FROM FABRIC TO FANTASTIC

HOW DBT MAKES LAKEHOUSES AND WAREHOUSES SHINE

Sam Debruyn

Fabric Global Online Conference
September 2024

Who am I?

Sam Debruyn

📍 Heist-op-den-Berg, BE

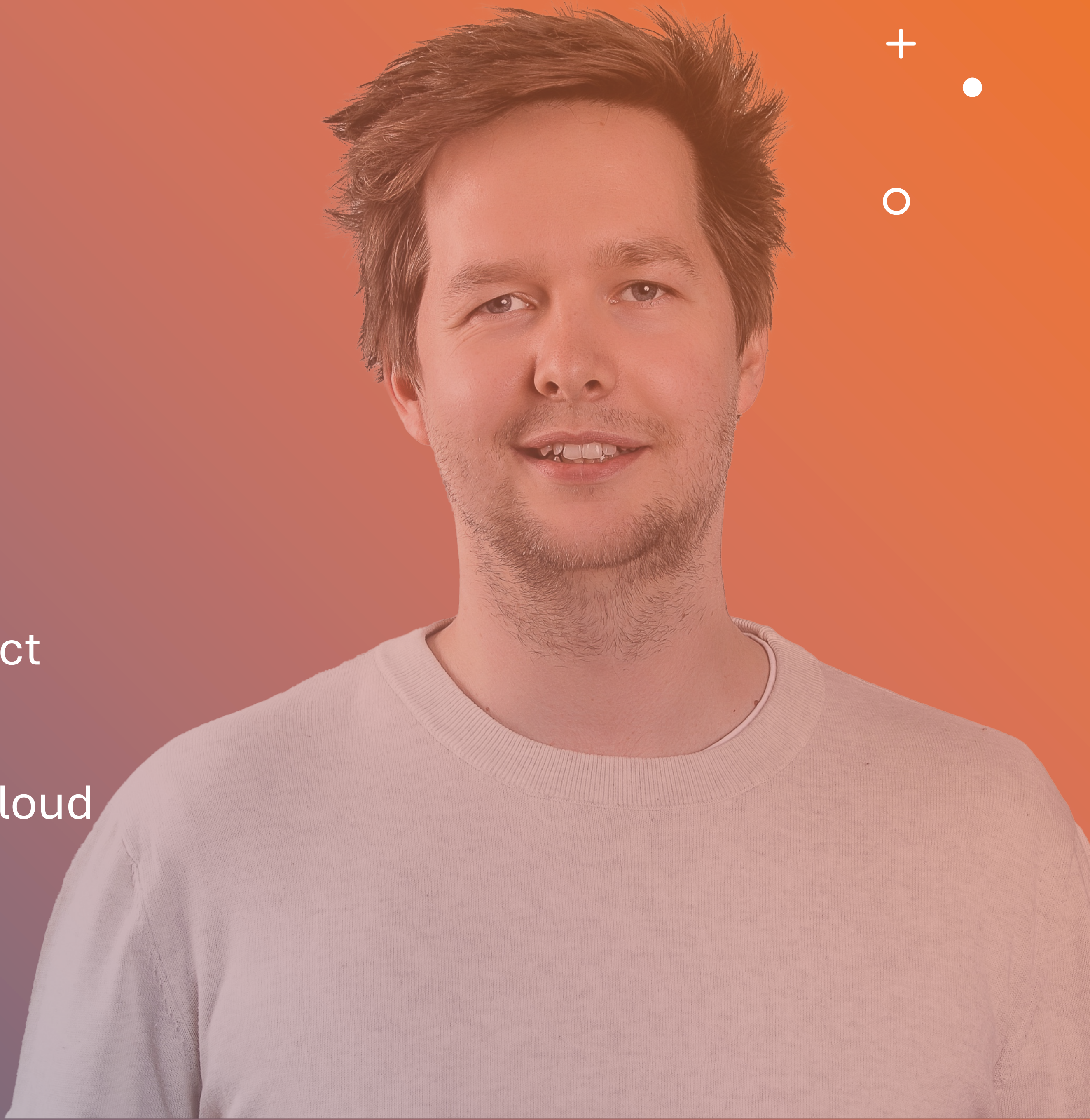
💼 Consultant / Data & Cloud Architect

5️⃣ years in data

🔟 years in software / architecture / cloud

🫶 dbt, Microsoft, modern data stack

MVP Microsoft® Most Valuable Professional

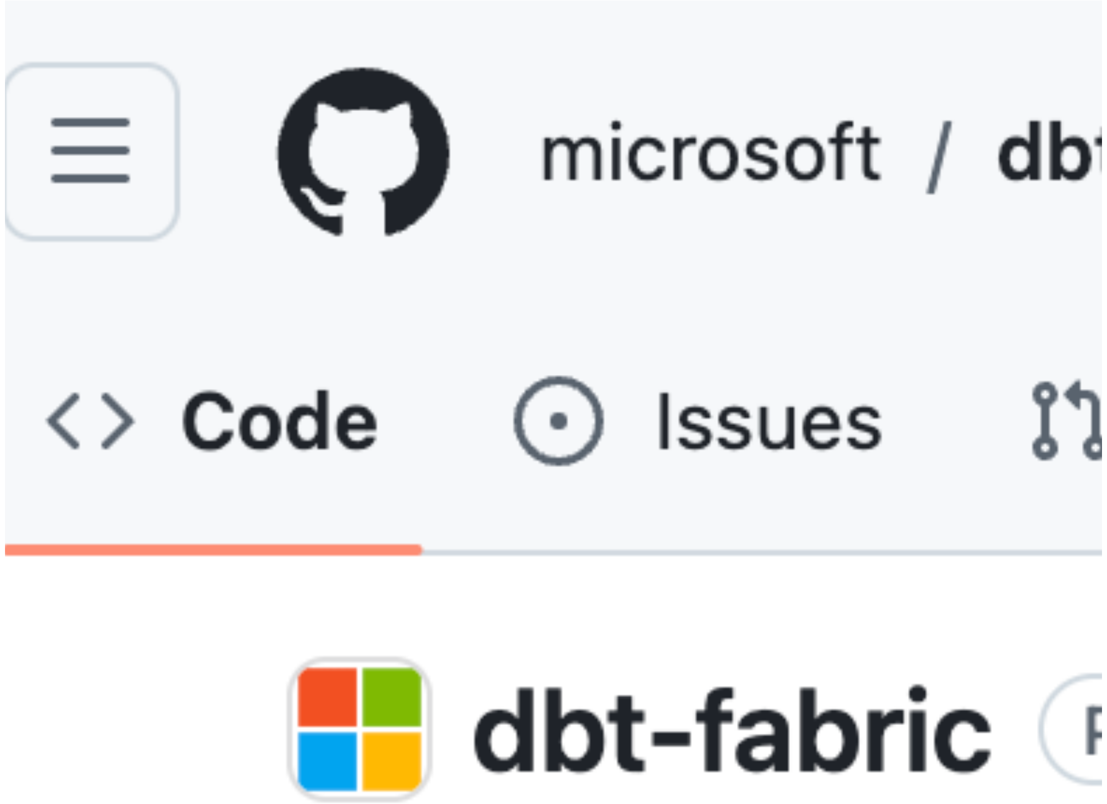


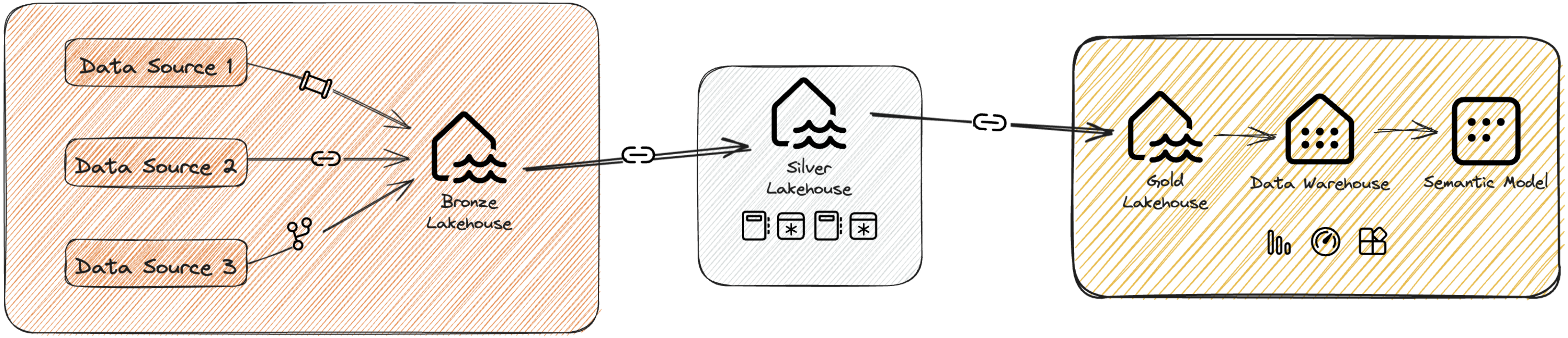
dbt-fabric: a quick lookback

OSS project (dbt adapter) created in 2019 to bring dbt support to SQL Server & Azure SQL

