5 häufige Fehler beim Erstellen von Information Views

Amsterdam – Juni 2019

Fragebogen

- Ich benutze immer Inner Joins
- Projektionen sind überflüssig
- SQL oder Column-Engine? Mir doch egal….
- Star Joins sind viel zu langsam
- Ich verwende regelmäßig Aggregatknoten
- Ich nehme immer alle Spalten in meine View
- Left Outer oder Inner Join? Das macht doch HANA….

5 häufige Fehler beim Erstellen von Information Views

Was sind die 5 häufigsten Fehler?

1. „Falsche" Calculation View für die Aufgabe
2. „Falsche" SQL-Engine verwenden
3. Häufige Verwendung von „heavy nodes"
4. Filter und Left-Outer-Joins an der „falschen" Stelle
5. Ignorieren zusätzlicher Persistenz

1.) Verwenden der „falschen“ Calculation View

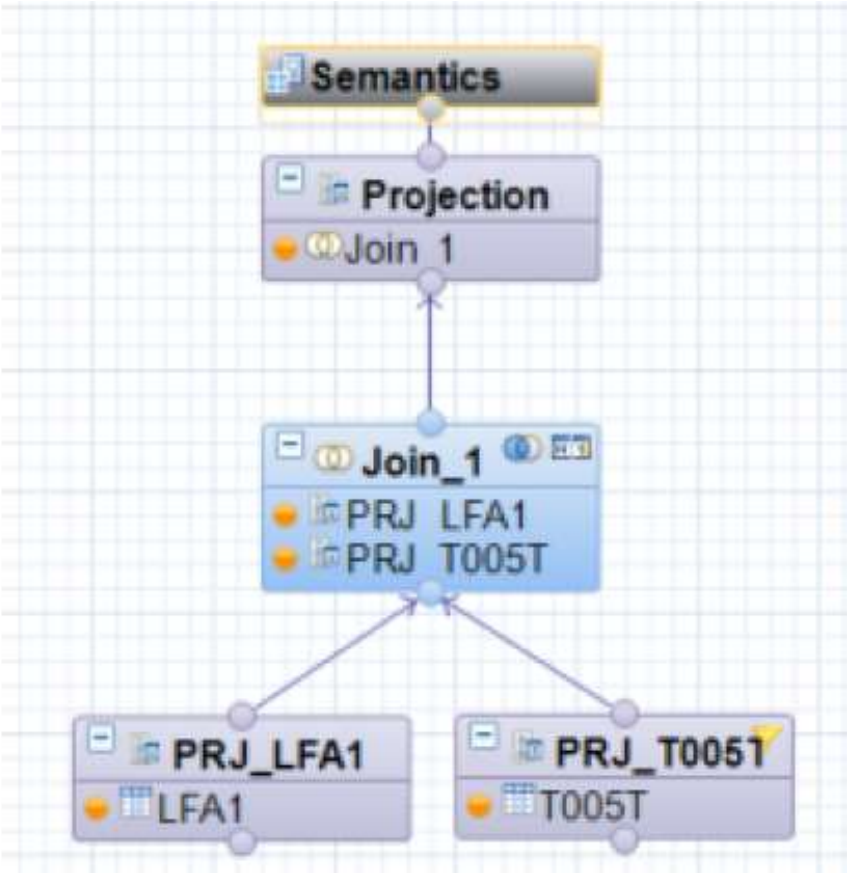
- Fehler bei Modellierung von Dimensional Calcuation Views (DCV)
- Fehler beim Design von Star Joins (SCV)
- Fehler beim Design von klassischen Calculation Views (TCV)

- Daumenregel:
  - DCV -> SCV -> TCV

Fehler bei Modellierung der Dimensional Calculation View (DCV)

- DCV dienen zum Modellieren von Stammdaten
  - Wiederverwendung!
- Best practices:
  - Kein Überladen der Dimension
  - Minimieren der Anzahl der Tabellen in der View
  - Kleinere Dimensionen haben bessere Laufzeiten!
  - Nur notwendige Spalten (Trade-Off bzgl. Wiederverwendbarkeit und Performance)
  - Keine geschachtelten DCVs

