

June 5-8, 2023 · Data platforms · Data engineering · Data management

BUDAPEST DATA FORUM



DBT and data model based design & development

Attila Berecz

DW Consultant
Meta Consulting

Danubius Hotel Helia + Streaming

budapestdata.hu

AGENDA

- **About us**
- **Data modeling – DV overview**
- **dbt overview**
- **Demo**
- **Q&A**

Meta Consulting Ltd.

- Established in 2002, privately owned company

Services

- DW & BI, Metadata management, Data governance
- System design, Development, Consulting, Training

Customers

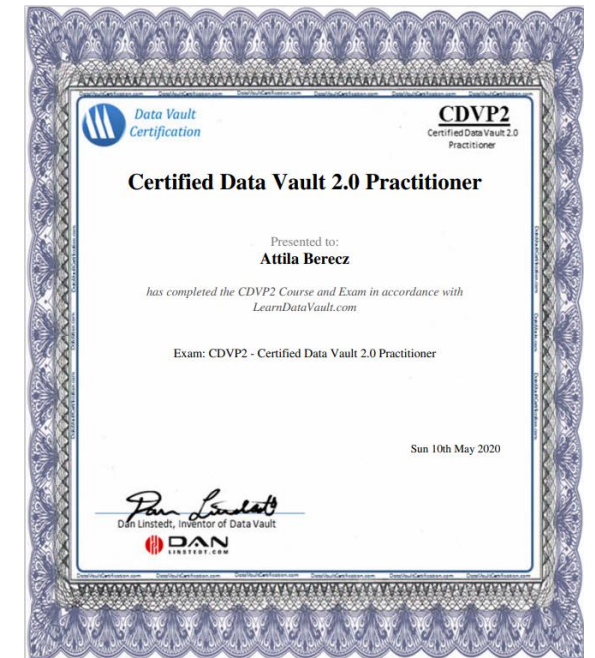
- Bank, Insurance, FMCG, Telecom, Media, Education, Government etc.
- Mostly in middle Europe (HU, DE, CZ, PL, SK, RO, MNE) + UK/USA
- 60+ customers, 12 countries, 130+ projects

Hungarian partner of Snowflake, WhereScape & Fivetran

Presented by...

Attila Berecz

- Working with DWH/BI since 2014
- Consulting, review, design & development
- Certified Data Vault 2.0 Practitioner
- Industries: Bank, Insurance, FMCG, Retail, ...
- Technologies: SQL, dbt, C#, java, WhereScape ...
- Databases: Snowflake, Oracle, SQL Server etc.



Design & Development

Source data reverse engineering

Profile and understand the data

Integrate sources and build a common data model
(preferably using Data Vault)

Automate the development based on the data model

Data modeling

Definition of data elements and their connections between them (Visual representation aka Diagram)

