Think BIGGER

DEEP DIVE FÜR FACHLEUTE

SAP CDS Views

Architektur, Syntax, Annotationen, DCL und Best Practices für SAP CDS

Autor	Erhard Rainer
Veröffentlicht	18.09.2024
Aktualisiert	12.04.2026
Umfang	ca. 23 Min. Lesezeit
Online	www.thinkbigger.at/wissen/sap-cds-views/

thinkbigger.at

Business Intelligence · Machine Learning · KI-Agenten · Schulungen

Erhard Rainer · Schönaugasse 106 · 8020 Graz · info@think-bigger.at

Inhalt

Der ultimative Guide zu SAP CDS 2
Inhaltsverzeichnis 2
TL;DR 2
1. Einleitung: Warum CDS der Gamechanger ist 2
2. Die Architektur: Was unter der Haube passiert 3
3. Syntax & Core Features (Das Handwerkszeug) 5
4. Annotationen: Die "Magie" der Metadaten 9
5. Berechtigungen mit DCL (Data Control Language) 11
6. On-Premise vs. ABAP Cloud: Wo liegen die Unterschiede? 11
7. Tooling & Entwicklungsumgebung 12
8. Best Practices & Performance-Tipps 13
References & Weiterführende Links 18

Der ultimative Guide zu SAP CDS

Inhaltsverzeichnis

- TL;DR
- 1. Einleitung
- 2. Die Architektur
- 3. Syntax & Core Features
- 4. Annotationen
- 5. Berechtigungen mit DCL
- 6. On-Premise vs. ABAP Cloud
- 7. Tooling & Entwicklungsumgebung
- 8. Best Practices & Performance-Tipps
- Troubleshooting-Checkliste
- References & Weiterführende Links

TL;DR

CDS (Core Data Services) sind das zentrale deklarative Datenmodell in S/4HANA: Schreibe Logik nah an der DB, nutze `DEFINE VIEW ENTITY` für moderne, leichtgewichtige Entities, schichte Basic→Composite→Consumption Views im VDM, und setze DCL für Zeilensicherheit ein. Nutze ADT/ADT-Tools für Entwicklung und HANA Plan Viz / ST05 zum Debugging.

Leserführung

- **Strategische Entscheider:** Fokus auf den Business-Mehrwert – Echtzeit-Analysen, geringerer Wartungsaufwand, Single-Source-of-Truth. Empfehlenswerte Kapitel: 1 Einleitung, 2 Architektur, 8 Best Practices & Performance-Tipps (inkl. Cloud-Strategie).
- **Klassischer ABAP-Entwickler:** Schau dir Kapitel 3 Syntax, 4 Annotationen, 7 Tooling und 5 DCL an; Beispiele und die Transition zu `DEFINE VIEW ENTITY` sind besonders relevant.
- **Data Scientist / Analyst:** Springe zu 4 Analytics-Annotationen, 8 Performance & Troubleshooting und die VDM-Tabellen; die Abschnitte zu Table Functions und Hierarchien sind nützlich.

1. Einleitung: Warum CDS der Gamechanger ist

In der Welt von SAP S/4HANA ist ein grundlegender Paradigmenwechsel im Gange. Weg vom traditionellen Data-Aging und den immer komplexer werdenden Berichten an der Applikationsschicht hin zu einem Modell, in dem die Datenlogik so nah wie möglich an der Datenbank ausgeführt wird. Core Data Services (CDS) sind dabei nicht nur ein technisches Featureset, sondern das Herzstück dieser Entwicklung. Sie sind der Gamechanger, der die Art und Weise, wie Entwickler und Fachanwender mit Daten umgehen, nachhaltig verändert.

Vom Data-Aging zur Echtzeit

Historisch wurden SAP-Anwendungen oft auf NetWeaver ABAP gebaut, bei denen umfangreiche Datenmengen in relationalen Tabellen liegen. Analysen und Reports wurden über Selektionen, klassische SQL-Views und ABAP-Programme realisiert. Mit dem Übergang zu S/4HANA und der In-Memory-Technologie entstehen neue Anforderungen: Es geht nicht mehr um Batch-Jobs, die über Nacht ausgeführt werden, sondern um Echtzeit-Auswertungen, die jederzeit reagieren müssen. Die klassische Datenbankabfrage über Open SQL stößt hier an Grenzen – Performance, Wartbarkeit und das „Code-to-Data"-Prinzip werden kritisch.

Das "Code-to-Data"-Prinzip

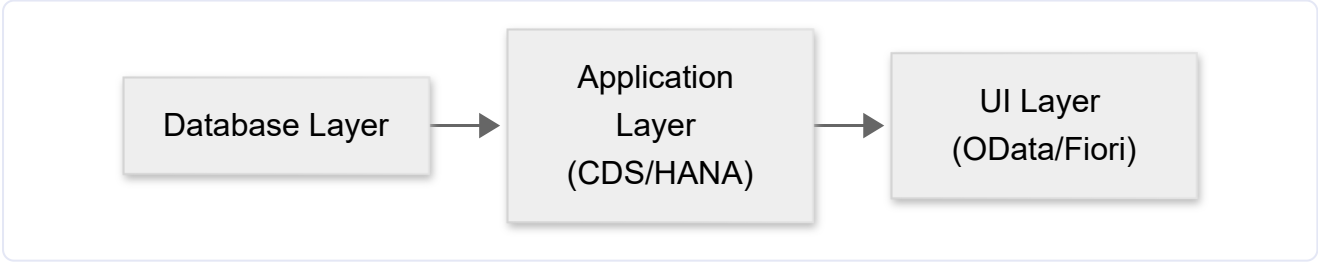
CDS verkörpert das „Code-to-Data"-Prinzip: Statt Daten in die Anwendung zu holen und dort zu filtern, werden alle möglichen Transformationen, Aggregationen und Berechnungen direkt in die Datenbank überführt. Dies verschiebt die Intelligenz vom ABAP-Server in die HANA-DB. Der Vorteil: Die massiv-parallele Verarbeitung von HANA kann vollständig genutzt werden, Daten-Redundanzen werden vermieden und der Netzwerkverkehr minimiert. Eine CDS-View ist damit nichts anderes als ein deklaratives Datenmodell, das sowohl das Backend (HANA) als auch das Frontend (ABAP, Fiori) gleichermaßen versteht.

Definition: CDS als Herzstück der Virtual Data Models (VDM)

Virtual Data Models (VDM) sind logische Schichten von Views, die Geschäftsobjekte und Prozesse abbilden. Sie dienen als Grundlage für Analytics, Reporting, OData-Services und Applikationen. CDS ist das Defining Language (DDL) Framework, mit dem diese VDM-Schichten modelliert werden. Ein CDS-View beschreibt semantisch, wie Daten zusammengehören – ohne sich um physische Tabellen oder Indizes zu kümmern. So entsteht ein virtuelles Schaufenster in die Datenwelt, das beliebig erweitert und wiederverwendet werden kann.

