Premium sponsors

dataon
Power your data

twoday

exmon

Fellowmind

Standard sponsors

devoteam
Data Driven

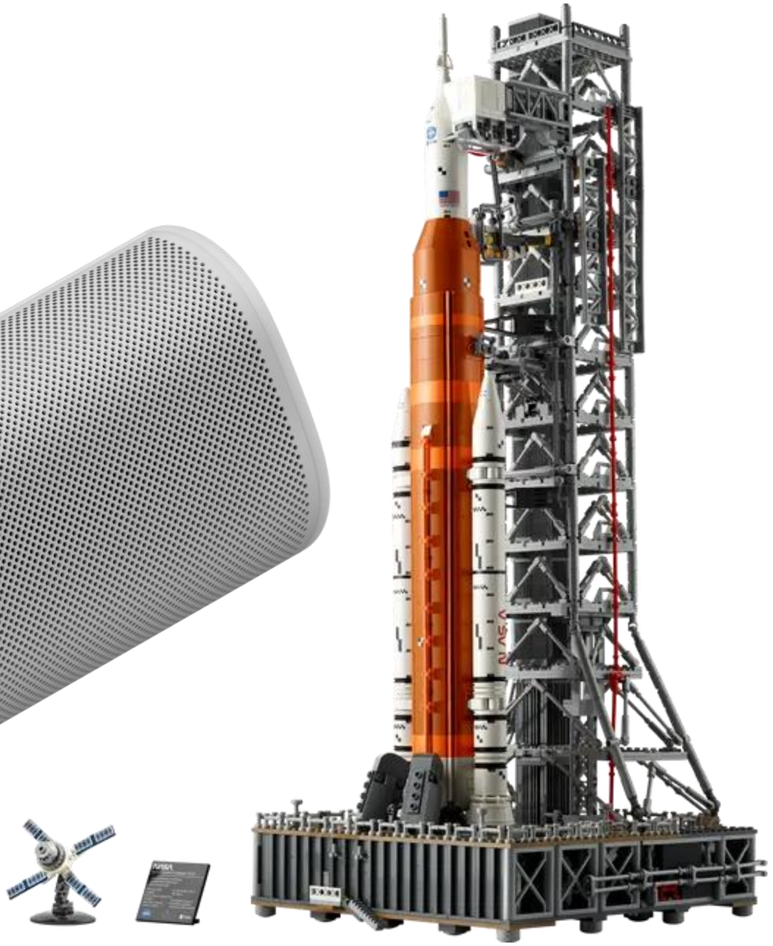
INVIXO

INSPARI
a valantic company

backstage

BIWISE
BASED ON INTELLIGENCE

Raffle Prizes



Enhancing your Fabric Warehouse with dbt

Sean Douglas Thomsen

Business Intelligence Solution Architect @ LINAK

sdt@linak.com

Agenda

Introduction to dbt

Tests

Contracts

Documentation

Deployment

Introduction to dbt

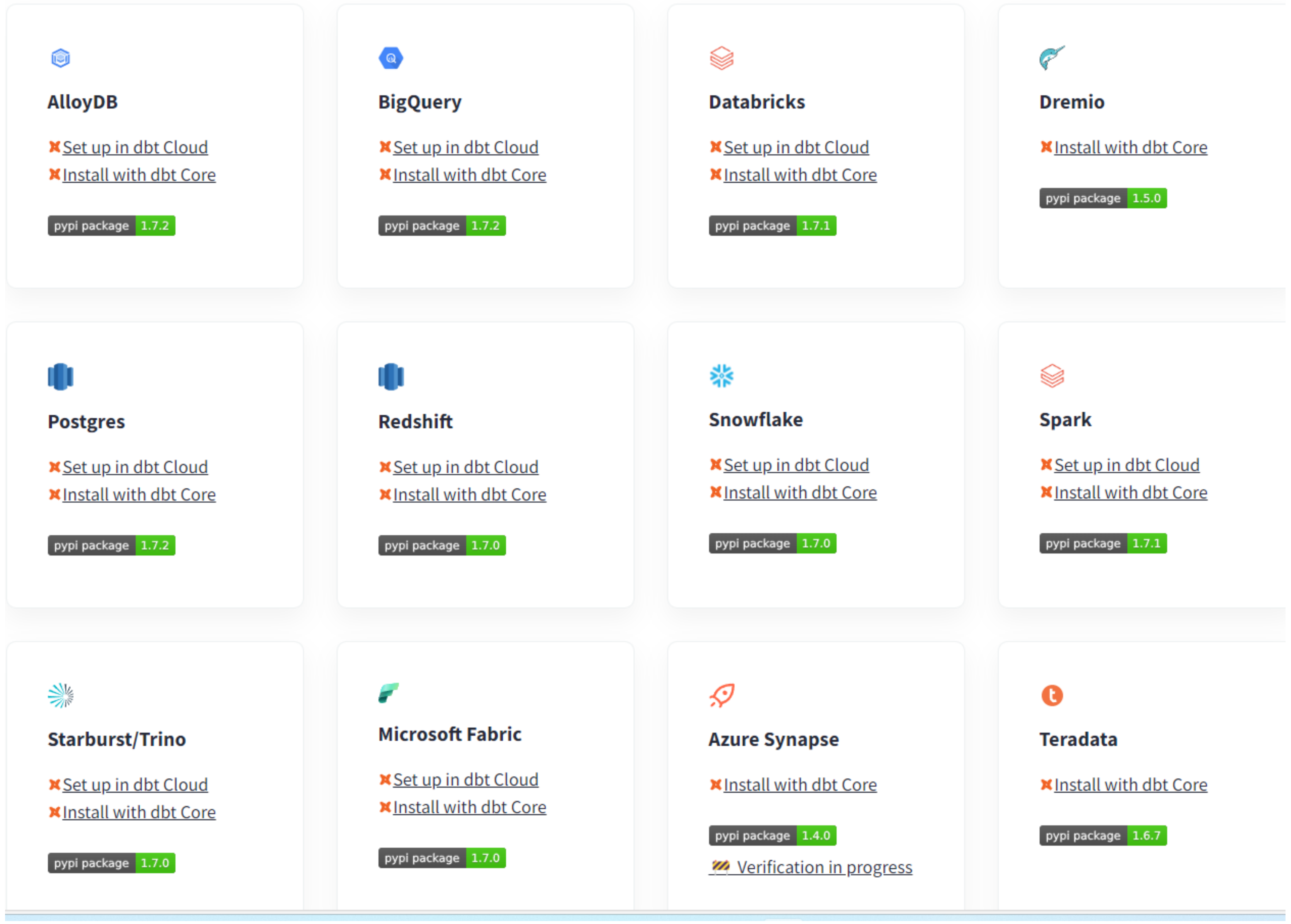
- Data build tools
- Open-source framework for data transformations within data warehouses.
- Only does the T in ETL
- Focuses on transformations using SQL, automates DDL
- Brings software development best practices to data engineering
  - Version control
  - Testing
  - Code reusability
- dbt is written in Python ➔ works well in CI/CD
- Data never leaves our data warehouse!
- Huge community >30.000 customers

Introduction to dbt

This session only discusses dbt-core features

dbt-core	dbt-cloud
• Open Source • Free to use • Command line only	• Commercial Product • Browser IDE • Job scheduling • CI/CD • Documentation hosting

Adapters



AlloyDB
- Set up in dbt Cloud
- Install with dbt Core

pypi package 1.7.2

BigQuery
- Set up in dbt Cloud
- Install with dbt Core

pypi package 1.7.2

Databricks
- Set up in dbt Cloud
- Install with dbt Core

pypi package 1.7.1

Dremio
- Install with dbt Core

pypi package 1.5.0

Postgres
- Set up in dbt Cloud
- Install with dbt Core

pypi package 1.7.2

Redshift
- Set up in dbt Cloud
- Install with dbt Core

pypi package 1.7.0

Snowflake
- Set up in dbt Cloud
- Install with dbt Core

pypi package 1.7.0

Spark
- Set up in dbt Cloud
- Install with dbt Core

pypi package 1.7.1

Starburst/Trino
- Set up in dbt Cloud
- Install with dbt Core

pypi package 1.7.0

Microsoft Fabric
- Set up in dbt Cloud
- Install with dbt Core

pypi package 1.7.0

Azure Synapse
- Install with dbt Core

pypi package 1.4.0

Verification in progress

Teradata
- Install with dbt Core

pypi package 1.6.7

dbt is templating engine

We write SQL templates using jinja

and configuration and metadata using YAML

SQL templates are compiled to runnable code during build

Compiled SQL is executed on target platform

dbt terminology

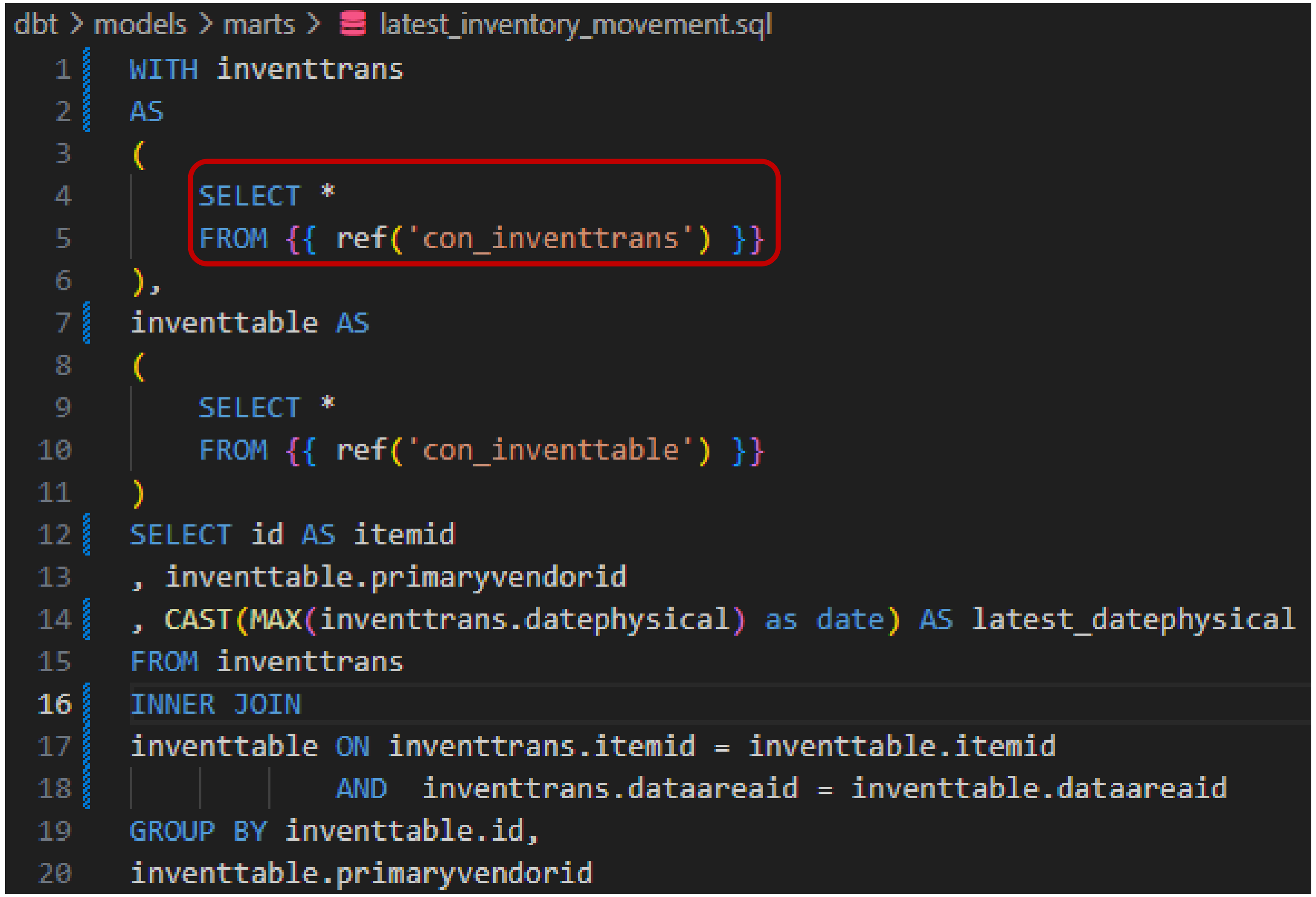
Model: A select statement

Materialization: How should a model be created in target (table, view etc)

