# End-to-End, Runnable Tutorial: BEAM → dbt (Semantic Layer + MetricFlow)

This tutorial is **fully executable**. It gives you schemas, seed data, macros, and exact commands to go from zero → a working dbt Semantic Layer with measures and metrics generated from **BEAM contracts** and a **KPI library**.

Assumptions - dbt Core installed; adapter configured (Snowflake/BigQuery/Redshift/Postgres).  
- You have a working profiles.yml target and a database/schema you can write to.  
- Project root named beam\_semantic\_demo/ (adjust paths if different).

## 0) Project Scaffolding

**File tree (you will create these):**

beam\_semantic\_demo/  
 dbt\_project.yml  
 models/  
 staging/  
 stg\_patient.sql  
 stg\_provider.sql  
 stg\_encounter.sql  
 stg\_claim.sql  
 staging.yml  
 semantic/  
 semantic\_models.yml # generated later via macro output (Step 6)  
 metrics.yml # generated later via macro output (Step 7)  
 macros/  
 validate\_beam\_contracts.sql  
 generate\_measures\_yaml.sql  
 generate\_metrics\_yaml.sql  
 data/  
 patient.csv  
 provider.csv  
 encounter.csv  
 claim.csv  
 beam\_contracts.csv  
 kpi\_library.csv

**dbt\_project.yml (minimal):**

name: beam\_semantic\_demo  
version: 1.0.0  
config-version: 2  
profile: beam\_semantic\_demo  
model-paths: ["models"]  
seed-paths: ["data"]  
macro-paths: ["macros"]  
models:  
 +materialized: view  
seeds:  
 +quote\_columns: false

Configure profiles.yml per your adapter (not shown here). Ensure your target schema/database exist.

## 1) EDM (UDM Healthcare) — Seeds as Source of Truth

Create **seed CSVs** for a minimal UDM-style subset. Place in data/.

**data/patient.csv**

patient\_id,birth\_date,gender  
P001,1980-06-01,F  
P002,1975-09-14,M  
P003,1990-01-22,F

**data/provider.csv**

provider\_id,specialty,organization\_id  
V001,Internal Medicine,O10  
V002,Cardiology,O11

**data/encounter.csv**

encounter\_id,patient\_id,provider\_id,encounter\_start\_timestamp,encounter\_type  
E100,P001,V001,2025-01-10 09:15:00,office\_visit  
E101,P002,V002,2025-01-11 10:30:00,consult  
E102,P001,V001,2025-01-12 14:05:00,office\_visit

**data/claim.csv**

claim\_id,encounter\_id,billed\_amount,paid\_amount,claim\_status,service\_date  
C900,E100,250.00,200.00,paid,2025-01-10  
C901,E101,500.00,0.00,denied,2025-01-11  
C902,E102,150.00,120.00,paid,2025-01-12

Run:

dbt seed --select patient provider encounter claim

This creates four tables (or equivalent) in your target schema.

## 2) Staging Models (Keys, Types, Basic Tests)

**models/staging/stg\_patient.sql**

select  
 cast(patient\_id as varchar) as patient\_id,  
 cast(birth\_date as date) as birth\_date,  
 cast(gender as varchar) as gender  
from {{ ref('patient') }}

**models/staging/stg\_provider.sql**

select  
 cast(provider\_id as varchar) as provider\_id,  
 cast(specialty as varchar) as specialty,  
 cast(organization\_id as varchar) as organization\_id  
from {{ ref('provider') }}

**models/staging/stg\_encounter.sql**

select  
 cast(encounter\_id as varchar) as encounter\_id,  
 cast(patient\_id as varchar) as patient\_id,  
 cast(provider\_id as varchar) as provider\_id,  
 cast(encounter\_start\_timestamp as timestamp) as encounter\_start\_timestamp,  
 cast(encounter\_type as varchar) as encounter\_type  
from {{ ref('encounter') }}

**models/staging/stg\_claim.sql**

select  
 cast(claim\_id as varchar) as claim\_id,  
 cast(encounter\_id as varchar) as encounter\_id,  
 cast(billed\_amount as numeric) as billed\_amount,  
 cast(paid\_amount as numeric) as paid\_amount,  
 cast(claim\_status as varchar) as claim\_status,  
 cast(service\_date as date) as service\_date  
from {{ ref('claim') }}

**models/staging/staging.yml** (tests)

version: 2  
models:  
 - name: stg\_patient  
 tests:  
 - not\_null:  
 column\_name: patient\_id  
 - unique:  
 column\_name: patient\_id  
  
 - name: stg\_provider  
 tests:  
 - unique:  
 column\_name: provider\_id  
  
 - name: stg\_encounter  
 tests:  
 - not\_null:  
 column\_name: encounter\_id  
 - unique:  
 column\_name: encounter\_id  
  
 - name: stg\_claim  
 tests:  
 - not\_null:  
 column\_name: claim\_id  
 - unique:  
 column\_name: claim\_id  
 - relationships:  
 to: ref('stg\_encounter')  
 field: encounter\_id  
 target\_field: encounter\_id

Run:

dbt build --select staging

## 3) Extended BEAM Contracts (with Contract Fields)

Create the BEAM contracts seed (drives **measures**). Place in data/.

