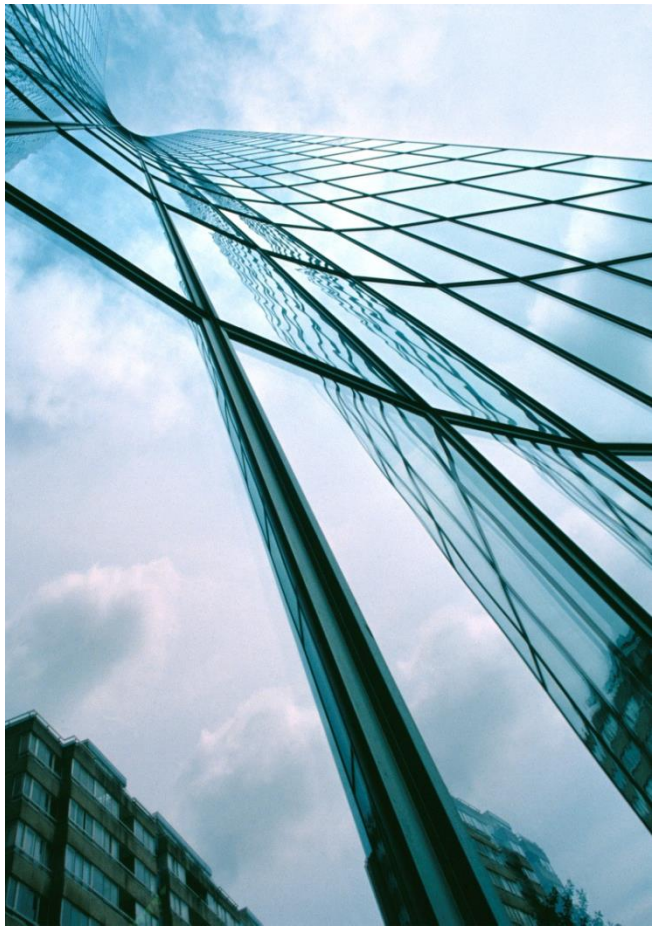
ABAP on SAP HANA

SAP Active Global Support

Welcome

to day 1 of the Workshop

ABAP on SAP HANA



What can you Expect…

- to understand the ABAP for SAP HANA
- to familiarize yourself with ABAP and SAP HANA Eclipse tools
- to gain some insights into ABAP tools e.g. debugger, profiling etc.
- to learn about modeling in SAP HANA
- to learn calling different HANA artifacts from ABAP using OPEN SQL
- to apply your knowledge in several exercises / an E2E example

- to find the answer to ‘all your SAP HANA questions’ ☺
- to learn about all (new) features during the workshop in detail
- to become an SAP HANA expert (modeling / development)
- to hear anything specific about the ‘Suite on HANA’ project

Agenda

Lessons	Topics
Welcome	Welcome, Introduction of Participants, Review Agenda and Topics
Introduction	ABAP on HANA flavors
ABAP Development tools (ADT)	Introduction to Studio, Introduction to ABAP in Eclipse, Writing simple ABAP reports in HANA studio, Features in ADT, ABAP Trace in HANA studio, Debugging in ADT
Exercise – Profiling and Debugging	Profile and debug a simple ABAP program using ADT
Golden Rules, Patterns and Anti-Patterns	Understanding the “Golden Rules” and Patterns & Anti-patterns
Internal tables	Optimized access on internal tables
Code Inspector	New checks in the code inspector for HANA
SQL Monitor and Work list (SWLT)	New tools - SQLM and SWLT to find the ABAP codes which need to be adapted/optimized for HANA
ABAP Database connectivity -ADBC	Why is ADBC needed, Usage of ADBC

Agenda

Lessons	Topics
Modeling/Code push down – Attribute view	Introduction to Attribute views, Calculated column, External views
Demos + Exercise	**Modeling, ABAP**: Creating a Attribute View to calculate the DAYS OPEN (difference between todays' date and date on which SO was created) for each SO. Accessing the attribute view with OPEN SQL.
Demos	**Performance:** Open days in ABAP v/s Open days in HANA
Modeling/Code push down – Analytical view	Introduction to Analytical view
Demos + Exercise	**Modeling , ABAP**: Creating Analytical view to calculate gross amount per business partner in USD. Accessing the analytical view through ADBC and open SQL.
Demos	**Performance:** Currency conversion in ABAP v/s Currency conversion in HANA
Modeling/Code push down – Calculation view	Introduction to Calculation view – graphical and scripted

Agenda

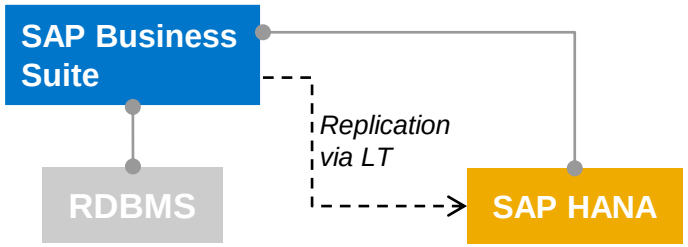
Lessons	Topics
Demos (Optional Exercise)	**Modeling**: Creating Calculation views to mark specific Business Partners as SPECIAL based on predefined conditions (Scripted and Graphical) Accessing the calculation view through open SQL.
Code push down – DB procedure	Introduction to database procedures, DB procedure proxies
Demos + Exercise	**Code Push down**: Writing DB procedure to find the Business partners with highest 5 and bottom 5 Gross Revenue
Demos+ Exercise	**Code Push down, ABAP**: Creating DB procedure Proxy and calling DB procedures from ABAP code.

Introduction

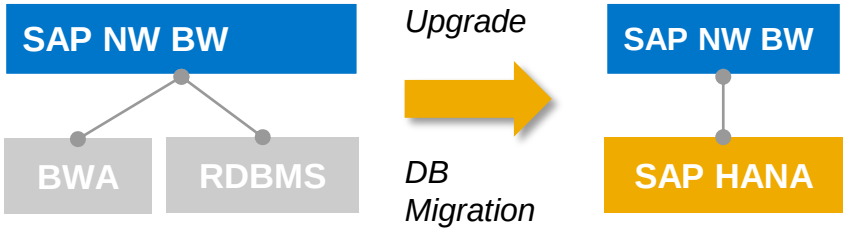
ABAP on HANA flavors

Main application scenarios (On Premise, AS ABAP 7.x)

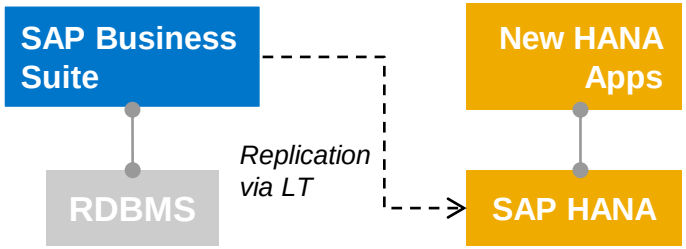
„Accelerators“



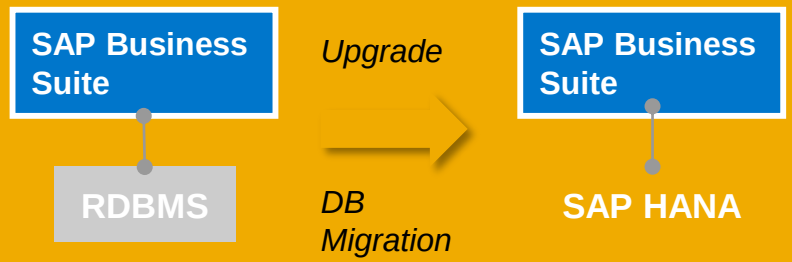
„BW on HANA“



„HANA Apps“



„ERP on HANA“



Side-by-Side ***Primary Persistence***

Golden Rules and Patterns & Anti-patterns

DAY 1
Learning Objectives: Golden Rules and Patterns & Anti-patterns

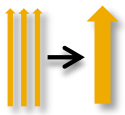
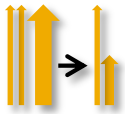
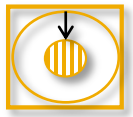
After completing this unit you will be able to:

- Explain the “golden rules” in ABAP
- Identify Anti-Patterns and ways to avoid/correct them.