Source: External data

Exposure: Downstream use of models outside of dbt

Models are defined in SQL



dbt > models > marts > latest_inventory_movement.sql

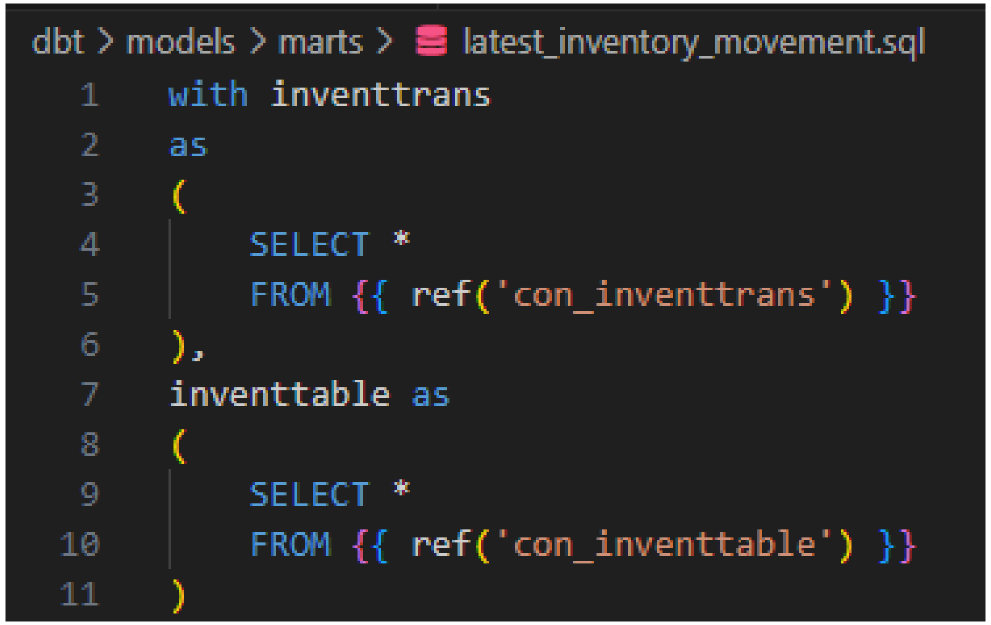
```sql
WITH inventtrans
AS
(
    SELECT *
    FROM {{ ref('con_inventtrans') }}
),
inventtable AS
(
    SELECT *
    FROM {{ ref('con_inventtable') }}
)
SELECT id AS itemid
, inventtable.primaryvendorid
, CAST(MAX(inventtrans.datephysical) as date) AS latest_datephysical
FROM inventtrans
INNER JOIN
inventtable ON inventtrans.itemid = inventtable.itemid
            AND  inventtrans.dataareaid = inventtable.dataareaid
GROUP BY inventtable.id,
inventtable.primaryvendorid
```

Macros – ref()

- Ref() is used to reference models in our project
- By using ref() we do not have to rewrite downstream models when we move target schemas

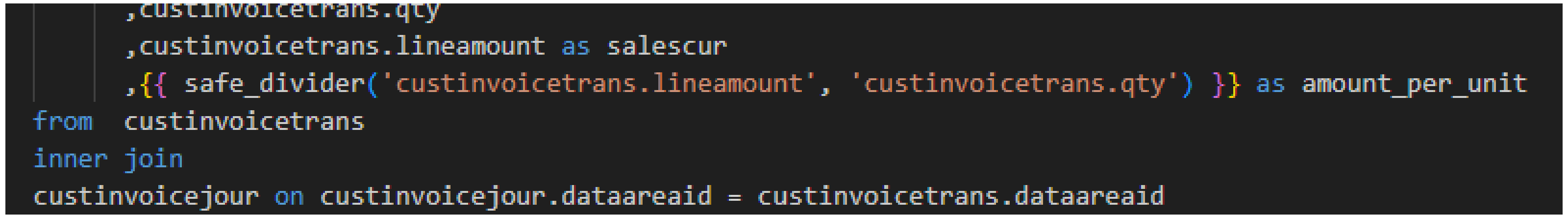
➔Model names need to be unique in dbt

➔Ref() used to build a DAG

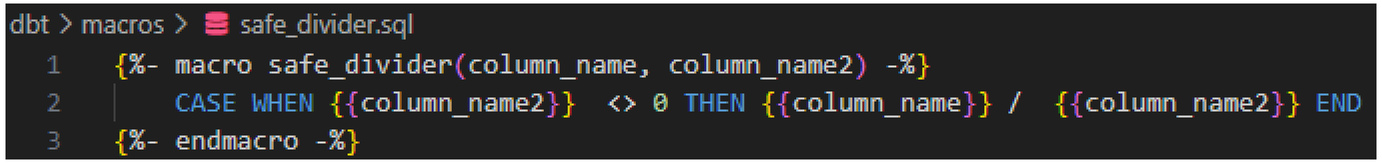


```
with inventtrans
as
(
    SELECT *
    FROM {{ ref('con_inventtrans') }}
),
inventtable as
(
    SELECT *
    FROM {{ ref('con_inventtable') }}
)
```

We can add our own custom macros

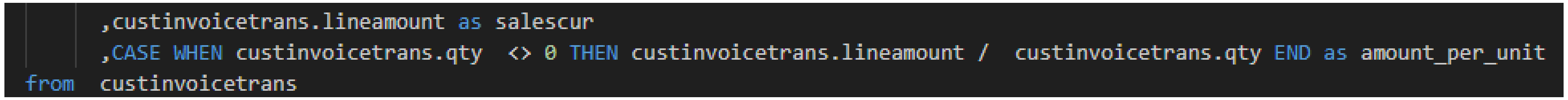


```sql
        ,custinvoicetrans.qty
        ,custinvoicetrans.lineamount as salescur
        ,{{ safe_divider('custinvoicetrans.lineamount', 'custinvoicetrans.qty') }} as amount_per_unit
from  custinvoicetrans
inner join
custinvoicejour on custinvoicejour.dataareaid = custinvoicetrans.dataareaid
```



```sql
{%- macro safe_divider(column_name, column_name2) -%}
    CASE WHEN {{column_name2}}  <> 0 THEN {{column_name}} /  {{column_name2}} END
{%- endmacro -%}
```

! dbt labs recommends not going overboard with macros. Favor readability over strict DRY adherence (Don't repeat yourself)

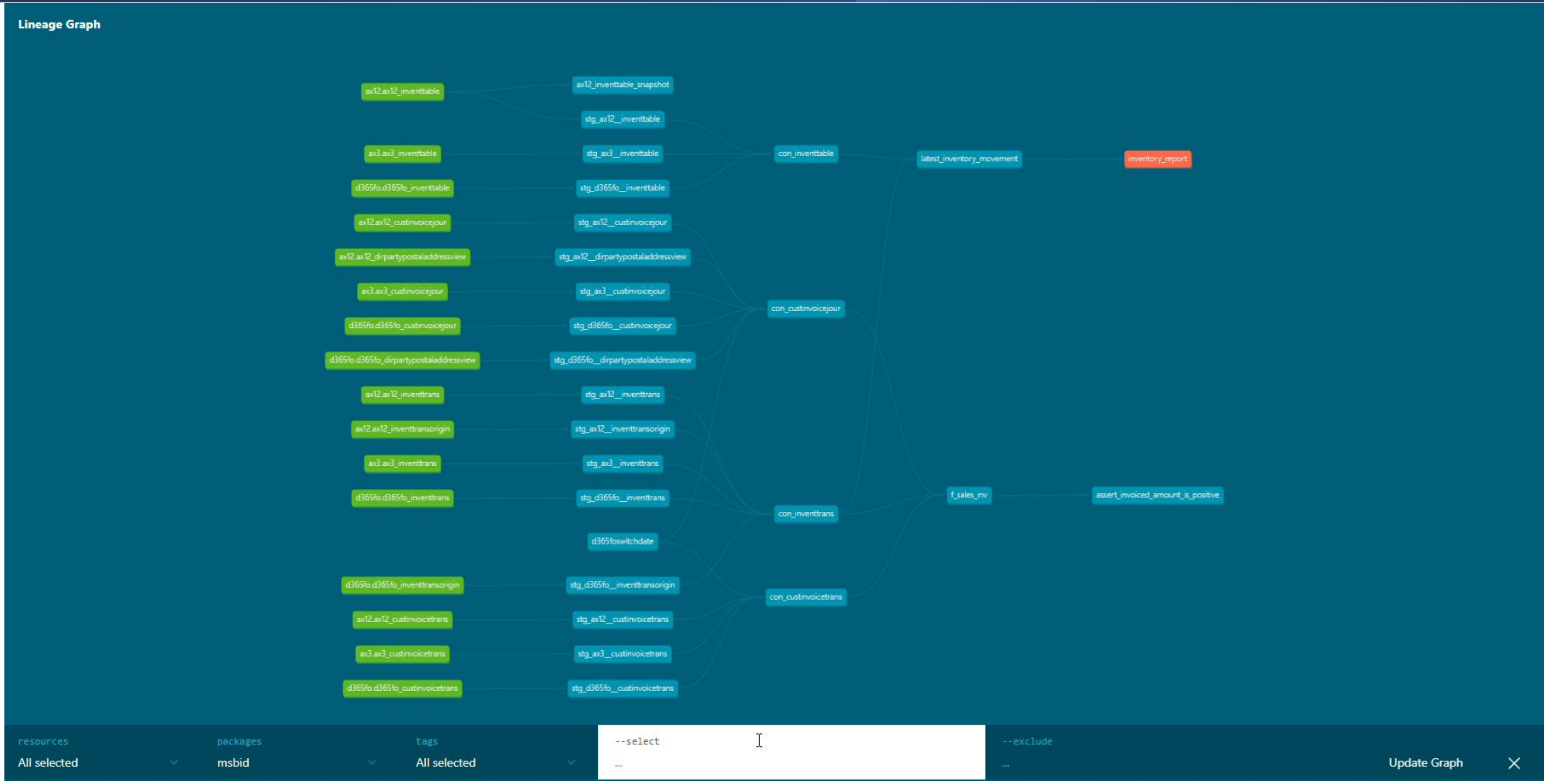


```sql
        ,custinvoicetrans.lineamount as salescur
        ,CASE WHEN custinvoicetrans.qty  <> 0 THEN custinvoicetrans.lineamount /  custinvoicetrans.qty END as amount_per_unit
from  custinvoicetrans
```

DAG enables…

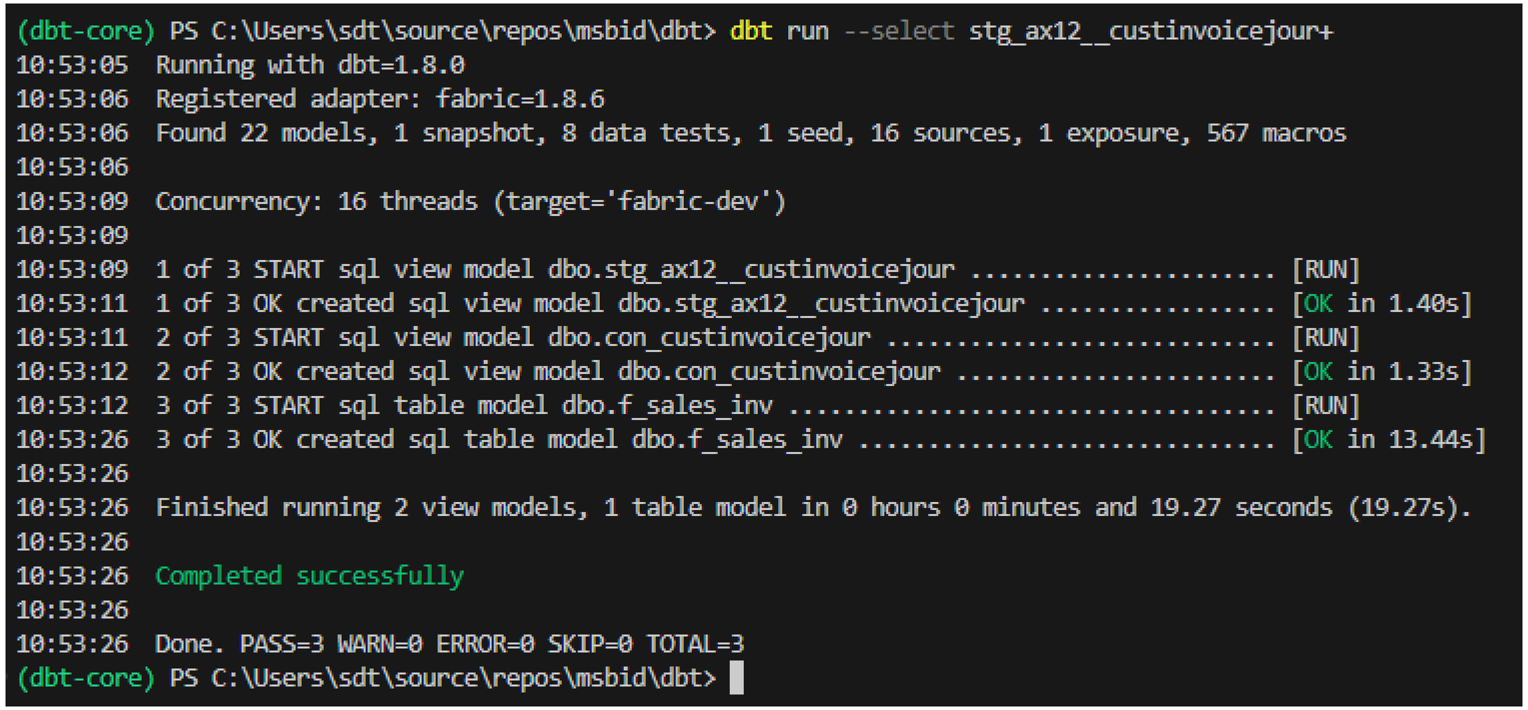
- Table lineage
- Execution order
- Graph operators

Select on: models, sources, exposures, tags, path, states, configs…


Lineage Graph
ax12.ax12_inventtable
ax3.ax3_inventtable
d365fo.d365fo_inventtable
ax12.ax12_custinvoicejour
ax12.ax12_dirpartypostaladdressview
ax3.ax3_custinvoicejour
d365fo.d365fo_custinvoicejour
d365fo.d365fo_dirpartypostaladdressview
ax12.ax12_inventtrans
ax12.ax12_inventtransorigin
ax3.ax3_inventtrans
d365fo.d365fo_inventtrans
d365fo.d365fo_inventtransorigin
ax12.ax12_custinvoicetrans
ax3.ax3_custinvoicetrans
d365fo.d365fo_custinvoicetrans
ax12_inventtable_snapshot
stg_ax12__inventtable
stg_ax3__inventtable
stg_d365fo__inventtable
stg_ax12__custinvoicejour
stg_ax12__dirpartypostaladdressview
stg_ax3__custinvoicejour
stg_d365fo__custinvoicejour
stg_d365fo__dirpartypostaladdressview
stg_ax12__inventtrans
stg_ax12__inventtransorigin
stg_ax3__inventtrans
stg_d365fo__inventtrans
d365foswitchdate
stg_d365fo__inventtransorigin
stg_ax12__custinvoicetrans
stg_ax3__custinvoicetrans
stg_d365fo__custinvoicetrans
con_inventtable
con_custinvoicejour
con_inventtrans
con_custinvoicetrans
latest_inventory_movement
f_sales_inv
inventory_report
assert_invoiced_amount_is_positive
Update Graph

resources: All selected
packages: msbid
tags: All selected
--select ...
--exclude ...