Conceptual/Logical/Physical type of data model

Helps to understand data better and the final picture

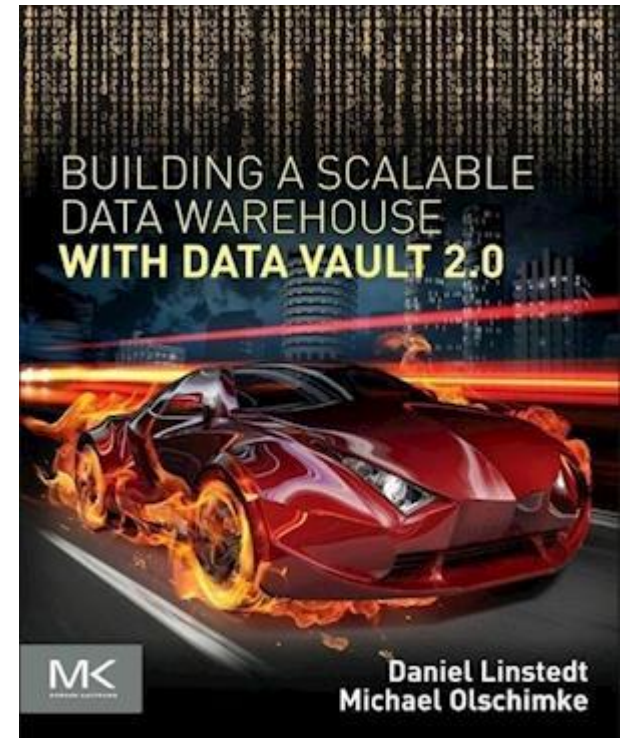
Data warehouse design methods

- Dimensional (star or snowflake schema)
- Data vault

Data Vault 2.0

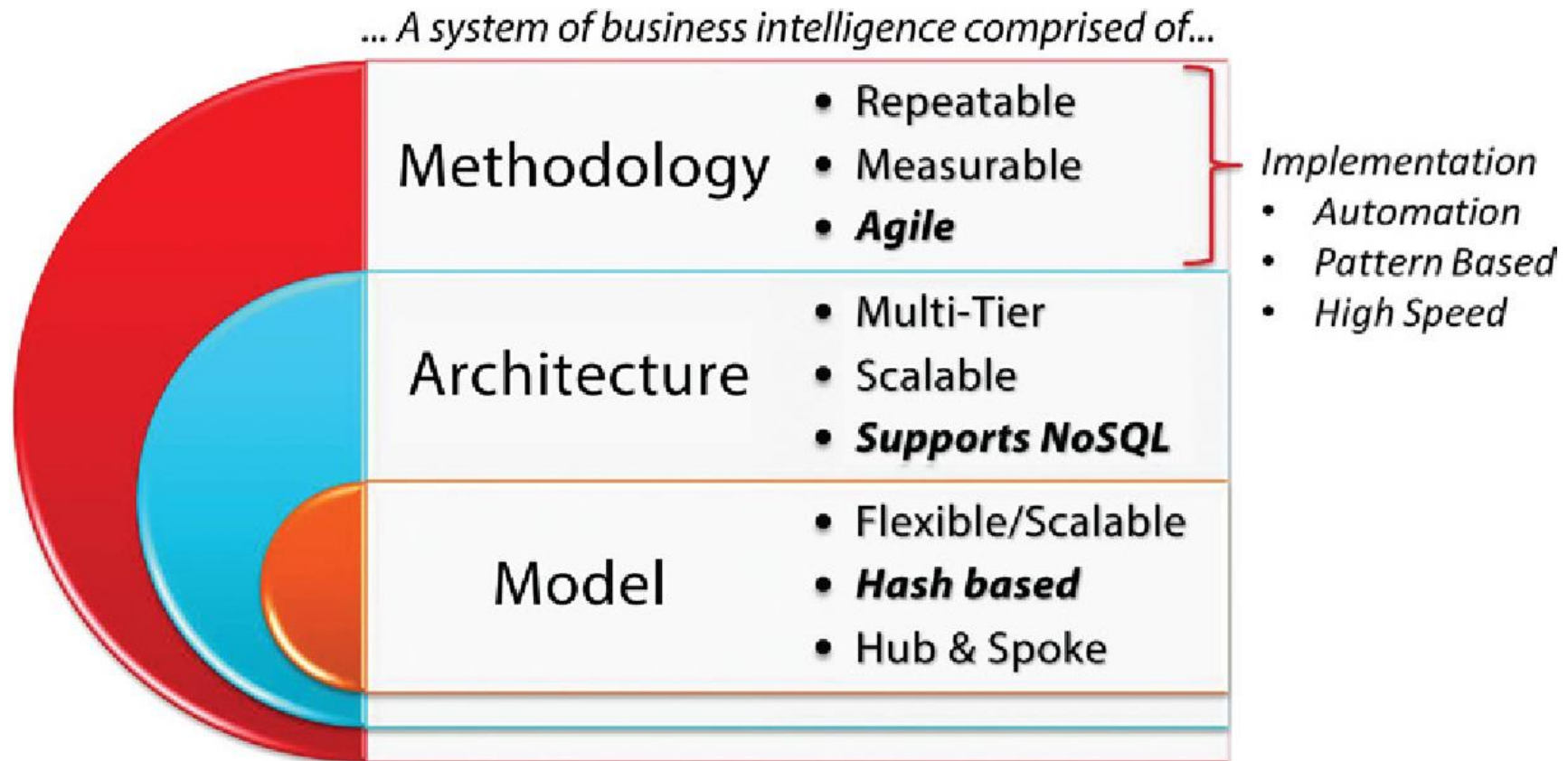
„Data Vault 2.0 is a system of Business Intelligence containing the necessary components needed to accomplish enterprise vision in Data Warehousing and Information Delivery”

