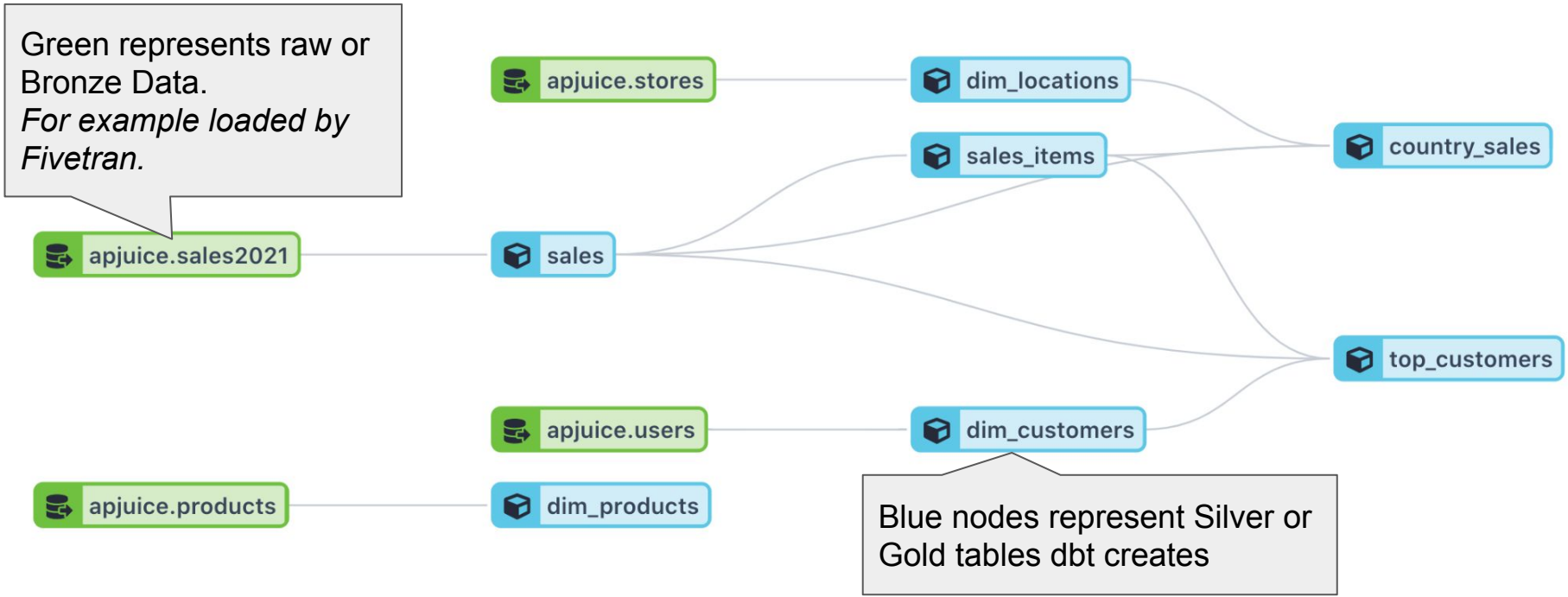
dim_customers.sql

```sql
select
    store_id || "-" || cast(id as string) as unique_id,
    id,
    store_id,
    name,
    email
from {{ source('apjuice', 'users') }}
```

top_customers.sql

```sql
select
  s.store_id,
  ss.unique_customer_id,
  c.name,
  sum(product_cost) total_spend
from
  {{ ref('sales_items') }} s
  join {{ ref('sales') }} ss        on s.sale_id = ss.id
  join {{ ref('dim_customers') }} c on ss.unique_customer_id = c.unique_id
where
  ss.unique_customer_id is not null
group by s.store_id, ss.unique_customer_id, c.name
order by total_spend
limit 100
```

Our final data lineage:



Optional: Starter SQL code for exercises

Link here:
https://gist.github.com/adinsmoor/60f33ebd8537c06fdd7edb1880e38f8e

Exercise 1: creating our Silver model

The Ask: “we need a single source of truth for customer details. Can you create a dim_customers model?”

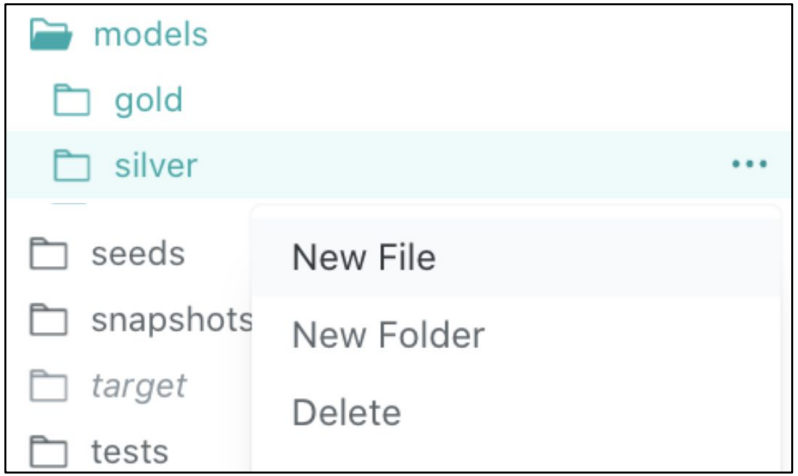
- Include: id, store_id, name, email
- You’ll need to add a new unique_key field that concatenates store_id and users id, since user id may not be unique across stores
- Define a source
- Execute *dbt run* once complete