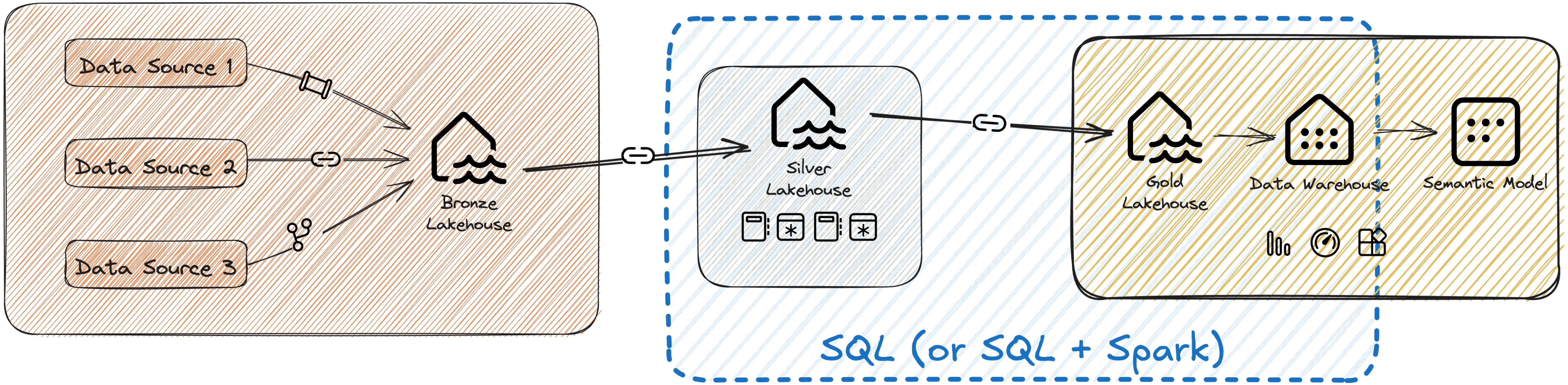
Led maintenance on adapters for SQL Server, Azure SQL, Synapse...

Worked with Microsoft to bring dbt to Fabric





Lakehouse/Warehouse



Where does SQL fit in?

Different ways to transform data

Programming languages

Python and Scala. High learning curves and often creates a boundary between business users and specialized engineers. Very powerful and easy to maintain.

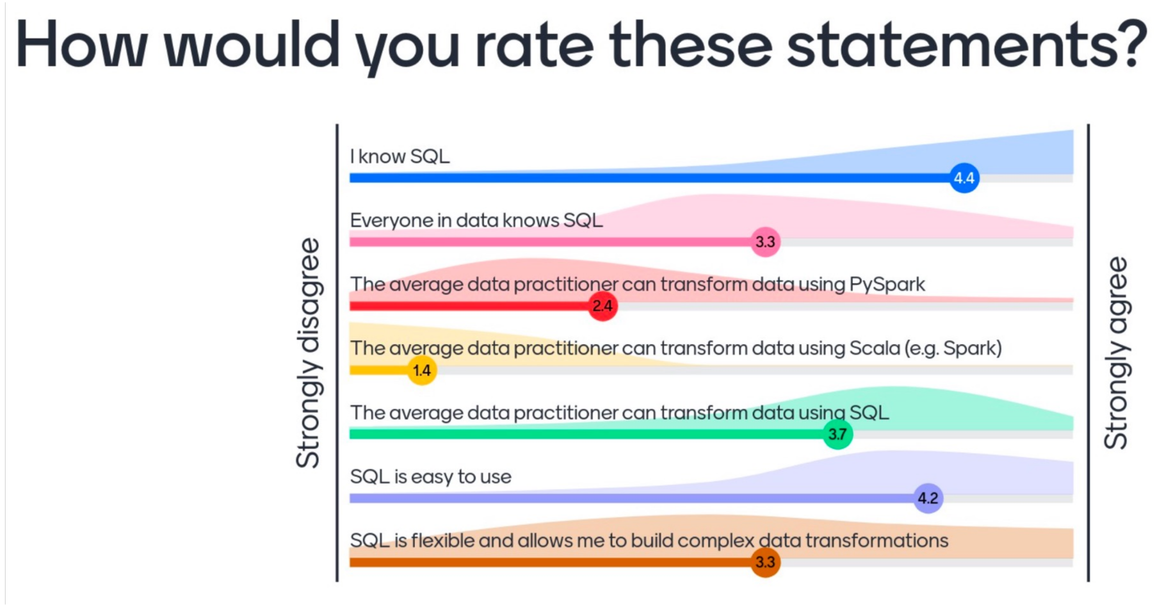
Declarative languages

SQL, SAS, and the likes. Code is easy to write and understand but offers limited flexibility and can be hard to maintain (adopting software eng. best practices).

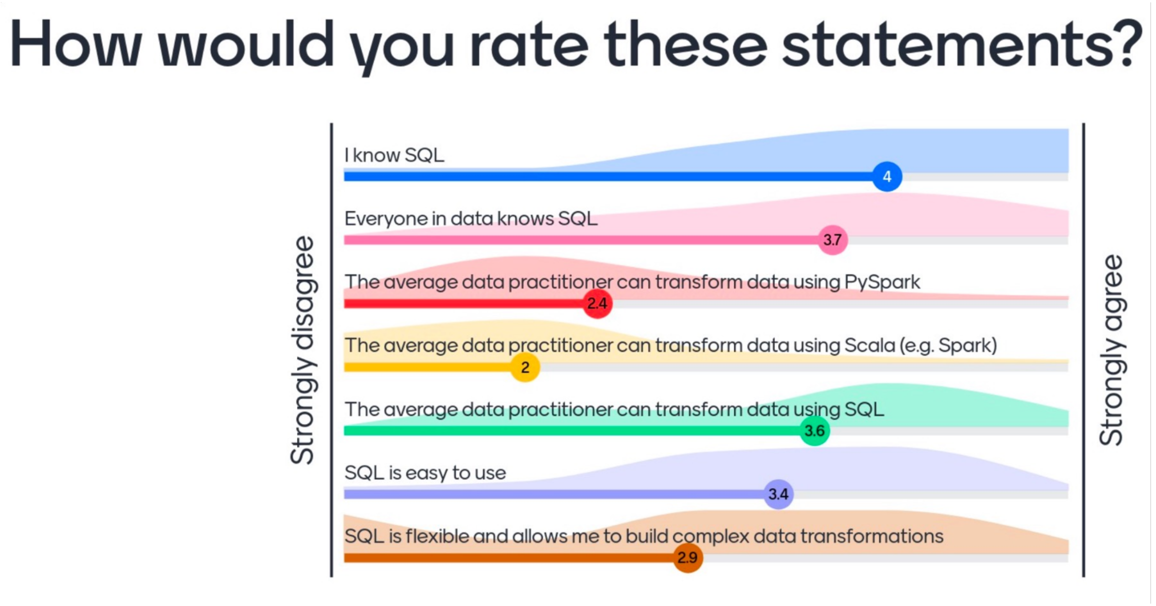
Low-code / UI-based

Easy to adopt, use, and achieve results. Very high vendor lock-in and limited flexibility and modularity.

A quick survey done at local meetups

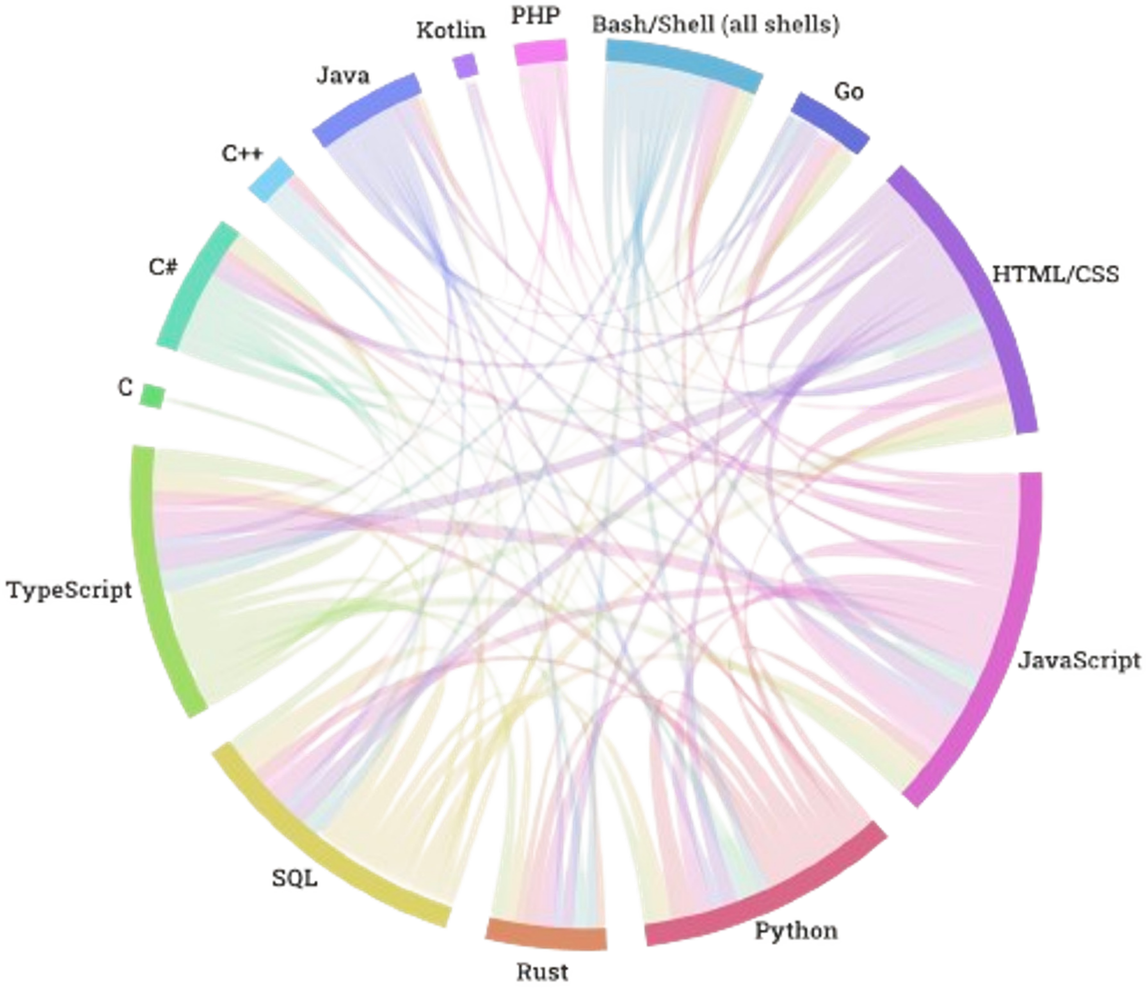


How would you rate these statements?



