📘 CDS Views and AMDP – Architectural Behaviour and Execution

For ABAP on HANA, S/4HANA, and RAP Developers

🔷 1. Introduction

In the SAP HANA era, **ABAP developers evolved from procedural logic builders to data architects**. Two key frameworks define this evolution:

- **Core Data Services (CDS)** — for **data modeling, abstraction, and declarative logic** on the database.
- **ABAP Managed Database Procedures (AMDP)** — for **imperative logic and complex computations** in HANA's native SQLScript.

While both aim to "push down" processing to the database, their architectural behavior, compilation, optimization, and runtime execution differ dramatically. Understanding these internal differences separates an *ABAP developer* from an *ABAP SME (Architect)*.

🔷 2. Conceptual Positioning in Architecture

Layer	CDS Views	AMDP
Definition Layer	Declarative SQL model (DDL source in ABAP repository)	Procedural SQLScript embedded in ABAP class
Runtime Container	SQL View generated in HANA	Stored Procedure generated in HANA
Execution Type	Lazy-evaluated, optimized by HANA View Engine	Explicit execution via method call
Abstraction	Semantic & Data modeling abstraction	Performance and logic abstraction
Usage Scope	Data modeling for OData, RAP, Analytics	Heavy data processing, aggregations, algorithmic logic

---

🔷 3. Compilation and Deployment Architecture

3.1 CDS Compilation Lifecycle

1. **ABAP DDL Source Creation**
   - Created using `@AbapCatalog.sqlViewName` and `@EndUserText.label`.
   - Stored in ABAP repository as a DDL source object (`.ddl`).
2. **DDL to SQL View Generation**

    - Upon activation, the DDL parser generates an equivalent **HANA SQL view**.
    - The ABAP Dictionary entry is created, linking the DDL source to a physical HANA view (`/1BCDS/<GUID>` or custom SQL view name).
3. **Dependency Graph Build**
    - CDS entities form a *dependency graph*; the activation process ensures top-down build sequence (base to composite views).
4. **HANA Optimization Phase**
    - The generated SQL view is parsed by **HANA's Calculation Engine**.
    - The optimizer rewrites the view into a **query plan tree**, eliminating redundant joins, flattening projections, and using **column pruning**.
5. **ABAP Integration**
    - At runtime, ABAP Open SQL directly consumes the CDS entity name (not the SQL view name).
    - The Data Dictionary and DDL source act as dual metadata providers.

📎 **Key Internal Mechanism:**
When you run:

```
SELECT * FROM z_i_sales INTO TABLE @lt_sales.
```

ABAP runtime looks up:

- The **ABAP Dictionary reference** for `Z_I_SALES`.
- Determines its underlying SQL view in HANA.
- Delegates execution to **DBI (Database Interface)** which calls **HANA's Query Engine**.
- HANA executes the *optimized graph* (not a naive SQL string).

3.2 AMDP Compilation Lifecycle

1. **ABAP Class Declaration**

```
CLASS zcl_sales_amdp DEFINITION PUBLIC.
  INTERFACES if_amdp_marker_hdb.
  METHODS get_top_sales FOR DATABASE PROCEDURE
    LANGUAGE SQLSCRIPT OPTIONS READ-ONLY
    USING vbak vbap.
ENDCLASS.
```

8. **Activation Phase**
    - The ABAP compiler validates syntax and converts SQLScript into a HANA procedure source.
    - Generated procedure name is typically:
    - `/1BCAMDP/<class>/<method>`
    - This is deployed to the HANA schema of the ABAP system user (usually `_SYS_BIC` or `_SYS_AFL`).
9. **Runtime Invocation**
    - When called in ABAP:
    - `lo_amdp->get_top_sales( EXPORTING iv_year = '2025' IMPORTING et_sales = lt_sales ).`
    - The ABAP runtime invokes the procedure directly via a **DBI call** using JDBC or native DB interface.

10. **Result Handling**
    - Data returned is **automatically mapped to ABAP structures**.
    - No implicit buffering or lazy loading – full result set transfer occurs.

🔷 4. Execution Behaviour at Runtime

CDS Execution Flow

```
ABAP Open SQL → DBI → HANA View Engine → Calculation Node Graph → Result
```

- **Lazy Execution:** CDS queries execute only when data is required (e.g., `SELECT` into internal table).
- **Optimized Join Handling:** HANA flattens nested views and performs join elimination.
- **Parameter Binding:** If CDS has parameters, they are passed as *bind variables*; HANA reuses execution plans for performance.
- **Annotations' Effect:**
    - `@AccessControl.authorizationCheck`: adds row-level filters.
    - `@Analytics.dataCategory`: informs optimizer about aggregation intent.
    - `@ObjectModel.foreignKey.association`: influences join path decisions.

AMDP Execution Flow

```
ABAP Call → DBI → Stored Procedure Engine → SQLScript Optimizer → Result
Set
```

- **Eager Execution:** The stored procedure executes immediately.
- **Intermediate Result Materialization:** Temporary tables may be created in HANA memory.
- **Parallelization:** SQLScript optimizer automatically distributes loops and joins across threads if possible.
- **Plan Cache Reuse:** If same parameters and structure, HANA reuses compiled execution plan.