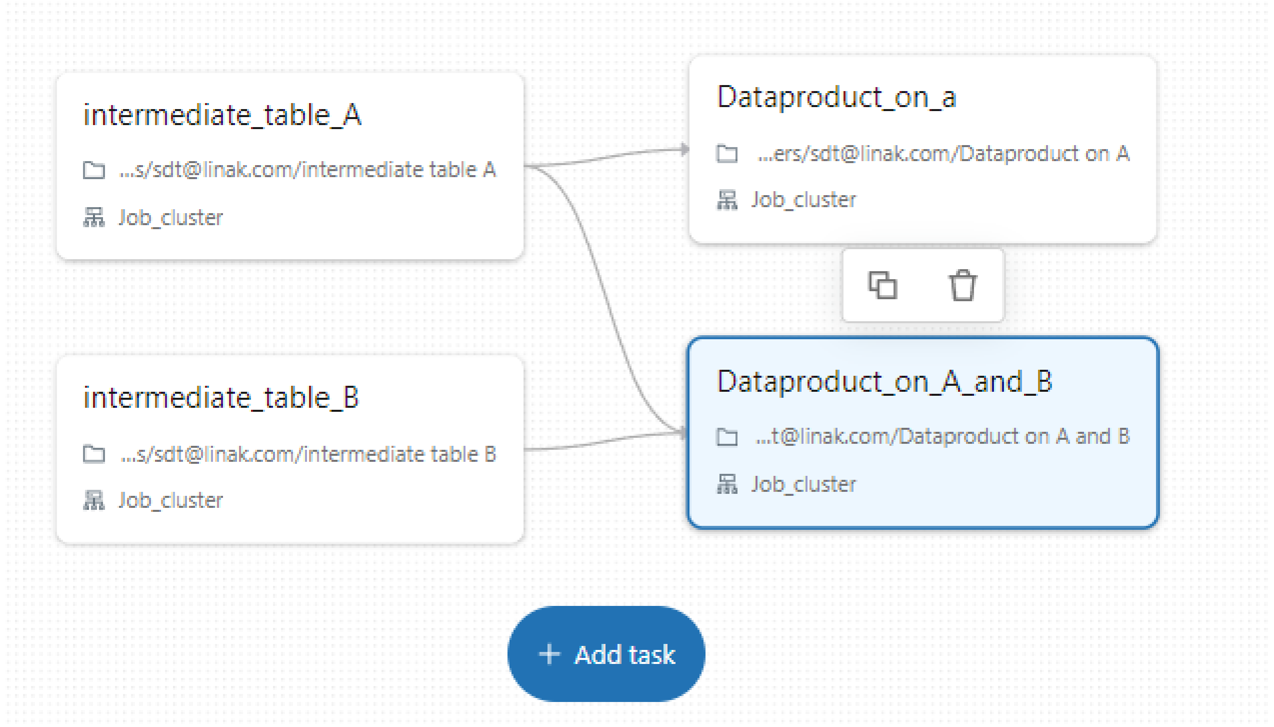
dbt-core is a command line tool



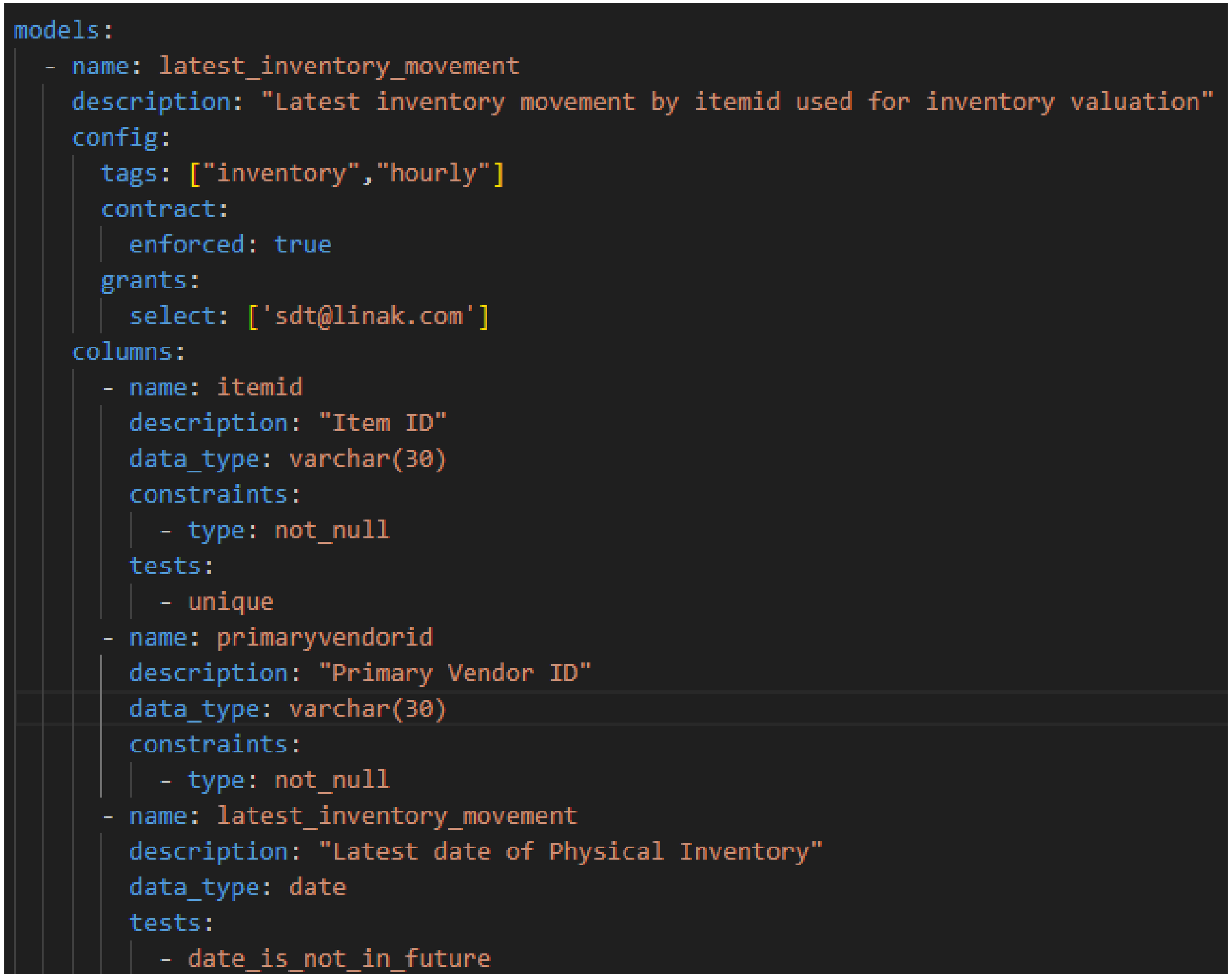
```
(dbt-core) PS C:\Users\sdt\source\repos\msbid\dbt> dbt run --select stg_ax12__custinvoicejour+
10:53:05  Running with dbt=1.8.0
10:53:06  Registered adapter: fabric=1.8.6
10:53:06  Found 22 models, 1 snapshot, 8 data tests, 1 seed, 16 sources, 1 exposure, 567 macros
10:53:06
10:53:09  Concurrency: 16 threads (target='fabric-dev')
10:53:09
10:53:09  1 of 3 START sql view model dbo.stg_ax12__custinvoicejour ...................... [RUN]
10:53:11  1 of 3 OK created sql view model dbo.stg_ax12__custinvoicejour ................. [OK in 1.40s]
10:53:11  2 of 3 START sql view model dbo.con_custinvoicejour ............................ [RUN]
10:53:12  2 of 3 OK created sql view model dbo.con_custinvoicejour ....................... [OK in 1.33s]
10:53:12  3 of 3 START sql table model dbo.f_sales_inv ................................... [RUN]
10:53:26  3 of 3 OK created sql table model dbo.f_sales_inv .............................. [OK in 13.44s]
10:53:26
10:53:26  Finished running 2 view models, 1 table model in 0 hours 0 minutes and 19.27 seconds (19.27s).
10:53:26
10:53:26  Completed successfully
10:53:26
10:53:26  Done. PASS=3 WARN=0 ERROR=0 SKIP=0 TOTAL=3
(dbt-core) PS C:\Users\sdt\source\repos\msbid\dbt>
```

Side note: dbt vs Databricks Jobs

- No DAG
- Data products on different schedules required hand crafted jobs


intermediate_table_A
...s/sdt@linak.com/intermediate table A
Job_cluster
intermediate_table_B
...s/sdt@linak.com/intermediate table B
Job_cluster
Dataproduct_on_a
...ers/sdt@linak.com/Dataproduct on A
Job_cluster
Dataproduct_on_A_and_B
...t@linak.com/Dataproduct on A and B
Job_cluster
+ Add task

Models are described with YAML



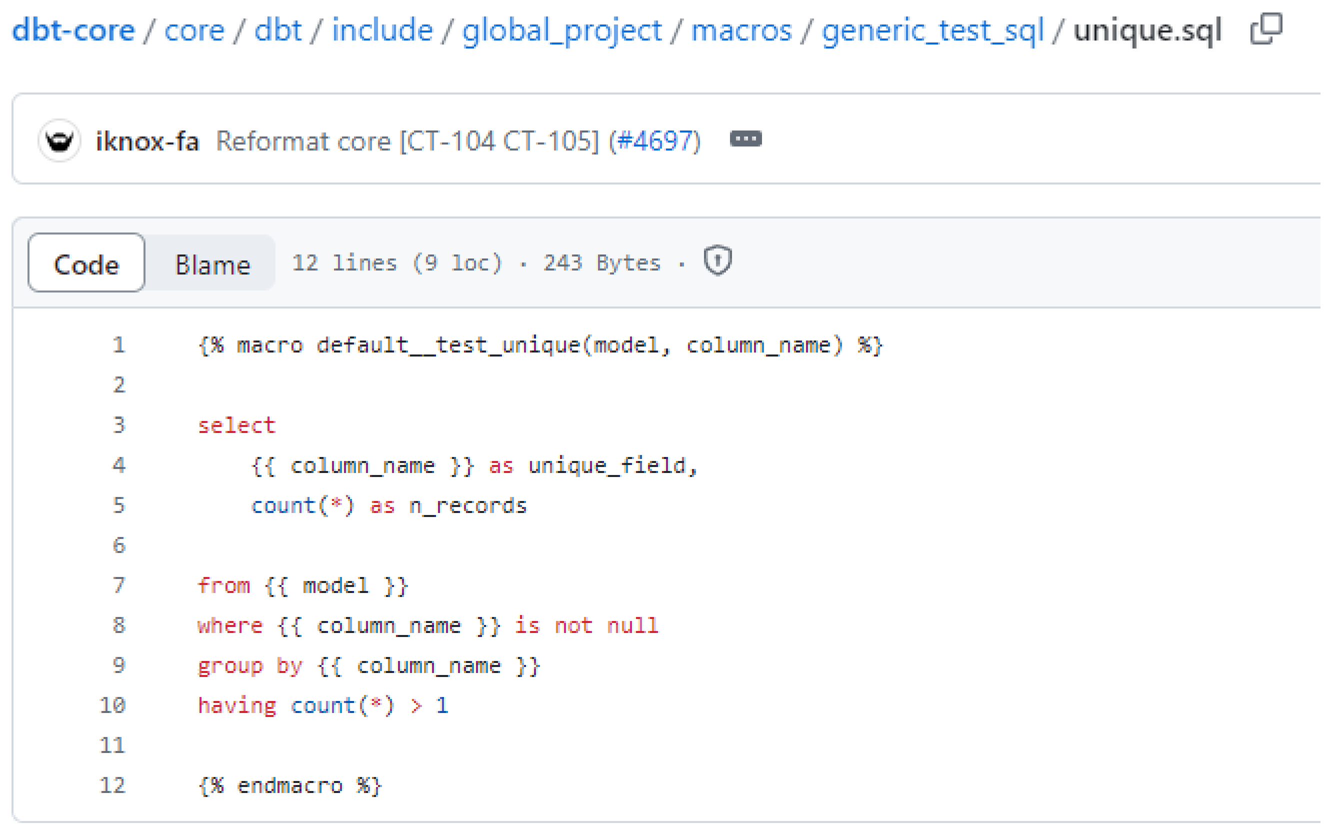
```yaml
models:
  - name: latest_inventory_movement
    description: "Latest inventory movement by itemid used for inventory valuation"
    config:
      tags: ["inventory","hourly"]
      contract:
        enforced: true
      grants:
        select: ['sdt@linak.com']
    columns:
      - name: itemid
        description: "Item ID"
        data_type: varchar(30)
        constraints:
          - type: not_null
        tests:
          - unique
      - name: primaryvendorid
        description: "Primary Vendor ID"
        data_type: varchar(30)
        constraints:
          - type: not_null
      - name: latest_inventory_movement
        description: "Latest date of Physical Inventory"
        data_type: date
        tests:
          - date_is_not_in_future
```