Dan Linstedt, creator of the Data Vault method



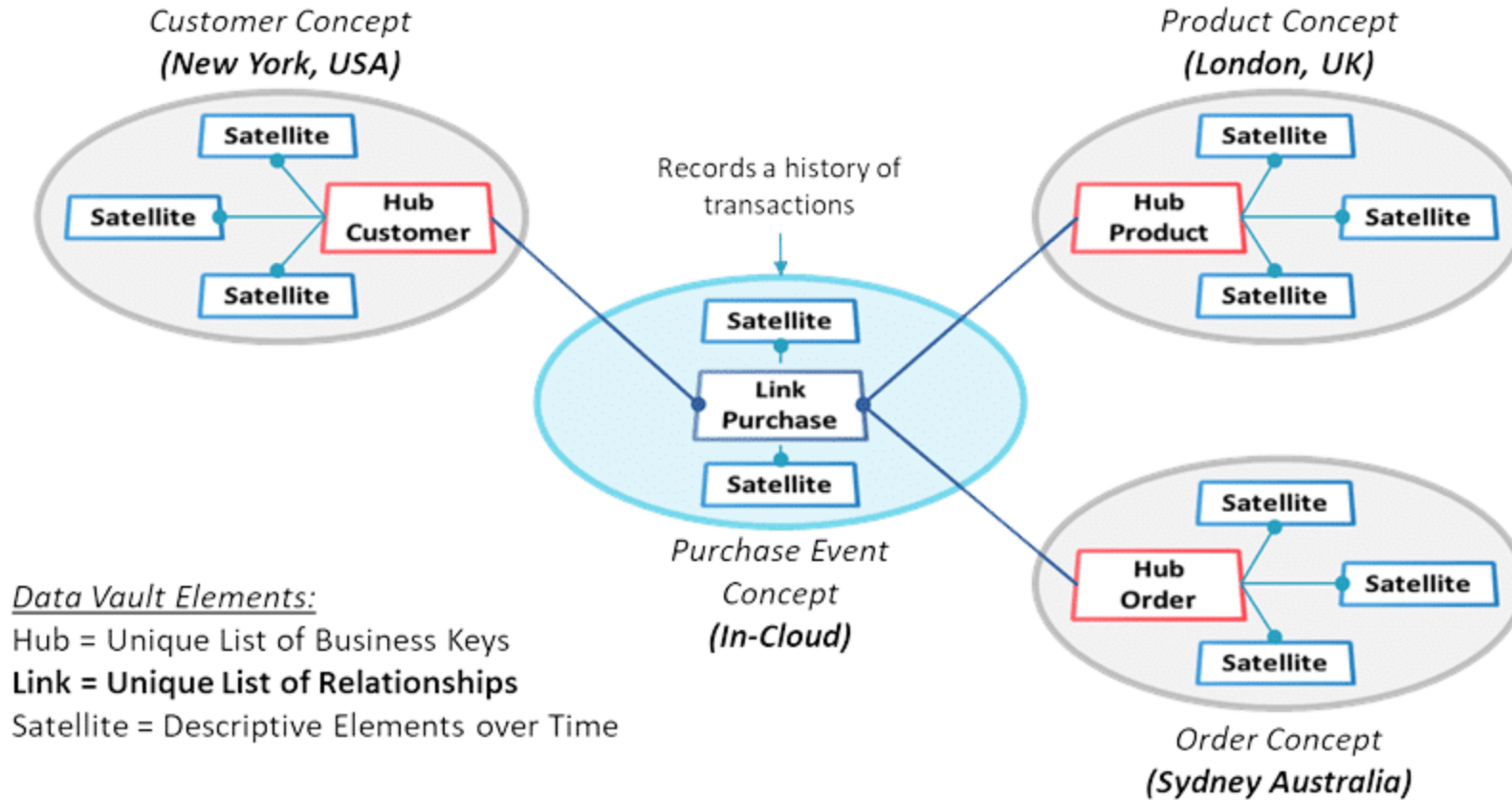
[Building a Scalable Data Warehouse with Data Vault 2.0](#)

Data Vault 2.0



Source: Data architecture: A primer for the data Scientist

Key Data Vault 2.0 elements



Data Vault Elements:

Hub = Unique List of Business Keys

Link = Unique List of Relationships

Satellite = Descriptive Elements over Time

Source: datavaultalliance.com

- SQL-first transformation tool also support template driven development
- Following software engineering best practices
- Built-in dependency handling and lineage
- Wide list of dbt resources (dbt_utils, data vault packages, etc)
- dbt Core & dbt Cloud

dbt - Data Vault packages



AutomateDV

- support standard DV table types
- dbtvault-generator



Datavault4dbt & Turbovault4dbt

- allowance of multiple deltas
- Meta data inputs
 - Snowflake
 - BigQuery
 - Google Sheets
 - Excel