How would you rate these statements?

Programming languages 2023



source: Stack Overflow
worked with / wants to
work with

The common language of data transformations is **not** drag-and-drop

Data architects

ERWIN
Wherescape

Data engineers

Informatica
Matillion

Analytics engineers

Alteryx
Talend

BI developers

Tableau
Qlik

Analysts

Excel / Sheets
Power BI

The common language of data transformations is SQL

Data architects

ERWIN
Wherescape
SQL

Data engineers

Informatica
Matillion
SQL

Analytics engineers

Alteryx
Talend
SQL

BI developers

Tableau
Qlik
SQL

Analysts

Excel / Sheets
Power BI
SQL

Introducing dbt

Open-source Python utility for building data transformations

Free/OSS version: dbt Core / version with all the bells & whistles included: dbt Cloud

The de facto default tool for analytics engineering

3 things to know

No compute

dbt requires a data warehouse to function, it only sends SQL queries

SQL with Jinja

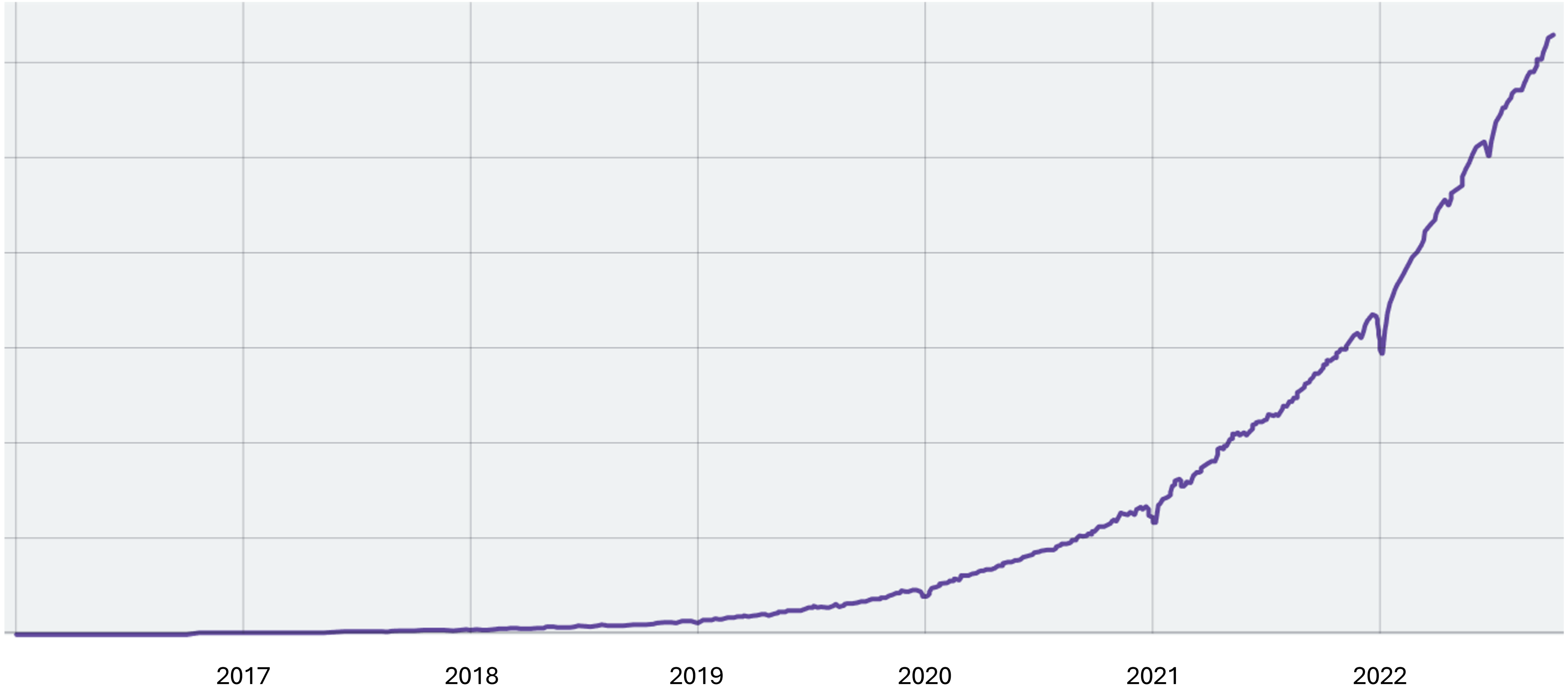
dbt is built for SQL, in some cases you can also use Python

Free/self-hosted or cloud

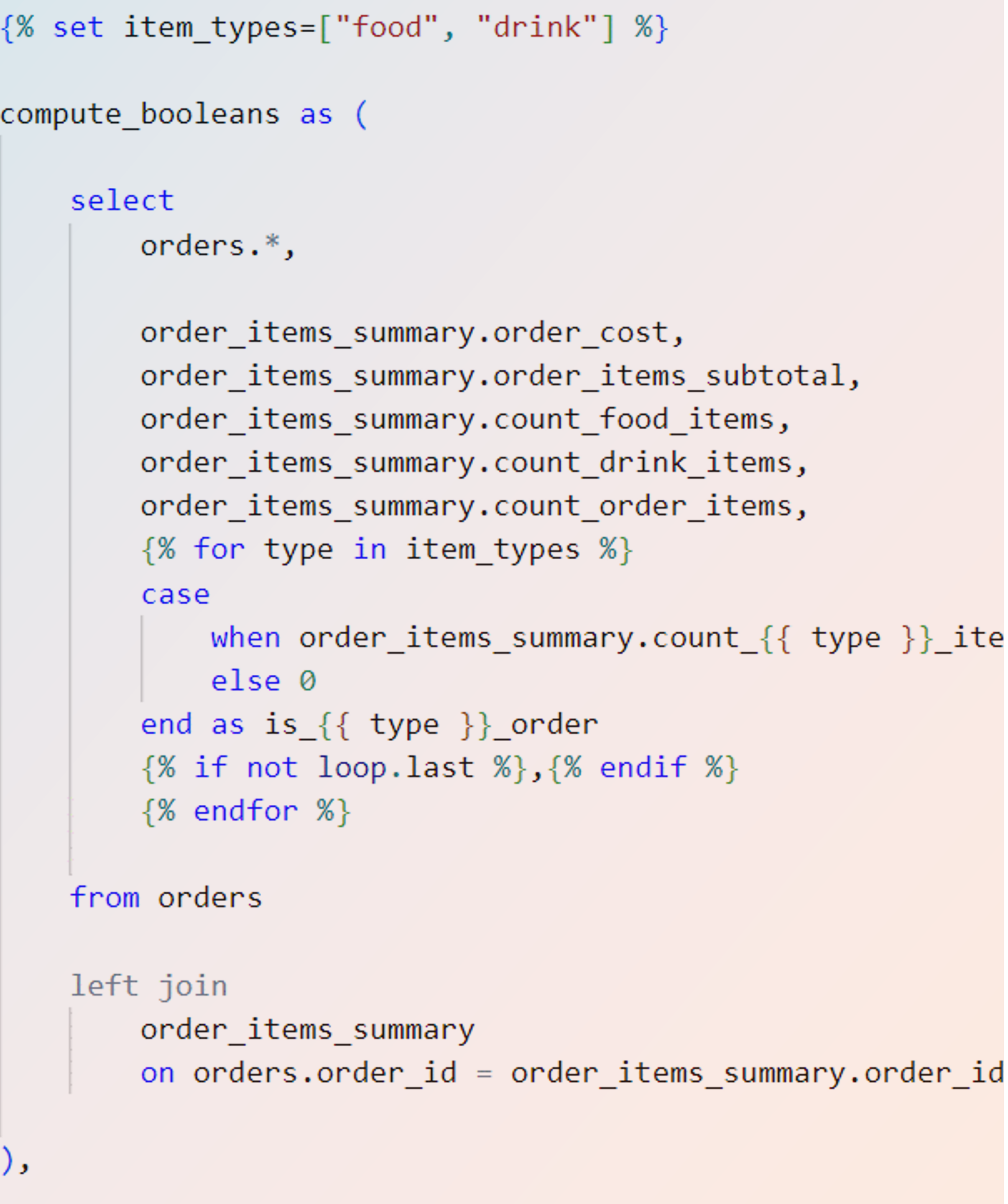
dbt Core is free but requires "plumbing" (e.g. an orchestrator)

dbt Cloud is paid, but will be cheaper than building everything around it manually

dbt adoption past 6 years



October 2023: 30000+ weekly active projects



```
{% set item_types=["food", "drink"] %}

compute_booleans as (

    select
        orders.*,

        order_items_summary.order_cost,
        order_items_summary.order_items_subtotal,
        order_items_summary.count_food_items,
        order_items_summary.count_drink_items,
        order_items_summary.count_order_items,
        {% for type in item_types %}
        case
            when order_items_summary.count_{{ type }}_ite
            else 0
        end as is_{{ type }}_order
        {% if not loop.last %},{% endif %}
        {% endfor %}

    from orders

    left join
        order_items_summary
        on orders.order_id = order_items_summary.order_id

),
```