Data tests in dbt

- Implemented as SQL queries
- Fail when query returns rows
- Tests can have different severities
- Executed after model has been materialized
- → Bad data can be in model accessible by users

Generic Tests

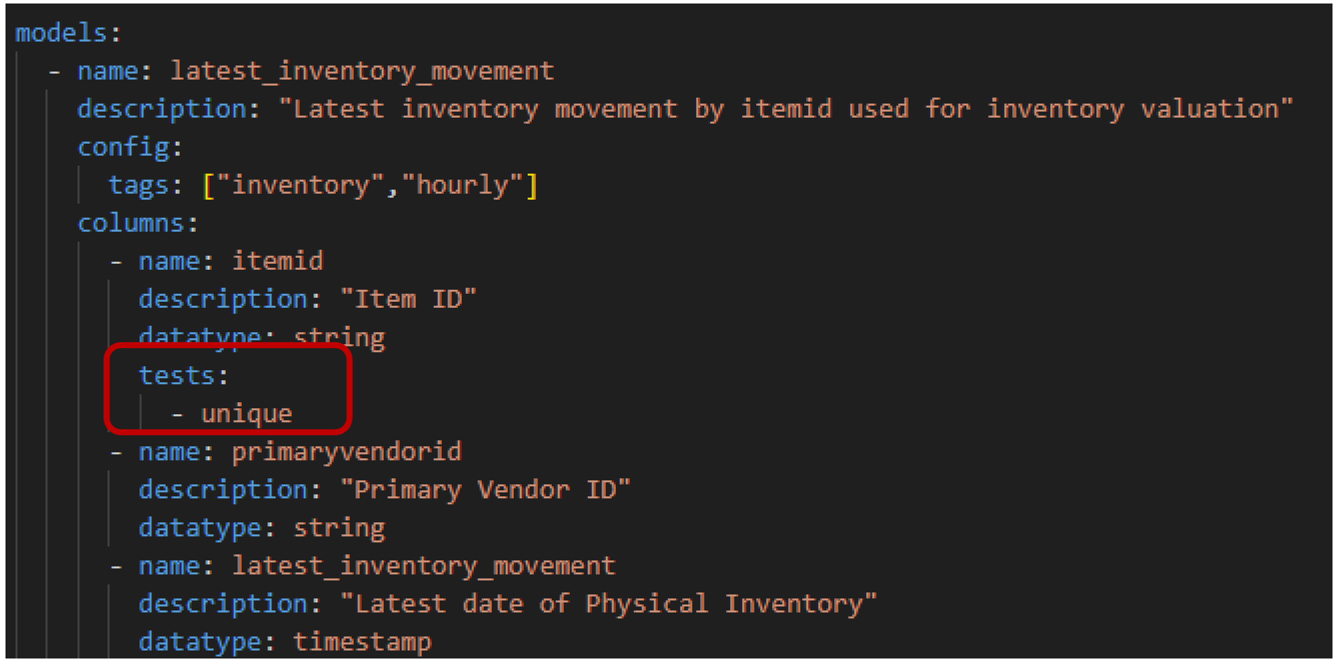
- SQL Templates
- 4 generic tests are included
  - Unique
  - Not_null
  - Accepted_values
  - Relationships
- More can be added manually or by packages



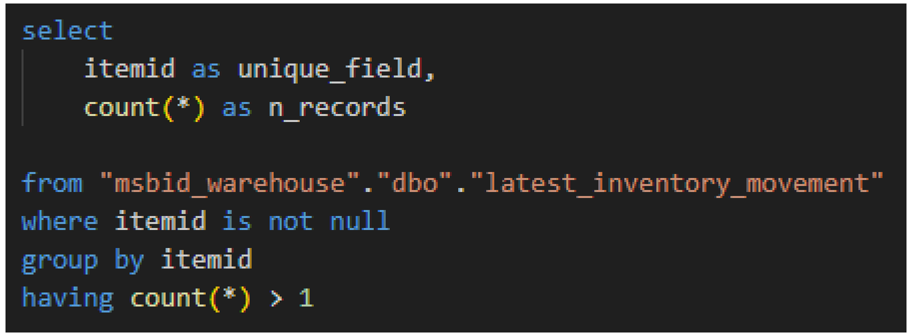
dbt-core / core / dbt / include / global_project / macros / generic_test_sql / unique.sql
iknox-fa Reformat core [CT-104 CT-105] (#4697)
Code
Blame
12 lines (9 loc) · 243 Bytes

```
{% macro default__test_unique(model, column_name) %}

select
    {{ column_name }} as unique_field,
    count(*) as n_records

from {{ model }}
where {{ column_name }} is not null
group by {{ column_name }}
having count(*) > 1

{% endmacro %}
```

Adding a generic test

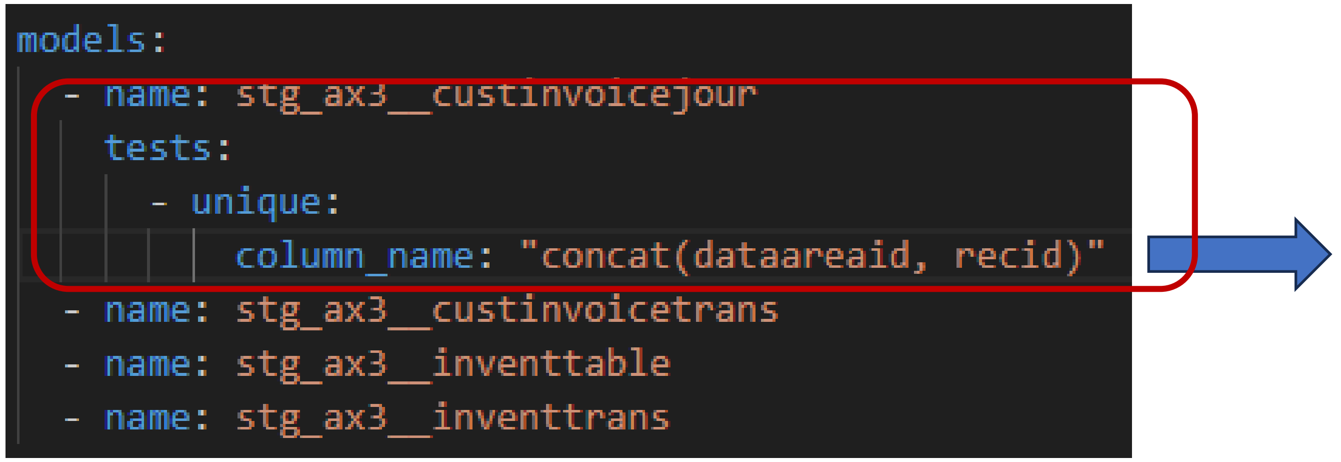


```
models:
  - name: latest_inventory_movement
    description: "Latest inventory movement by itemid used for inventory valuation"
    config:
      tags: ["inventory","hourly"]
    columns:
      - name: itemid
        description: "Item ID"
        datatype: string
        tests:
          - unique
      - name: primaryvendorid
        description: "Primary Vendor ID"
        datatype: string
      - name: latest_inventory_movement
        description: "Latest date of Physical Inventory"
        datatype: timestamp
```

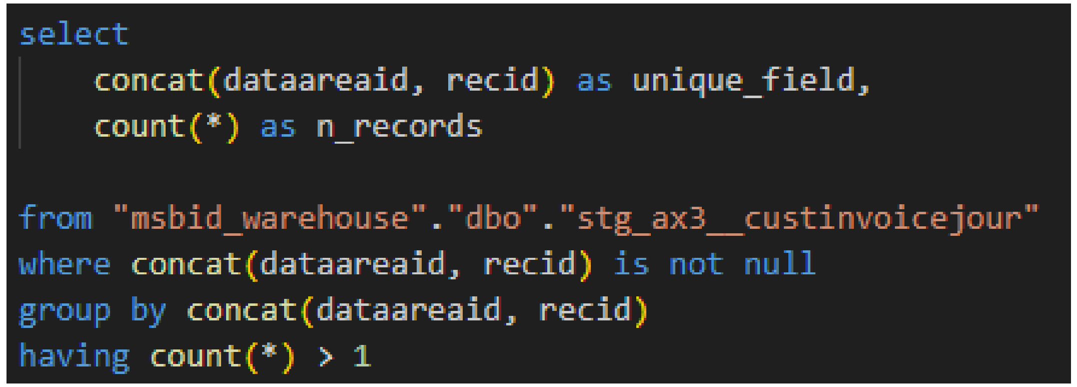


```
select
    itemid as unique_field,
    count(*) as n_records

from "msbid_warehouse"."dbo"."latest_inventory_movement"
where itemid is not null
group by itemid
having count(*) > 1
```

Generic Test



```
models:
  - name: stg_ax3__custinvoicejour
    tests:
      - unique:
          column_name: "concat(dataareaid, recid)"
  - name: stg_ax3__custinvoicetrans
  - name: stg_ax3__inventtable
  - name: stg_ax3__inventtrans
```

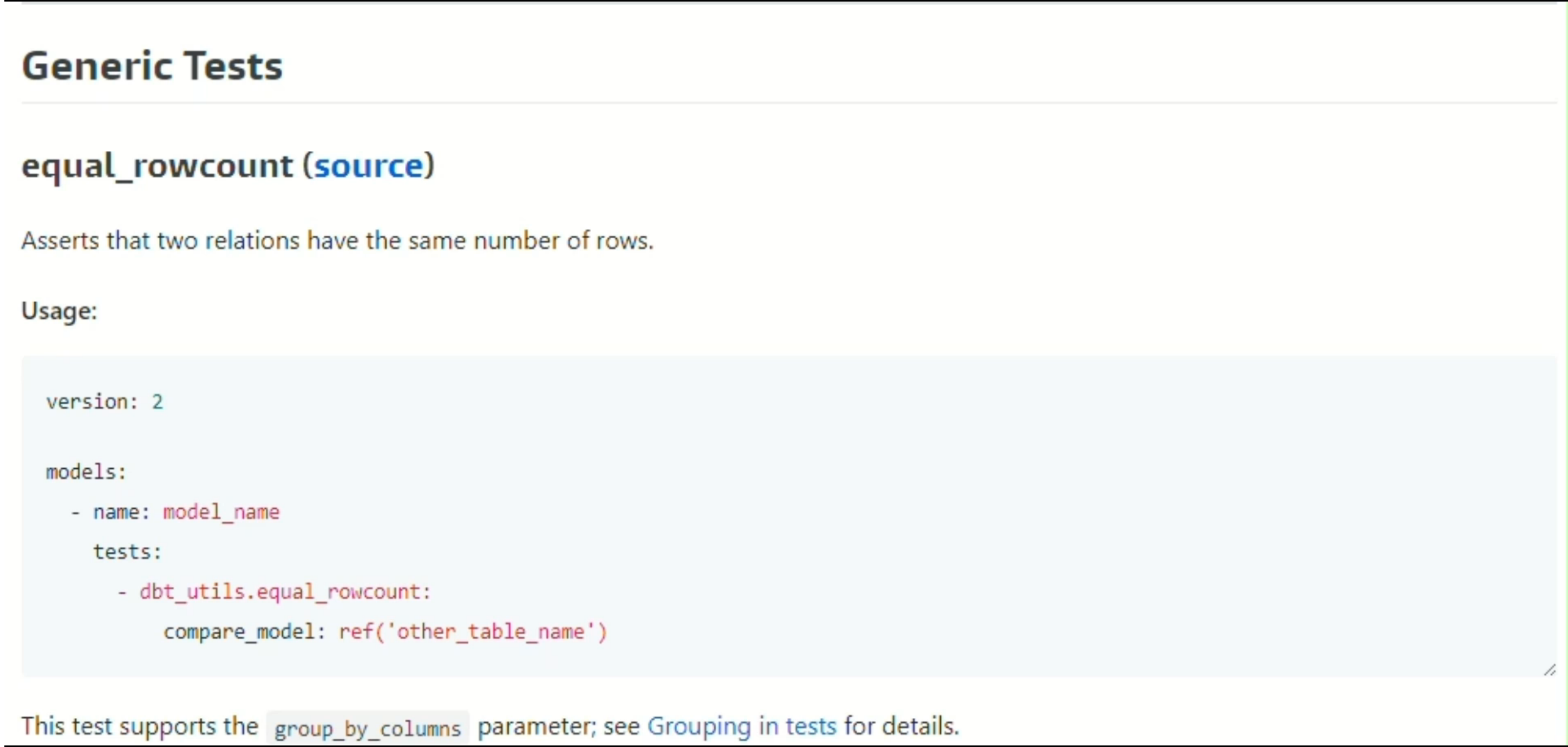


```
select
    concat(dataareaid, recid) as unique_field,
    count(*) as n_records

from "msbid_warehouse"."dbo"."stg_ax3__custinvoicejour"
where concat(dataareaid, recid) is not null
group by concat(dataareaid, recid)
having count(*) > 1
```

Generic Tests from dbt_utils

dbt_expectations contains dozens of tests inspired by Great Expectations

Some templates use unsupported SQL syntax on Fabric



Generic Tests

equal_rowcount (source)

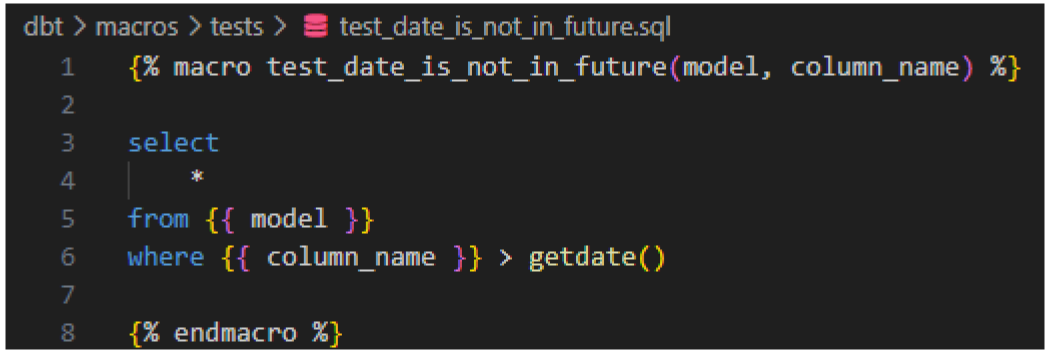
Asserts that two relations have the same number of rows.

