The Dynamics of AMDP – Advanced Runtime Flexibility and Adaptive SQLScript Design in S/4HANA On-Premise

Disclaimer :- Dynamic SQL incurs **runtime parsing cost** (milliseconds to seconds depending on query).
Hence, it should be used **only for logic that genuinely cannot be handled statically**.

---

1. Introduction – What "Dynamic" Means in AMDP Context

In traditional ABAP, "dynamic" means:

- Dynamic table names
- Dynamic field selection
- Dynamic WHERE or ORDER BY clauses

In AMDP (SQLScript within HANA), "dynamic" has **three levels**:

Level	Description	Example
L1 – Parametric	Parameters drive query logic	Filters, currency, company code
L2 – Conditional Logic	Control structures influence SQL generation	IF, CASE, LOOP
L3 – Dynamic SQL Execution	Entire SQL statements built and executed at runtime	EXECUTE IMMEDIATE / dynamic queries

Goal: achieve logic flexibility *without* regenerating or redeploying AMDP procedures.

---

2. Architectural Behaviour of Dynamic AMDP

Where Dynamics Are Processed

AMDP code is compiled at activation, but dynamic parts are executed **at runtime within the HANA SQLScript Engine**.
This means:

- Static SQL is optimized by the HANA compiler.
- Dynamic SQL is parsed and optimized *just before execution*.

Lifecycle View

```
ABAP → AMDP Call → SQLScript Engine
   → Dynamic Statement Build → Parse → Execute → Return Result
```

---

3. Dimensions of Dynamic Behavior in AMDP

3.1 Dynamic Parameterization

Parameters can be used anywhere — WHERE clauses, JOINS, aggregations, or even in function names.

Example:

```
METHOD get_data BY DATABASE PROCEDURE FOR HDB LANGUAGE SQLSCRIPT
  OPTIONS READ-ONLY USING mara.
  SELECT * FROM mara WHERE mtart = :iv_type;
ENDMETHOD.
```

Static structure, but dynamic execution.
No runtime SQL compilation required.

3.2 Dynamic Filtering and Sorting

To dynamically apply filtering, we can:

- Pass parameters as part of the query logic.
- Or construct filters dynamically using conditional logic.

Example:

```
IF :iv_filter_type = 'MATERIAL' THEN
    result = SELECT * FROM mara WHERE matnr LIKE :iv_value;
ELSEIF :iv_filter_type = 'DESCRIPTION' THEN
    result = SELECT * FROM mara WHERE maktx LIKE :iv_value;
END IF;
```

Performance-friendly: both branches precompiled.

3.3 Dynamic SQL Execution (Runtime Statement Construction)

This is the **most powerful and risky** form of dynamic behaviour.

Example:

```
DECLARE lv_sql NVARCHAR(5000);
lv_sql = 'SELECT * FROM ' || :iv_table || ' WHERE bukrs = ''' || :iv_bukrs
|| '''';
EXECUTE IMMEDIATE :lv_sql INTO result;
```

Behavior:

- The SQL string is created at runtime.
- HANA compiles and executes it dynamically.
- Returns a result set (requires predefined table type).

Use Cases:

- Cross-table analytics where table name changes (e.g., partitioned by year).
- Config-driven reporting logic.
- Runtime switching between views and tables.

Caution:
Dynamic SQL bypasses compile-time validation → syntax issues only at runtime.
To mitigate:

- Validate SQL parts in ABAP before passing to AMDP.
- Use `TRY...CATCH` to handle execution errors.

3.4 Dynamic Aggregations

Use Case: Aggregating different key figures based on input parameters.

```
IF :iv_kpi = 'SALES' THEN
    SELECT SUM(netwr) INTO lv_result FROM vbap;
ELSEIF :iv_kpi = 'MARGIN' THEN
    SELECT SUM(margin) INTO lv_result FROM zsales;
END IF;
```

AMDP allows complete logical branching with SQLScript IF statements.
Each branch is statically analyzed and optimized separately.