2. Die Architektur: Was unter der Haube passiert

Um CDS wirklich verstehen zu können, muss man die Architektur kennen. Was startet beim Erzeugen einer CDS-View in Eclipse oder HANA Studio, und wie wird daraus ein View in der Datenbank sowie eine ABAP-Entität im Backend?



DDL, DQL, DCL & VDM: Das Buchstabensalat-Rätsel gelöst

- **DDL (Data Definition Language):** In CDS beschreibt DDL die Struktur der View – FELDER, ASSOCIATIONS, JOINS, ANNOTATIONS. Mit den SQL-ähnlichen `DEFINE VIEW` - oder `DEFINE VIEW ENTITY` -Statements entsteht ein persistentes Repository-Objekt. **Seit ABAP 7.55/7.56 hat sich ein „Breaking Change" eingeschlichen:** `DEFINE VIEW ENTITY` ist der neue Standard. Bei dieser Variante wird keine klassische SQL-View im DDIC mehr angelegt, sondern nur noch eine *CDS Entity*. Dadurch fällt die Aktivierung

schneller aus und das Namensraumchaos (weniger zusätzlicher `SQLVIEW_<nnn>`-Einträge im Dictionary) wird vermindert. Alte Perspektiven, die noch `DEFINE VIEW` nutzen, bleiben funktionieren, aber neue Entwicklungen sollten auf View Entities umstellen.

:::note **Migrationstipp** – DDIC-basierte Views → View Entities Beim Umstieg entferne unbedingt die `@AbapCatalog.sqlViewName`-Annotation aus der alten Definition. Ist sie noch vorhanden, versucht das System beim Aktivieren eine SQL-View zu generieren und bricht mit Fehlern ab (z.B. "object already exists"). Lege stattdessen ggf. ein separates Consumption-Entity mit dem gewünschten SQL-Namen an. :::

Beispiel: `DEFINE VIEW ENTITY` (vereinfachtes Beispiel)

```
@AbapCatalog.sqlViewName: 'ZV_MARA_ENT'    " optional, nur für On-Premise readability
@VDM.viewType: #BASIC
define view entity Z_CDS_MaterialEntity as select from mara {
  key matnr,
  mtart,
  matkl
}
```

Hinweis: Bei reinen View Entities kann die `sqlViewName`-Angabe optional sein; das Objekt existiert primär als CDS-Entity in ADT.

- **DQL (Data Query Language):** Zur Laufzeit kann aus ABAP heraus gegen ein CDS-Objekt mittels Open SQL `SELECT ... FROM <view>` bzw. `SELECT ... FROM <entity>` abgefragt werden. Die eigentliche Ausführung passiert in HANA.
- **DCL (Data Control Language):** Sicherheitsaspekte werden über CDS-DCL geregelt. Zeilenlevel-Security und Authorisationsprüfungen lassen sich deklarativ definieren.
- **VDM (Virtual Data Model):** Eine Schicht aus CDS-Views, die von der rohesten Basis bis zur UI-sicht reichen. SAP unterscheidet hier heute streng:
  - **Basic Views** (Typ `#BASIC`): direkter Zugriff auf Tabellen; keine Logik, reine Felddefinitionen.
  - **Composite Views** (Typ `#COMPOSITE`): Zusammenschaltungen mehrerer Basic Views, Berechnung von Kennzahlen, Aggregationen.
  - **Consumption Views** (Typ `#CONSUMPTION`): Die oberste Schicht; sie liefern Annotations für OData/Fiori und sind der Einstiegspunkt für Anwender.

Diese Nomenklatur erklärt auch die Namensgebung in SAP-Standard-VDM (z.B. `I_Material` =Consumption, `C_Material` =Composite, `I_MaterialBasic` =Basic).

Die Doppelnatur: Wie aus einer Definition ein Datenbank-View (HANA) und eine ABAP-Entität entsteht

Ein CDS-Objekt wird im DDIC (Data Dictionary) als sogenanntes DDL-Source-Objekt gespeichert. Beim Aktivieren generiert der Aktivierungsdienst zwei Dinge (dabei zeigt sich bereits der Übergang von der DB-Ebene zur Applikations- und UI-Ebene):

1. **Datenbank-View / Repräsentation:** In HANA entsteht immer ein SQL-View bzw. eine interne Repräsentation, die Tabellen, Joins und Filter zusammenführt. Bei klassischen `DEFINE VIEW`-Objekten ist dieser View gleichzeitig im DDIC als „SQLVIEW_“ sichtbar – man kann ihn in der SE11 öffnen. **Wichtig:** Bei modernen `DEFINE VIEW ENTITY`-Objekten existiert zwar weiterhin intern eine HANA-Repräsentation, **aber** es gibt **keinen** entsprechenden Dictionary-Eintrag mehr. In der SE11 findet man dann keine View mehr; das Objekt lebt ausschließlich als CDS-Entity in ADT. Dieser Unterschied reduziert den administrativen Overhead und vermeidet unerwartete SE11-Einträge.

2. **ABAP-Entität:** Im ABAP Dictionary entsteht – abhängig von der Art des CDS-Objekts – entweder eine abstrakte View-Definition oder eine Entity-Struktur, die aus ABAP Open SQL konsumiert werden kann. Die Felder und Assoziationen werden als Strukturen bzw. Tabellentypen im DDIC verfügbar gemacht.

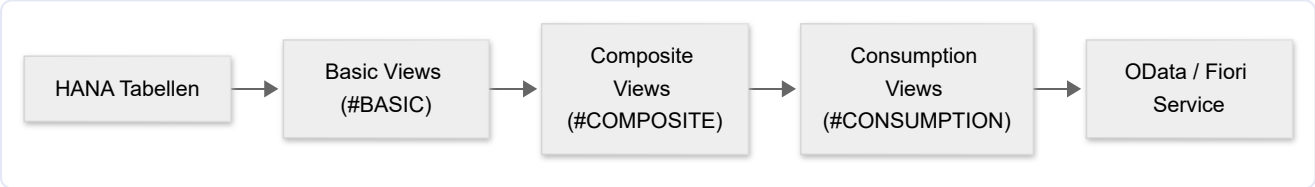
Dadurch ist es möglich, in ABAP, in OData-Services oder in anderen SAP-Tools mit derselben Definition zu arbeiten – eine echte Single-Source-of-Truth.