Cheat code: use dbt Cloud shortcuts! Type “__s”, select *source*, and tab.

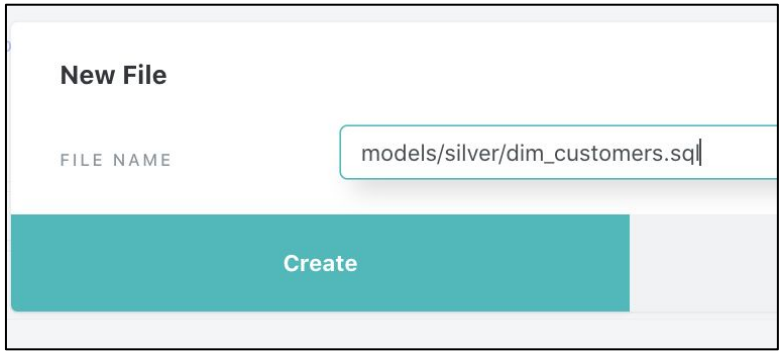
Bonus: optionally, try adding a custom configuration at the top of the model file to define this model as a *view* (instead of the *table* default we have set for the project).

Exercise 1: Solution

*1. Select **New File** from the dropdown*



*2. Give the file a **name** with a **.sql extension***



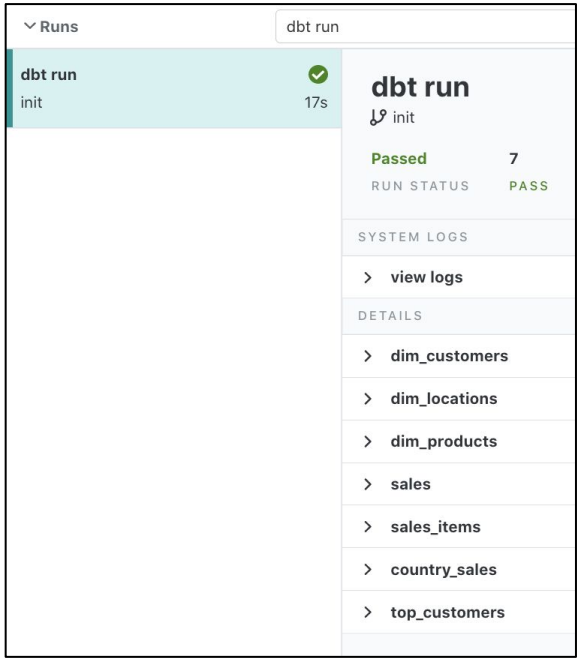
Exercise 1: Solution (cont.)

*3. Add your **SQL and Save***

dim_customers.sql

```
select
    store_id || "-" || cast(id as string) as unique_id,
    id,
    store_id,
    name,
    email
from {{ source('apjuice', 'users') }}
```

*4. Execute the model with **dbt run***



Exercise 2: Testing & Documenting Our Models

The Ask: "We want to share context about the data with users and verify that the source data conforms to our expectations over time. Can you set that up?"

1. ***Open silver_docs.yml*** *and add* ***docs & a not_null*** *for your dim_customers model, follow the example for locations*

2. *Execute the test with* ***dbt test***

silver_docs.yml

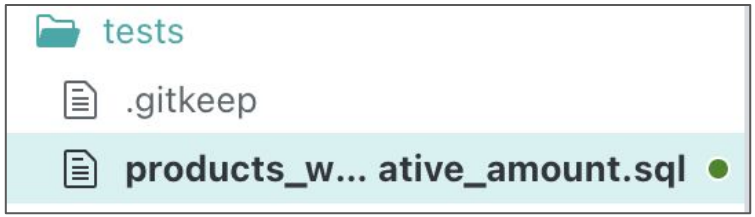
```yaml
version: 2

models:
  - name: dim_locations
    description: store locations further classified by country
    columns:
      - name: id
        tests:
          - not_null
```

Bonus: optionally, try adding a custom singular test (in the *tests* folder) that checks for any records in sales_items with a *product_cost* < 0.

Is there a way to set the test up to error if > 10 records have negative amounts, but only warn otherwise?