Modular development

- Write transformations in separate version-controlled files

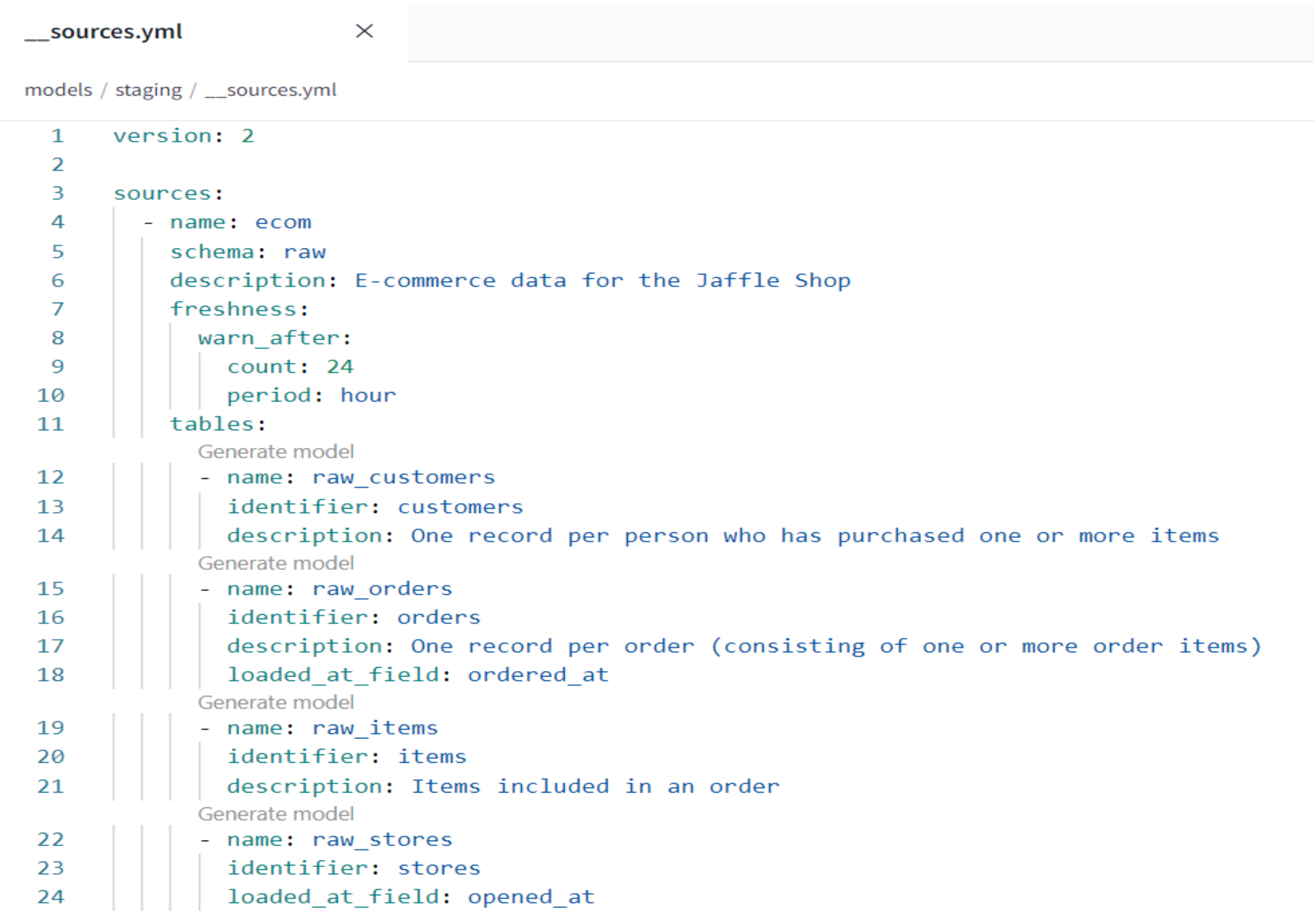
- SQL on steroids with Jinja: control logic, loops

- Customize and parametrize with variables

- Reusable code blocks with macros

- Easy to follow DRY principles

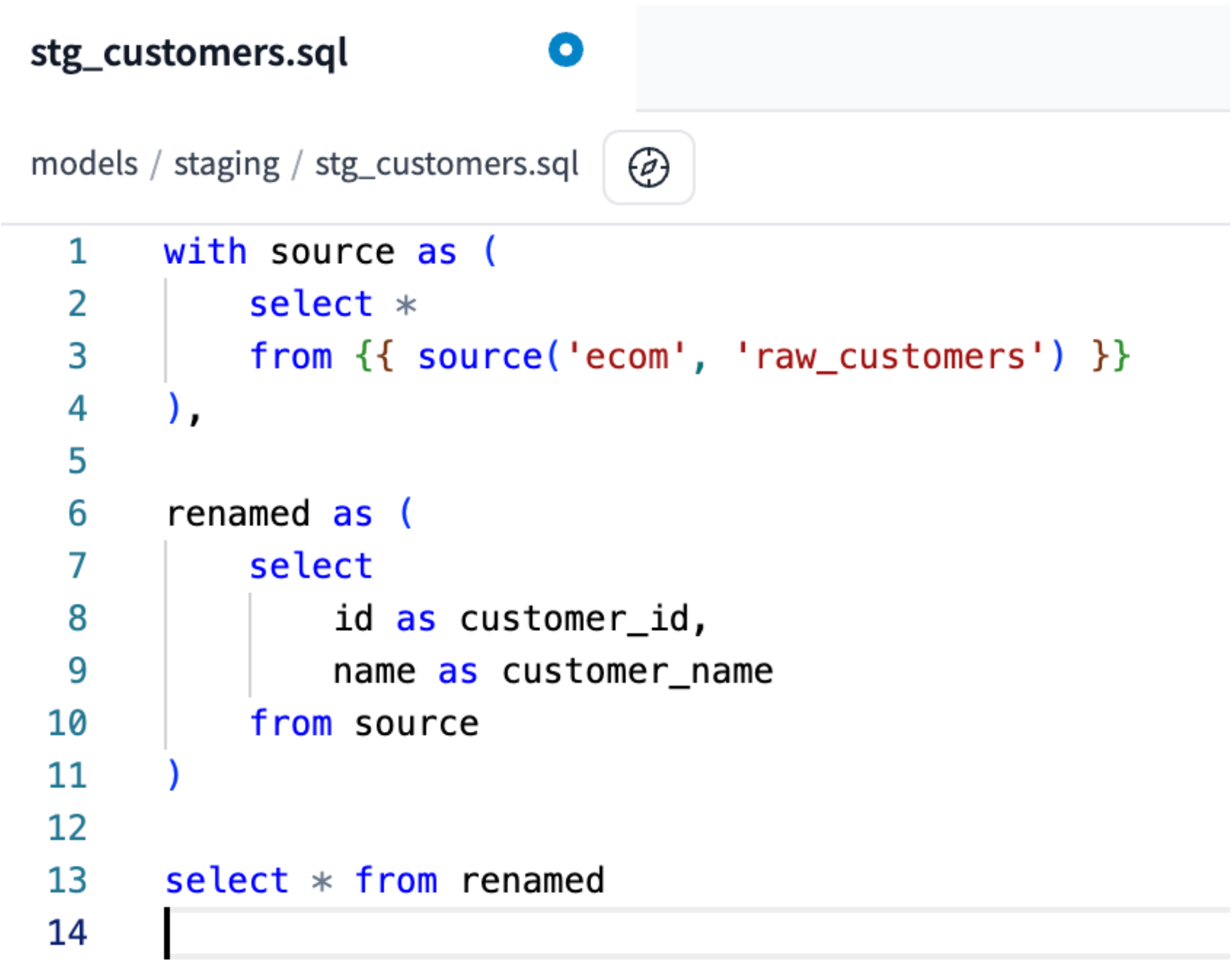
Sources



```yaml
version: 2

sources:
  - name: ecom
    schema: raw
    description: E-commerce data for the Jaffle Shop
    freshness:
      warn_after:
        count: 24
        period: hour
    tables:
      - name: raw_customers
        identifier: customers
        description: One record per person who has purchased one or more items
      - name: raw_orders
        identifier: orders
        description: One record per order (consisting of one or more order items)
        loaded_at_field: ordered_at
      - name: raw_items
        identifier: items
        description: Items included in an order
      - name: raw_stores
        identifier: stores
        loaded_at_field: opened_at
```