Usage:

```
version: 2

models:
  - name: model_name
    tests:
      - dbt_utils.equal_rowcount:
          compare_model: ref('other_table_name')
```

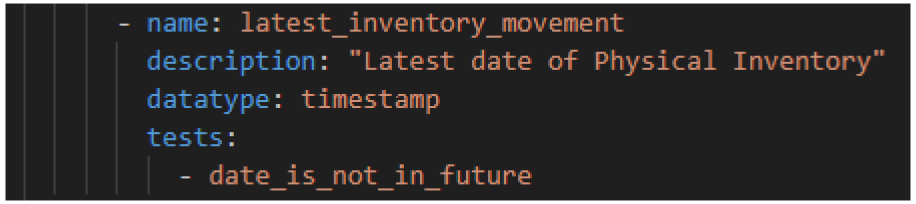
This test supports the `group_by_columns` parameter; see Grouping in tests for details.

Custom Generic Test

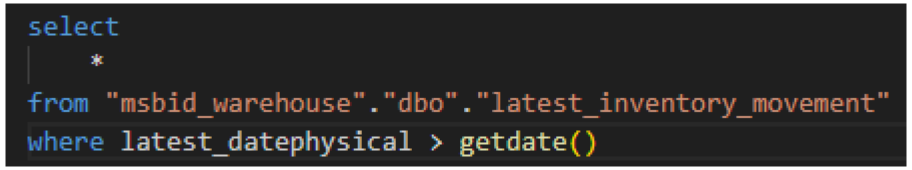


dbt > macros > tests > test_date_is_not_in_future.sql

```
{% macro test_date_is_not_in_future(model, column_name) %}

select
    *
from {{ model }}
where {{ column_name }} > getdate()

{% endmacro %}
```

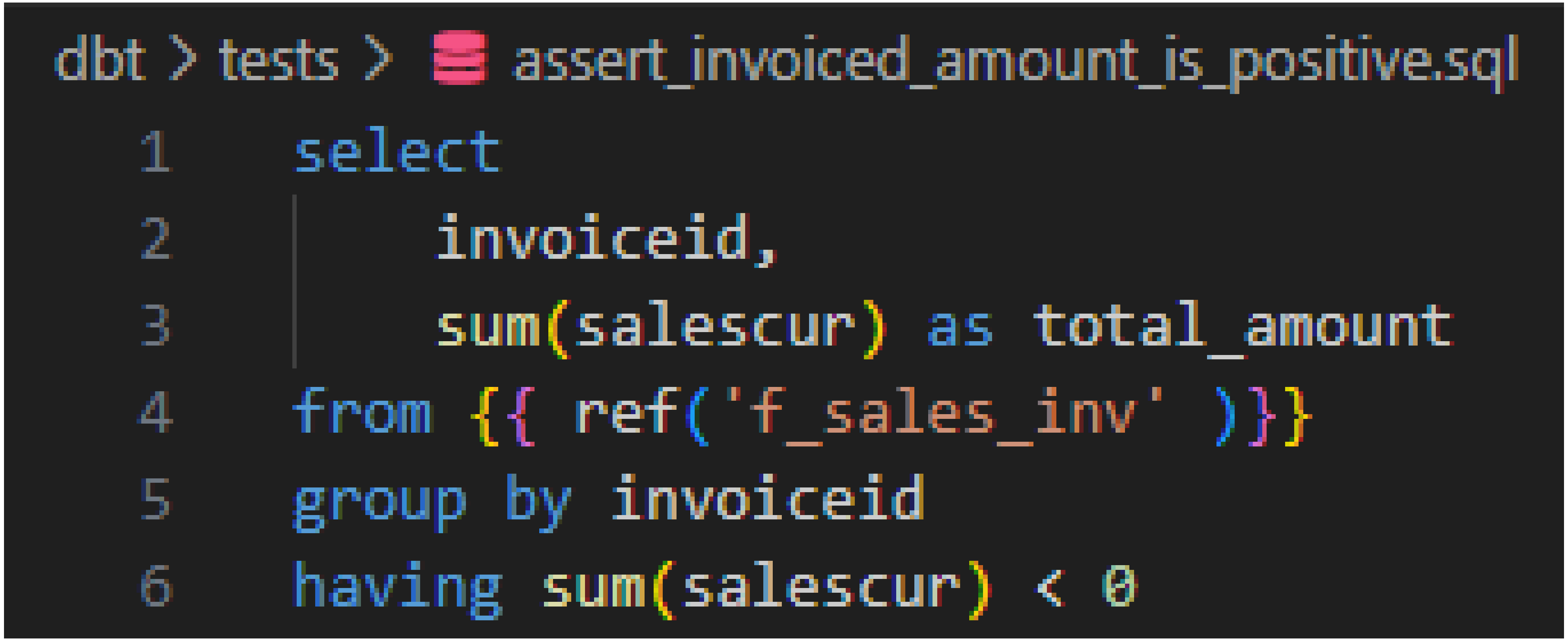


```
- name: latest_inventory_movement
  description: "Latest date of Physical Inventory"
  datatype: timestamp
  tests:
    - date_is_not_in_future
```



```
select
    *
from "msbid_warehouse"."dbo"."latest_inventory_movement"
where latest_datephysical > getdate()
```

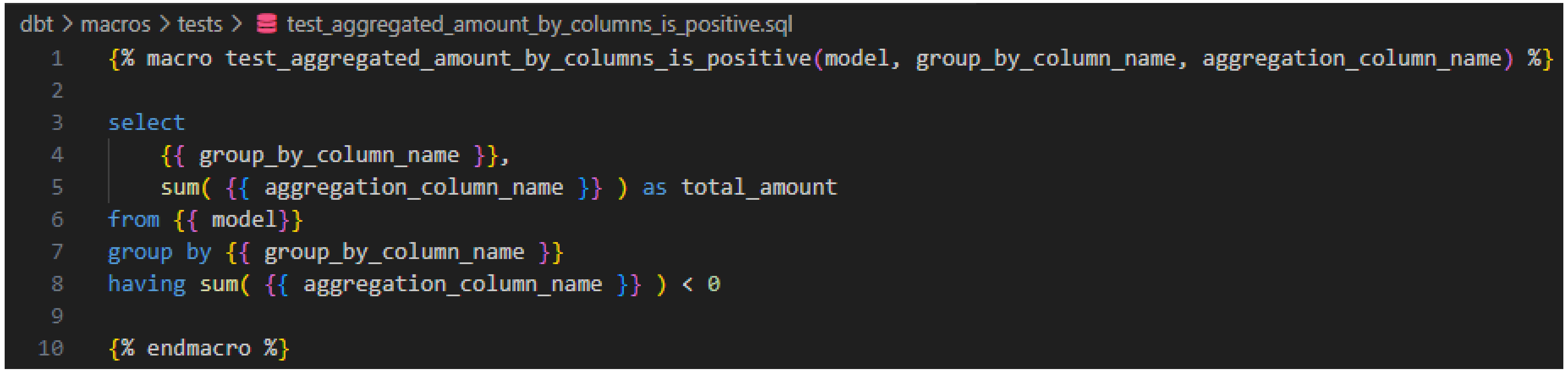
Singular Tests – Work only on a specific model



dbt > tests > assert_invoiced_amount_is_positive.sql

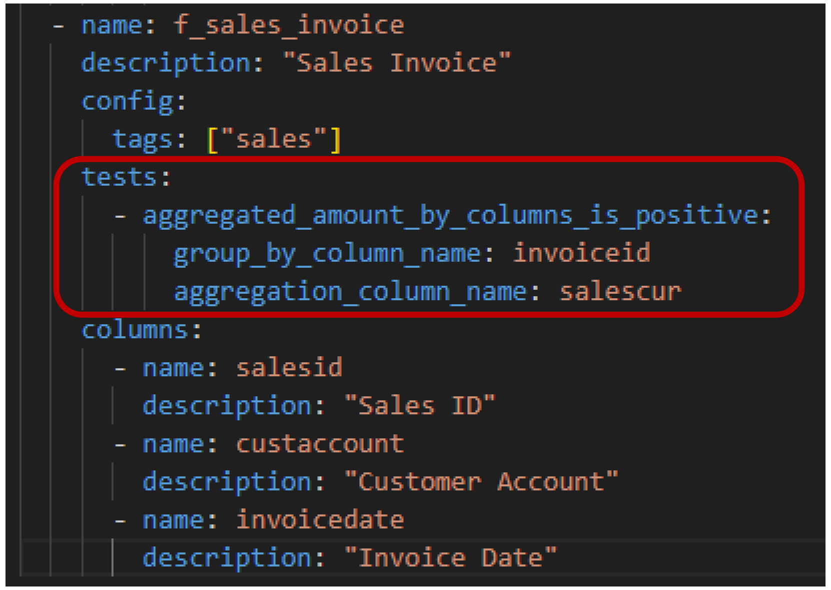
```sql
select
    invoiceid,
    sum(salescur) as total_amount
from {{ ref('f_sales_inv' )}}
group by invoiceid
having sum(salescur) < 0
```

Generic Version of Singular Test



dbt > macros > tests > test_aggregated_amount_by_columns_is_positive.sql

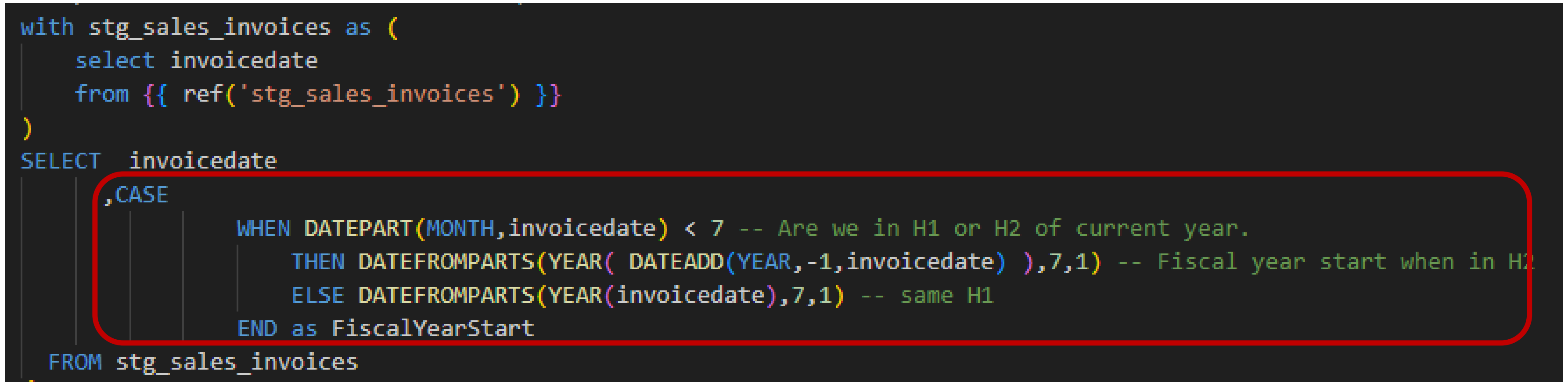
```sql
{% macro test_aggregated_amount_by_columns_is_positive(model, group_by_column_name, aggregation_column_name) %}

select
    {{ group_by_column_name }},
    sum( {{ aggregation_column_name }} ) as total_amount
from {{ model}}
group by {{ group_by_column_name }}
having sum( {{ aggregation_column_name }} ) < 0

{% endmacro %}
```



```yaml
- name: f_sales_invoice
  description: "Sales Invoice"
  config:
    tags: ["sales"]
  tests:
    - aggregated_amount_by_columns_is_positive:
        group_by_column_name: invoiceid
        aggregation_column_name: salescur
  columns:
    - name: salesid
      description: "Sales ID"
    - name: custaccount
      description: "Customer Account"
    - name: invoicedate
      description: "Invoice Date"
```