3.5 Dynamic Table Names and Metadata

While AMDP does not allow arbitrary dynamic table schema creation (for security),
it supports **runtime table name substitution** within controlled constructs.

Pattern Example:

```
lv_table = 'ZSALES_' || :iv_year;
lv_sql = 'SELECT * FROM "' || :lv_table || '"';
EXECUTE IMMEDIATE :lv_sql INTO result;
```

Used in **partitioned data models** (e.g., ZSALES_2024, ZSALES_2025).

3.6 Dynamic Column Selection

In HANA SQLScript, you cannot SELECT a variable column name directly.
Instead, dynamic SQL execution is used.

```
lv_sql = 'SELECT ' || :iv_column || ' FROM mara';
EXECUTE IMMEDIATE :lv_sql INTO result;
```

This is powerful for **metadata-driven frameworks**, where columns differ per configuration.

---

4. Real-World Dynamic AMDP Scenarios

Scenario 1 – Multi-Year Table Switch

Business Need:
Transactional data is stored in yearly partitioned tables like `ZGL_2023`, `ZGL_2024`, `ZGL_2025`. The logic should pick the right table dynamically.

AMDP Logic:

```
lv_table = 'ZGL_' || :iv_year;
lv_sql = 'SELECT * FROM "' || :lv_table || '" WHERE bukrs = :iv_bukrs';
EXECUTE IMMEDIATE :lv_sql INTO result;
```

Outcome:
No annual code deployment. The same AMDP runs across years dynamically.

Scenario 2 – Configuration-Driven Filters

Business Need:
ZCONFIG defines filter conditions (field, operator, value). AMDP reads config and dynamically applies filters.

```
FOR row AS SELECT * FROM zconfig DO
  lv_sql = 'SELECT * FROM mara WHERE ' || row.field || row.op || '''' ||
row.val || '''';
  EXECUTE IMMEDIATE :lv_sql INTO result;
END FOR;
```

Outcome:
Dynamic rule-based query engine. Perfect for generic validation frameworks.

Scenario 3 – Dynamic Join Between Modules

Business Need:
Join tables from FI or MM dynamically based on calling program context.

```
IF :iv_source = 'FI' THEN
    SELECT * FROM bseg INNER JOIN bkpf ON bseg.belnr = bkpf.belnr;
ELSEIF :iv_source = 'MM' THEN
    SELECT * FROM ekko INNER JOIN ekpo ON ekko.ebeln = ekpo.ebeln;
END IF;
```

Outcome:
One AMDP serves multi-module reporting requirements.

Scenario 4 – Dynamic CDS Table Function Selection

Business Need:
AMDP should pick which CDS table function (or underlying view) to use dynamically.

```
lv_view = CASE :iv_mode
             WHEN 'PURCHASE' THEN 'ZI_PURCHASE_VIEW'
             WHEN 'SALES' THEN 'ZI_SALES_VIEW'
           END;
lv_sql = 'SELECT * FROM "' || :lv_view || '"';
EXECUTE IMMEDIATE :lv_sql INTO result;
```

Outcome:
Same logic, different analytical datasets — controlled by configuration.

Scenario 5 – Dynamic Field List for Generic Reports

Business Need:
Users define which fields to include via configuration. AMDP constructs dynamic SELECT list.

```
lv_fieldlist = 'matnr, mtart, ' || :iv_extra_fields;
lv_sql = 'SELECT ' || :lv_fieldlist || ' FROM mara';
EXECUTE IMMEDIATE :lv_sql INTO result;
```

Outcome:
Generic report builder using one AMDP across multiple datasets.