Manage data sources and monitor data freshness

Sources

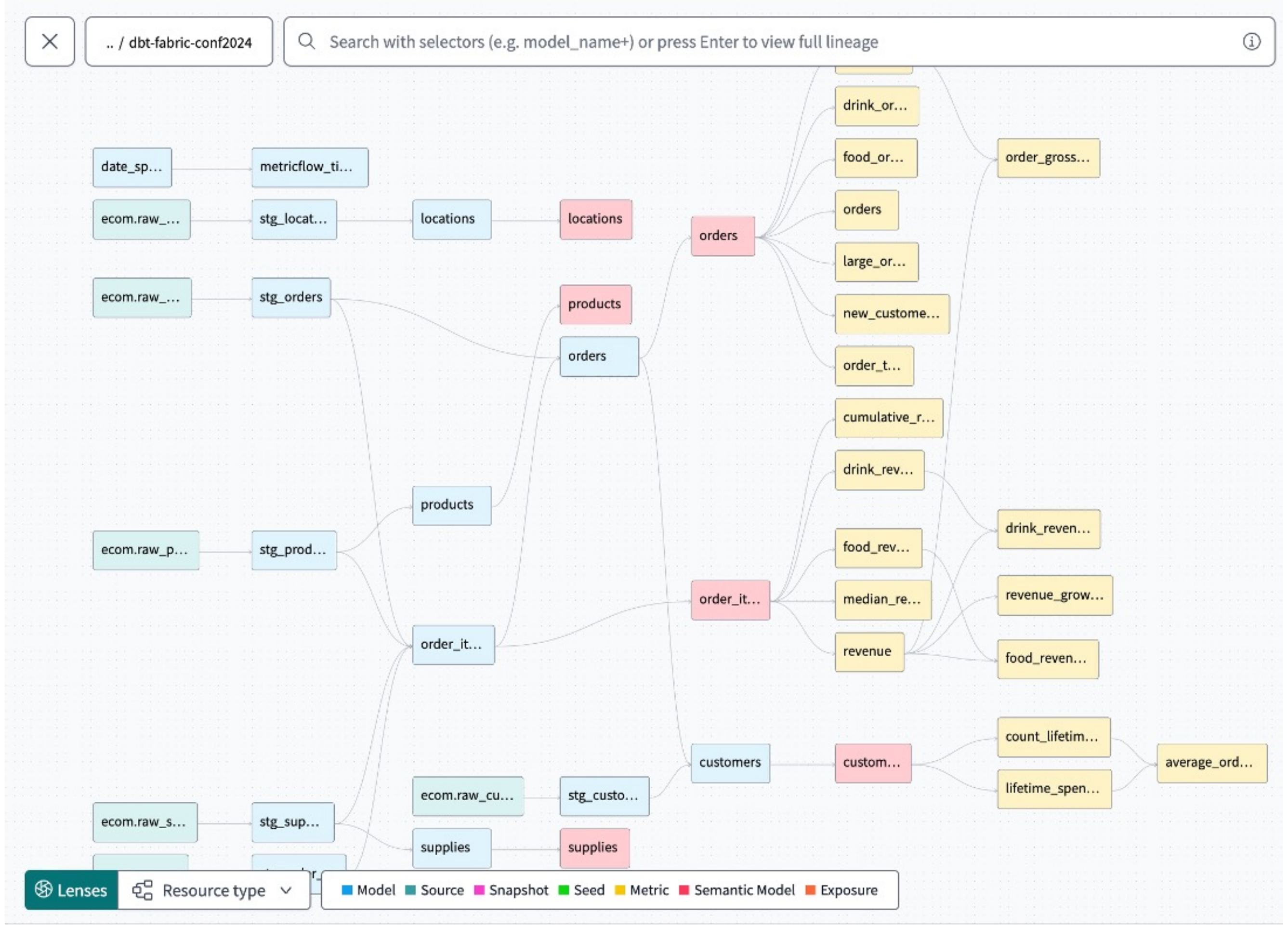


```
with source as (
    select *
    from {{ source('ecom', 'raw_customers') }}
),

renamed as (
    select
        id as customer_id,
        name as customer_name
    from source
)

select * from renamed