Golden Rules

- Existing performance guidelines for traditional databases (as defined in the product standard “Performance”, e.g.) are mostly still valid for HANA. There is only a shift in priority for certain rules.
- No major change from ABAP performance tuning perspective on HANA. Do what always should have been done.
- Golden Rules / SQL best practices, are summarized on the following pages.

Golden Rules – Overview

	Minimize the number of round trips	• Avoid nested Selects by using Joins, Subqueries, or SELECT FOR ALL ENTRIES • Use array variants of INSERT, UPDATE, MODIFY, and DELETE
	Minimize amount of transferred data	• Don't read unused columns using SELECT *, but specify the required fields using a field list. • Use aggregate functions (COUNT, MIN, MAX, SUM, AVG) instead of transferring all the rows to the application server
	Keep the result set small	• Don't read records from the database that you don't need in your application. Restrict the data set by an appropriate WHERE clause.
	Keep load away from the database	• Avoid reading data redundantly • Use table buffering (if applicable) and do not bypass it • Consider sorting in ABAP
	Minimize the search overhead	• Define and use appropriate indexes

SQL Statements
Best Practices

In General, “golden rules” for optimizing SQL statements have to be applied:

- Keep the result set small
  - Don’t retrieve rows from database and discard them on application server
  - Make the “Where” clause as specific as possible
  - Check for empty driver table before each FOR ALL ENTRIES.

- Minimize amount of transferred data
  - Use Select with field list
  - Use aggregate functions instead of transferring everything on application server
  - Check for duplicates in driver table before each FAE.

SQL Statements
Best Practices

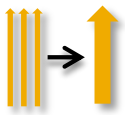
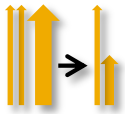
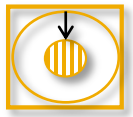
- Reduce the number of round trips
  - Use Joins or Subqueries instead of nested SELECT loops
  - Use For All Entries instead of lots of SELECTs or SELECT SINGLEs.
  - Use Array variants of INSERT, UPDATE, MODIFY, and DELETE

- Keep the load away from the Database
  - Avoid reading data redundantly
  - Use table buffering (if possible). DO NOT BYPASS BUFFERS
  - SORT in ABAP or HANA based on the number of records to be sorted

SQL Statements
Best Practices

- Minimize the search overhead
  - Define and use appropriate indexes.

Golden Rules – Overview

	Minimize the number of round trips	• Avoid nested Selects by using Joins, Subqueries, or SELECT FOR ALL ENTRIES • Use array variants of INSERT, UPDATE, MODIFY, and DELETE
	Minimize amount of transferred data	• Don’t read unused columns using SELECT *, but specify the required fields using a field list. • Use aggregate functions (COUNT, MIN, MAX, SUM, AVG) instead of transferring all the rows to the application server
	Keep the result set small	• Don’t read records from the database that you don’t need in your application. Restrict the data set by an appropriate WHERE clause.
	Keep load away from the database	• Avoid reading data redundantly • Use table buffering (if applicable) and do not bypass it • Consider sorting in ABAP
	Minimize the search overhead	• Define and use appropriate indexes

Rule: Keep the result set small

Anti Patterns: No or not selective where clause

- In order to minimize the number of data records which have to be transferred, it is necessary to specify a selective WHERE clause in each SQL statement.

- Don’t check conditions on application layer
  → always specify WHERE conditions !

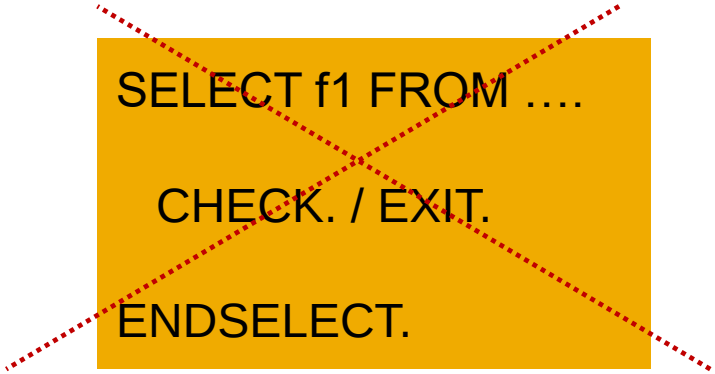
```
SELECT * FROM table.
IF itab-field1 = ‘ABC’.
…
ENDIF.
ENDSELECT.
```

```
SELECT * FROM table WHERE field1 = ‘ABC’.
```

Rule: Keep the result set small

Anti Patterns: EXIT, CHECK in SELECT

- No CHECKs or EXITs should be used in SELECT …ENDSELECT loops.



- All filters should be pushed down to the Database as early as possible to avoid fetching data which is not needed.

```
SELECT f1 FROM … WHERE …
  INTO...
```

Rule: Keep the result set small

Anti Patterns: Filtering data too late

- Do not Select the all data from the database and delete later from the internal table.

```
SELECT…. FROM dbtab WHERE….INTO itab.
……
……
LOOP AT itab.
   IF…..
   DELETE… itab.
   ENDIF.
ENDLOOP.
```

- All filters should be pushed down to the Database as early as possible to avoid fetching data which is not needed.

Rule: Keep the result set small

Anti Patterns: Empty FAE driver Table

- Empty FAE table means that the complete where clause disappears. If the FAE driver table is empty, then NO WHERE condition is generated. Thus ALL ROWS will be selected (full table scan).
- Even for additional fields in the WHERE conditions, not referencing the driver table , no WHERE condition will be generated

```
SELECT carrid connid fldate seatsocc
   FROM sflight
   INTO TABLE seatsocc_tab (target
table)
    FOR ALL ENTRIES IN conn_tab
(driver table)
    WHERE carrid = conn_tab-carrid
      AND connid = conn_tab-connid.
```

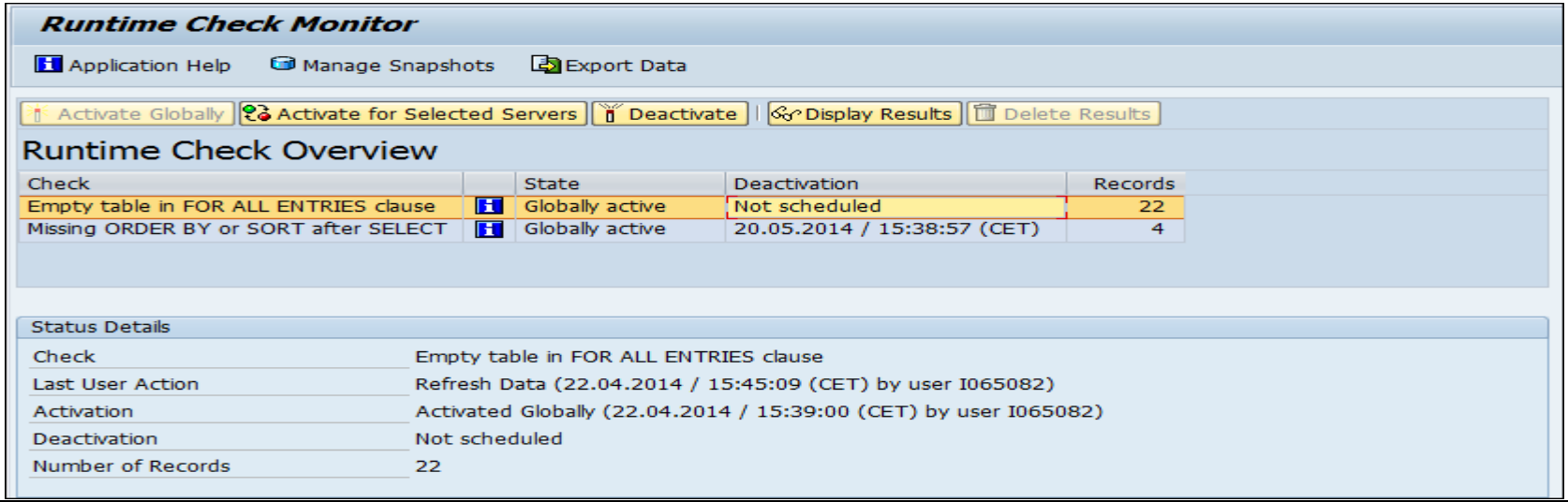
IF conn_tab (driver table) is not initial .

```
SELECT carrid connid fldate seatsocc
   FROM sflight
   INTO TABLE seatsocc_tab (target
table)
   FOR ALL ENTRIES IN conn_tab
(driver table)
   WHERE carrid = conn_tab-carrid
     AND connid = conn_tab-connid.
```