Scenario 6 – Dynamic Aggregation Key

Business Need:
Aggregation key changes by business function — customer, material, region.

```
lv_group = CASE :iv_key WHEN 'CUST' THEN 'kunnr' WHEN 'MAT' THEN 'matnr'
ELSE 'vkorg' END;
lv_sql = 'SELECT ' || :lv_group || ', SUM(netwr) AS total FROM vbap GROUP
BY ' || :lv_group;
EXECUTE IMMEDIATE :lv_sql INTO result;
```

Outcome:
Highly flexible reporting without separate AMDPs per KPI.

Scenario 7 – Dynamic Table Join from Configuration Table

Business Need:
Table relationships are stored in ZJOIN_CONFIG (source, target, join field).

```
SELECT * INTO TABLE @DATA(lt_join) FROM zjoin_config;
FOR row AS SELECT * FROM :lt_join DO
  lv_sql = 'SELECT * FROM ' || row.source || ' INNER JOIN ' || row.target
||
            ' ON ' || row.source || '.' || row.field || ' = ' || row.target
|| '.' || row.field;
  EXECUTE IMMEDIATE :lv_sql INTO result;
END FOR;
```

Outcome:
Dynamic model builder — no static dependency on tables.

Scenario 8 – Dynamic WHERE Construction from ABAP Table Input

Business Need:
Pass 100 dynamic conditions from ABAP table (field, value) into AMDP.

```
FOR row AS SELECT * FROM :it_conditions DO
  lv_where = lv_where || row.field || ' = ''' || row.value || ''' AND ';
END FOR;
lv_sql = 'SELECT * FROM mara WHERE ' || LEFT(:lv_where, LENGTH(:lv_where)-
4);
EXECUTE IMMEDIATE :lv_sql INTO result;
```

Outcome:
Query changes per user input at runtime.

Scenario 9 – Dynamic Calculation Formula

Business Need:
Business rules define formula at runtime (e.g., profit = revenue - cost).

```
lv_formula = 'revenue - cost';
lv_sql = 'SELECT kunnr, (' || :lv_formula || ') AS profit FROM zsales';
EXECUTE IMMEDIATE :lv_sql INTO result;
```

Outcome:
Rule-based, formula-driven KPI engine.

Scenario 10 – Multi-Company Schema Access

Business Need:
Each company code data resides in separate schema.

```
lv_schema = 'ZSCHEMA_' || :iv_bukrs;
lv_sql = 'SELECT * FROM "' || :lv_schema || '"."zfinance"';
EXECUTE IMMEDIATE :lv_sql INTO result;
```

Outcome:
Single AMDP that accesses multiple schemas dynamically.

Scenario 11 – Dynamic Pivot Generation

Business Need:
Convert rows to columns dynamically for report output.

```
lv_sql = 'SELECT kunnr, SUM(CASE WHEN region=''APAC'' THEN netwr ELSE 0
END) AS APAC,
                  SUM(CASE WHEN region=''EU'' THEN netwr ELSE 0 END) AS
EUROPE
           FROM vbak GROUP BY kunnr';
EXECUTE IMMEDIATE :lv_sql INTO result;
```

Outcome:
Pivot generation inside HANA layer — replaces ABAP ALV pivot.

Scenario 12 – Dynamic Ranking Criteria

Business Need:
Users decide ranking metric (sales, margin, profit).

```
lv_sql = 'SELECT kunnr, ' || :iv_metric || ', RANK() OVER (ORDER BY ' ||
:iv_metric || ' DESC) AS rank FROM zsales';
EXECUTE IMMEDIATE :lv_sql INTO result;
```

Outcome:
Single AMDP supports multiple ranking KPIs.

Scenario 13 – Dynamic Exception Rule Evaluation

Business Need:
Validation rules differ by business area; each rule is stored as SQL condition in ZRULES.

```
FOR row AS SELECT * FROM zrules WHERE active = 'X' DO
  lv_sql = 'SELECT COUNT(*) AS cnt FROM ' || row.tab || ' WHERE ' ||
row.rule_cond;
  EXECUTE IMMEDIATE :lv_sql INTO result;
  IF result.cnt > 0 THEN
    INSERT INTO zlog VALUES (:row.tab, :row.rule_cond, result.cnt);
  END IF;
END FOR;
```

Outcome:
Dynamic data quality framework fully configurable.

