



# Jinja-Ninjas - die Makro-Templates in DBT

27. März 2025



# Vorstellung

- Sebastian Flucke
  - Dipl. oec. in "Wirtschaftsinformatik" 1986
  - BI Consultant seit 1995
  - Schwerpunkte:
    - Microsoft BI Stack
    - Azure SQL Databases, DBT, Databricks
  - [SFlucke@online.de](mailto:SFlucke@online.de)

# Inhalt

- kurze Einordnung DBT sowie Jinja
- Kurzüberblick Jinja-Features
- Beispiele
  - CTE
  - Scalar Function
  - HDA-Generierung
  - recursive CTE

# [www.getdbt.com/product/what-is-dbt](https://www.getdbt.com/product/what-is-dbt)

## **Centralized**

dbt centralizes your business logic, making collaboration and updates easy. It transforms your data inside your cloud data platform — never moving it.

## **Modular**

dbt breaks business logic into modular chunks that are easy to build on. Built-in testing ensures data quality, while documentation provides shared context.

## **Open**

dbt is open-source and compiles directly into transparent SQL. It's supported by a thriving and growing community.

# DBT: Datenbank-Plattformen

- AlloyDB, Apache Spark, Athena, Azure Synapse, BigQuery, Databricks, Dremio, Glue, IBM Netezza, Materialize, Microsoft Fabric, Oracle Autonomous Database, Postgres, Redshift, Snowflake, Starburst/Trino, Teradataeinfache Datumsliste
- [docs.getdbt.com/docs/supported-data-platforms](https://docs.getdbt.com/docs/supported-data-platforms)

<https://www.palletsprojects.com/projects/jinja>

- Jinja is a fast, expressive, extensible templating engine. Special placeholders in the template allow writing code similar to Python syntax. Then the template is passed data to render the final document.
- Jinja's philosophy is that while application logic belongs in Python if possible, it shouldn't make the template designer's job difficult by restricting functionality too much.

<https://www.palletsprojects.com/projects/jinja>

- Jinja is a fast, expressive, extensible templating engine. Special placeholders in the template allow writing code similar to Python syntax. Then the template is passed data to render the final document.
- Jinja's philosophy is that while application logic belongs in Python if possible, it shouldn't make the template designer's job difficult by restricting functionality too much.

**Warum? Jinja kommt aus der HTML-Ecke (auch die Doku) - und da ist die Logik natürlich besser in Python als in HTML aufgehoben!**

# DBT: Erzeugung von Datenbank-Objekten

## Erzeugung von Datenbank-Objekten

- Schritt 0: Schreiben von SQL-SELECT-Code (mehr oder weniger plattform-spezifisch)
- Schritt 1: COMPILE - Umsetzung in lauffähiges SQL, ggf. inkl. Auflösung von Abhängigkeiten
- Schritt 2: RUN - Erzeugung und ggf. Processing der Objekte auf der zugeordneten Datenbank-Plattform

# DBT: Erzeugung von Datenbank-Objekten

## Erzeugung von Datenbank-Objekten

- Schritt 0: Schreiben von SQL-SELECT-Code (mehr oder weniger plattform-spezifisch)
- Schritt 1: COMPILER - Umsetzung in lauffähiges SQL, ggf. inkl. Auflösung von Abhängigkeiten
- Schritt 2: RUN - Erzeugung und ggf. Processing der Objekte auf der zugeordneten Datenbank-Plattform

## Einsatz von Jinja in DBT

- Nutzen von Jinja-Makros in Schritt 0
- Auflösung der Makros in Schritt 1

# Jinja: relevante formale Features

- grundsätzlich: reine Text-Ersetzung (offline, ohne Datenbank-Verbindung)
- Verschachtelung von Makro-Aufrufen: ja
  - also in einem Makro andere Makros aufrufen
- mehrstufiges Parsing: nein
  - mit einem Makro einen weiteren Makro-Aufruf generieren
- Schleifen-Bildung
- strukturierte Makro-Datentypen: Liste, Dictionary

# Jinja: relevante inhaltliche Features

- Objekt-Lineage durch Source- & Ref-Makros
- Definition von Tests
- Iterieren von Dictionaries und Listen

# Jinja-DBT-Beispiel 1 - eine CTE bauen

Warum dafür ein Makro?