```

Dynamic schema selection

Start tracking lineage from the source

Data lineage

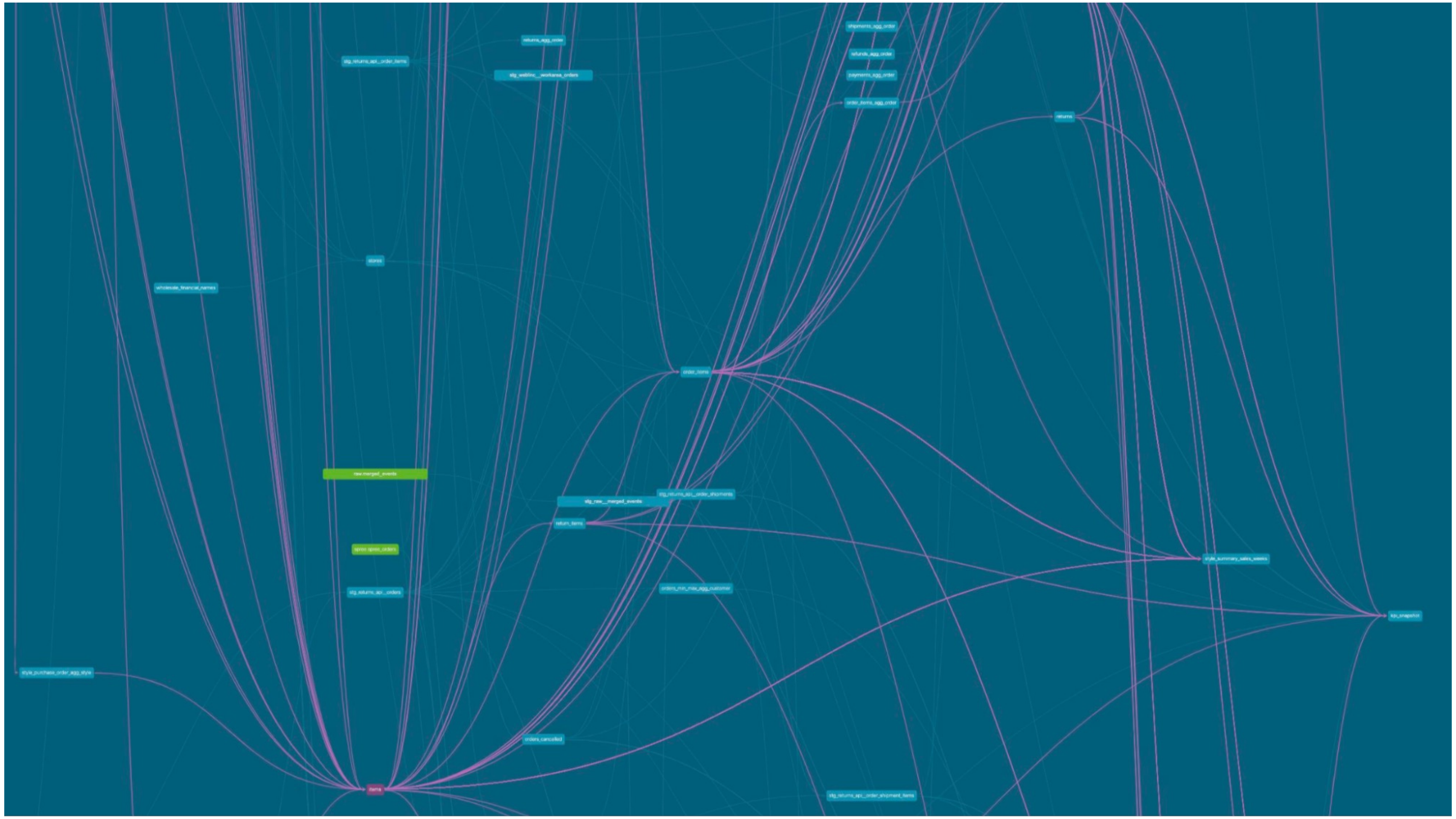


Understand the flow of data

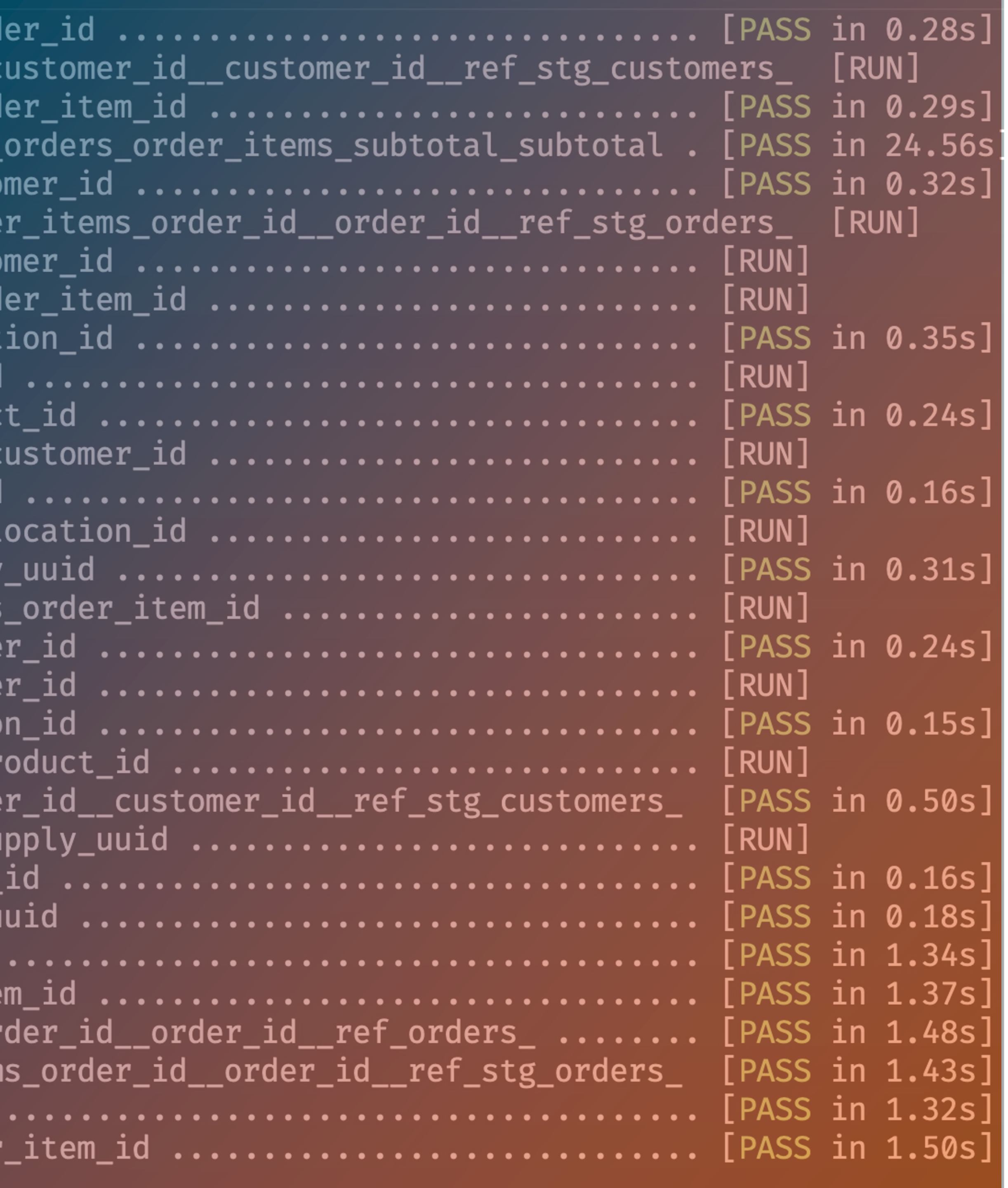
Impact of modifying a transformation

How a dimension/fact is constructed

Data lineage



Spot and detect bad data model design



```
ler_id ................................ [PASS in 0.28s]
ustomer_id__customer_id__ref_stg_customers_  [RUN]
ler_item_id ........................... [PASS in 0.29s]
orders_order_items_subtotal_subtotal  . [PASS in 24.56s
omer_id ............................... [PASS in 0.32s]
er_items_order_id__order_id__ref_stg_orders_ [RUN]
omer_id ............................... [RUN]
ler_item_id ........................... [RUN]
ion_id ................................ [PASS in 0.35s]
l ..................................... [RUN]
t_id .................................. [PASS in 0.24s]
ustomer_id ............................ [RUN]
l ..................................... [PASS in 0.16s]
ocation_id ............................ [RUN]
_uuid ................................. [PASS in 0.31s]
_order_item_id ........................ [RUN]
er_id ................................. [PASS in 0.24s]
er_id ................................. [RUN]
on_id ................................. [PASS in 0.15s]
roduct_id ............................. [RUN]
er_id__customer_id__ref_stg_customers_  [PASS in 0.50s]
pply_uuid ............................. [RUN]
id .................................... [PASS in 0.16s]
uid ................................... [PASS in 0.18s]
....................................... [PASS in 1.34s]
m_id .................................. [PASS in 1.37s]
der_id__order_id__ref_orders_ ........  [PASS in 1.48s]
s_order_id__order_id__ref_stg_orders_   [PASS in 1.43s]
....................................... [PASS in 1.32s]
_item_id .............................. [PASS in 1.50s]
```

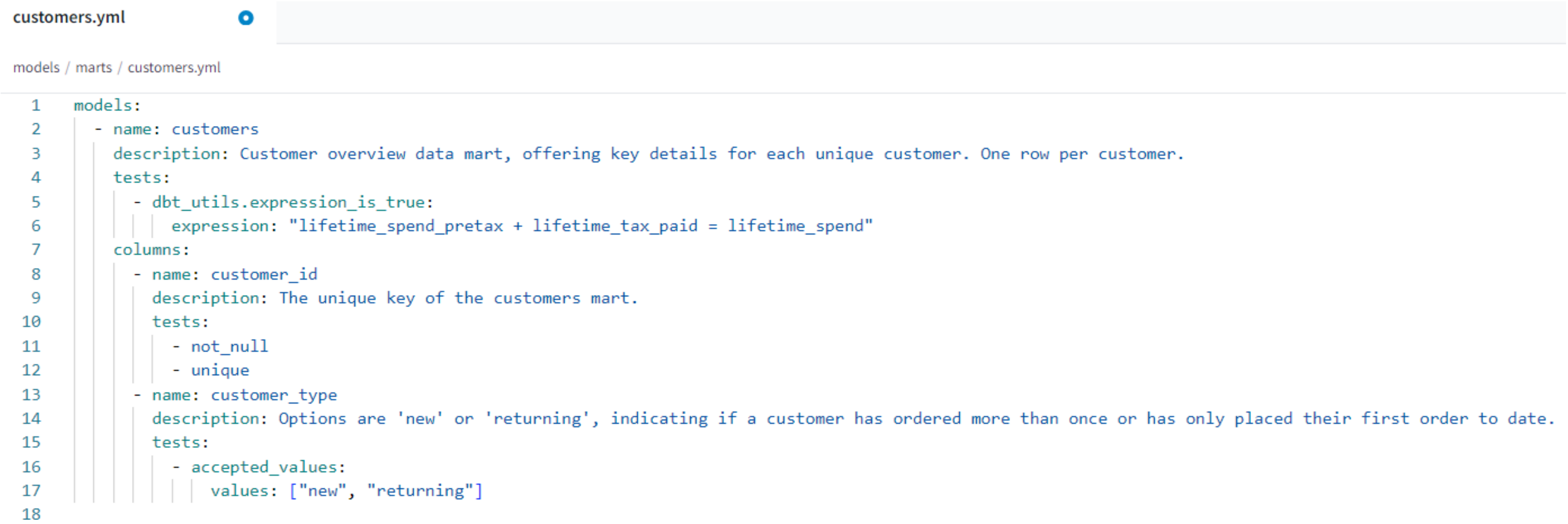
Data tests & unit tests

Automated testing for your code, as well as for your data

Tests can be integrated in other tooling to get a good view on your data quality

Simple YAML- or SQL-based syntax to define tests

Documentation and tests

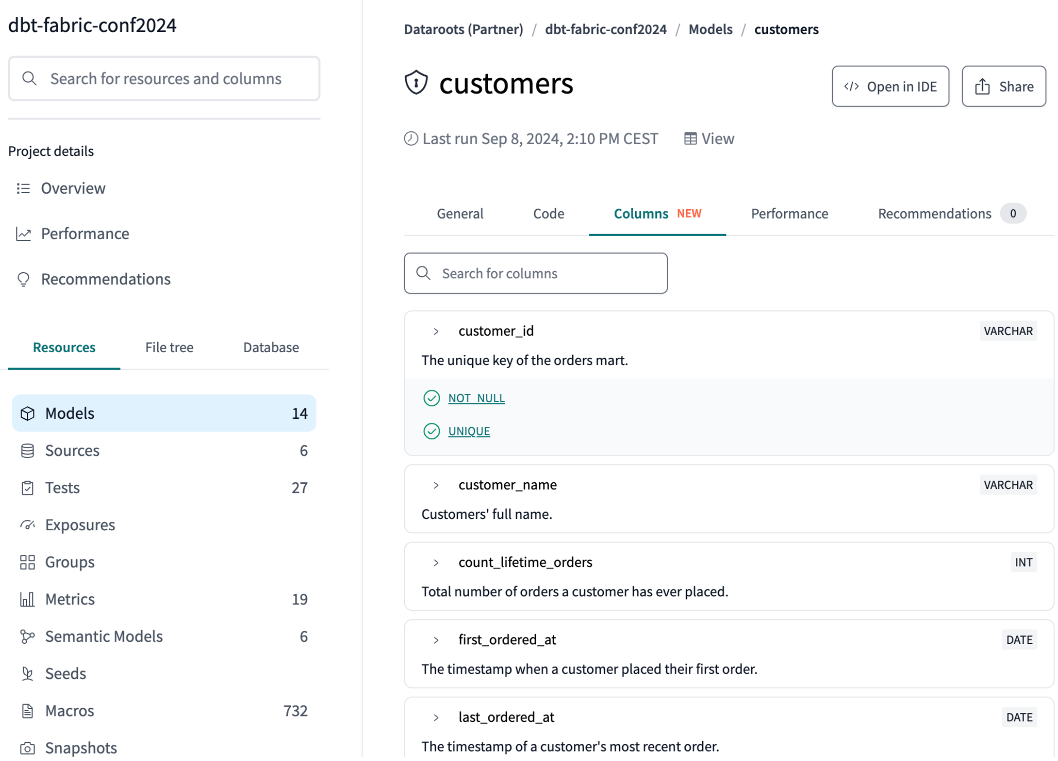


customers.yml

models / marts / customers.yml

```yaml
models:
  - name: customers
    description: Customer overview data mart, offering key details for each unique customer. One row per customer.
    tests:
      - dbt_utils.expression_is_true:
          expression: "lifetime_spend_pretax + lifetime_tax_paid = lifetime_spend"
    columns:
      - name: customer_id
        description: The unique key of the customers mart.
        tests:
          - not_null
          - unique
      - name: customer_type
        description: Options are 'new' or 'returning', indicating if a customer has ordered more than once or has only placed their first order to date.
        tests:
          - accepted_values:
              values: ["new", "returning"]