Demo overview

- Source data reverse engineering (Snowflake TPCH_SF1 sample database)
- Create (design) initial data model
- Generate & review generated Data Vault model (tables, references, mappings w/ lineage)
- Generate dbt files (yml, sql)
- Compile and run dbt models
- Check the results (dbt & snowflake database)
- Docs generation (dbt docs w/ lineage)

Demo

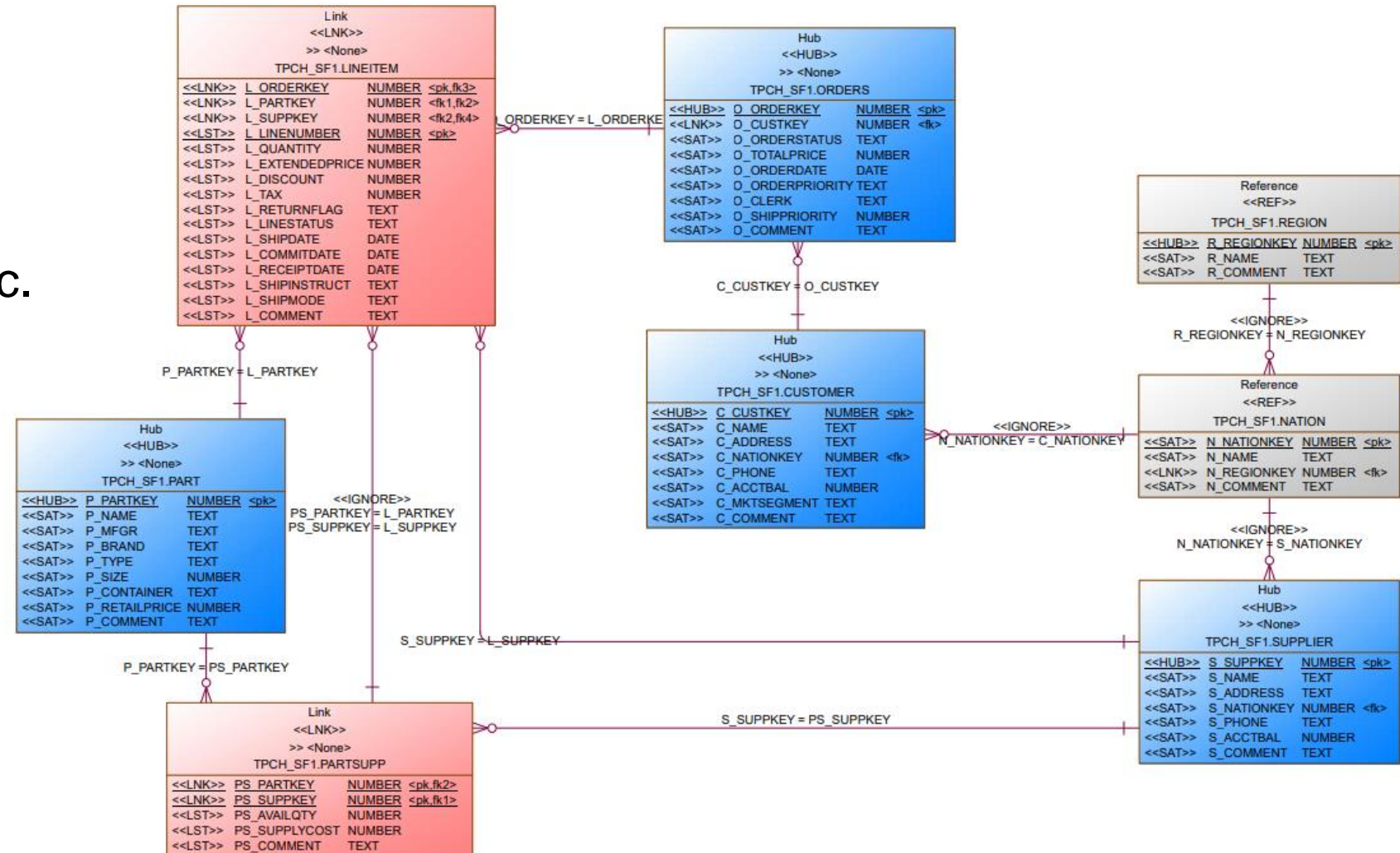
Demo: design data model

Source data model

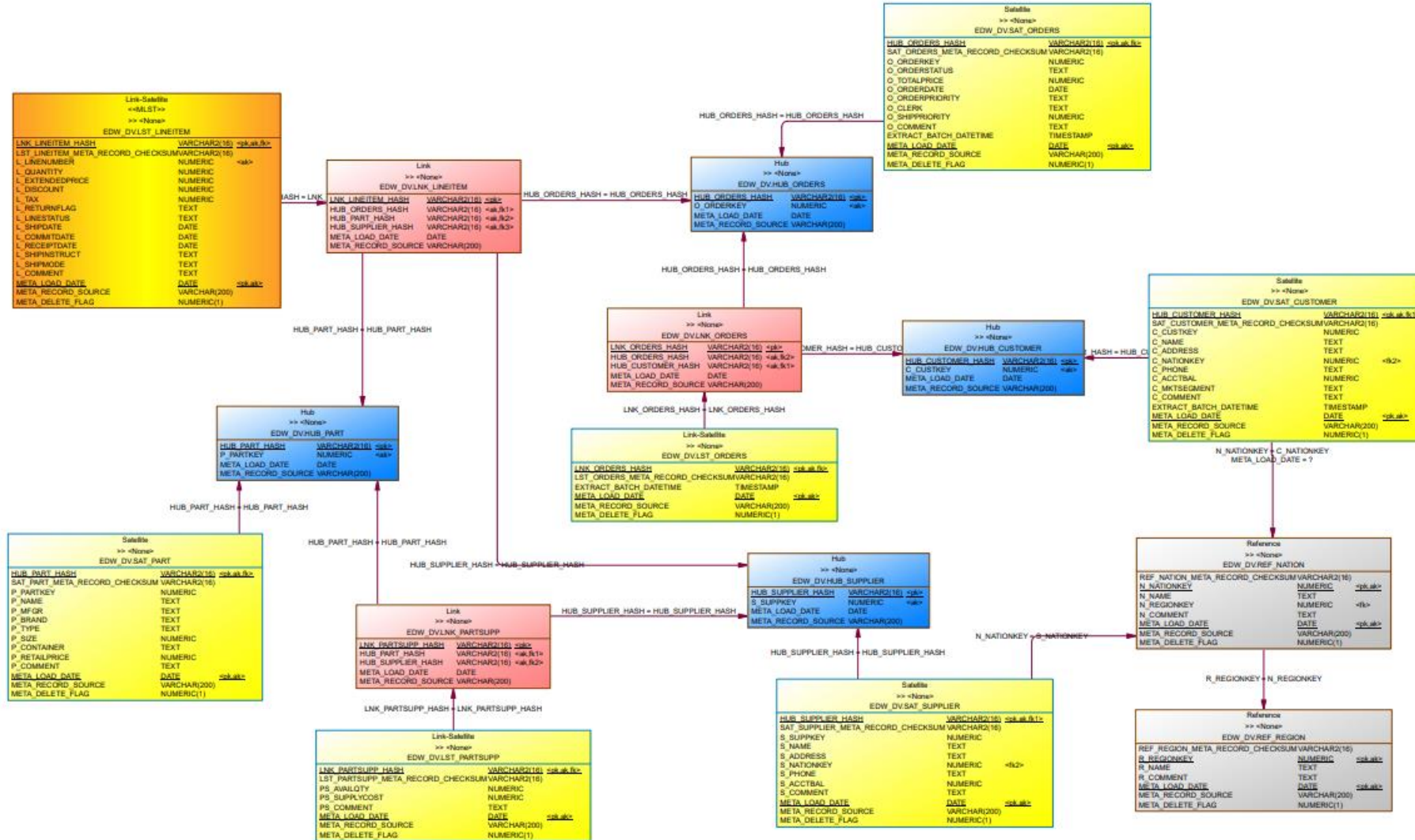
- Reverse engineered

Design model

- Structure, references etc. set-up by the designer
- Colorcoding automated



Demo: data vault model (generated)



Demo: data mapping (generated)

Mapping Editor

Table Mappings - HUB_CUSTOMER (HUB_CUSTOMER)