- man muß nur die "Inhalte" angeben...
- ...und sich nicht um die Syntax kümmern
- man hat eine sich automatisch erweiternde ColumnList und muß zusätzliche Felder nur einmalig an der Entstehungsstelle eintragen
- wenn Generatoren Objekte erzeugen, ist ein Makro zum CTE-Bauen auch ein hilfreicher Single Point of Definition

Beispiel - CTE

# Jinja-DBT-Beispiel 2 - Scalar Function

- Scalar-Funktion-Problematik in jedem Datenbank-System:
  - pro: single point of definition
  - contra: der Tod jeden QueryOptimizers
- Ausweg
  - die Funktion-Logik als Jinja-Makro abbilden
  - das Parsing beim COMPILE/ RUN erzeugt den optimizer-tauglichen Code...

# Beispiel - Scalar Function

# Jinja-DBT-Beispiel 2 - Scalar Function

- Scalar-Funktion-Problematik in jedem Datenbank-System:
  - pro: single point of definition
  - contra: der Tod jeden QueryOptimizers
- Ausweg
  - die Funktion-Logik als Jinja-Makro abbilden
  - das Parsing beim COMPILE/ RUN erzeugt den optimizer-tauglichen Code...
  - ... sieht ggf. etwas gruselig aus, ist aber an allen Stellen des Makro-Einsatzes identisch!

## Jinja-DBT-Beispiel 3 - Erstellung von HDA-Objekten (1)

HDA-Objekte (leicht vereinfacht)

- Landezone-Tabelle (alle Input-Daten drin)
- "...\_tec"-Table -> Change Detection
- "...\_hda\_base"-Table -> detected history inkl. Delete-Records
- "...\_hda"-View -> Join "...\_hda\_base" mit Landezone-Tabelle
- "...cur"-View -> Filter auf "...\_hda" bzgl. aktuell gültiger Datensätze

(je nach LoadType kann es mehr oder weniger dieser Objekte geben)

## Jinja-DBT-Beispiel 3 - Erstellung von HDA-Objekten (2)

Generierung von HDA Stufe 1:

- Python-Skript zur Erstellung von relevanten SQL-Modells
- Steuer-Parameter kommen aus einer Source-YML
- Ergebnis: SQL-Modelle, die nur aus Makro-Aufrufen bestehen

# Beispiel - HDA-Generierung Stufe 1

## Jinja-DBT-Beispiel 3 - Erstellung von HDA-Objekten (2)

Generierung von HDA Stufe 1:

- Python-Skript zur Erstellung von relevanten SQL-Modells
- Steuer-Parameter kommen aus einer Source-YML
- Ergebnis: SQL-Modelle, die nur aus Makro-Aufrufen bestehen

Generierung von HDA Stufe 2:

- COMPILE der Modelle -> Erzeugung des SQL-Codes

## Beispiel - HDA-Generierung Stufe 2

# Jinja-DBT-Beispiel 4 - Recursive CTE (1)

Thema "rekursives SQL-Select"

- wer kennt sich damit aus?
- Anwendungsfälle
  - inhaltliches ParentChild (z.B Team-Hierarchie)
  - technisches ParentChild
    - als Schleifenersatz, z.B. Erzeugung einer Datumstabelle
    - für Extrapolationen
    - für reihenfolgen-basierte Verarbeitungen
    - etc.

# Recursive CTE - T-SQL-Beispiel

## Jinja-DBT-Beispiel 4 - Recursive CTE (2)

- kann Databricks nativ leider nicht
- der Weg über PySpark ist
  - fehleranfällig
  - führt wieder weg vom Query-Optimizer
- deshalb ein Jinja-Makro...

## Jinja-DBT-Beispiel 4 - Recursive CTE (3)

- ...deshalb ein Jinja-Makro
- Auflösung in Blöcke
  - Root
  - Schleife je Level
  - extra-Level wg. Überlauf
  - abschließendes Union

## Beispiel - Recursive CTE

Und bei Fragen - fragen!



Danke für Eure  
Aufmerksamkeit!