```

dbt docs



dbt-fabric-conf2024

Search for resources and columns

Project details

- Overview
- Performance
- Recommendations

Models	14
Sources	6
Tests	27
Exposures	
Groups	
Metrics	19
Semantic Models	6
Seeds	
Macros	732
Snapshots	

Dataroots (Partner) / dbt-fabric-conf2024 / Models / customers

customers

Last run Sep 8, 2024, 2:10 PM CEST   View

Search for columns

customer_id — VARCHAR
The unique key of the orders mart.
✓ NOT_NULL
✓ UNIQUE

customer_name — VARCHAR
Customers' full name.

count_lifetime_orders — INT
Total number of orders a customer has ever placed.

first_ordered_at — DATE
The timestamp when a customer placed their first order.

last_ordered_at — DATE
The timestamp of a customer's most recent order.

Clear convention-based data documentation

Good step-up to a data catalog

dbt packages: don’t reinvent the wheel

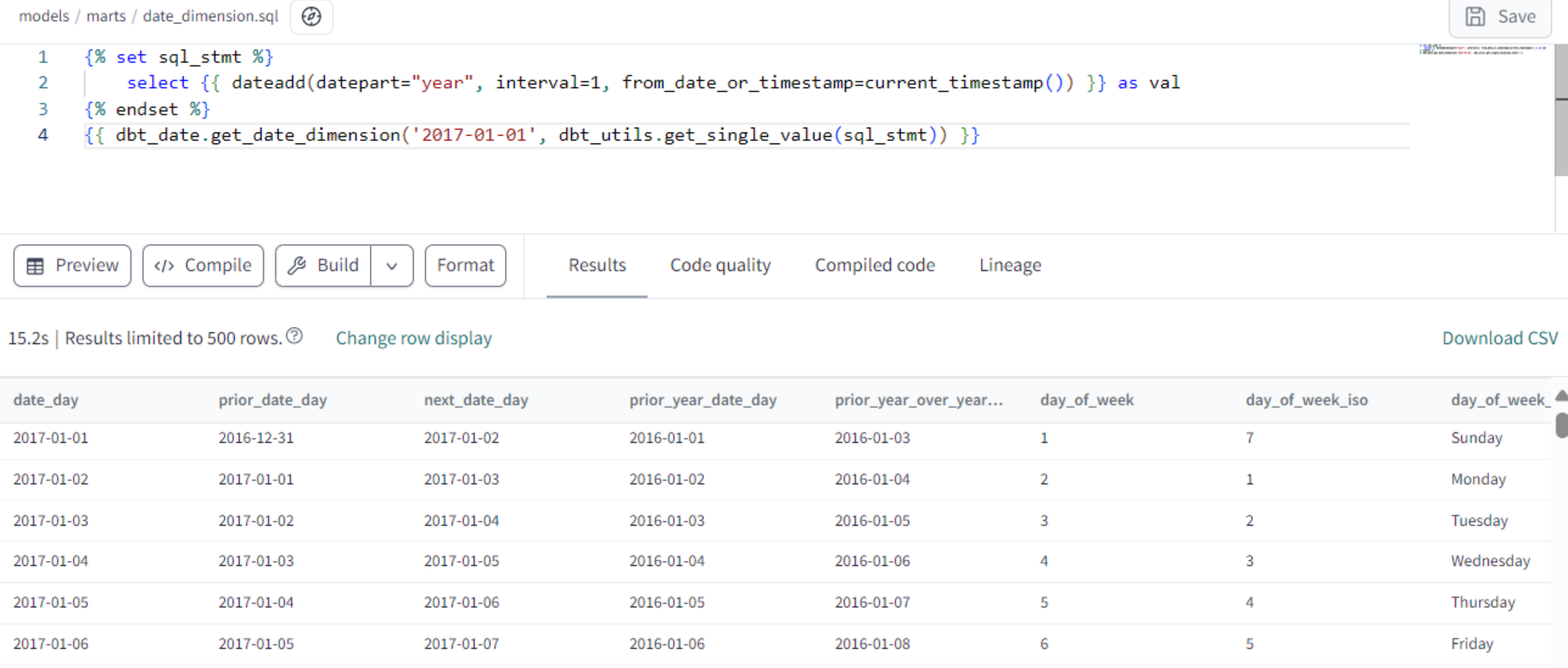
Similar to libraries in software development

Benefit from global knowledge by using pre-built common data transformations and data modelling techniques

Share publicly or privately within your organization

Can contain models (transformations), macros, tests, …

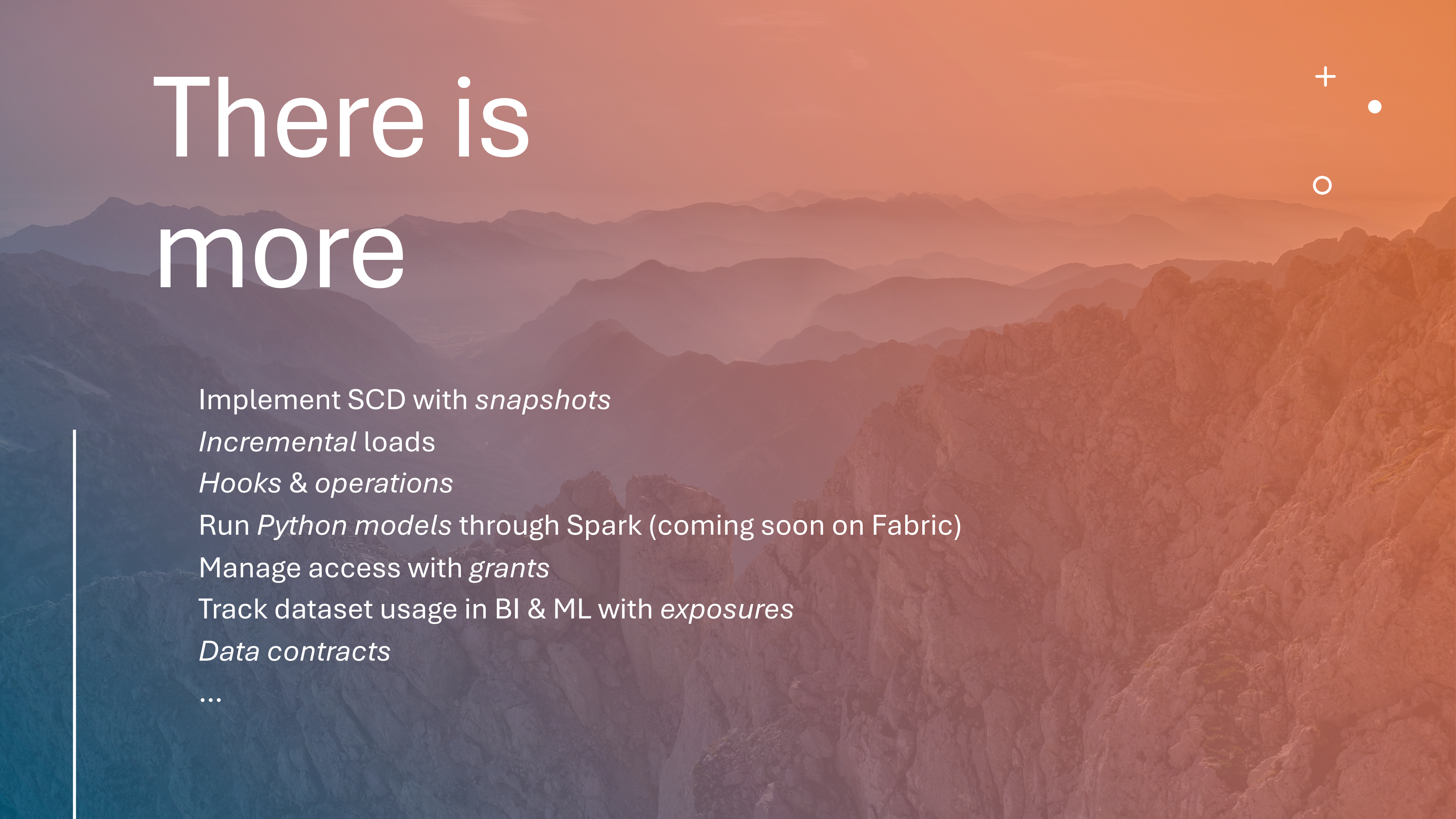
Date dimension in 1 line



```
{% set sql_stmt %}
    select {{ dateadd(datepart="year", interval=1, from_date_or_timestamp=current_timestamp()) }} as val
{% endset %}
{{ dbt_date.get_date_dimension('2017-01-01', dbt_utils.get_single_value(sql_stmt)) }}
```