Der Stack: Die Schichtung von der physischen Tabelle bis zum OData-Service

Ein typischer Datenfluss sieht wie folgt aus:

- **Schema E: Tabellen:** Klassische HANA-Tabellen (z.B. MARA, BKPF) enthalten Rohdaten.
- **CDS-Basis-Views:** Views mit `@AbapCatalog.sqlViewName : 'ZV_MARA'` verbinden Tabellen und bieten semantische Felder.
- **Operationale Views:** Zusätzliche Logik, z.B. Berechnungen, Filter, die für bestimmte Szenarien optimiert sind.
- **Consume Views:** Sichtbar für Anwendungen, definieren oft OData-Annotations für Fiori.
- **OData-Service:** Ein Gateway-Service nutzt die Consume Views als Datenquelle. In einer RAP-Umgebung werden lokale Query- und Behavior-Definitionen hinzugefügt.

Zwischen diesen Ebenen können mehrere CDS-Views gestapelt werden; jede Ebene fügt Semantik oder Funktionalität hinzu.



3. Syntax & Core Features (Das Handwerkszeug)

Der Kern jeder CDS-View ist die Sprache selbst. Sie ähnelt SQL, enthält aber spezielle Erweiterungen. Zusätzlich zu den klassischen Bausteinen bietet CDS einige moderne Konstrukte, die typische Datenbankgrenzen durchbrechen – etwa Virtual Elements (siehe weiter unten).

📌 **Virtual Elements** Manchmal sind Berechnungen oder Datenquellen so exotisch, dass „reines" SQL nicht ausreicht. Virtual Elements sind Felder, die mit ABAP-Code berechnet werden, sobald sie angefordert werden. Sie erlauben z.B. Entfernungsberechnungen via Funktionsbaustein oder komplexe Zugriffskontrollen, die HANA nicht leisten kann.

```
@AbapCatalog.virtualElement: true
@ObjectModel.virtualElementClass: 'ZCL_MY_VE'
      distance_in_km  : decfloat34;
```

Die Klasse `ZCL_MY_VE` enthält dann eine Methode `CALCULATE` mit ABAP-Logik. Virtual Elements eignen sich, wenn andere CDS-Techniken versagen – sie sollten aber sparsam eingesetzt werden, da die Berechnung im ABAP-Stack erfolgt. Technisch muss die begleitende Klasse `IF_SADL_EXIT_CALC_ELEMENT_READ` implementieren, die das Interface und dessen Methoden zur Berechnung bereitstellt, nicht nur eine generische `CALCULATE`-Methode.

Extensibility: Standard-Views erweitern

Ähnlich wie man Tabellen mit Append-Strukturen ergänzt, bietet CDS die Möglichkeit, bestehende Views zu erweitern. Besonders praktisch beim Arbeiten mit SAP-Standard, weil man Anpassungen ohne Modifikationen an den Originalobjekten realisieren kann:

```
extend view entity I_MATERIAL with
{
  @EndUserText.label: 'Custom Flag'
  z_my_flag : abap_bool;
}
```

Der Zusatzfeldname beginnt typischerweise mit `Z_` oder `Y_` und die Erweiterung wird separat aktiviert. In Cloud-Systemen sind nur `extend view entity`-Erweiterungen erlaubt, klassische `append`-Strukturen entfallen.

Table Functions: Wenn CDS an seine Grenzen stößt

CDS deckt sehr viele Szenarien ab, aber manchmal sind komplexe Algorithmen, Iterationen oder prozedurale Logik notwendig. In solchen Fällen bietet HANA `DEFINE TABLE FUNCTION` (SQLScript/AMDP) die Möglichkeit, eine „funktionale" Tabelle zu implementieren, die CDS als Datenquelle referenzieren kann. Typische Einsatzfälle:

- Komplexe Berechnungen mit Schleifen oder Zwischentabellen
- Performance-kritische Set-Based-Operationen, die SQLScript effizienter abbildet
- Nutzung von HANA-spezifischen Funktionen, die in rein deklarativen CDS schwer darstellbar sind

Wichtig: Table Functions bringen zusätzliche Komplexität (Deployment, Traceability, Berechtigungen) und sollten gut dokumentiert werden. In ABAP-Kontexten werden solche Funktionen oft über AMDPs (ABAP Managed Database Procedures) erstellt und als Datenquelle in CDS eingebunden.

CDS-Hierarchien (DEFINE HIERARCHY)

Hierarchische Datenmodelle (z.B. Organisationsstruktur, Produktkataloge) lassen sich explizit mit CDS/HANA-Hierarchien modellieren. `DEFINE HIERARCHY` erlaubt die Abbildung von Parent-Child-Beziehungen, Pfad- und Level-Berechnungen und ist besonders nützlich für analytische Auswertungen, die Tiefenbegrenzung oder Roll-Up/Drill-Down-Szenarien benötigen.

Kurzvorteile:

- Native Unterstützung von Level/Path-Funktionen
- Effiziente Auswertung großer Hierarchien in HANA
- Integration in Analytic Queries und Fiori-Reports

Hinweis: Die genaue Syntax und das Verhalten können zwischen HANA-CDS und ABAP-CDS leicht variieren — prüfe die Systemdokumentation und test die Hierarchie-Abfragen im HANA-Studio bzw. ADT.

Joins vs. Associations: Warum "Lazy Loading" die Performance rettet

Vergleich	JOIN	ASSOCIATION
Join-Typ	explizit	implizit, bei Bedarf
Performance-Impact	immer ausgeführt	nur bei Verwendung der Felder
Lazy Loading	❌	✅

Vergleich	JOIN	ASSOCIATION
Kardinalitäts-Risiko	hoch	mittel (Optimierer kann umwandeln)

Anschließend folgt der normale Text:

In herkömmlichem SQL werden Joins verwendet, um Tabellen zu verbinden. CDS bietet neben klassischen `JOIN`-Statements auch `ASSOCIATION`-Beziehungen. Der Unterschied:

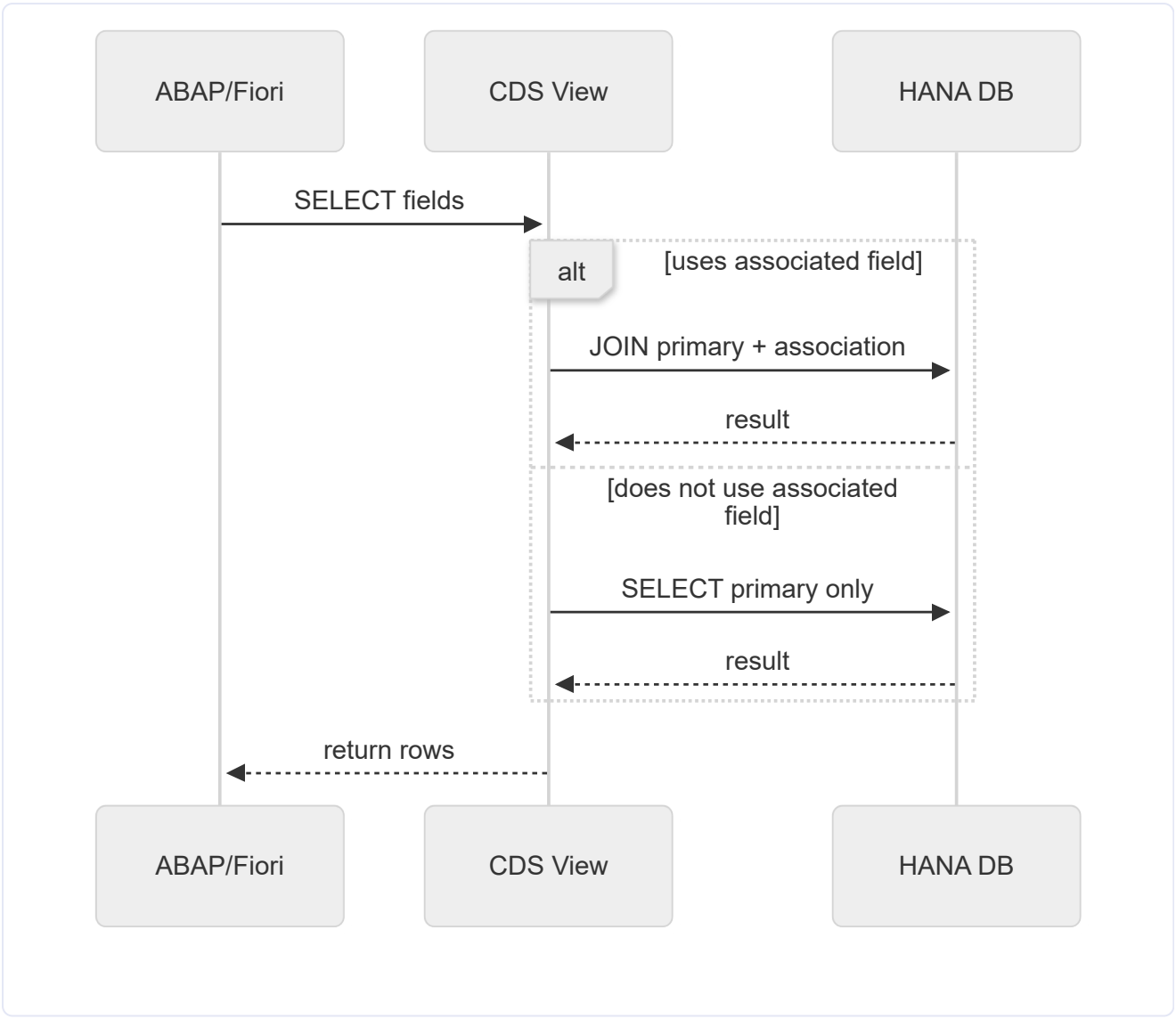
- **JOIN:** Wird in der View direkt ausgeführt. Alle Daten werden in der generierten SQL-Anweisung zusammengeführt.
- **ASSOCIATION:** Definiert eine Beziehung, die erst bei Bedarf (nämlich bei Verwendung der assoziierten Felder) ausgeführt wird. Im Open SQL wird dann ein zusätzlicher `LEFT OUTER JOIN` erzeugt.

Beispiel:

```
@AbapCatalog.sqlViewName: 'ZV_MARA'
define view Z_CDS_Material as select from mara
  association[0..1] to makt as _desc on $projection.matnr = _desc.matnr
{
  key mara.matnr,
  mara.mtart,
  _desc.maktx
}
```

Im oben gezeigten View wird `_desc` nur aktiviert, wenn `maktx` im SELECT-Statement verwendet wird. Das reduziert den Datenverkehr und nutzt das Prinzip des "Lazy Loading" (Bedarfsauswertung) – ähnlich wie bei ORM-Frameworks.

> **Tipp für Perfektionisten:** Die korrekte Kardinalität der Association (`[0..1]`, `[1..*]` usw.) ist nicht nur Dokumentation, sondern hintet den Optimierer von HANA. Mit passenden Kardinalitäten können Left-Outer-Joins automatisch in Inner-Joins umgewandelt werden (Filter-Pushdown), was massive Performance-Vorteile bringt.
>
> **Warnung:** Eine zu enge Kardinalität (z.B. `[1..1]` statt `[0..1]`) kann den Optimierer dazu veranlassen, einen Join ganz zu *prunen* – das heißt, bei bestimmten Bedingungen überhaupt nicht mehr auszuführen. Ergebnis: fehlende Zeilen und falsche Daten. Trage nur solche Kardinalitäten ein, die durch Faktdaten abgesichert sind.



Eine fehlerhafte Kardinalität kann hingegen zu falschen Plänen führen — prüfe daher Fremdschlüssel, Unique Constraints und reale Datenverteilungen beim Festlegen der Cardinalities.

Aggregationen & Ausdrücke: Rechnen direkt auf der Datenbank

Union/Intersect Beispiel:

```
define view Z_CDS_Combined as
  select from table1
    union all
    select from table2
{
  key field1,
  field2
}
```

CDS erlaubt Aggregationen (`SUM`, `COUNT`, `AVG`) sowie CASE-Ausdrücke und CAST-Operationen direkt in der Definition anzuwenden. Damit lassen sich Kennzahlen vorberechnen, z.B.:

```
define view Z_CDS_Sales as select from vbrk
  left join vbrp on vbrk.vbeln = vbrp.vbeln
{
  key vbrk.vbeln,
```

```
  vbrk.fkdat,
  sum( vbrp.netwr ) as total_value,
  case when vbrk.fkdat >= :p_date then 'Recent' else 'Old' end as age_category
}
```

Parameter (`:p_date`) können zur Laufzeit gesetzt werden und ermöglichen flexible Analysen.

Parameter & Filter: Dynamische Views erstellen

Parameter in einer CDS-View werden mit `@Analytics.query: { ... }` und `define view ... with parameters` angegeben. Beispiel:

```
define view Z_CDS_PModel
  with parameters p_bukrs : bukrs_d;
select from bkpf
where bukrs = :p_bukrs
{
  key bukrs,
  key belnr,
  budat
}
```

Solche Views werden häufig in Analytic Queries oder als Input für Fiori-Kacheln genutzt – Benutzer geben Werte ein oder SAP liefert Defaults.

4. Annotationen: Die "Magie" der Metadaten

Ohne Annotationen wäre eine CDS-View nur ein technisches Gebilde. Mit ihnen wird aus der View ein semantisches Artefakt, das andere Tools verstehen.