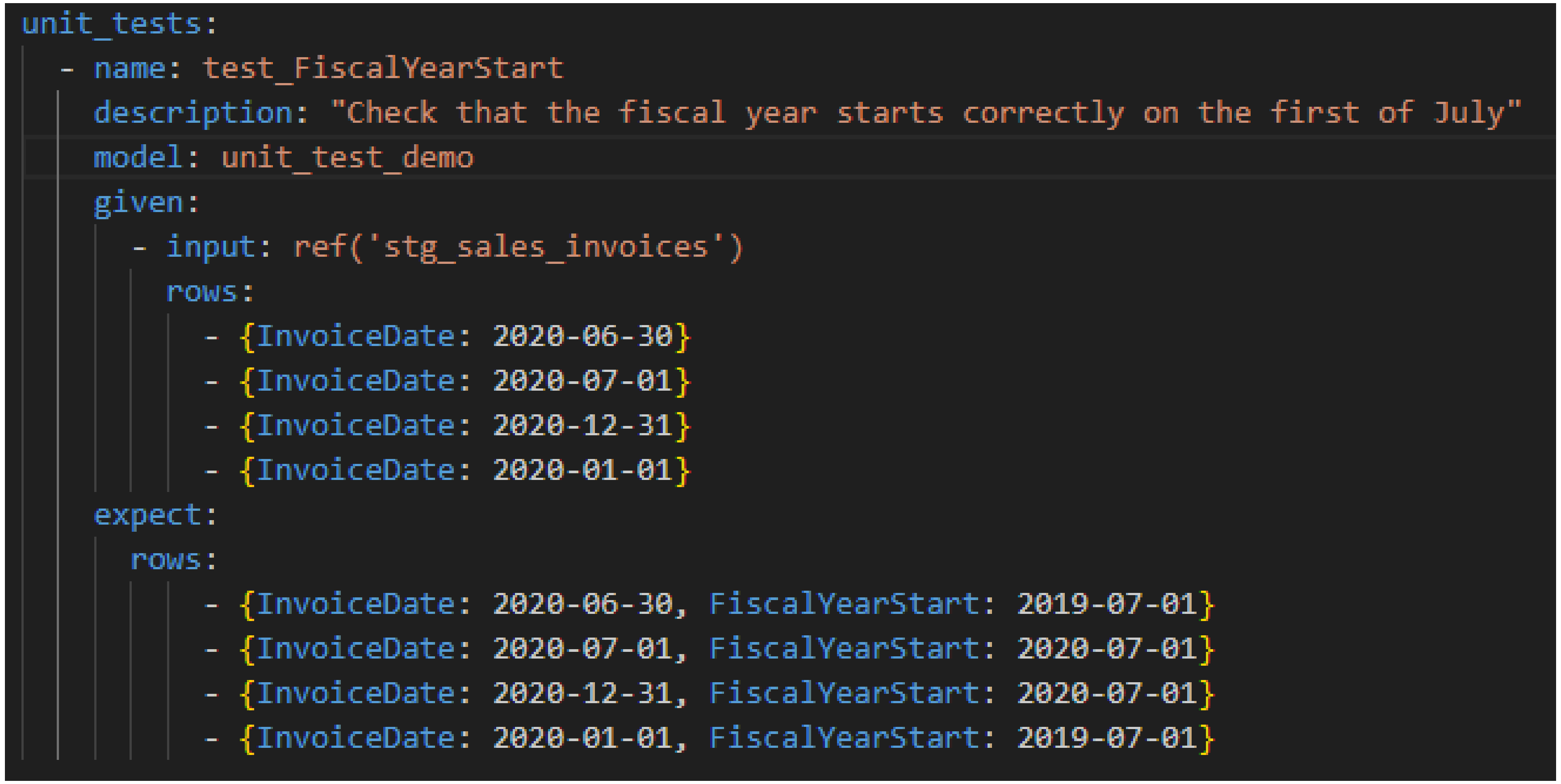
Unit Tests

- Brand new feature
- Instead of testing on data in warehouse we test on well defined data with a known result
- Are we sure that the logic to find the start of fiscal year always works?



```
with stg_sales_invoices as (
    select invoicedate
    from {{ ref('stg_sales_invoices') }}
)
SELECT  invoicedate
      ,CASE
                WHEN DATEPART(MONTH,invoicedate) < 7 -- Are we in H1 or H2 of current year.
                    THEN DATEFROMPARTS(YEAR( DATEADD(YEAR,-1,invoicedate) ),7,1) -- Fiscal year start when in H
                    ELSE DATEFROMPARTS(YEAR(invoicedate),7,1) -- same H1
                END as FiscalYearStart
  FROM stg_sales_invoices
```

Unit Tests



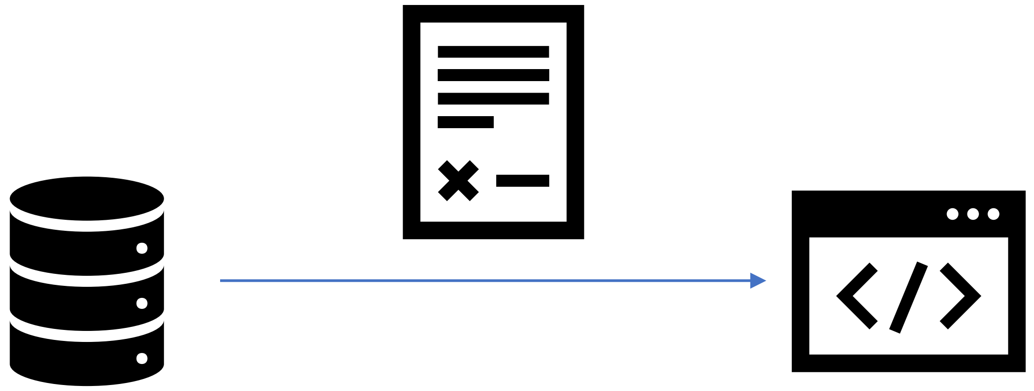
```
unit_tests:
  - name: test_FiscalYearStart
    description: "Check that the fiscal year starts correctly on the first of July"
    model: unit_test_demo
    given:
      - input: ref('stg_sales_invoices')
        rows:
          - {InvoiceDate: 2020-06-30}
          - {InvoiceDate: 2020-07-01}
          - {InvoiceDate: 2020-12-31}
          - {InvoiceDate: 2020-01-01}
    expect:
      rows:
          - {InvoiceDate: 2020-06-30, FiscalYearStart: 2019-07-01}
          - {InvoiceDate: 2020-07-01, FiscalYearStart: 2020-07-01}
          - {InvoiceDate: 2020-12-31, FiscalYearStart: 2020-07-01}
          - {InvoiceDate: 2020-01-01, FiscalYearStart: 2019-07-01}
```

Known edge cases

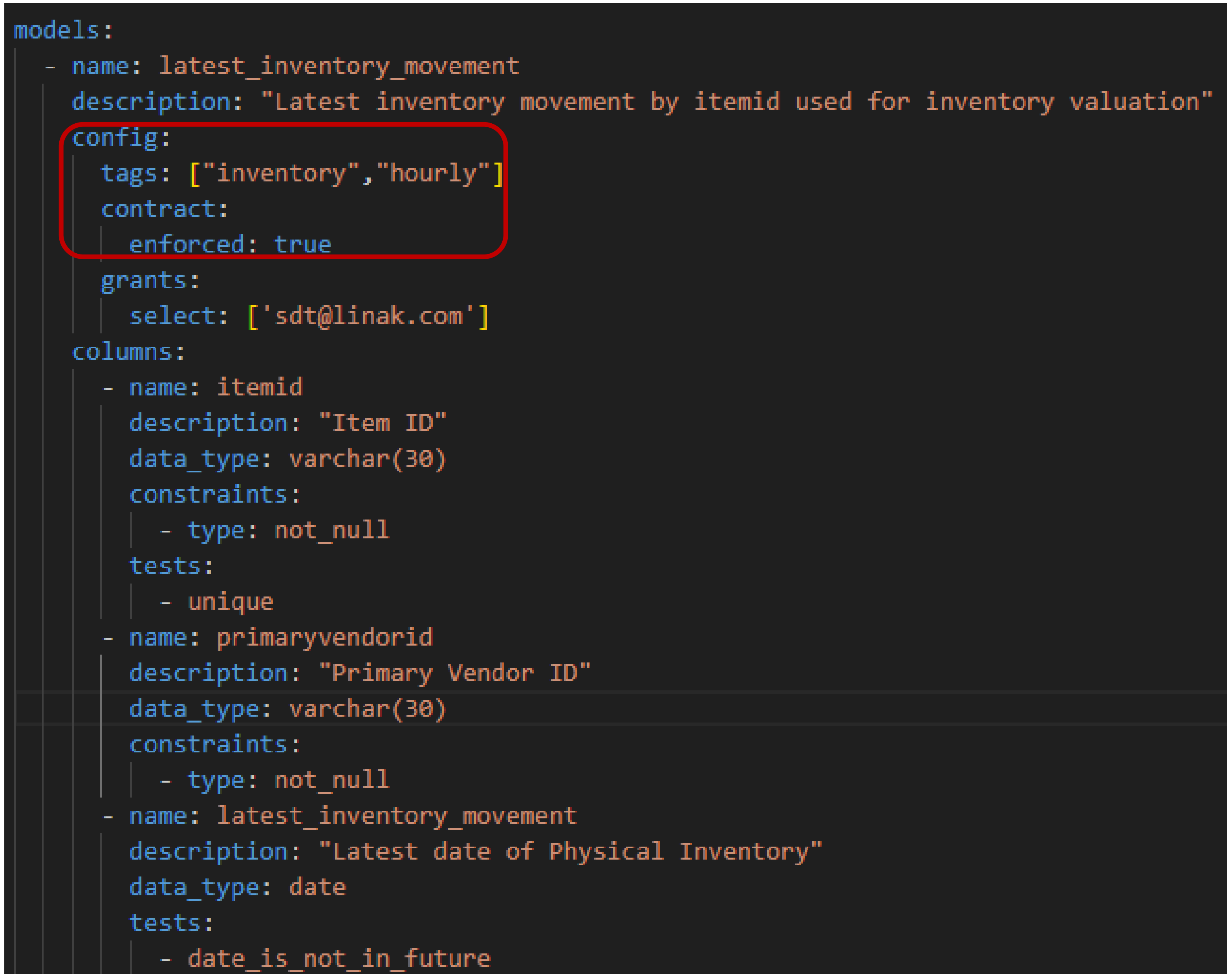
Expected result

Contracts

- Data platforms are used by down stream applications
- Any changes to data shapes may break integrations
- Contracts define and enforce a specified data shape

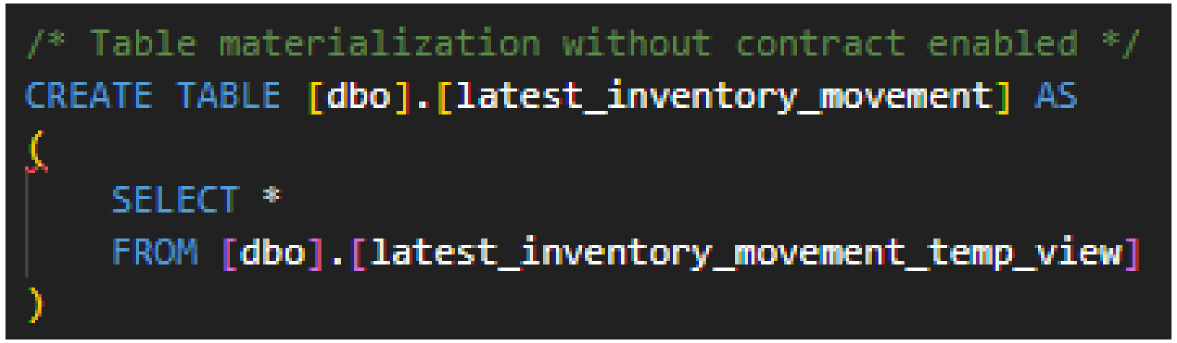


Contracts are specified in YAML

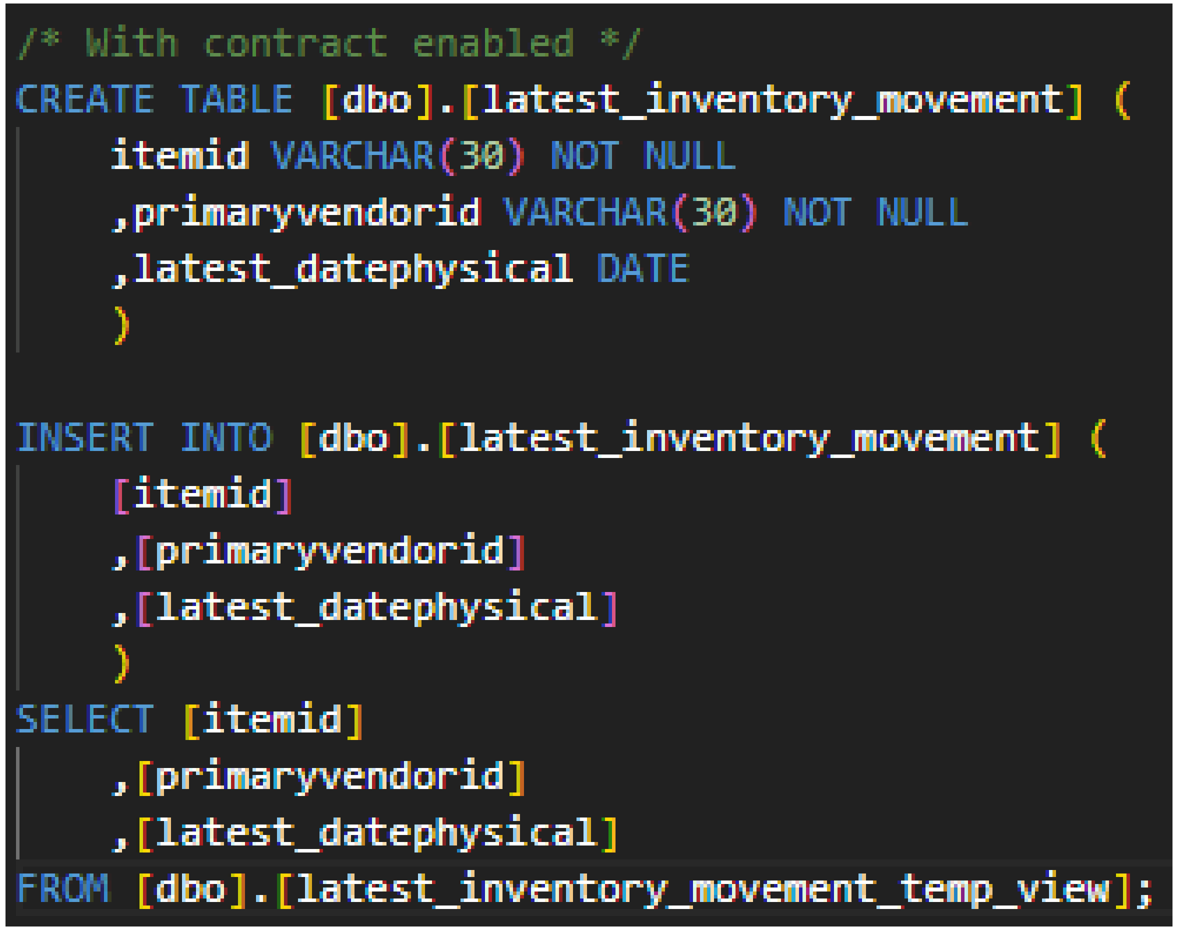


```yaml
models:
  - name: latest_inventory_movement
    description: "Latest inventory movement by itemid used for inventory valuation"
    config:
      tags: ["inventory","hourly"]
      contract:
        enforced: true
      grants:
        select: ['sdt@linak.com']
    columns:
      - name: itemid
        description: "Item ID"
        data_type: varchar(30)
        constraints:
          - type: not_null
        tests:
          - unique
      - name: primaryvendorid
        description: "Primary Vendor ID"
        data_type: varchar(30)
        constraints:
          - type: not_null
      - name: latest_inventory_movement
        description: "Latest date of Physical Inventory"
        data_type: date
        tests:
          - date_is_not_in_future
```