15.2s | Results limited to 500 rows.

date_day	prior_date_day	next_date_day	prior_year_date_day	prior_year_over_year...	day_of_week	day_of_week_iso	day_of_week_
2017-01-01	2016-12-31	2017-01-02	2016-01-01	2016-01-03	1	7	Sunday
2017-01-02	2017-01-01	2017-01-03	2016-01-02	2016-01-04	2	1	Monday
2017-01-03	2017-01-02	2017-01-04	2016-01-03	2016-01-05	3	2	Tuesday
2017-01-04	2017-01-03	2017-01-05	2016-01-04	2016-01-06	4	3	Wednesday
2017-01-05	2017-01-04	2017-01-06	2016-01-05	2016-01-07	5	4	Thursday
2017-01-06	2017-01-05	2017-01-07	2016-01-06	2016-01-08	6	5	Friday



There is more

Implement SCD with *snapshots*

Incremental loads

Hooks & operations

Run *Python models* through Spark (coming soon on Fabric)

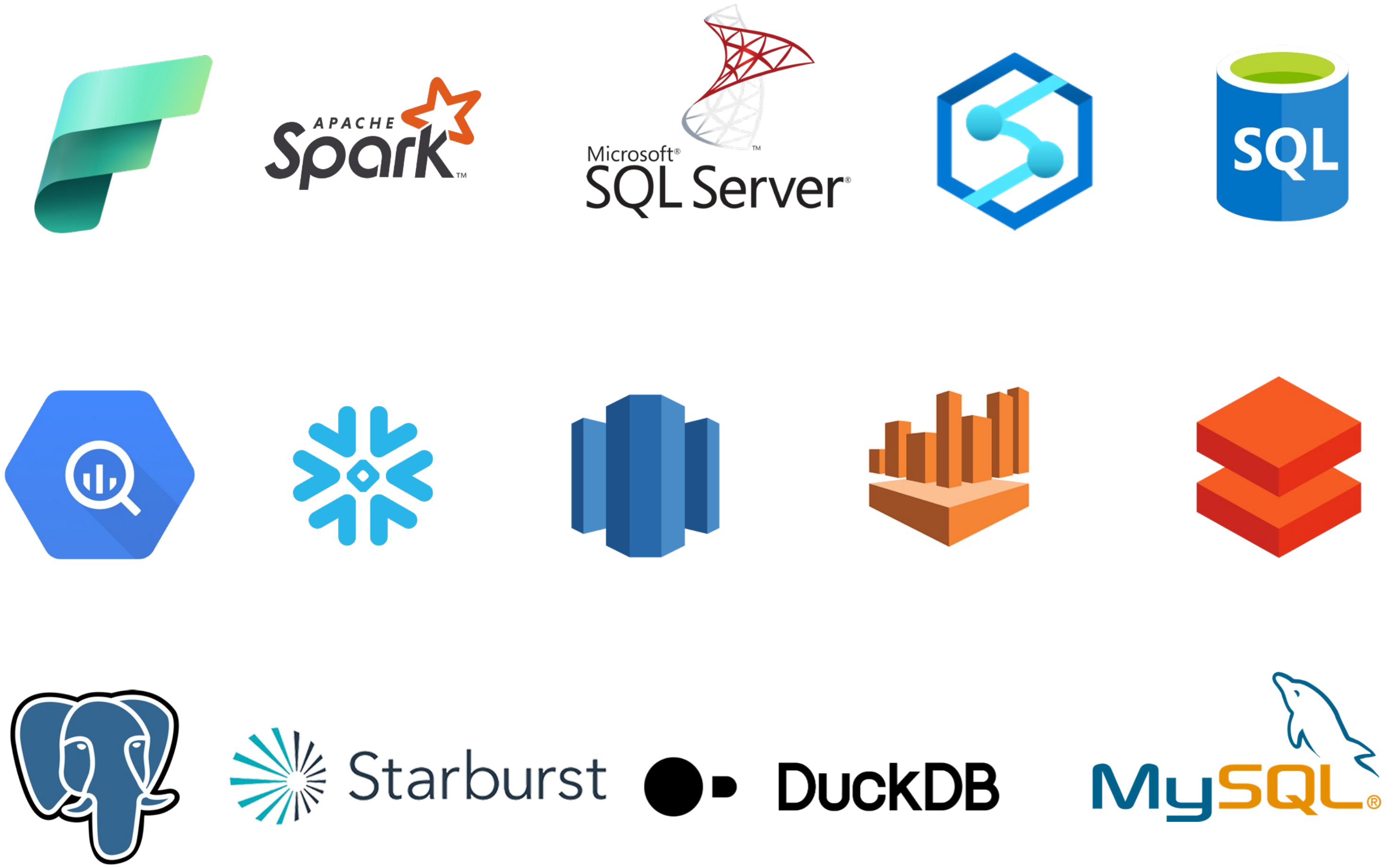
Manage access with *grants*

Track dataset usage in BI & ML with *exposures*

Data contracts

…

Compatibility



Accomplish great things

Version controlled and reproducible

↗️ Collaboration within the team & other teams

Built-in docs & lineage

↗️ Know and understand your data

Test code & data

↗️ Deploy & run with confidence

Modular & easy to use

↗️ Easy to extend and maintain

Your next steps

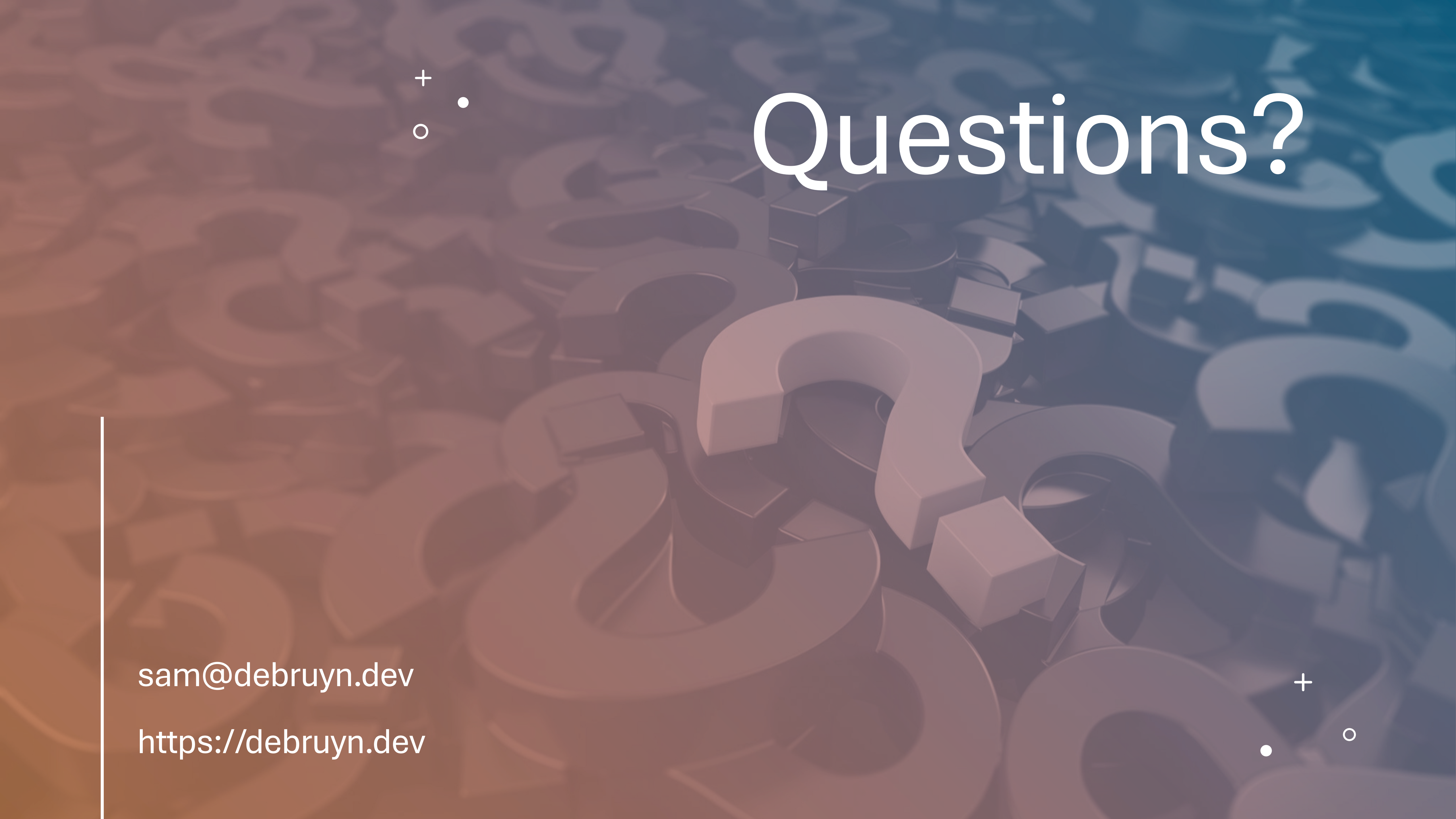
dbt Community: over 100K members

Active and helpful Slack channels

A lot of development in the open-source space

Local meetups all over the world

learn.getdbt.com: free courses to get started with dbt



Questions?

sam@debruyn.dev

https://debruyn.dev