**data/beam\_contracts.csv**

event\_name,physical\_edm\_table,physical\_edm\_column,aggregation\_type,primary\_key\_column,time\_column,time\_grains  
Claim Paid,stg\_claim,paid\_amount,sum,claim\_id,service\_date,day|month|year  
Claim Billed,stg\_claim,billed\_amount,sum,claim\_id,service\_date,day|month|year  
Denied Claim Count,stg\_claim,claim\_id,count,claim\_id,service\_date,month|year

Run:

dbt seed --select beam\_contracts

## 4) KPI Library (drives **metrics**)

**data/kpi\_library.csv**

metric\_name,description,numerator,denominator,metric\_type  
claims\_denial\_rate,% of claims denied,denied\_claims,total\_claims,ratio  
avg\_paid\_amount,Average paid amount over period,total\_paid\_amount,,simple

denied\_claims, total\_claims, total\_paid\_amount will be resolved from the **measures** emitted from BEAM contracts.

Run:

dbt seed --select kpi\_library

## 5) Info Schema Validation Macro

**macros/validate\_beam\_contracts.sql**

{% macro validate\_beam\_contracts() %}  
 {# Generic validation using adapter-aware information schema #}  
 {% set sql %}  
 select b.event\_name, b.physical\_edm\_table, b.physical\_edm\_column  
 from {{ ref('beam\_contracts') }} b  
 left join (  
 {% if target.type in ['postgres','redshift'] %}  
 select table\_name, column\_name  
 from information\_schema.columns  
 where table\_schema = '{{ target.schema }}'  
 {% elif target.type == 'snowflake' %}  
 select table\_name, column\_name  
 from information\_schema.columns  
 where table\_schema = '{{ target.schema | upper }}'  
 {% elif target.type == 'bigquery' %}  
 select table\_name, column\_name  
 from `{{ target.project }}.{{ target.dataset }}.INFORMATION\_SCHEMA.COLUMNS`  
 {% else %}  
 select table\_name, column\_name from information\_schema.columns  
 {% endif %}  
 ) i  
 on i.table\_name = upper(split\_part(b.physical\_edm\_table, '.', -1))  
 and i.column\_name = upper(b.physical\_edm\_column)  
 where i.column\_name is null  
 {% endset %}  
 {% set res = run\_query(sql) %}  
 {% if res and res.columns[0].values()|length > 0 %}  
 {{ log('Missing columns found in BEAM contracts:', true) }}  
 {{ log(res, true) }}  
 {% do exceptions.raise\_compiler\_error('Validation failed: some BEAM mappings do not exist in info schema.') %}  
 {% else %}  
 {{ log('BEAM contract validation passed.', true) }}  
 {% endif %}  
{% endmacro %}

Run:

dbt run-operation validate\_beam\_contracts

## 6) Measures Generator (dbt + Jinja → YAML to stdout)

**macros/generate\_measures\_yaml.sql**

{% macro generate\_measures\_yaml() %}  
 {% set sql %}  
 select event\_name,  
 physical\_edm\_table,  
 physical\_edm\_column,  
 aggregation\_type,  
 primary\_key\_column,  
 time\_column,  
 time\_grains  
 from {{ ref('beam\_contracts') }}  
 {% endset %}  
 {% set res = run\_query(sql) %}  
 {% if not res %}  
 {% do exceptions.raise\_compiler\_error('No BEAM contracts found.') %}  
 {% endif %}  
  
 version: 2  
 semantic\_models:  
 {% for i in range(res.columns[0].values()|length) %}  
 {% set event\_name = res.columns[0].values()[i] %}  
 {% set model\_name = res.columns[1].values()[i] %}  
 {% set expr\_col = res.columns[2].values()[i] %}  
 {% set agg = res.columns[3].values()[i] %}  
 {% set pk\_col = res.columns[4].values()[i] %}  
 {% set time\_col = res.columns[5].values()[i] %}  
 {% set grains\_str = res.columns[6].values()[i] %}  
 {% set grains = grains\_str.split('|') if grains\_str else ['day','month','year'] %}  
  