Was sind Annotationen? Das @-Zeichen und seine Macht

Annotationen sind Metadaten, die dem CDS-Objekt zusätzliche Informationen geben. Sie werden mit `@` eingeleitet und können nahezu jedes Element der View schmücken: Felder, Associations, Views selbst.

Beispiel:

```
@AbapCatalog.sqlViewName: 'ZV_EMP'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'Employee View'
define view Z_CDS_Employee as select from pa0001
{
  key pernr,
  vorna,
  nachn,
  @Semantics.email
  smtp_addr
}
```

Hier definieren wir einen Label für die View, aktivieren die Vergleichsfilterung und kennzeichnen das Feld `smtp_addr` als E-Mail.

UI-Annotationen: Wie man Fiori-Oberflächen direkt im Backend "programmiert"

CDS erlaubt, UI-relevante Informationen zu beschreiben. Fiori-Elemente wie Tabellen, Formulare und OData-Services bauen auf diesen Annotationen auf. Einige Beispiele:

- `@UI.lineItem`: Definiert die Spalten für eine Listenansicht.
- `@UI.selectionField`: Bestimmt Auswahlfelder für Filterleisten.
- `@UI.identification`: Informationen für Detailansichten.
- `@UI.headerInfo`: Key-Infos für Kopfzeilen.

Annotation Cheat-Sheet (Top 10)

Kategorie	Annotation	Zweck
UI	`@UI.lineItem`	Spalten in Listenansichten
UI	`@UI.selectionField`	Filterfelder
UI	`@UI.identification`	Detailansichten
UI	`@UI.headerInfo`	Kopfzeilen
Analytics	`@Analytics.dataCategory`	CUBE/OLAP vs. DIMENSION
Analytics	`@Consumption.filter`	Default-Filter für Queries
Authorization	`@AccessControl.authorizationCheck`	Aktiviert DCL
Authorization	`@EndUserText.label`	Lesbarer Name für Policies
SADL	`@AbapCatalog.sqlViewName`	Name des generierten SQL-Views
SADL	`@AbapCatalog.buffering`	ABAP-Buffering-Parameter

Beispiel für eine Fiori List-Report-Konfiguration:

```
@UI: {
  lineItem: [{ position: 10, label: 'Employee ID', value: pernr },
             { position: 20, label: 'Name', value: concat( vorna, ' ', nachn ) }],
  selectionField: [{ position: 10, value: pernr }, { position: 20, value: nachn }]
}
define view Z_CDS_EmployeeList as select from Z_CDS_Employee { ... }
```

Diese UI-Annotations steuern dann automatisch, wie eine Liste in SAP Fiori gerendert wird – ohne JavaScript, XML oder UI5-Programmierung auf den ersten Blick.

Analytics & Search: Daten für SAP Analytics Cloud oder die Enterprise Search vorbereiten

Für analytische Zwecke gibt es Annotationen wie `@Analytics.dataCategory`, `@Consumption.filter`, und weitere. Damit lassen sich in SAP Analytics Cloud oder BW/4HANA Query-Elemente automatisch erkennen. Für die Enterprise Search können `@Search`-Annotationen Felder indexieren oder als Facetten definieren.

Ein Beispiel für eine SAP Analytics Cloud-fähige View:

```
@Analytics.dataCategory: #CUBE
@Analytics.query: { enabled: true }
@VDM.viewType: #CONSUMPTION


@@DIAGRAM3@@



define view Z_CDS_SalesCube as select from ... { ... }
```

Die Annotation `#CUBE` signalisiert, dass die View als analytisches DataCube zu behandeln ist.

5. Berechtigungen mit DCL (Data Control Language)

Sicherheit ist in Unternehmensanwendungen unerlässlich. CDS bietet hierfür eine eigenständige Sprache – die Data Control Language – um Zugriffskontrollen bereits auf Datenbankebene zu definieren.

Sicherheit auf Zeilenebene

Traditionelle ABAP-Autorisierungsprüfungen basierten auf `AUTHORITY-CHECK` oder Berechtigungsobjekten. In CDS ist es möglich, Regeln für Zeilensicherheit zu definieren:

```
@EndUserText.label: 'Purchase Order Access'
define access policy Z_ACL_PO_ACCESS {
  grant select on Z_CDS_PO
    where z_cds_po.bukrs = $session.system_field_costcenter
      or z_cds_po.ernam = $session.user;
}
```

Diese Policys können komplexe Bedingungen enthalten und verwenden Session-Variablen (z.B. `$session.user`, `$session.system_field_costcenter`). Weitere Systemfelder sind `$session.system_date`, `$session.client`, `$session.currency` etc.; sie lassen sich in WHERE-Klauseln oder Berechnungen verwenden, z.B.

```
where bukrs = :$session.client
  and budat = $session.system_date
```

Sie schützen automatisch jede Abfrage gegen die View – egal ob von ABAP, Fiori oder Extern.

Access Policies: Buchungskreis, Werk, etc.

Besonders nützlich sind Zugriffsregeln, die Organisationsdaten berücksichtigen. Beispielsweise: Ein Vertriebsmitarbeiter soll nur Aufträge seines eigenen Vertriebsbereichs sehen.

Ein wichtiger Mechanismus ist die **Vererbung** von Policies. Mit `AS INHERITED FROM` lässt sich eine Access Policy von einer Basis-View auf eine Consumption-View übertragen, wodurch Duplikate entfallen und Änderungen zentral vorgenommen werden können.

```
define access policy Z_ACL_SALES where
  z_sales_order.vkorg in (
    select distinct vkorg from z_user_sales org
    where org.user = $session.user
  );
```

Durch Verknüpfung mit Modellen für Organisationszugehörigkeit lassen sich sehr feingranulare Berechtigungen realisieren.

6. On-Premise vs. ABAP Cloud: Wo liegen die Unterschiede?

Mit der Einführung der ABAP Cloud (SAP BTP) und der Veröffentlichung von S/4HANA Cloud hat CDS weitere Facetten erhalten.

Classic CDS vs. CDS Projection Views

In On-Premise-Systemen kann eine CDS-View direkt aus anderen Views oder Tabellen bestehen (einschließlich Joins, Aggregate, etc.). In der Cloud hingegen sind sogenannte *CDS Projection Views* der Standard: eine API, die nur bestimmte Felder einer bestehenden View oder eines technischen Objekts (z.B. ABAP-Struct) auswählt. Projection Views haben den Vorteil, dass sie stabiler und einfacher versionierbar sind, eignen sich aber nicht für komplexe Logik.