Contracts are enforced by platform

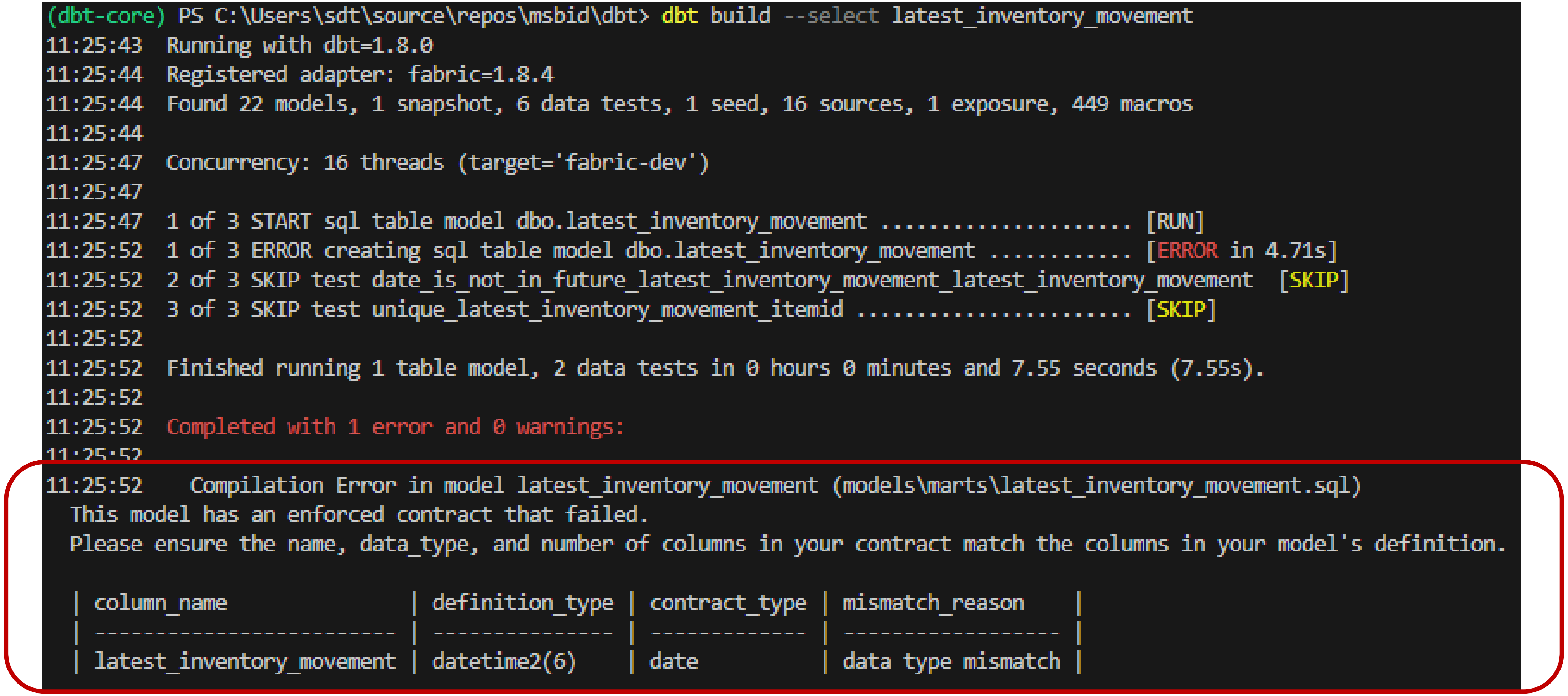


```sql
/* Table materialization without contract enabled */
CREATE TABLE [dbo].[latest_inventory_movement] AS
(
    SELECT *
    FROM [dbo].[latest_inventory_movement_temp_view]
)
```



```sql
/* With contract enabled */
CREATE TABLE [dbo].[latest_inventory_movement] (
    itemid VARCHAR(30) NOT NULL
    ,primaryvendorid VARCHAR(30) NOT NULL
    ,latest_datephysical DATE
    )

INSERT INTO [dbo].[latest_inventory_movement] (
    [itemid]
    ,[primaryvendorid]
    ,[latest_datephysical]
    )
SELECT [itemid]
    ,[primaryvendorid]
    ,[latest_datephysical]
FROM [dbo].[latest_inventory_movement_temp_view];
```

Breaking contracts causes failure



```
(dbt-core) PS C:\Users\sdt\source\repos\msbid\dbt> dbt build --select latest_inventory_movement
11:25:43  Running with dbt=1.8.0
11:25:44  Registered adapter: fabric=1.8.4
11:25:44  Found 22 models, 1 snapshot, 6 data tests, 1 seed, 16 sources, 1 exposure, 449 macros
11:25:44
11:25:47  Concurrency: 16 threads (target='fabric-dev')
11:25:47
11:25:47  1 of 3 START sql table model dbo.latest_inventory_movement ...................... [RUN]
11:25:52  1 of 3 ERROR creating sql table model dbo.latest_inventory_movement ............. [ERROR in 4.71s]
11:25:52  2 of 3 SKIP test date_is_not_in_future_latest_inventory_movement_latest_inventory_movement  [SKIP]
11:25:52  3 of 3 SKIP test unique_latest_inventory_movement_itemid ........................ [SKIP]
11:25:52
11:25:52  Finished running 1 table model, 2 data tests in 0 hours 0 minutes and 7.55 seconds (7.55s).
11:25:52
11:25:52  Completed with 1 error and 0 warnings:
11:25:52
11:25:52    Compilation Error in model latest_inventory_movement (models\marts\latest_inventory_movement.sql)
  This model has an enforced contract that failed.
  Please ensure the name, data_type, and number of columns in your contract match the columns in your model's definition.

  | column_name               | definition_type | contract_type | mismatch_reason    |
  | ------------------------- | --------------- | ------------- | ------------------ |
  | latest_inventory_movement | datetime2(6)    | date          | data type mismatch |
```

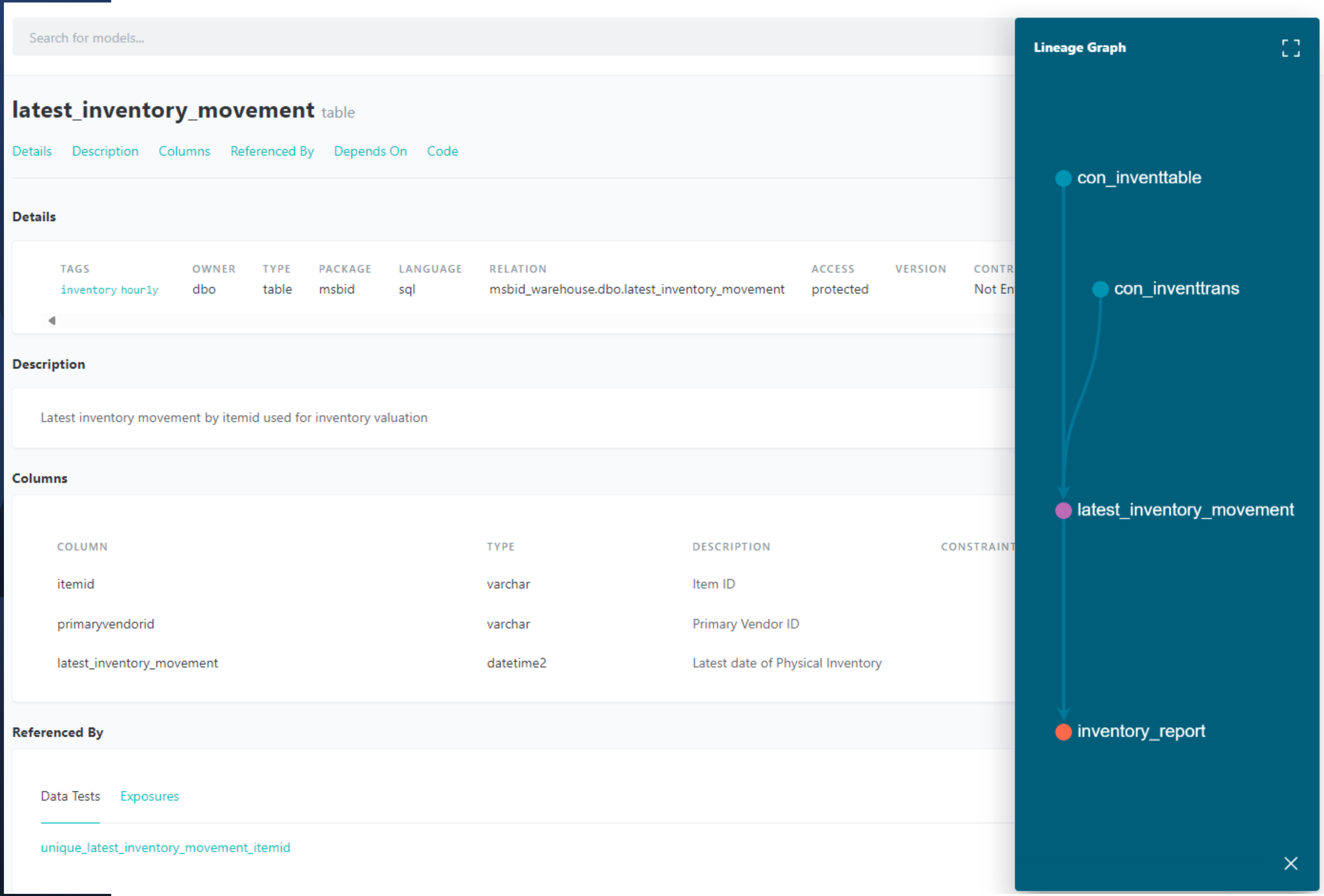
Tests compared to Contracts

	Contracts	Tests
What?	Data shape	Data quality
When?	During materialization.	After materialization.
How often?	On every materialization	As specified
Support?	Constraints depend on target platform	Works on every platform (if valid SQL)

Documentation

dbt docs generates a html page, containing all information and dependencies present in project

! dbt docs does not contain any data from your data warehouse.



Search for models...

latest_inventory_movement table

Details Description Columns Referenced By Depends On Code

Details

TAGS	OWNER	TYPE	PACKAGE	LANGUAGE	RELATION	ACCESS	VERSION	CONTR
`inventory` `hourly`	dbo	table	msbid	sql	msbid_warehouse.dbo.latest_inventory_movement	protected		Not En

Description

Latest inventory movement by itemid used for inventory valuation

Columns

COLUMN	TYPE	DESCRIPTION	CONSTRAINT
itemid	varchar	Item ID	
primaryvendorid	varchar	Primary Vendor ID	
latest_inventory_movement	datetime2	Latest date of Physical Inventory	