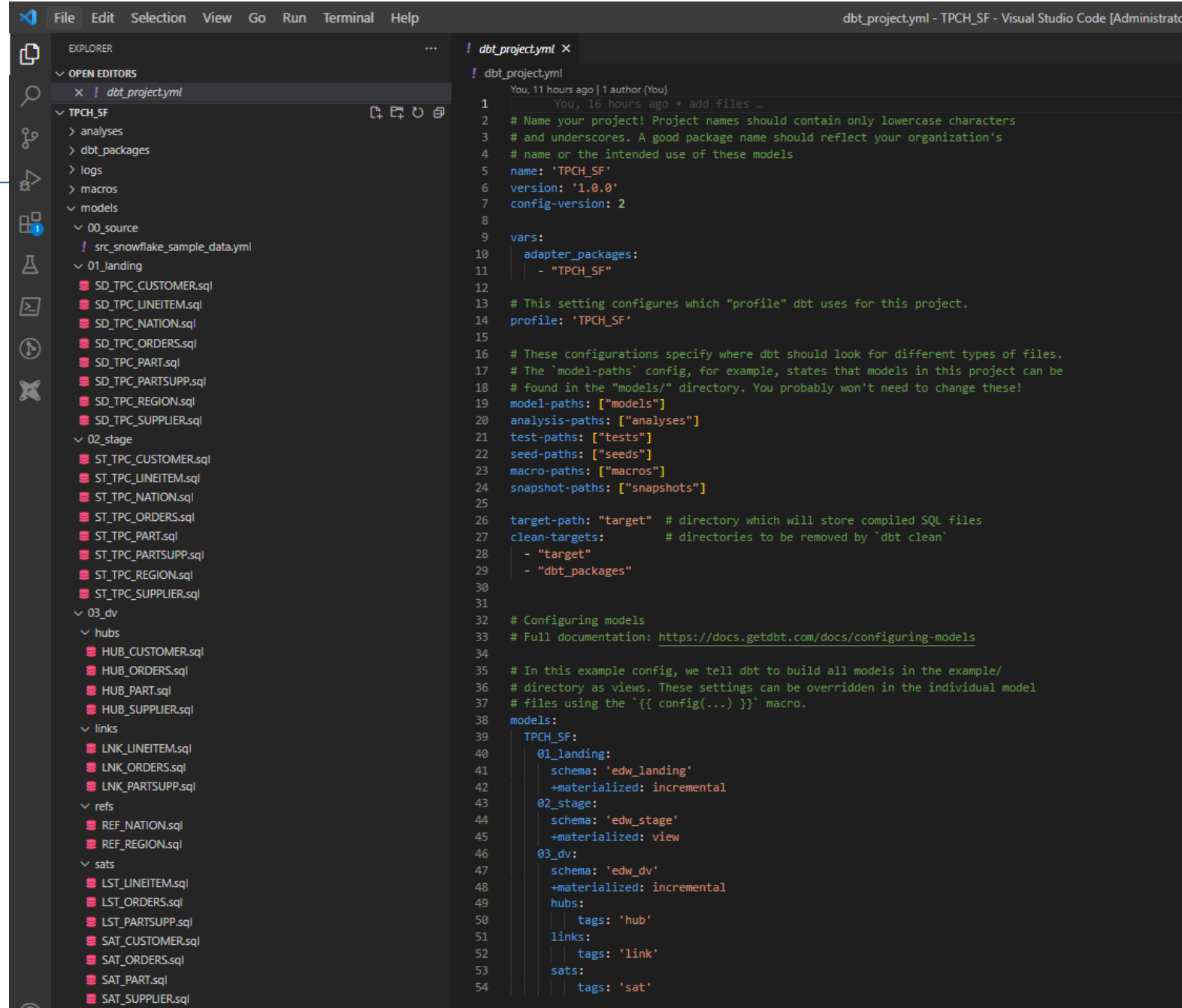
Mapping: DV.HUB_CUSTOMER.HDWCUSTO_GTSCUSTO

	Column	Mapped to	Comment
→	C_CUSTKEY	EDW_STAGE.ST_TPC_CUSTOMER.C_CUSTKEY	
2	HUB_CUSTOMER_HASH	EDW_STAGE.ST_TPC_CUSTOMER.HUB_CUSTOMER_HASH	-- hash calculated in STAGE table; but only after all DV tables generated -- AddM
3	META_LOAD_DATE	EDW_STAGE.ST_TPC_CUSTOMER.META_LOAD_DATE	
4	META_RECORD_SOURCE	EDW_STAGE.ST_TPC_CUSTOMER.META_RECORD_SOURCE	

Demo: dbt files (generated)

Generated dbt files

- Folders:
in sync with the db layers
- Model files:
.sql with proper SQL
using the designed mappings



The screenshot shows the Visual Studio Code interface with a dbt project. The Explorer view on the left displays the project structure:

- TPCH_SF
 - analyses
 - dbt_packages
 - logs
 - macros
 - models
 - 00_source
 - src_snowflake_sample_data.yml
 - 01_landing
 - SD_TPC_CUSTOMER.sql
 - SD_TPC_LINEITEM.sql
 - SD_TPC_NATION.sql
 - SD_TPC_ORDERS.sql
 - SD_TPC_PART.sql
 - SD_TPC_PARTSUPP.sql
 - SD_TPC_REGION.sql
 - SD_TPC_SUPPLIER.sql
 - 02_stage
 - ST_TPC_CUSTOMER.sql
 - ST_TPC_LINEITEM.sql
 - ST_TPC_NATION.sql
 - ST_TPC_ORDERS.sql
 - ST_TPC_PART.sql
 - ST_TPC_PARTSUPP.sql
 - ST_TPC_REGION.sql
 - ST_TPC_SUPPLIER.sql
 - 03_dv
 - hubs
 - HUB_CUSTOMER.sql
 - HUB_ORDERS.sql
 - HUB_PART.sql
 - HUB_SUPPLIER.sql
 - links
 - LNK_LINEITEM.sql
 - LNK_ORDERS.sql
 - LNK_PARTSUPP.sql
 - refs
 - REF_NATION.sql
 - REF_REGION.sql
 - sats
 - LST_LINEITEM.sql
 - LST_ORDERS.sql
 - LST_PARTSUPP.sql
 - SAT_CUSTOMER.sql
 - SAT_ORDERS.sql
 - SAT_PART.sql
 - SAT_SUPPLIER.sql

The dbt_project.yml file is open in the editor, showing the following configuration:

```
1 You, 11 hours ago | 1 author (You)
2 You, 16 hours ago * add files ...
3 # Name your project! Project names should contain only lowercase characters
4 # and underscores. A good package name should reflect your organization's
5 # name or the intended use of these models
6 name: 'TPCH_SF'
7 version: '1.0.0'
8 config-version: 2
9
10 vars:
11   adapter_packages:
12     - "TPCH_SF"
13
14 # This setting configures which "profile" dbt uses for this project.
15 profile: 'TPCH_SF'
16
17 # These configurations specify where dbt should look for different types of files.
18 # The `model-paths` config, for example, states that models in this project can be
19 # found in the "models/" directory. You probably won't need to change these!
20 model-paths: ["models"]
21 analysis-paths: ["analyses"]
22 test-paths: ["tests"]
23 seed-paths: ["seeds"]
24 macro-paths: ["macros"]
25 snapshot-paths: ["snapshots"]
26
27 target-path: "target" # directory which will store compiled SQL files
28 clean-targets: # directories to be removed by `dbt clean`
29   - "target"
30   - "dbt_packages"
31
32 # Configuring models
33 # Full documentation: https://docs.getdbt.com/docs/configuring-models
34
35 # In this example config, we tell dbt to build all models in the example/
36 # directory as views. These settings can be overridden in the individual model
37 # files using the `{{ config(...) }}` macro.
38 models:
39   TPCH_SF:
40     01_landing:
41       schema: 'edw_landing'
42       +materialized: incremental
43     02_stage:
44       schema: 'edw_stage'
45       +materialized: view
46     03_dv:
47       schema: 'edw_dv'
48       +materialized: incremental
49       hubs:
50         tags: 'hub'
51       links:
52         tags: 'link'
53       sats:
54         tags: 'sat'
```

Demo: dbt file dependency in dbtCloud (generated)

The screenshot displays the dbtCloud interface for a project named 'bdf2023-dbt-demo'. The main editor shows the SQL file 'HUB_CUSTOMER.sql' with the following content:

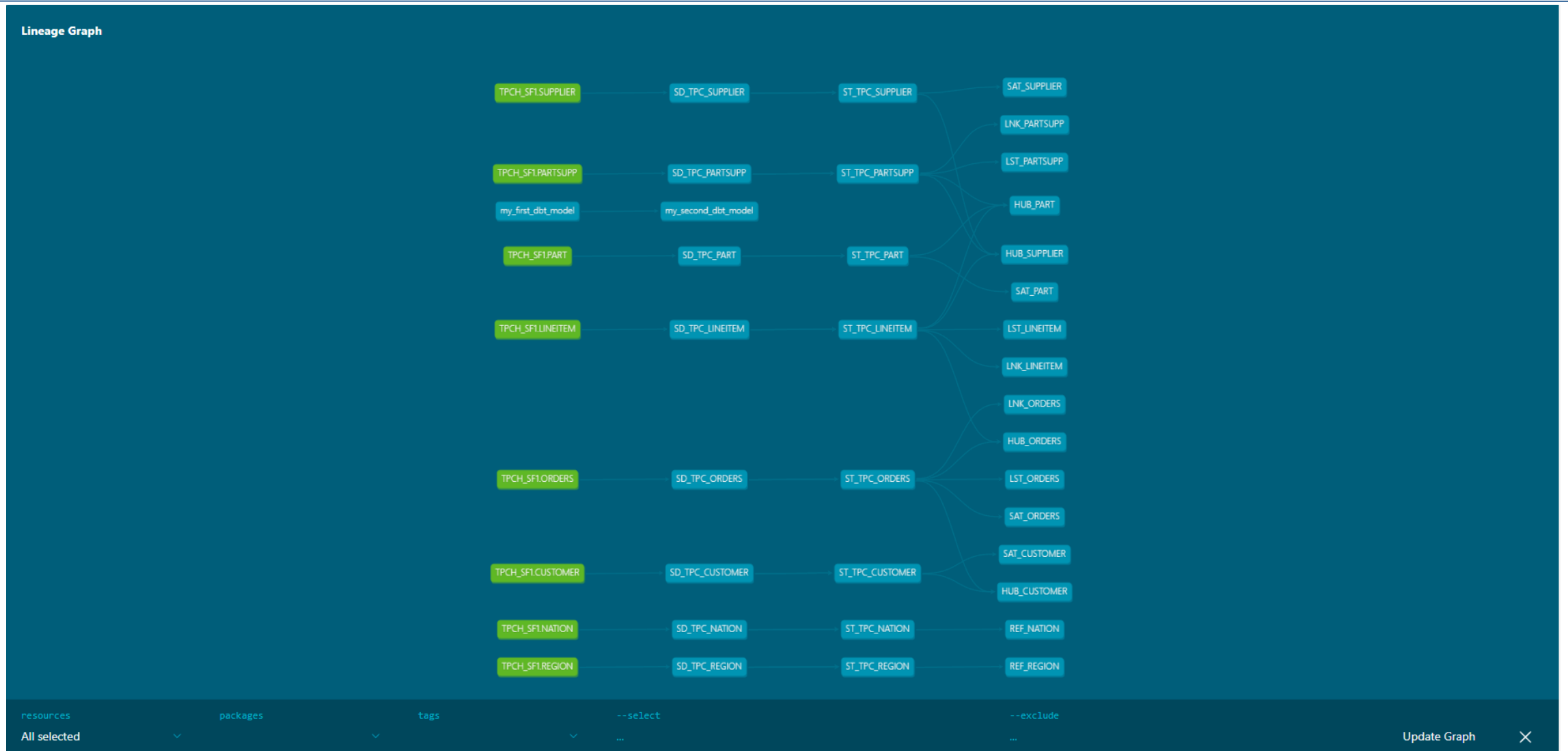
```
1 {% set source_model = ["ST_TPC_ORDERS", "ST_TPC_CUSTOMER"] -%}  
2 {% set src_pk = "HUB_CUSTOMER_HASH" -%}  
3 {% set src_nk = ["C_CUSTKEY"] -%}  
4 {% set src_ldts = "META_LOAD_DATE" -%}  
5 {% set src_source = "META_RECORD_SOURCE" -%}  
6  
7 {{ dbtvault.hub(src_pk=src_pk, src_nk=src_nk, src_ldts=src_ldts,  
8 | | | | src_source=src_source, source_model=source_model) }}
```

The left sidebar shows the 'File Explorer' with a tree view of the project structure, including folders like 'models' and 'hubs', and files like 'HUB_CUSTOMER.sql'. The bottom right of the interface shows a 'Lineage' graph with the following nodes and dependencies:

- SD_TPC_ORDERS (Source) depends on ST_TPC_ORDERS (Staging)
- SD_TPC_CUSTOMER (Source) depends on ST_TPC_CUSTOMER (Staging)
- ST_TPC_ORDERS (Staging) and ST_TPC_CUSTOMER (Staging) both depend on HUB_CUSTOMER (Hub)

The graph also includes a '2+HUB_CUSTOMER+2' node and an 'Update Graph' button. The bottom status bar shows 'dbt build --select <model_name>' and a 'Ready' indicator.

Demo: dbt models lineage in dbtCloud (generated)



Conclusion

Benefits

- faster development
- improved the code quality (templates)
- things are connected (data model + dbt models are in sync)

Risks

- a bad model produces a bad result (but it can be fixed quickly by regenerating)
- data modeling can be a bottleneck

Usability

- data modeling takes some time
- we need to learn more components
- still some tasks that we need to implement manually (complex business logic)

Thanks!



info@metaconsulting.hu