Einschränkungen in der Cloud

In der ABAP Cloud (BTP) darf der Entwickler weiterhin DDL-Statements schreiben – die Aktivierung läuft ebenfalls in ADT – aber das Resultat ist immer eine *Entity*, keine klassische SQL-View. Es gibt keine SE11-Ansicht und es ist nicht erlaubt, per Open SQL auf die interne SQL-View-Tabelle (`SQLVIEW_<nnn>`) zuzugreifen. Außerdem kann man in der Cloud keine eigenen physischen Tabellen anlegen und muss sich strikt an die vorgegebenen Standard-Entitäten halten. Der gesamte Prozess bleibt für den Entwickler im ADT visuell ähnlich, die generierten Objekte sind jedoch leichtgewichtiger und besser versionierbar.

Das RAP-Modell: CDS als Fundament für transaktionale Apps

Das RESTful ABAP Programming Model (RAP) nutzt CDS als Basis für CRUD-Operationen. Hierfür werden neben *Projection Views* auch *Behavior Definitions* geschrieben. Ein typischer RAP-Stack:

1. **CDS-View (Basic):** Modelliert die Datenstruktur.
2. **CDS-View (Projection/Consumption):** Exponiert die Felder, oft mit UI-Annotations.
3. **Behavior Definition:** Legt fest, welche Operationen (`CREATE`, `UPDATE`, `DELETE`) erlaubt sind, plus Prüfungen und Validierungen.
4. **Service Definition/Binding:** Macht das Objekt als OData-Service verfügbar.

RAP ermöglicht damit transaktionale Fiori-Apps ohne klassisches Dynpro oder WebDynpro.

7. Tooling & Entwicklungsumgebung

CDS-Entwicklung unterscheidet sich stark von der klassischen ABAP-Arbeit. Alte Werkzeuge wie die SE11 sind für CDS weitgehend unbrauchbar geworden.

Abschied von der SE11

In der SE11 konnte man Views, Tabellen, Strukturen und Suchhilfen anlegen. CDS-Entwicklung hat dagegen eine eigene Umgebung:

- **DDLS (ABAP Dictionary -> DDL-Source):** Der Code der View wird in einem speziellen Editor gespeichert.
- **SE80:** bietet eingeschränkte Unterstützung; ABAP in Eclipse (ADT) ist der primäre Ort.

Core Data Services in ADT

ABAP Development Tools (ADT) in Eclipse ist die bevorzugte Entwicklungsumgebung.

- **Editor:** Syntaxhervorhebung, Code-Completion (`Ctrl+Space`), Quick-Info.
- **Dependency Analyzer:** Zeigt, welche Objekte von einer View abhängen und wo sie verwendet wird.
- **Data Preview:** Lässt zu Testzwecken einen SELECT auf der View ausführen und zeigt Daten an.
- **Refactoring-Unterstützung:** Rename, Move, Extract.

Zusätzlich gibt es Tools wie `ABAPGit` für Versionskontrolle, und das ABAP Test Cockpit für Analysen.

8. Best Practices & Performance-Tipps

CDS ist mächtig, aber mit großer Macht kommt Verantwortung. Neben den Schicht- und Namensregeln (siehe oben) gibt es auch Hilfsmittel zur Fehlersuche und Analyse, denn "warum liefert mein View keine Daten?" ist ein klassischer Schmerzpunkt.

Vermeidung von "Cyclic Dependencies"

Wenn View A VON View B und View B VON View A referenziert, entsteht ein Zyklus, der Aktivierungen unmöglich macht. Solche Abhängigkeiten sollten vermieden werden, indem man klare Schichten definiert:

- Basis-Views nur auf Tabellen
- Operationale Views auf Basis-Views
- Konsum-Views auf Operationale

Regelmäßige Überprüfung mittels Dependency Analyzer ist wichtig.

Sizing & Komplexität: Wann ist ein View zu tief geschachtelt?

Tief geschachtelte Views führen oft zu schlechter Performance, da die generierte SQL-Abfrage sehr komplex wird. Regeln:

- Nicht mehr als drei Ebenen ohne guten Grund
- Fördere Wiederverwendbarkeit, aber nicht um jeden Preis
- Verwende `WITH PARAMETERS` statt dynamische Filter in jeder Ebene

Debugging & Fehleranalyse

Wenn ein View „keine Daten liefert“ oder plötzlich langsam wird, helfen folgende Werkzeuge:

- **HANA Plan Viz**: Analyse des SQL-Plans; zeigt, welche Joins/Filter wie teuer sind.
- **ST05 / SQL Trace**: Erkennen, welche SQL-Statements vom ABAP-Server erzeugt wurden.
- **Query Monitor (RSRT)**: Für analytische Views (OLAP) zeigt er Parameter, Filter und Performance-Werte.
- **Dependency Analyzer**: Prüft, ob eine View von einem Objekt abhängt, das geändert oder gelöscht wurde.

Kurze Diagnose-Tipps:

1. Im ADT mit *Data Preview* testen – liefert der interne SELECT schon keine Zeilen?
2. Den generierten SELECT per ST05 nachbauen und in HANA direkt ausführen.
3. Bei `@AccessControl`-Policies prüfen, ob die Session-Variablen erwartungsgemäß gefüllt sind.

Diese Schritte ersparen frustrierende Stunden und geben schnell Einblicke in die Ursache.

Troubleshooting-Checkliste (Kurzreferenz)

- **Aktivierungsfehler**: "View entity contains cyclical dependencies" bedeutet, dass zwei Views sich gegenseitig referenzieren; löse die Schichtordnung. Fehler beim CAST z.B. "Field length mismatch" erfordern Anpassung der Datentypen.