Rule: Keep the result set small

Anti Patterns: Empty FAE driver Table (TA SRTCM)

- There is Runtime Check Monitor available (SAP T-code: SRTCM) which can capture the empty FAE executions.
- Since the checks are performed at Kernel level, the overall system performance is not affected by activating these checks.
- To activate the check:
  - Open T-code SRTCM
  - Select a particular check
  - *Activate it on a particular server or Globally.*



Rule: Keep the result set small

Anti Patterns: Empty FAE driver Table (TA SRTCM)

- To check the results:
  - ➢ Double click on the check for which you want to see the results.
  - ➢You can select the range of development objects that are suitable for analysis of runtime check monitoring data using various criteria. For example, you can limit the development objects to a particular set of packages or a particular object type.

Results for Check "EMPTY_4ALLNTRIES": 10 Records

Occs.	Last Occ. Date	Occ. Time	Package	Obj. Type	Object Name	Include	Incl. Line	Changed
9.182	22.04.2014	15:35:38	$TMP	PROG	Z_PERFORMANCE	Z_PERFORMANCE	42	
3	16.02.2014	12:31:40	$TMP	PROG	Z_PERFORMANCE	Z_PERFORMANCE	47	
2	06.11.2013	17:52:59	Z_VIC_MY_PACK	PROG	Z_VIC_TASK_2	Z_VIC_TASK_2	75	
2	06.11.2013	17:52:59	Z_VIC_MY_PACK	PROG	Z_VIC_TASK_2	Z_VIC_TASK_2	90	
2	07.11.2013	10:20:08	Z_VIC_MY_PACK	PROG	Z_VIC_TASK_3	Z_VIC_TASK_3	73	
2	07.11.2013	10:20:08	Z_VIC_MY_PACK	PROG	Z_VIC_TASK_3	Z_VIC_TASK_3	88	
4	15.02.2014	19:13:18	Z_VIC_MY_PACK	PROG	Z_VIC_TASK_BAD_SAMPLE	Z_VIC_TASK_BAD_SAMPLE	51	
4	15.02.2014	19:13:18	Z_VIC_MY_PACK	PROG	Z_VIC_TASK_BAD_SAMPLE	Z_VIC_TASK_BAD_SAMPLE	81	
5	20.04.2014	15:35:45	ZMA_APRIL	PROG	ZMA_FAE_SNDW_SO	ZMA_FAE_SNDW_SO	20	
7	14.04.2014	14:36:52	ZMA_APRIL	PROG	ZMA_SFLIGHT_0	ZMA_SFLIGHT_0	20	

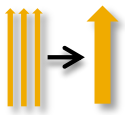
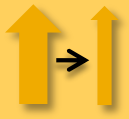
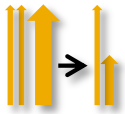
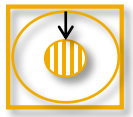
- On the results screen, you can use the hotspot links in order to directly navigate to the relevant source code position.

Rule: Keep the result set small

Anti Patterns: Empty FAE driver Table (TA SRTCM)

- Availability :
  - Transaction SRTCM is available with SAP BASIS 740, SP5. #
  - It can be downported to SAP BASIS 740 SP2, SP3,SP4. Refer to SAP Note 1931870 - Downport of transaction SRTCM to SP02 / 03 / 04.

Golden Rules – Overview

	Minimize the number of round trips	• Avoid nested Selects by using Joins, Subqueries, or SELECT FOR ALL ENTRIES • Use array variants of INSERT, UPDATE, MODIFY, and DELETE
	Minimize amount of transferred data	• Don’t read unused columns using SELECT *, but specify the required fields using a field list. • Use aggregate functions (COUNT, MIN, MAX, SUM, AVG) instead of transferring all the rows to the application server
	Keep the result set small	• Don’t read records from the database that you don’t need in your application. Restrict the data set by an appropriate WHERE clause.
	Keep load away from the database	• Avoid reading data redundantly • Use table buffering (if applicable) and do not bypass it • Consider sorting in ABAP
	Minimize the search overhead	• Define and use appropriate indexes

Rule: Minimize amount of transferred data

Anti Patterns: SUM, MIN, MAX, AVG etc.. DB content in ABAP

- Do not fetch data to ABAP level and then perform aggregations.

```
SELECT …
     FROM table …
     WHERE …
  sum = sum + table-amount.
ENDSELECT.
```

- Use aggregate functions (like COUNT, SUM, MAX, …) to perform calculations on database layer.

```
SELECT SUM(amount) …
   FROM table
   WHERE…
```

Rule: Minimize amount of transferred data

Anti Patterns: SUM, MIN, MAX, AVG etc.. DB content in ABAP

- Aggregate functions are often used together with GROUP BY.
- HANA shows its greatest strength when using operations like GROUP BY. Transferring calculations and aggregations to HDB will increase overall performance and can be very efficient for memory.

```
SELECT c1 max( c2 ) min( c3 ) FROM table
WHERE … GROUP BY c1
```

- *Keep in mind that buffered tables should be read from the SAP table buffer*.
- *SELECT with aggregate function (COUNT, MIN, MAX, AVG, SUM, GROUP BY, HAVING) bypass the buffer. These are detected by SCI.*

Rule: Minimize amount of transferred data

Anti Patterns: FAE driver table with duplicate entries

- The result set of FAE is always distinct.
- The DBI and ABAP runtime guarantee a distinct result set.
- After ALL records have been selected, the duplicates are removed in DBI before the final distinct result set is transferred back to ABAP.
- Duplicates in driver FAE table thus leads to unnecessary fetching of more (redundant/ duplicate) records from the DB which are later deleted by DBI before giving the data to ABAP.