 - name: {{ event\_name | lower | replace(' ', '\_') }}  
 model: ref('{{ model\_name }}')  
 entities:  
 - name: {{ event\_name | lower | replace(' ', '\_') }}  
 type: primary  
 expr: {{ pk\_col }}  
 dimensions:  
 - name: {{ time\_col }}  
 type: time  
 expr: {{ time\_col }}  
 type\_params:  
 time\_granularity: [{{ grains | map('trim') | map('lower') | join(', ') }}]  
 measures:  
 - name: {{  
 ( 'total\_' ~ expr\_col ) if agg in ['sum','avg','min','max'] else  
 ( 'count\_' ~ expr\_col )  
 | lower | replace(' ', '\_') }}  
 agg: {{ agg }}  
 expr: {{ expr\_col }}  
 {% endfor %}  
{% endmacro %}

Generate YAML into a file (redirect stdout):

mkdir -p models/semantic  
# write semantic\_models.yml  
{ dbt run-operation generate\_measures\_yaml ; } > models/semantic/semantic\_models.yml

## 7) Metrics Generator (dbt + Jinja → YAML to stdout)

**macros/generate\_metrics\_yaml.sql**

{% macro generate\_metrics\_yaml() %}  
 {% set sql %}  
 select metric\_name, description, numerator, denominator, metric\_type  
 from {{ ref('kpi\_library') }}  
 {% endset %}  
 {% set res = run\_query(sql) %}  
 {% if not res %}  
 {% do exceptions.raise\_compiler\_error('No KPI library rows found.') %}  
 {% endif %}  
  
 metrics:  
 {% for i in range(res.columns[0].values()|length) %}  
 {% set name = res.columns[0].values()[i] | lower | replace(' ', '\_') %}  
 {% set desc = res.columns[1].values()[i] %}  
 {% set num = res.columns[2].values()[i] | lower | replace(' ', '\_') %}  
 {% set den = res.columns[3].values()[i] | lower | replace(' ', '\_') %}  
 {% set mtyp = (res.columns[4].values()[i] or 'simple') %}  
  
 - name: {{ name }}  
 label: {{ name | replace('\_',' ') | title }}  
 description: {{ desc }}  
 {% if mtyp == 'ratio' %}  
 type: ratio  
 type\_params:  
 numerator: {{ num }}  
 denominator: {{ den }}  
 {% else %}  
 type: simple  
 type\_params:  
 measure: {{ num }}  
 {% endif %}  
 {% endfor %}  
{% endmacro %}

Generate YAML into a file:

# write metrics.yml  
{ dbt run-operation generate\_metrics\_yaml ; } > models/semantic/metrics.yml

## 8) Build & Validate

Run the whole flow:

# 1) Load seeds  
dbt seed  
  
# 2) Build staging + tests  
dbt build --select staging  
  
# 3) Validate BEAM contracts vs information\_schema  
dbt run-operation validate\_beam\_contracts  
  
# 4) Generate semantic YAML (measures + metrics)  
{ dbt run-operation generate\_measures\_yaml ; } > models/semantic/semantic\_models.yml  
{ dbt run-operation generate\_metrics\_yaml ; } > models/semantic/metrics.yml  
  
# 5) (Re-)Parse project so dbt sees new YAML, then compile/build  
# Depending on your environment, re-run build to ensure parsing picks up YAML  
dbt build --select semantic

**Sanity check (example MetricFlow usage):** - Confirm that total\_paid\_amount, total\_billed\_amount, count\_claim\_id measures exist under the generated semantic model. - Confirm that claims\_denial\_rate and avg\_paid\_amount appear under metrics.

## 9) Troubleshooting

* **Missing columns in validation:** Fix physical\_edm\_table/column in beam\_contracts.csv and re-seed.
* **No metrics generated:** Ensure kpi\_library.csv uses measure names that the generator produces (e.g., total\_paid\_amount).
* **Adapter differences:** The information\_schema query block includes adapter-specific branches; adjust for your warehouse if needed.
* **Parsing new YAML:** After generating models/semantic/\*.yml, re-run dbt build so dbt parses the new files.

## 10) What You Now Have

* A minimal **EDM subset** (patient, provider, encounter, claim) with real seed data.
* **Staging models** with basic tests.
* A **BEAM contracts seed** using the three contract fields (+ keys/time config).
* A **KPI library seed** with a ratio and simple metric.
* **Macros** that generate **semantic\_models.yml** and **metrics.yml** from the seeds.
* A repeatable process to refresh and extend metrics by updating metadata—not code.

Extend by adding rows to beam\_contracts.csv and kpi\_library.csv, then re-run Steps 8.4–8.5.