Exercise 2: Bonus Solution

3. Add your ***product_with_negative_amount.sql*** to the tests folder



4. Define your SQL-based test. Save. Execute your test with ***dbt test –select sales_items***

products_with_negative_amount.sq

```
1
2   {{
3       config(
4           error_if = '>10'
5           , warn_if = '>0'
6       )
7   }}
8
9   select *
10  from {{ ref('sales_items') }}
11  where product_cost < 0
12
```

Exercise 3: creating our Gold model

The Ask: “the marketing team runs special promotions for the top 100 customers. Can you add this to our Gold layer?”

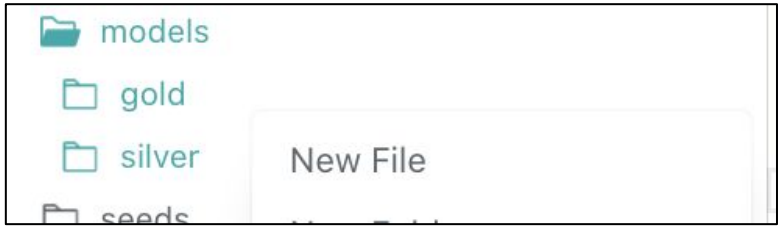
- Include: store_id, unique_customer_id, name, total spend
- Hint: start with *sales_items* and join in *sales* and *dim_customers*
- Remember to use the *ref* command!
- Execute *dbt run* once complete

Advanced Bonus: optionally, create a dbt *snapshot* (in the snapshots directory) on dim_customers to preserve a history of how customers have changed over time. Config you’ll need: *unique_key = ‘unique_id*, *strategy='check'*, *check_cols='all'*

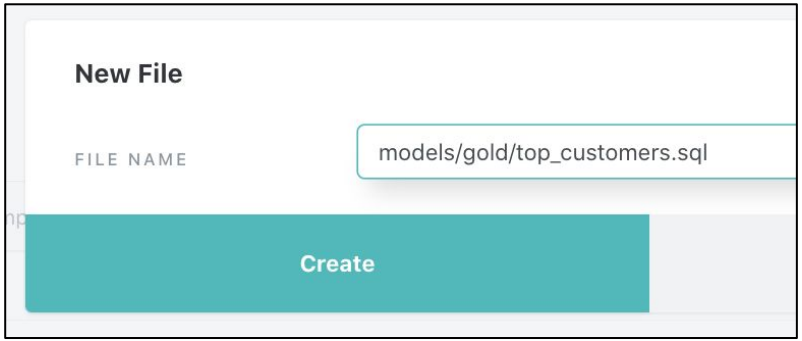
Cheat code: use dbt Cloud shortcuts! Type “__r”, select *ref*, and tab. For the bonus: type “__s”, select *snapshot*, and tab.

Exercise 3: Solution

*1. Select **New File** from the dropdown*



*2. Give the file a **name** with a **.sql extension***



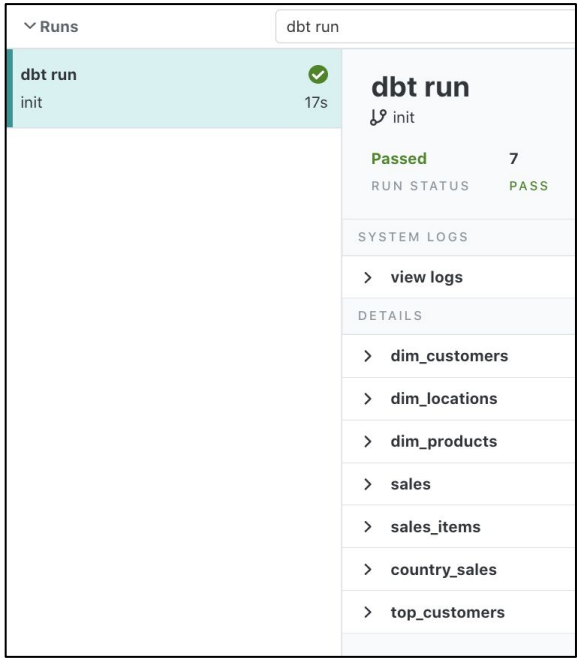
Exercise 3: Solution (cont.)