Rule: Minimize amount of transferred data

Anti Patterns: FAE driver table with duplicate entries

- This FOR ALL ENTRIES

```
SELECT carrid connid fldate seatsocc FROM sflight
    INTO TABLE seatsocc_tab (target table)
    FOR ALL ENTRIES IN conn_tab (driver table)
    WHERE carrid = conn_tab-carrid.
```

- has the same effect as

```
SELECT DISTINCT carrid connid fldate seatsocc
  FROM sflight
  INTO TABLE seatsocc_tab
  WHERE
   ( carrid IN (<row 1 of conn_tab>-carrid, <row 2 of
conn_tab>-carrid,… , <row n of conn_tab>-carrid))
```

Rule: Minimize amount of transferred data

Anti Patterns: FAE driver table with duplicate entries

- This FOR ALL ENTRIES

```
SELECT carrid connid fldate seatsocc FROM sflight
    INTO TABLE seatsocc_tab (target table)
    FOR ALL ENTRIES IN conn_tab (driver table)
    WHERE carrid = conn_tab-carrid
      AND connid = conn_tab-connid.
```

- has the same effect as

```
SELECT DISTINCT carrid connid fldate seatsocc
  FROM sflight
  INTO TABLE seatsocc_tab
  WHERE
   ( carrid = <row 1 of conn_tab>-carrid AND
     connid = <row 1 of conn_tab>-connid)
   OR
   ( carrid = <row 2 of conn_tab>-carrid AND
     connid = <row 2 of conn_tab>-connid)
   OR ...
   OR
   ( carrid = <row n of conn_tab>-carrid AND
     connid = <row n of conn_tab>-connid ).
```

Rule: Minimize amount of transferred data

Anti Patterns: FAE driver table with duplicate entries

- Always delete duplicates from the driver table before using it in FAE.

```
SORT (driver table) BY carrid.

DELETE ADJACENT DUPLICATES FROM (driver table)
COMPARING carrid.

SELECT carrid connid fldate seatsocc FROM sflight
    INTO TABLE seatsocc_tab (target table)
    FOR ALL ENTRIES IN conn_tab (driver table)
    WHERE carrid = conn_tab-carrid.
```

Rule: Minimize amount of transferred data

Anti Patterns: SELECT of unneeded columns (select *)

- When using Select *, unnecessary data/columns are transferred from database server to application server.

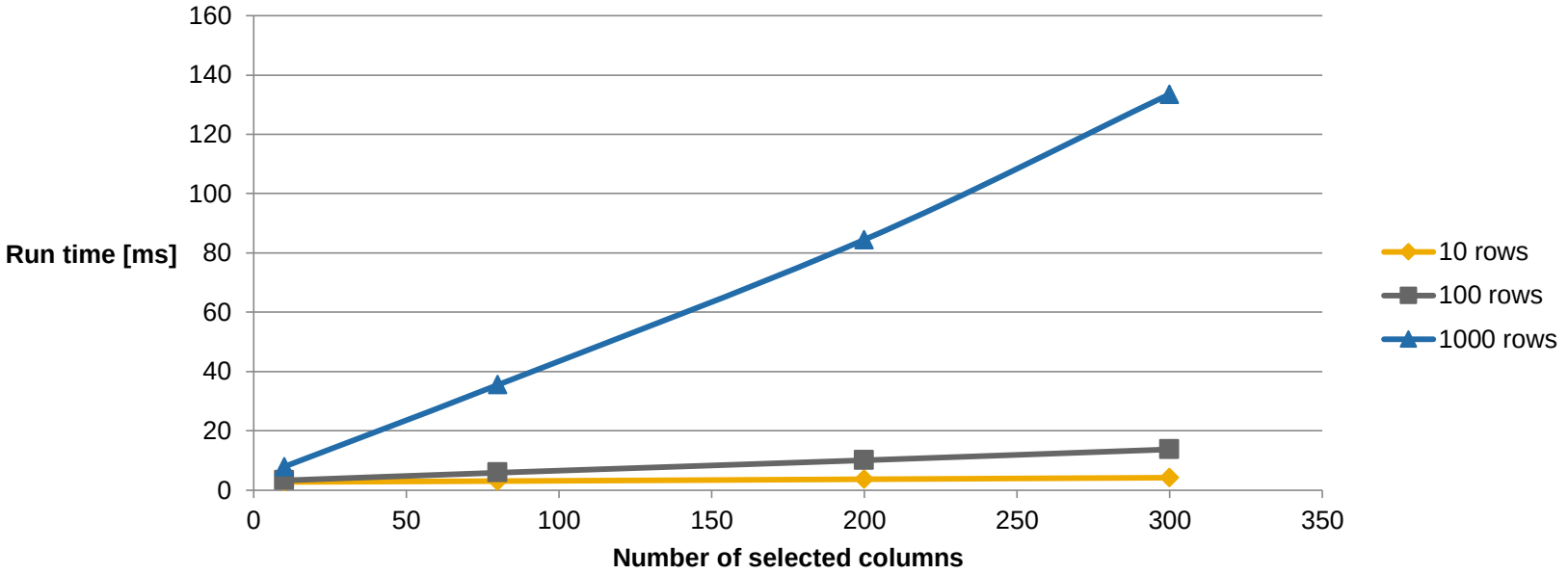
SELECT * FROM table WHERE …

- Specification of few columns (only the one's needed) will reduce amount of data being transferred.

SELECT c1 c2 c3 FROM table WHERE …

- Especially HANA gains when less columns are requested, as data is stored column based.

Select from … where … up to N rows
Run time with varying number of rows and columns



Conclusion:

The more rows are selected, the more important becomes the optimization for field lists. Large factors (>20) are possible for 1000 rows.

Rule: Minimize amount of transferred data

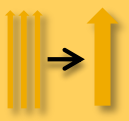
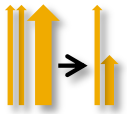
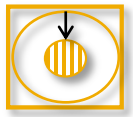
Anti Patterns: SELECT of unneeded columns (select *)

- However statement executions are more important than fields. All fields that are needed should be selected in one SQL statement.

```
SELECT f1, f2 WHERE f7 = ?;
SELECT f3, f4 WHERE f7 = ?;
```

```
SELECT f1, f2, f3 ,f4 WHERE f7 = ?;
```

Golden Rules – Overview

	Minimize the number of round trips	• Avoid nested Selects by using Joins, Subqueries, or SELECT FOR ALL ENTRIES • Use array variants of INSERT, UPDATE, MODIFY, and DELETE
	Minimize amount of transferred data	• Don’t read unused columns using SELECT *, but specify the required fields using a field list. • Use aggregate functions (COUNT, MIN, MAX, SUM, AVG) instead of transferring all the rows to the application server
	Keep the result set small	• Don’t read records from the database that you don’t need in your application. Restrict the data set by an appropriate WHERE clause.
	Keep load away from the database	• Avoid reading data redundantly • Use table buffering (if applicable) and do not bypass it • Consider sorting in ABAP
	Minimize the search overhead	• Define and use appropriate indexes

Rule: Reduce the number of round trips

Anti Patterns: SELECT and SELECT SINGLE‘s in SELECT/LOOP/DO/While

- Avoid SELECTs in LOOPs and nested SELECTs.

```
SELECT … FROM t_head … WHERE
  SELECT… FROM t_item … WHERE

  ENDSELECT.
ENDSELECT.
```

- Strive to use array SELECTs, Joins, Subqueries, FAE, Views where ever possible.

```
SELECT … FROM t_head
  JOIN t_item ON …
WHERE…
```

Rule: Reduce the number of round trips

Anti Patterns: SELECT and SELECT SINGLE‘s in SELECT/LOOP/DO/While

- Avoid SELECT SINGLEs in LOOP.

```
LOOP AT itab1.
  SELECT SINGLE… FROM dbtab… WHERE…..
ENDLOOP.
```

- Strive to use array SELECTs, Joins, FAE outside the loop. Read inside the loop using BINARY SEARCH (declare the temporary internal table as sorted/hashed)

```
SELECT … FROM dbtab INTO itab_temp…
  FOR ALL ENTRIES IN itab1 WHERE ….

LOOP AT itab1.
  READ TABLE itab_temp WITH KEY……(BINARY SEARCH)
```

Rule: Reduce the number of round trips

Anti Patterns: Execute inserts/updates/deletes in Loop

- Do not execute inserts/updates/deletes in LOOPs.

```
SELECT * FROM dbtab..WHERE…
DELETE dbtab .
ENDSELECT.
```

```
LOOP AT itab.
…….
INSERT dbtab.
……
ENDLOOP.
```

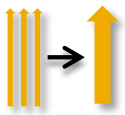
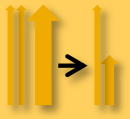
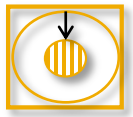
- Strive to use ARRAY variants instead.

```
DELETE FROM dbtab WHERE….
```

```
LOOP AT itab.
…….
   INSERT wa_itab1 INTO TABLE itab1
…….
ENDLOOP.

INSERT dbtab FROM TABLE itab1.
```