🔷 5. Data Flow in System Architecture

```
[ABAP Layer]
   ↓  Open SQL or AMDP Call
[DBI Interface Layer]
   ↓
[HANA SQL/Calc Engine]
   ↓
[Column Store Processing]
   ↓
[Result Set → ABAP Internal Table]
```

CDS sits closer to the **data modeling** layer (passive, optimized, declarative), whereas AMDP sits in the **processing** layer (active, procedural, dynamic).

🔷 6. Performance Architecture

Feature	CDS View	AMDP
Optimizer Level	Query Optimizer (Cost-Based)	SQLScript Optimizer (Procedure-Level)
Result Caching	Possible via HANA calculation view caching	Not applicable (procedures are non-cacheable)
Execution Plan	Visible in HANA PlanViz (as calculation graph)	Visible in PlanViz (procedural flow)
Parallelism	Automatic join parallelization	Loop and cursor-level parallelization
Pushdown Degree	High for declarative logic	Highest for procedural heavy computation

---

🔷 7. Comparison Summary (Architectural Behaviour)

Dimension	CDS	AMDP
Nature	Declarative (What to do)	Imperative (How to do)
Reusability	High – used in multiple layers (OData, RAP, Analytics)	Medium – callable units
Optimization Control	HANA-controlled	Developer-controlled
Debugging	ABAP SQL trace (ST05)	SQLScript Debugger
Security Integration	DCL, Authorization Annotations	Manual SQL filters
Extensibility	via Extend View or Metadata Extension	via method redefinition or new AMDP
Maintenance Overhead	Low	High
Use Case	Data models, projections, associations	Intensive calculations, algorithmic logic

---

🔷 8. Integration in RAP (Restful ABAP Programming Model)

CDS in RAP

- Entity definitions (`@EndUserText.label`, `@OData.publish`, `@ObjectModel`) form **Root and Child entities**.
- Acts as **Data Source Provider (DSP)** for behavior definitions.

- Consumed by **Projection Views** and **Behavior Definition Implementations**.

AMDP in RAP

- Used in **Query Implementations** or **Custom Actions** where:
  - CDS cannot express logic efficiently.
  - SQLScript offers better performance (e.g., forecasting, predictive algorithms).

🔷 9. Advanced Internal Mechanisms

1. **Delta Propagation in CDS**
   - When CDS participates in Change Data Capture (CDC), HANA internally marks delta nodes using transaction metadata tables.
2. **CDS Dependency Graph Optimizer**
   - Detects redundant associations and flattens them for runtime efficiency.
3. **AMDP Table Function Binding**
   - AMDP can define **table functions** returning structured datasets for CDS to consume.
   - Hybrid architecture example:
   - `CDS → Table Function (AMDP) → HANA Logic → Return Dataset`
4. **Memory Pushdown**
   - Both CDS and AMDP operate directly on HANA column-store memory; no intermediate buffers exist in ABAP.
5. **Plan Reuse**
   - HANA stores query execution plans; similar CDS or AMDP executions reuse cached plans for faster response.

🔷 10. Real-World Performance Patterns

Scenario	Best Practice
Large joins (millions of records)	Use CDS with associations and filters early in path
Complex loops/calculations	Use AMDP (SQLScript)
Layered CDS models (Data → Interface → Consumption)	Keep <4 levels for optimizer efficiency
Parameterized reports	CDS Parameters with `@Consumption.filter`
Predictive algorithms	AMDP using HANA PAL or APL libraries

🔷 11. Pitfalls and Misconceptions

1. **Myth:** "CDS replaces AMDP."
   ✅ Reality: They complement each other — CDS models data; AMDP processes it.

2. **Myth:** “All CDS logic runs in ABAP.”
   ❌ False — execution is fully in HANA once SQL is generated.
3. **Pitfall:** Overlayered CDS models → cause **optimizer flattening overload**.
4. **Pitfall:** Using dynamic SQL in AMDP — breaks plan caching.
5. **Pitfall:** Forgetting to define `READ-ONLY` — may trigger unintended commit scopes.

🔷 12. Future Direction

- **CDS GraphQL Integration:** SAP roadmap includes CDS serving as backend for GraphQL APIs.
- **AMDP + AI Integration:** Future AMDPs can interface with embedded AI libraries in HANA Cloud (TensorFlow integration).
- **Cloud Optimization:** In ABAP Cloud, AMDP is restricted; CDS table functions and RAP queries dominate.

🔷 13. Final Summary

Aspect	CDS	AMDP
Core Purpose	Semantic data modelling	Complex procedural logic
Architectural Behaviour	Declarative, metadata-driven	Procedural, execution-driven
Execution Model	Compiled SQL View → Query Graph	Compiled SQLScript → Stored Procedure
Performance Anchor	HANA View Optimizer	SQLScript Engine Optimizer
Primary Consumer	RAP, OData, Analytics, BW	ABAP logic, Reporting, Predictive apps

---

🔷 14. Reference Architecture Diagram (Descriptive)