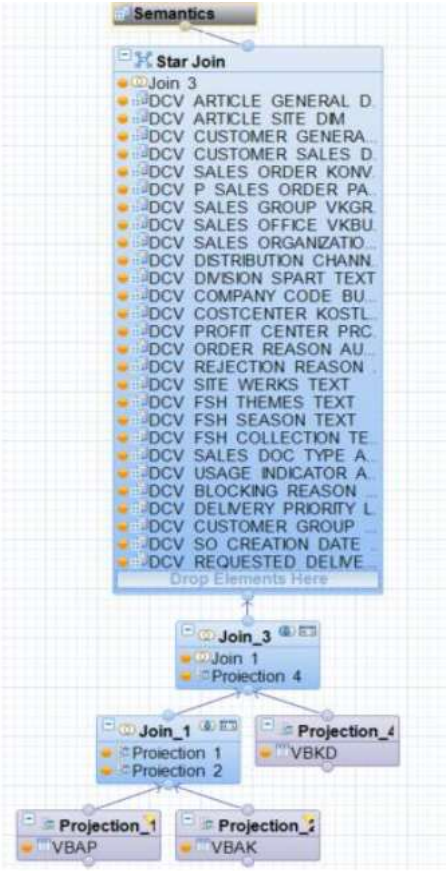
Beispiel für gutes Design mit Dimensional Calculation Views



Fehler beim Modellieren von Star Joins

- Star Join Calculation Views (SCV) verbinden ein oder mehrere DCVs mit der Faktentabelle
- SCV ist der vorletzte Knoten im Modell
  - Kommt unmittelbar vor Semantics Knoten
- Best practices
  - Tabellengrößen der zu joinenden Tabellen beachten
    - Kleinere Tabellen eignen sich besser
  - Nur benötigte Spalten
  - Einsatz von Filtern und Input-Parametern auf den Projektionsknoten
    - Reduktion der Datenmenge und des Joins
  - Nach Möglichkeit Left-Outer-Joins benutzen
  - Inner Joins so früh wie möglich
    - Unterhalb des Star Joins
  - Spalten, die sowohl im Star join als auch in der Dimension vorkommen, sollten im Ergebnis nur aus der Dimension kommen

Beispiel gutes Design Star Join



- Einsatz von Projektion (DCV)
- Inner Join kommt zuerst
- Star Join mit notwendigen Feldern

2.) Verwendung der „falschen“ Engine

- SQL Engine -> Standard SQL
- Column Engine -> SQLScript
- Frühzeitig überlegen, welche Engine in der View verwendet wird
- Wahl abhängig von
  - Persönlichen Präferenzen und Know-How
  - Column Engine bietet größere Funktionsbibliothek
    - Geo-spatiale Funktionen nur in Column Engine
  - Ansonsten ist SQL Engine schneller
- Switch zwischen Column und SQL Engine im Modell vermeiden
  - Switch zwischen Engines kann sehr zeitintensv sein

3.) Häufige Verwendung von „heavy nodes“

- „Heavy Nodes“ sind Knoten, die z.Bsp. die Aggregatfunktionoder Rankingfunktion nutzen
- Brauchen sehr viel Rechenzeit
- Filter können nicht an „heavy nodes“ weitergereicht werden

- Best Practice
  - Aggregation so früh wie möglich
  - Datenmenge möglichst klein halten
  - Möglichst wenige Joins, um Datenmenge klein zu halten

4.) Filter und Left-Outer-Joins an der falschen Stelle

- Filter können an jeder Attributspalte außer im Semantik-Knoten und im Knoten vor dem Semantik-Knoten angelegt werden
- Input Parameter erlauben es, den Filter auf den Tabellen auszuführen
- Immer wenn möglich Left-Outer-Joins verwenden
  - Datenbank-Pruning

5.) Ignorieren zusätzlicher Persistenz

- Manchmal ist die Modellierung als CV zu komplex
- Zusätzliche Persistenz?
  - Nichts ist schneller als direktes Lesen aus einer HANA-Tabelle/View
  - Zusätzliche ETL kann deutliche Performancegewinne bringen
  - Für Real-Time-Analyse kann SAP HANA SDI genutzt werden
  - Einsatz von zusätzlicher ETL birgt Fehlerquelle
  - Einsatz von ETL bringt zusätzliche Kosten
  - Abwägung der Kosten gegenüber Nutzen

Zusätzliche Quellen

- Blog post consolidating HANA modeling best practices
  - https://blogs.sap.com/2016/11/04/hana-modelling-consolidated-best-practices-for-better-performance/