Scenario 14 – Dynamic CDS Exposure from AMDP

Business Need:
Expose AMDP logic via CDS table function whose logic changes by parameter.

AMDP Implementation:

```
IF :iv_mode = 'FIN' THEN
   result = SELECT * FROM zfi_data;
ELSE
   result = SELECT * FROM zmm_data;
END IF;
```

Outcome:
One CDS table function serves multiple business contexts.

Scenario 15 – Dynamic Data Validation Framework

Business Need:
Run data validation rules dynamically based on entries in ZVALIDATION_CONFIG.

```
FOR row AS SELECT * FROM zvalidation_config WHERE active='X' DO
  lv_sql = 'SELECT COUNT(*) AS err_count FROM ' || row.table_name || '
WHERE ' || row.condition;
  EXECUTE IMMEDIATE :lv_sql INTO result;
  INSERT INTO zval_log VALUES (:row.table_name, :row.condition,
result.err_count);
END FOR;
```

Outcome:
Zero code changes when new validation added.

Scenario 16 – Dynamic UNION Construction

Business Need:
Combine multiple similar tables dynamically into one dataset.

```
FOR row AS SELECT tabname FROM zunion_source DO
  lv_sql = lv_sql || 'SELECT * FROM ' || row.tabname || ' UNION ALL ';
END FOR;
lv_sql = LEFT(:lv_sql, LENGTH(:lv_sql)-10);
EXECUTE IMMEDIATE :lv_sql INTO result;
```

Outcome:
Scalable consolidation logic without code duplication.

Scenario 17 – Dynamic Lookup Based on Master Key

Business Need:
Perform lookups in variable tables depending on master data category.

```
CASE :iv_type
  WHEN 'MATERIAL' THEN SELECT * FROM mara;
  WHEN 'CUSTOMER' THEN SELECT * FROM kna1;
  WHEN 'VENDOR' THEN SELECT * FROM lfa1;
END CASE;
```

Outcome:
Adaptive lookup framework integrated in validation AMDP.

Scenario 18 – Dynamic Top-N Analysis

Business Need:
Extract top N records dynamically based on user-selected field.

```
lv_sql = 'SELECT ' || :iv_field || ', SUM(netwr) AS val FROM vbap GROUP BY
' || :iv_field ||
          ' ORDER BY val DESC LIMIT ' || :iv_topn;
EXECUTE IMMEDIATE :lv_sql INTO result;
```

Outcome:
Top-N ranking configurable per user input.

Scenario 19 – Dynamic Audit Trail Query

Business Need:
Dynamic selection of history tables for audit reports (e.g., MARA_H, VBAP_H).

```
lv_sql = 'SELECT * FROM "' || :iv_table || '_H" WHERE changenr > :iv_last';
EXECUTE IMMEDIATE :lv_sql INTO result;
```

Outcome:
Single AMDP caters to multiple audit trail tables.

Scenario 20 – Dynamic Exception Trigger Based on Metadata

Business Need:
Trigger alerts based on dynamically constructed SQL conditions.

```
FOR row AS SELECT * FROM zalert_rules DO
  lv_sql = 'SELECT COUNT(*) FROM ' || row.table_name || ' WHERE ' ||
row.condition;
  EXECUTE IMMEDIATE :lv_sql INTO result;
  IF result.COUNT > 0 THEN
     INSERT INTO zalerts VALUES (:row.table_name, :row.condition,
result.COUNT, CURRENT_TIMESTAMP);
  END IF;
END FOR;
```

Outcome:
Self-learning monitoring logic driven entirely by configuration.