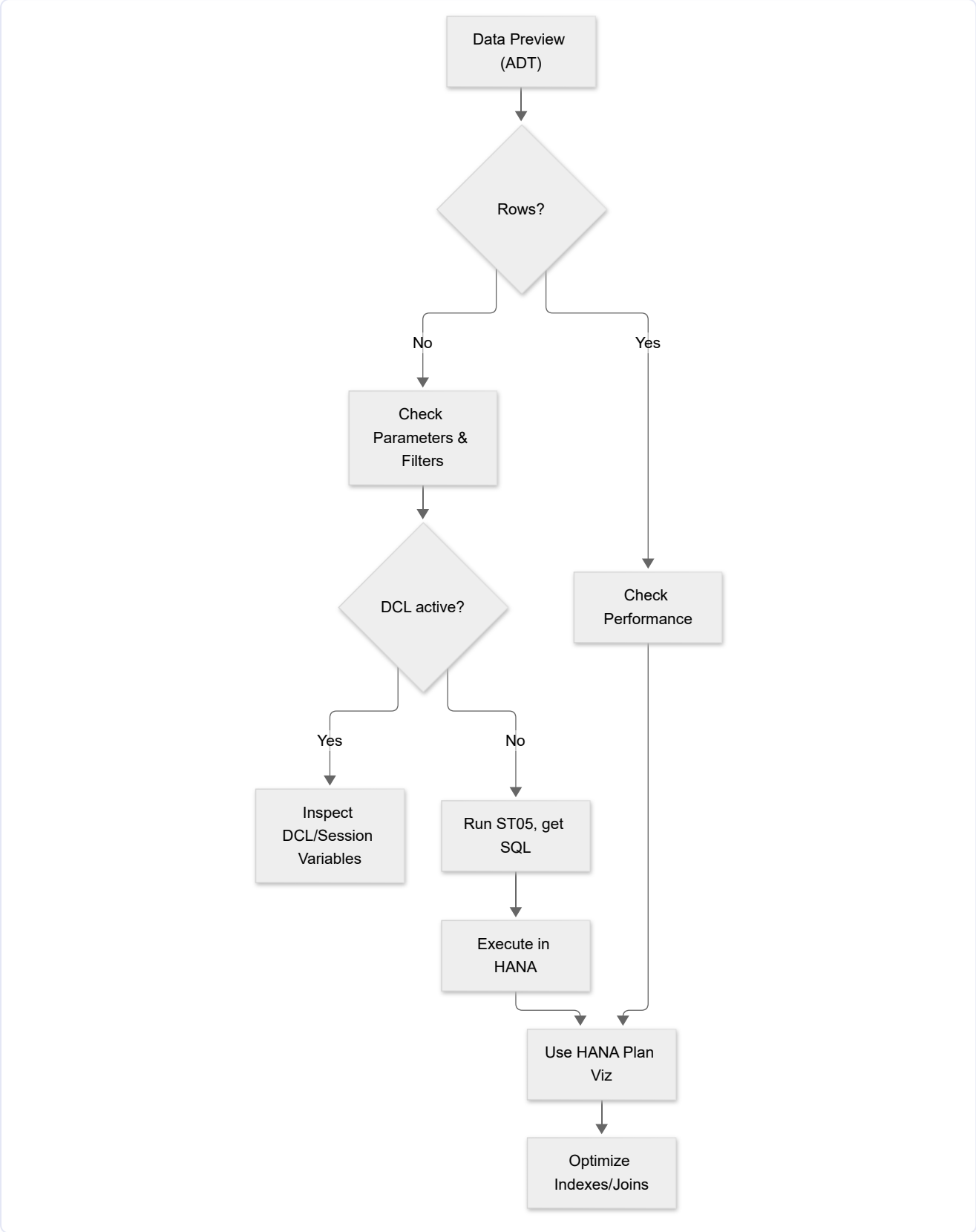
- **Shadowing/Namenskonflikte**: Vermeide, dass eine Association denselben Alias wie ein Feld hat; das führt zu Syntaxfehlern beim Aktivieren.

Troubleshooting-Checkliste (Kurzreferenz)

> **CDS-Test und Metadata Extensions**
>
> - Nutze das **CDS Test Double Framework**, um Views isoliert zu testen und Systemabhängigkeiten zu simulieren.
> - **Metadata Extensions (MDE)** erlauben eigene Annotationen, z.B. UI-Labels, auszulagern; ideal für Separation of Concerns.

1. Data Preview im ADT ausführen: Zeigt der View intern schon keine Zeilen? Wenn ja, weiter unten prüfen.
2. Parameter & Filter prüfen: Werden `WITH PARAMETERS` richtig übergeben? Sind Default-Werte aktiv?
3. Access Control prüfen: Sind DCL-Policies aktiv und schränken die Ergebnismenge ein?
4. Session-Variablen prüfen: `$session.user`, `$session.company` etc. sind korrekt gefüllt?
5. ST05/SQL Trace: Generierten SQL-String identifizieren und in HANA ausführen.
6. HANA Plan Viz: Expensive operators (HASH JOIN, TABLE SCAN) beobachten und Indizes prüfen.
7. Dependency Analyzer: Hat ein Objekt in der Chain Änderungen/Fehler?

Kurzdiagnose: Wenn der View lokal in HANA funktioniert, aber über ABAP leer bleibt, liegt die Ursache oft bei DCL oder Sessionvariablen. Wenn ABAP-SELECTs langsam sind, analysiere den Plan und achte auf Join-Reihenfolge und Datenvolumen.



Naming Conventions: Ordnung halten im VDM-Dschungel

Ein konsistentes Namensschema hilft bei der Navigation und Pflege:

- `ZV_` für Basis-Views (`#BASIC`)
- `ZVOP_` oder `ZC_` für Composite Views (`#COMPOSITE`)

- `ZVC_` für Consumption Views (`#CONSUMPTION`)
- Suffixe wie `_CDS`, `_IDX`

Diese Konventionen spiegeln die offiziellen VDM-Typen wider und erleichtern das Verständnis der Schichten in großen Projekten.

Dokumentation & Kommentare

Nutze `@EndUserText.label` und `@EndUserText.quickInfo` für lesbare Beschriftungen. Füge erklärende Kommentare über `/* ... */` hinzu, besonders bei komplexen Kalkulationen.

Performance-Optimierung (Anti-Patterns beachten)

> **Scalar Functions** Ab Releases 2022 und später können benutzerdefinierte HANA Scalar Functions (`CREATE FUNCTION`) in CDS eingebunden werden – ideal für komplexe Berechnungen, die nicht mit CAST/CASE abbildbar sind.

Performance-Optimierung (Anti-Patterns beachten)

Ein zusätzlicher Hebel ist das **Buffering**: CDS-Views können mit der Annotation `@AbapCatalog.buffering: #ALWAYS` oder `#ONCE_PER_CLIENT` gepuffert werden. Dies bedeutet, dass eine Kopie im ABAP-Buffer gehalten wird – sinnvoll für kleine, häufig gelesene Stammdatensichten. Bei großen Operativdaten sollte man sich eher auf HANA verlassen; Buffers erhöhen sonst die Workload und führen zu Synchronisationsproblemen.

- **SELECT * vermeiden:** Explizit Felder angeben
- **Joins minimieren:** Nutze Associations und nur bei Bedarf
- **Indexanalyse:** Stelle sicher, dass die zugrunde liegenden Tabellen passende Indizes haben
- **Analyse-Tools:** Verwende SQL-Trace (ST05) und den HANA Plan Viz für Optimierungen

> **Metrik:** Associations können Compile- und Laufzeit erheblich reduzieren, weil der Optimizer nur die benötigten Pfade ins SQL übersetzt. Statt jedes Mal einen großen Join zu erzeugen, wird nur bei Nutzung des assoziierten Feldes ein zusätzlicher Join eingefügt – das verkürzt die Kompilierzeit und verringert die Komplexität des erzeugten SQLs.