Referenced By

Data Tests Exposures

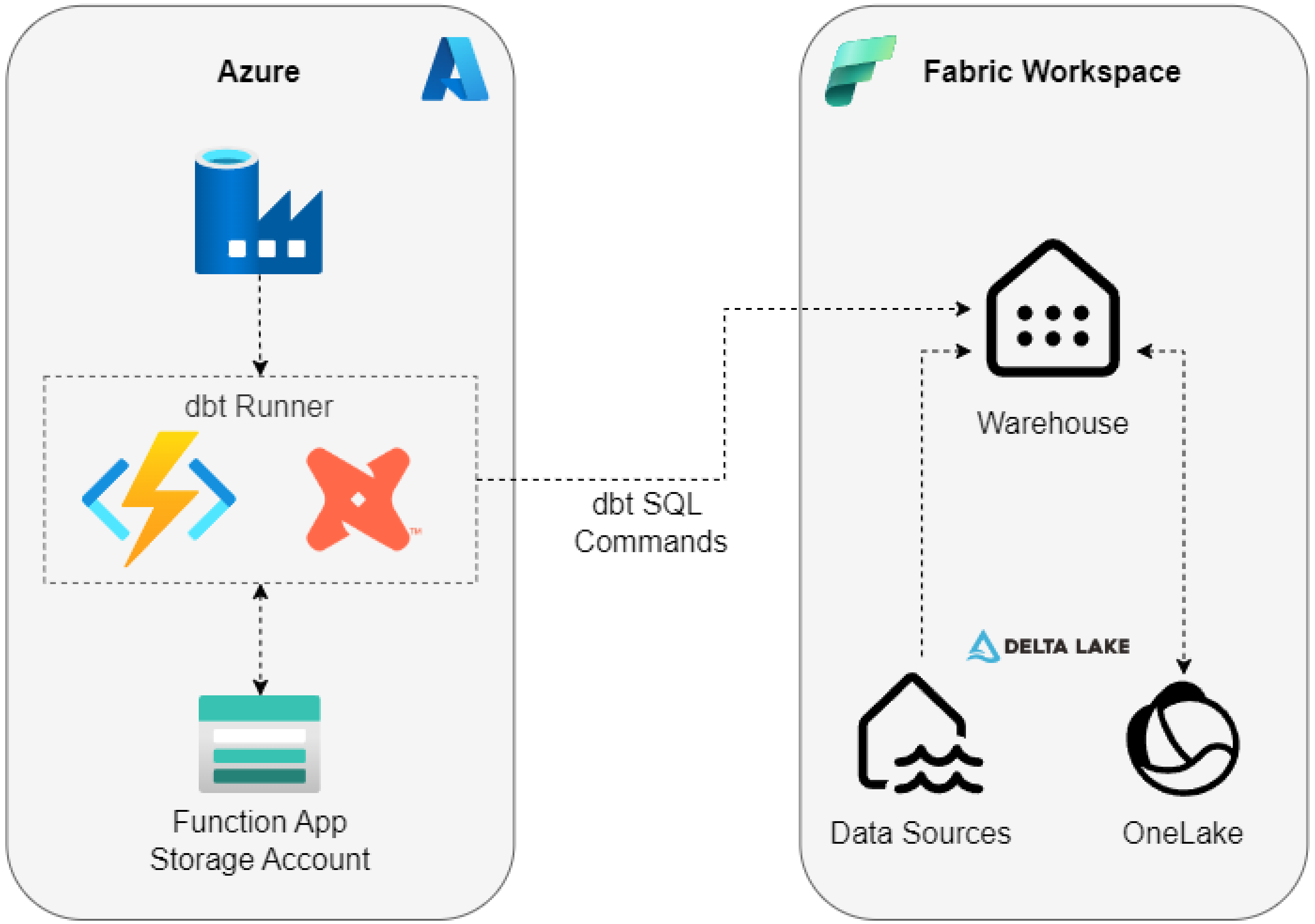
unique_latest_inventory_movement_itemid

Lineage Graph

Deploying dbt to Fabric

- There is no Fabric native way to run dbt
- We need python environment
- LINAK Requirement: Run needs to be scheduled by Azure Data Factory

Deploying to Azure Function

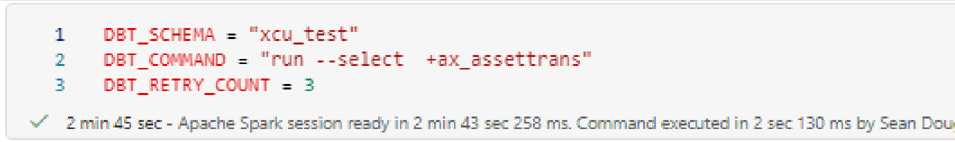


- My colleague Allan created a docker container with dbt installed
- Dbt runs can be started using API
- Solution is cheap, reliable and fast
- Requires knowledge on Azure Functions and docker containers

https://www.linkedin.com/pulse/deploy-dbt-core-workloads-azure-using-durable-allan-rasmussen-6oflf/

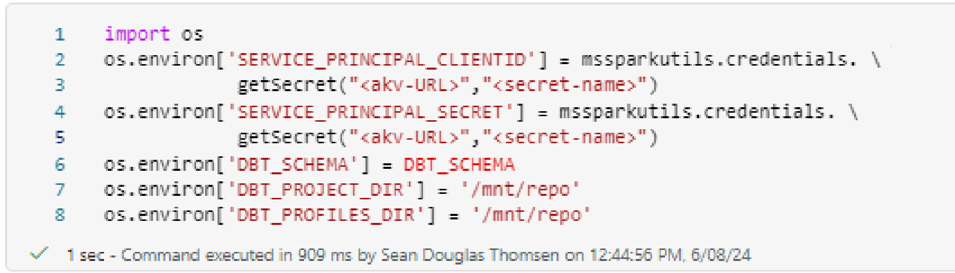
Deploying to Fabric Notebook

- pip install dbt-fabric (or use an environment)
- az copy or git clone to bring in project files
- Use dbt programmatic interface
- Simple, but notebook execution can only be triggered inside Fabric due to API limitations



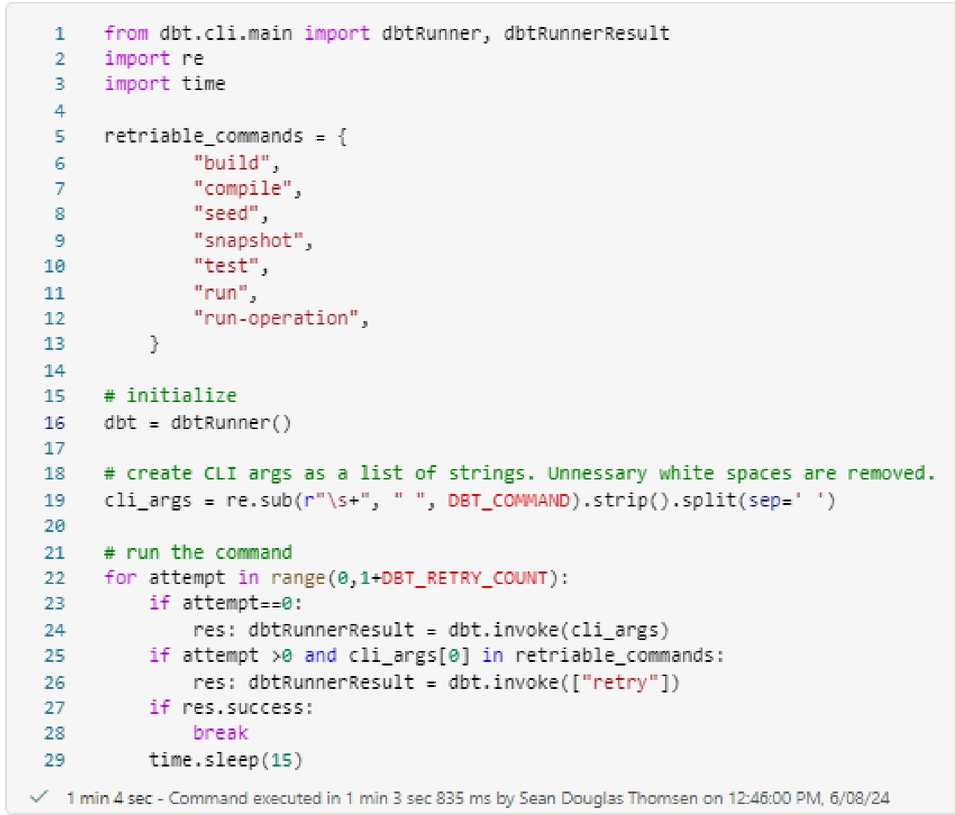
```
DBT_SCHEMA = "xcu_test"
DBT_COMMAND = "run --select  +ax_assettrans"
DBT_RETRY_COUNT = 3
```

✓ 2 min 45 sec - Apache Spark session ready in 2 min 43 sec 258 ms. Command executed in 2 sec 130 ms by Sean Dou



```
import os
os.environ['SERVICE_PRINCIPAL_CLIENTID'] = mssparkutils.credentials. \
            getSecret("<akv-URL>","<secret-name>")
os.environ['SERVICE_PRINCIPAL_SECRET'] = mssparkutils.credentials. \
            getSecret("<akv-URL>","<secret-name>")
os.environ['DBT_SCHEMA'] = DBT_SCHEMA
os.environ['DBT_PROJECT_DIR'] = '/mnt/repo'
os.environ['DBT_PROFILES_DIR'] = '/mnt/repo'
```

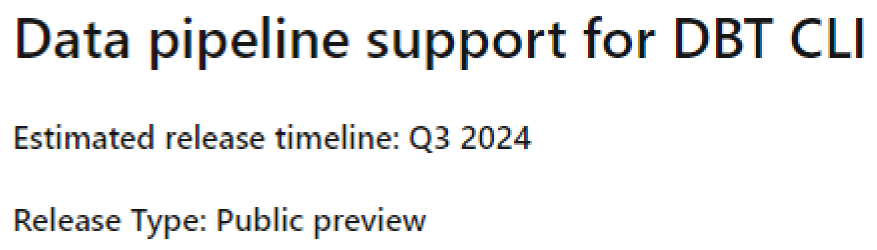
✓ 1 sec - Command executed in 909 ms by Sean Douglas Thomsen on 12:44:56 PM, 6/08/24



```
from dbt.cli.main import dbtRunner, dbtRunnerResult
import re
import time

retriable_commands = {
        "build",
        "compile",
        "seed",
        "snapshot",
        "test",
        "run",
        "run-operation",
    }

# initialize
dbt = dbtRunner()

# create CLI args as a list of strings. Unnessary white spaces are removed.
cli_args = re.sub(r"\s+", " ", DBT_COMMAND).strip().split(sep=' ')

# run the command
for attempt in range(0,1+DBT_RETRY_COUNT):
    if attempt==0:
        res: dbtRunnerResult = dbt.invoke(cli_args)
    if attempt >0 and cli_args[0] in retriable_commands:
        res: dbtRunnerResult = dbt.invoke(["retry"])
    if res.success:
        break
    time.sleep(15)
```

✓ 1 min 4 sec - Command executed in 1 min 3 sec 835 ms by Sean Douglas Thomsen on 12:46:00 PM, 6/08/24

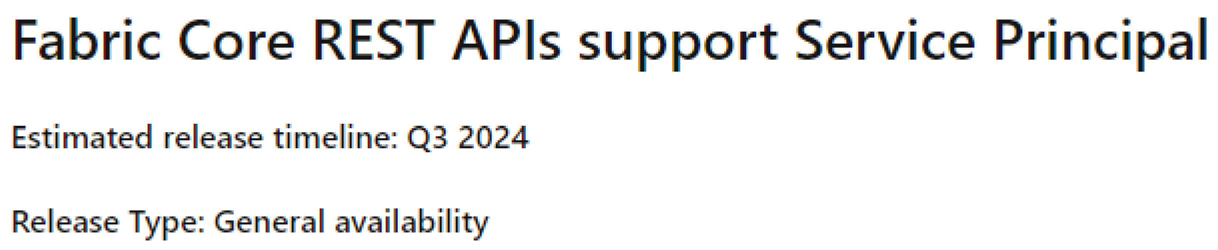
New Announcements



Data pipeline support for DBT CLI

Estimated release timeline: Q3 2024

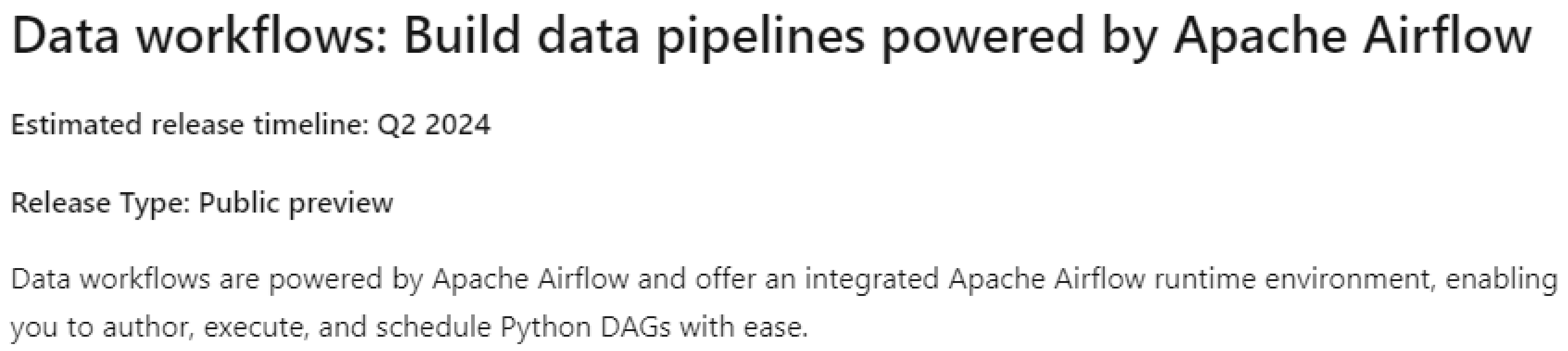
Release Type: Public preview



Fabric Core REST APIs support Service Principal

Estimated release timeline: Q3 2024

Release Type: General availability



Data workflows: Build data pipelines powered by Apache Airflow

Estimated release timeline: Q2 2024

Release Type: Public preview

Data workflows are powered by Apache Airflow and offer an integrated Apache Airflow runtime environment, enabling you to author, execute, and schedule Python DAGs with ease.

dbt is much more

- dbt seeds – Define manual data in your projects
- Logging – have dbt log run executions to your warehouse
- Snapshots – use dbt to snapshot your source data or create SCD
- Packages – include open-source packages in your project
- Hooks
- Grant – use dbt to manage access to your models
- Incremental models
- Slim ci

…

Key Take-aways

- ☺ dbt is in essence a templating engine
- ☺ Well suited for teams that prefer SQL
- ☺ Encourages reusing code, by using templates and macros
- ☺ Less boiler plate code, more focus on transformations
- ☺ Provides a data quality testing framework
- ☺ Can enforce data contracts
- ☺ Makes documentation a part of development workflow
- ☹ Deployment to Fabric could be easier
- ☹ Many popular packages do not support dbt-fabric yet

Thank you for your time!