Bonus: Pattern-Level Insights

Use Case Type	Dynamic Technique	Typical Tables
Multi-year or partitioned data	Dynamic table name	Z*, BKPF, VBAP
Validation rules	Dynamic condition builder	ZRULES, ZCONFIG
KPI Framework	Dynamic formula & aggregation	VBAK, CE1*
Reporting	Dynamic field list	Custom views, CDS
Monitoring	Dynamic SQL evaluation	ZALERTS, ZLOG

Key Learning Summary

- **Dynamic AMDP** lets you design *generic, reusable, self-evolving logic*.
- **Static validation** and **error handling** are crucial to prevent runtime failures.
- Proper architecture ensures **no performance penalty**, even with runtime SQL construction.
- Think of AMDP dynamics as **configurable logic layers**, not “string hacks.”

---

5. Balancing Dynamics and Optimization

Dynamic SQL disables **plan caching** — each unique string compiles separately. Hence, for performance:

- Keep dynamic parts minimal (WHERE conditions, not full SELECT).
- Use parameterized placeholders (: syntax) wherever possible.
- Avoid generating too many SQL variants.

Best Practice Pattern:

```
lv_sql = 'SELECT * FROM mara WHERE matnr LIKE ?';
EXECUTE IMMEDIATE :lv_sql USING :iv_matnr INTO result;
```

Parameter substitution keeps query plan reusable.

---

6. Error Handling for Dynamic AMDP

Common Error Patterns

Issue	Cause	Solution
SQL: invalid identifier	Incorrect runtime column/table	Validate before EXEC
SQL: unexpected end of command	Missing quotes or delimiters	Use `
SQL: insufficient privilege	Cross-schema access	Ensure proper GRANTs
Empty result	No matching data	Always check `ROWCOUNT`

---

7. Hybrid Dynamic Model (CDS + AMDP)

A hybrid pattern often used in real projects:

```
CDS Table Function → AMDP → Dynamic SQL
```

- CDS exposes a fixed structure to ABAP / RAP layer.
- AMDP internally handles runtime logic dynamically.
- Result: semantic stability + execution flexibility.

---

8. Advanced Dynamic Techniques

Dynamic View Switching

Based on runtime role or configuration:

```
CASE :iv_role
  WHEN 'FINANCE' THEN SELECT * FROM zfi_view;
  WHEN 'LOGISTICS' THEN SELECT * FROM zmm_view;
```

```
END CASE;
```

Dynamic Function Invocation

```
lv_function = 'ZFN_' || :iv_region;
lv_sql = 'CALL "' || :lv_function || '"()';
EXECUTE IMMEDIATE :lv_sql;
```

Dynamic Column Pivot

Transform rows into columns dynamically using EXEC and string aggregation.

---

9. Governance and Security in Dynamic AMDP

Dynamic SQL increases risk:

- **SQL injection potential** — always sanitize parameters.
- **Authorization bypass** — dynamic queries can access unintended objects.
- **Transport unpredictability** — dynamic objects may not exist in target system.

Always:

- Limit dynamic table names to known patterns.
- Validate every user input in ABAP layer before passing to AMDP.
- Log every dynamic statement in a custom trace table (ZAMDP_LOG).

---

10. Performance and Maintainability Matrix

Dynamic Level	Flexibility	Performance	Maintainability
Static SQL	Low	Best	Best
Conditional SQL	Medium	Very Good	Good
Dynamic SQL (EXEC)	Highest	Medium–Low	Complex

Rule of thumb:

Use dynamic SQL only where configuration drives logic. Else, use parameterization.

---

11. Conclusion – Controlled Chaos

AMDP dynamics are like a double-edged sword:
they empower the system to adapt **without redevelopment**, but they also challenge optimization and maintainability.

A true ABAP SME understands:

- When to leverage **runtime dynamics** for business flexibility.
- When to enforce **static design** for stability and caching.

Dynamic AMDPs, when architected properly, create **self-adapting, configuration-driven ABAP systems** — the foundation of modern, intelligent S/4HANA design