Golden Rules – Overview

	Minimize the number of round trips	• Avoid nested Selects by using Joins, Subqueries, or SELECT FOR ALL ENTRIES • Use array variants of INSERT, UPDATE, MODIFY, and DELETE
	Minimize amount of transferred data	• Don’t read unused columns using SELECT *, but specify the required fields using a field list. • Use aggregate functions (COUNT, MIN, MAX, SUM, AVG) instead of transferring all the rows to the application server
	Keep the result set small	• Don’t read records from the database that you don’t need in your application. Restrict the data set by an appropriate WHERE clause.
	Keep load away from the database	• Avoid reading data redundantly • Use table buffering (if applicable) and do not bypass it • Consider sorting in ABAP
	Minimize the search overhead	• Define and use appropriate indexes

Rule: Keep the load away from the Database

Anti Patterns: Bypass the SAP table buffer in the select

- Use the SAP table buffer, avoid bypassing the table buffer.
- Statements that bypass table buffers are:
  - If the WHERE condition does not fit: SCI
    - for generic key buffered tables
      - if not all fields of the generic buffered key fields are specified with ‚=' and ‚AND'
      - SELECT * from table1 WHERE bkey1 IN (val1, val2,…)
    - single record buffered
      - SELECT * from table1 WHERE bkey1 IN (val1, val2,…)
      - FOR ALL ENTRIES
      - Release <= 7.00: SELECT **SINGLE** not specified: SELECT … WHERE bkey1 = val1 (uses buffer as of rel. 7.02)

The where condition has to be written in a way that it finds exactly ONE area:

Fully buffered tables = the whole table is ONE area

generic buffered tables = one generic region, all key fields to address ONE region

Sinlge record tables = one record is one area, complete key is needed

Rule: Keep the load away from the Database

Anti Patterns: Bypass the SAP table buffer in the select

- If the SQL statement contains one of the following key words
  - SELECT … BYPASSING BUFFER
  - IN-lists for buffer key fields with more than 1 element SCI
  - SELECT … FOR UPDATE
  - SELECT with aggregate function (COUNT, MIN, MAX, AVG, SUM, GROUP GY, HAVING) SCI
  - SELECT DISTINCT SCI
  - SELECT … WHERE … IS NULL SCI
  - SELECT with subqueries SCI
  - SELECT … ORDER BY … (except PRIMARY KEY) SCI
  - JOINS SCI
  - SELECT … CLIENT-SPECIFIED but client condition not provided SCI
  - any native-SQL statement SCI
  - INSERT, UPDATE, DELETE, MODIFY
  - Function module ‚DB_SET_ISOLATION_LEVEL‘ was called before the SELECT (note 1376858)

Rule: Keep the load away from the Database

Anti Patterns: Reading data redundantly

- Avoid reading data redundantly from the Database.
- Use Reading modules to avoid redundant selects.
- For many tables SAP offers standard reading modules. SAP Note 332856 lists some of these, but there are more. You can search for these modules in SE37 with the below Patterns:
  - *<table_name>*single* or *single*<table_name>* e.g. *MARA*SINGLE* or *SINGLE*MARA*
  - Sometimes it is worth to search only with *<table_name>* because the available reading modules not always containt the key word single or buffer
  - For some tables there are as well array read modules available e.g. MARA_ARRAY_READ

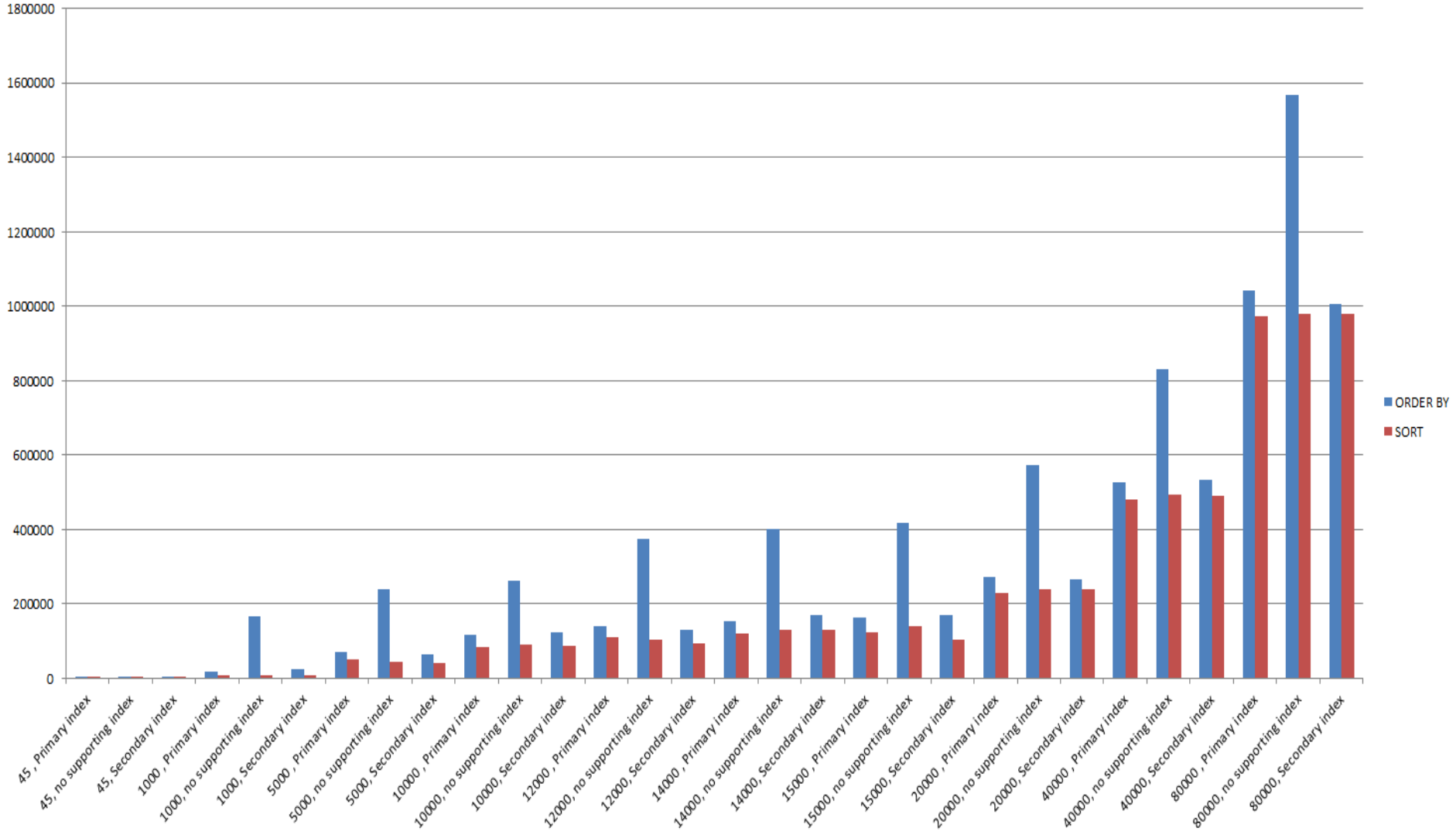
Rule: Keep the load away from the Database

Anti Patterns: Always sort data in ABAP programs

- Decide based on Data volume where to SORT – ABAP or HANA.
- Index supporting the SORT sequence helps HANA to sort faster as compared to Sorting without index support.
- So decide based on Data volume where to SORT – ABAP or HANA. And then to further improve SORTING in HANA, maybe think about index supporting the SORT order.
- If SORT is a part of code pushdown to HANA (e.g. DB procedure) let HANA do it.