Programmatische Erstellung & KI-Ansätze für CDS

In modernen Entwicklungsszenarien (z. B. Massengenerierung von Views oder Migrationstools) stellt sich oft die Frage, ob CDS-Views auch ohne manuelles Tippen in Eclipse (ADT) erstellt werden können.

1. Programmatische Erstellung im SAP-System

Eine öffentliche REST-API gibt es nicht; alles läuft über die ADT-Services bzw. spezielle ABAP-Frameworks. CDS-Views dienen nicht nur OData-Services; sie sind generische Datenquellen, etwa für RESTful Application Programming Interfaces oder Third-Party-Integrationen.

Minimaler ABAP-XCO-Generator

```
CLASS zcl_cds_factory DEFINITION.
  PUBLIC SECTION.
    METHODS create_view
      IMPORTING
        iv_name  TYPE scdscds
        iv_table TYPE tabname.
```

```
ENDCLASS.

CLASS zcl_cds_factory IMPLEMENTATION.
  METHOD create_view.
    DATA(lo_view) = cl_xco_cds_view_entity=>create( iv_name = iv_name ).
    lo_view->set_sql_view_name( iv_name ).
    lo_view->add_element( iv_table ).
    lo_view->activate( ).
  ENDMETHOD.
ENDCLASS.
```

und ein schnelles **Python-Skript** für JSON→CDS:

```
import json

template = '''define view entity {viewName} as select from {baseTable} {{
  key {fields[0][name]},
  {fields[1][name]}
}}'''

def gen(json_file):
    data = json.load(open(json_file))
    print(template.format(**data))
```

- **XCO-Framework (ABAP Cloud & On-Premise)**
  - Modernster Weg im SAP-System, um Repository-Objekte inklusive CDS View Entities programmatisch anzulegen.
  - Typsicher, verwaltet Lebenszyklus (Anlegen, Prüfen, Aktivieren).
  - Beispielkonzept: Ein ABAP-Programm liest Konfigurationszeilen und erzeugt 50 verschiedene CDS-Views via `IF_XCO_CDS_VIEW_ENTITY`
  - Nachteil: Tiefe Kenntnisse der XCO-Klassen erforderlich.
- **ADT Backend APIs**
  - Das Eclipse-Plugin kommuniziert via HTTP/XML mit dem Backend. Diese APIs können (inoffiziell) dazu missbraucht werden, `.ddls`-Dateien an das System zu senden und zu aktivieren.
  - Risikofaktor: Nicht dokumentiert, können sich bei System-Upgrades ändern.

2. KI-Ansätze: JSON zu CDS

Weil CDS deklarativ ist, eignet sich die Sprache hervorragend für Generierung durch KI oder Template-Engines.

- **Mapping-Ansatz (JSON-to-CDS)**
  - Definiere ein JSON-Schema, das die ViewSemantik beschreibt.
  - Ein Skript (Python, Node.js, ABAP) wandelt das JSON in DDL um.

```
{
  "viewName": "Z_I_InvoiceQuery",
  "baseTable": "VBRK",
  "fields": [
    { "name": "vbeln", "alias": "InvoiceNumber", "label": "Rechnungsnummer" },
    { "name": "waerk", "alias": "Currency" }
  ],
  "associations": [
    { "name": "_Items", "target": "VBRP", "on": "vbeln = _Items.vbeln" }
  ]
}
```

- Der Generator erzeugt:

```
define view entity Z_I_InvoiceQuery as
  select from vbrk
  association [1..*] to vbrp as _Items on vbrk.vbeln = _Items.vbeln
  {
    key vbrk.vbeln,
    vbrk.waerk
  }
```

- **Generative KI (LLMs)**
  - Modelle wie GPT-4 oder spezialisierte Coding-Llamas können CDS-Syntax generieren, sobald man ihnen ein Tabellenschema und gewünschte Annotationen vorgibt.
  - Few-Shot-Prompting: Drei Beispiele für gute CDS-Views plus JSON-Schema als Input; das Modell fungiert als Übersetzer von fachlichen Anforderungen in DDL.

3. SAP Build & Low-Code

Für Nicht-Entwickler bietet SAP mit **SAP Build** Werkzeuge an, die im Hintergrund CDS-Modelle generieren, ohne dass der Nutzer eine Zeile DDL sieht. Dies ist der „programmatische" Ansatz der SAP für Fachanwender.

Fazit: Während die Aktivierung immer noch ein SAP-Backend (via XCO oder ADT) erfordert, findet die Definition zunehmend außerhalb statt – in JSON-basierten Modellierungstools, durch Skripte oder über KI-Assistenten.

References & Weiterführende Links

- SAP Help Portal — Core Data Services (CDS): https://help.sap.com/viewer/product/SAP%C2%AE%20S/4HANA/latest/en-US/ (Suche nach "Core Data Services")
- SAP Help Portal – ABAP RESTful Programming Model (RAP)
- ABAP Development Tools (ADT) – CDS-Editor: https://help.sap.com/docs/adt (Kapitel "Core Data Services")
- SAP Help – HANA PlanViz / Performance Analysis
- SAP Community Blog (XCO & ADT APIs)

Dieser Guide stellt einen umfassenden Einstieg in CDS-Views dar. Er erklärt die Architektur, Syntax, Annotationen, Berechtigungskonzepte und das moderne Tooling, das Entwicklern zur Verfügung steht. Die Best Practices schließen mit Ratschlägen zur Performance und Strukturierung eines stabilen, wartbaren Virtual Data Models.

Mit CDS hat SAP einen Mechanismus geschaffen, der nicht nur Daten modelliert, sondern die komplette Anwendungslogik in eine leicht konsumierbare Form bringt. Unternehmen, die diesen Gamechanger verstehen und richtig einsetzen, werden in der Lage sein, ihre S/4HANA-Systeme effizient zu betreiben und zukunftssicher weiterzuentwickeln.

Über den Autor

Erhard Rainer berät kleine und mittlere Unternehmen in Österreich zu Business Intelligence, Machine Learning und KI-Agenten. Er hat 15 Jahre Erfahrung in Data Analytics, davon 10 Jahre in der Leitung von Teams und Departments in Konzernen, und ist Dozent für betriebswirtschaftliche und Data-Analytics-Fächer.

Sprechen Sie mit mir über Ihre Daten: Das Erstgespräch ist kostenlos.

https://www.thinkbigger.at/contact.html#anfrage

© 2026 Think BIGGER – Erhard Rainer. Dieses PDF gibt den Stand des Online-Artikels vom 12.04.2026 wieder; maßgeblich ist die Online-Fassung unter https://www.thinkbigger.at/wissen/sap-cds-views/