*3. Add your **SQL and Save***

top_customers.sql

```sql
select
  s.store_id,
  ss.unique_customer_id,
  c.name,
  sum(product_cost) total_spend
from
  {{ ref('sales_items') }} s
  join {{ ref('sales') }} ss        on s.sale_id = ss.id
  join {{ ref('dim_customers') }} c on ss.unique_customer_id = c.unique_id
where
  ss.unique_customer_id is not null
group by s.store_id, ss.unique_customer_id, c.name
order by total_spend
limit 100
```

*4. Execute the model with **dbt run***



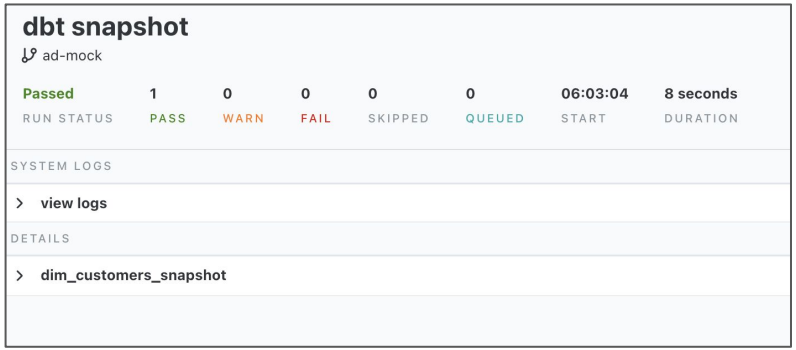
Exercise 3: Bonus Solution

5. Add a *dim_customer_snapshot.sql* file with the appropriate logic to the *snapshots* directory.

dim_customer_snapshot.sql

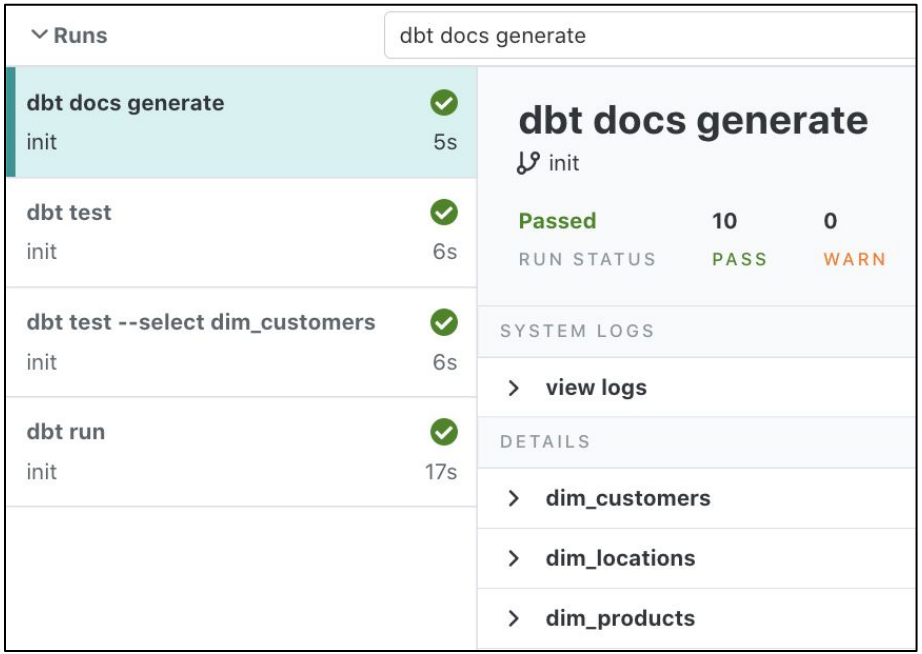
```
1
2   {% snapshot dim_customers_snapshot %}
3       {{
4           config(
5               unique_key='unique_id',
6               strategy='check',
7               check_cols='all'
8           )
9       }}
10
11      select * from {{ ref('dim_customers') }}
12
13  {% endsnapshot %}
14
```

*6. Execute the snapshot with **dbt snapshot***

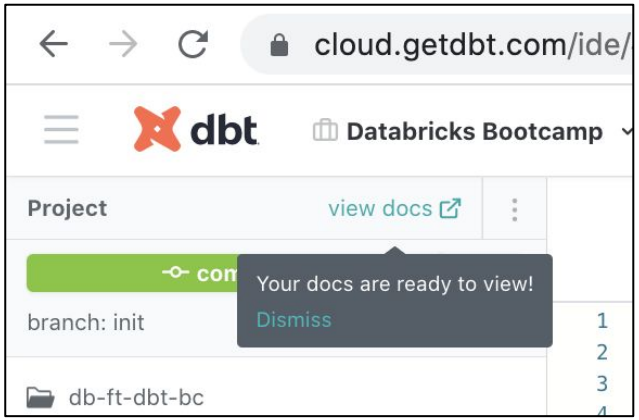


Wrap up: building our docs

1. *Run dbt Docs*



2. Click on the View Docs Link to view dbt Docs



Solution code (including bonuses!)

Link here:
https://gist.github.com/adinsmoor/93cd64264307b005bbe896b1f7804519

Wrapping Up

- What have we built?
- How do we deploy to Production?
- Can we test our changes prior to deployment (CI Testing)
- How do we share documentation with stakeholders?
- Open Q&A