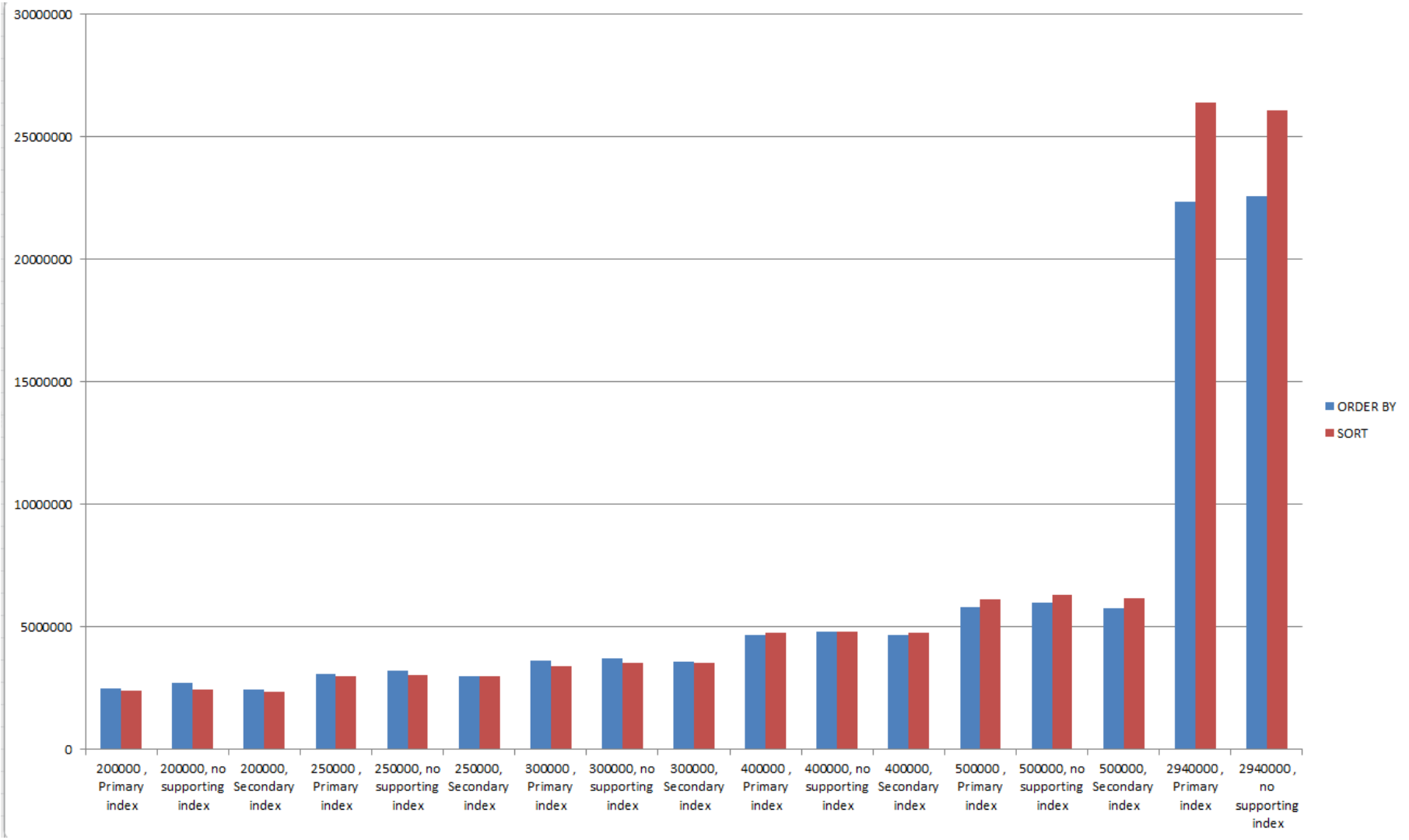
Rule: Keep the load away from the Database

Anti Patterns: Always sort data in ABAP programs

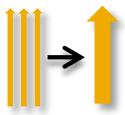
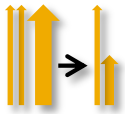


Rule: Keep the load away from the Database

Anti Patterns: Always sort data in ABAP programs



Golden Rules – Overview

	Minimize the number of round trips	• Avoid nested Selects by using Joins, Subqueries, or SELECT FOR ALL ENTRIES • Use array variants of INSERT, UPDATE, MODIFY, and DELETE
	Minimize amount of transferred data	• Don’t read unused columns using SELECT *, but specify the required fields using a field list. • Use aggregate functions (COUNT, MIN, MAX, SUM, AVG) instead of transferring all the rows to the application server
	Keep the result set small	• Don’t read records from the database that you don’t need in your application. Restrict the data set by an appropriate WHERE clause.
	Keep load away from the database	• Avoid reading data redundantly • Use table buffering (if applicable) and do not bypass it • Consider sorting in ABAP
	Minimize the search overhead	• Define and use appropriate indexes

Rule: Minimize the search overhead

Usage of appropriate indices

- SAP HANA does not necessarily require secondary indices for good search performance. As in column store a full column scan can be performed quite quickly, the creation of a secondary index is often not required.

- To reduce main memory consumption and to improve insert performance all existing secondary database indices on columnar tables are removed during migration or do not get created during installation for all AS ABAP systems from SAP NetWeaver 7.40 onwards.

Rule: Minimize the search overhead

Usage of appropriate indices

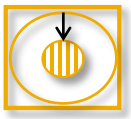
- For some use cases secondary indexes can still be beneficial. This is especially true for highly selective queries on non-primary key fields. To minimize memory consumption it is preferred to create single column-indexes.
- SAP Note 1794297 describes the procedure to find and create these indexes.
- For tables in the row store there is no change to the index design compared to traditional databases.

Golden Rules

Key Message

- **All the classical performance recommendations are still valid as general guidelines**
- Only the priority changes for some of the rules

An acceleration is expected for the following type of statements:



Statements that involve physical I/O on traditional databases, e.g. WHERE clauses w/o index support

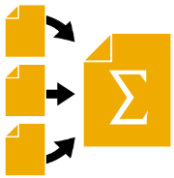
Statements that scan a large dataset, but provide a small result set, e.g. aggregations over single fields like COUNT, SUM, MAX, AVG, …

But some statements might perform slower in the column store:

The selection of unnecessary fields (Select * instead of specification of the Field list

Frequent database requests (Nested selects, inserts, updates)

DAY 1
Summary: Golden Rules and Patterns & Anti-patterns



You should now be able to

- Write optimized ABAP code keeping “Golden rules” in mind.
- Identify different Anti Patterns and correct them.

Internal Tables

DAY 1
Learning Objectives: Internal Tables

After completing this unit you will be able to:

- Understand optimized usage of internal tables for important ABAP commands

ABAP Commands
APPEND

	Standard	Sorted	Hashed
APPEND	O(1)	O(1) (higher costs than for standard tables) Needs to check if appended lines are in the right sort order. Otherwise we will get dump ITAB_ILLEGAL_SORT_ORDER	N/A
APPEND LINES OF itab	Similar to the above ones multiplied with the number of lines	Similar to the above ones multiplied with the number of lines. If the content to be appended is sorted, the APPEND is optimized further	N/A

An APPEND is always fast but on sorted tables the sort order must be preserved!

O(1) = constant **O(n) = linear** **O(log n) = logarithmic**

ABAP Commands
READ

	Standard	Sorted	Hashed
READ ... INDEX	O(1)	O(1)	N/A
READ ... WITH (TABLE) KEY ... (Complete key)	O(n)	O(log n)	O(1)
READ … WITH KEY … BINARY SEARCH	O(log n)	O(log n) Leading part of the key must be specified	N/A
READ ... WITH KEY ... (Incomplete key, initial part)	O(n)	O(log n) – O(n) For many duplicates a linear part O(n) must be added, since the first entry has to be retrieved	O(n)
READ ... WITH KEY ... (Incomplete key, no initial part)	O(n)	O(n)	O(n)

Use sorted tables whenever possible !

O(1) = constant **O(n) = linear** **O(log n) = logarithmic**

ABAP Commands
LOOP

	Standard	Sorted	Hashed
LOOP ... ENDLOOP (all rows)	O(n) (full table scan)	O(n) (full table scan)	O(n) (full table scan)
LOOP ... WHERE ENDLOOP (complete key) (assuming selective cond.)	O(n) (full table scan)	O(log n)	O(1) (returns 0 or 1 line, corresponding to a READ)
LOOP ... WHERE ENDLOOP (incomplete key, initial part)	O(n) (full table scan) Can be optimized manually using a sorted standard table and a binary search O(log n)	O(log n) – O(n) For many duplicates a linear part O(n) must be added, since the whole range is relevant	O(n) (full table scan)
LOOP ... WHERE ENDLOOP (incomplete key, no initial part)	O(n) (full table scan)	O(n) (full table scan)	O(n) (full table scan)
LOOP ... FROM n1 TO n2	O(n2 – n1) (depending on the size of the range to be looped)	O(n2 – n1) (depending on the size of the range to b	N/A

O(1) = constant **O(n) = linear** **O(log n) = logarithmic**

Use sorted tables whenever possible !

ABAP Commands
MODIFY

	Standard	Sorted	Hashed
MODIFY ... TRANSPORTING ... WHERE (complete key)	O(n) (full table scan) Can be optimized manually using a sorted standard table and a binary search O(log n)	O(log n)	O(1)
MODIFY... TRANSPORTING... WHERE (incomplete key, initial part)	O(n) (full table scan) Can be optimized manually using a sorted standard table and a binary search O(log n)	O(log n) – O(n) For many duplicates a linear part O(n) must be added, since the whole range is relevant	O(n) (full table scan)
MODIFY... TRANSPORTING... WHERE (incomplete key, no initial part)	O(n) (full table scan)	O(n) (full table scan)	O(n) (full table scan)
MODIFY ... [INDEX n] FROM wa (index access)	O(1)	O(1)	N/A
MODIFY TABLE... FROM wa (search effort as for WHERE)	O(n) (full table scan)	O(log n)	O(1)

O(1) = constant **O(n) = linear** **O(log n) = logarithmic**

Use sorted tables whenever possible !

ABAP Commands
DELETE

	Standard	Sorted	Hashed
DELETE ... WHERE (complete key)	O(n) (full table scan)	O(log n)	O(1)
DELETE ... WHERE (incomplete key, initial part)	O(n) (full table scan) Can be optimized manually using a sorted standard table and a binary search O(log n)	O(log n) – O(n) For many duplicates a linear part O(n) must be added, since the whole range is relevant	O(n) (full table scan)
DELETE ... WHERE (incomplete key, no initial part)	O(n) (full table scan)	O(n) (full table scan)	O(n) (full table scan)
DELETE ... INDEX	O(1)	O(1)	N/A
DELETE ... FROM n1 TO n2	O(n2 – n1)	O(n2 –n1)	
DELETE FROM WA / DELETE TABLE WITH TABLE KEY	O(n) (full table scan)	O(log n)	O(1)
DELETE ADJACENT DUPLICATES	O(n) (full table scan)	O(n) (full table s	O(n) (full table scan)

O(1) = constant **O(n) = linear** **O(log n) = logarithmic**

Use sorted tables whenever possible !

ABAP Commands
COLLECT

	Standard	Sorted	Hashed
COLLECT	O(1) - O(n) (constant - full table scan)  Depends on whether the temporary hash administration is valid or not	O(log n) 	O(1) 

Collect is recommended with hashed tables

O(1) = constant **O(n) = linear** **O(log n) = logarithmic**

ABAP Commands
SORT

	Standard	Sorted	Hashed
SORT	O(n * log n) 	N/A	O(n * log n) 

O(1) = constant **O(n) = linear** **O(log n) = logarithmic**

DAY 1
Summary: Internal Tables

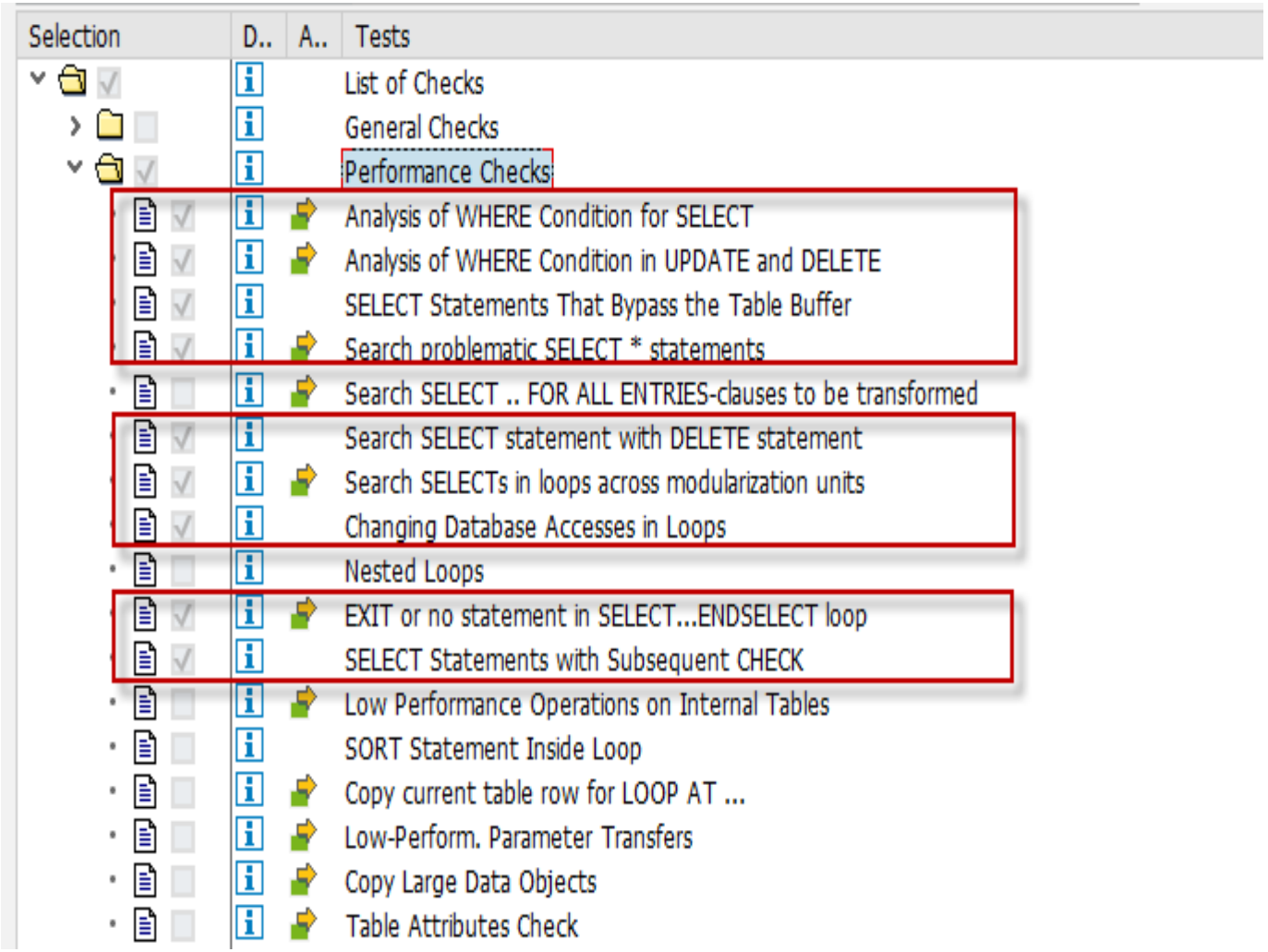
You should now be able to:

- Use internal tables in optimized way for important ABAP commands

Code Inspector – HANA Specific Checks

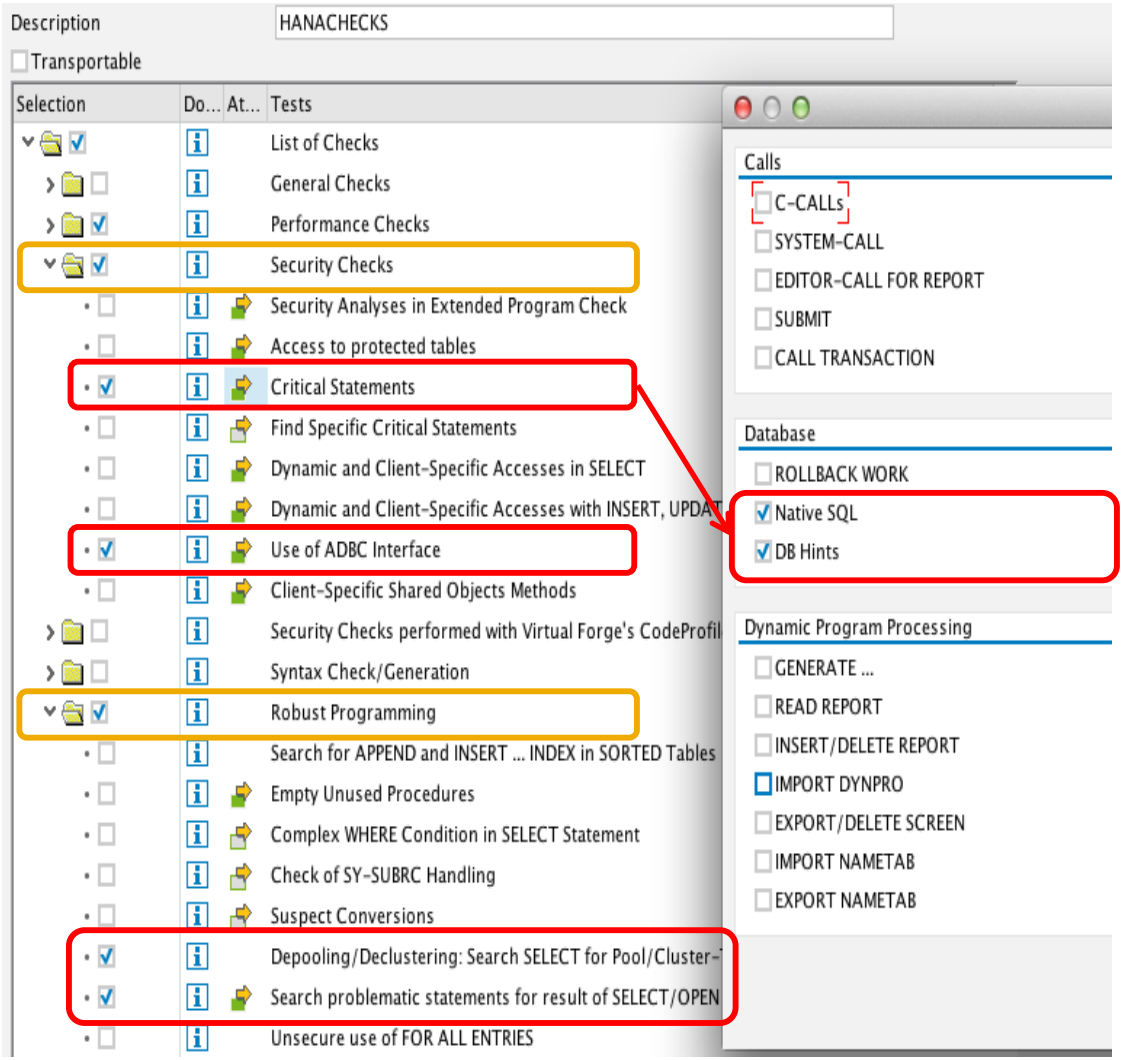
Code Inspector Performance Checks

SAP_BASIS 7.40 SP2



Code Inspector HANA Checks

SAP_BASIS 7.40 SP2



Detail Information for HANA Check : Critical Statements

Code Inspector Performance Checks

SAP_BASIS 7.40 SP2

	Minimize the number of round trips	Search SELECTs in loops across modularization units
	Minimize amount of transferred data	Search problematic SELECT * statements
	Minimize the search overhead	Analysis of WHERE Condition for SELECT Analysis of WHERE Condition in UPDATE and DELETE
	Keep the result set small	Search SELECT statement with DELETE statement EXIT or no statement in SELECT...ENDSELECT loop SELECT Statements with Subsequent CHECK
	Keep load away from the database	SELECT Statements That Bypass the Table Buffer

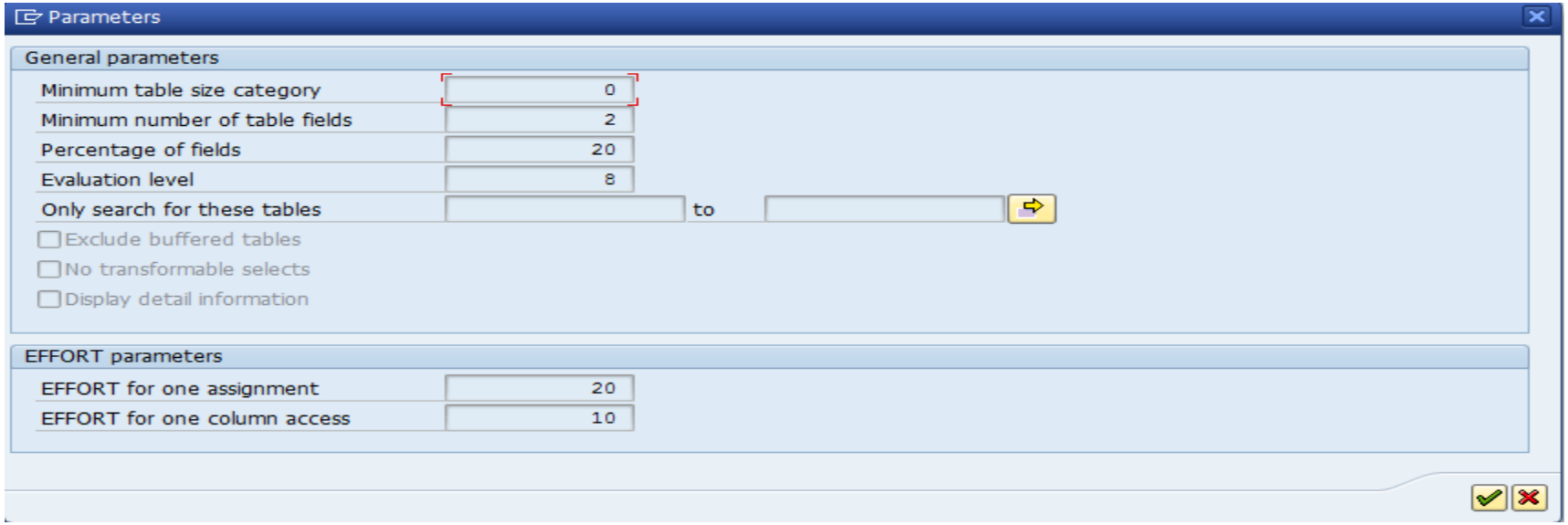
Specific checks for HANA
Corresponding Code Inspector Checks

Database Specific enhancements	☑ Critical Statements
Database Specific SQL Statements	☑ Use of ADBC Interface
Expecting sorted SQL result	☑ Depooling/Declustering: Search SELECT for Pool/Cluster-Tables w/o ORDER BY ☑ Search problematic statements for result of SELECT/OPEN CURSOR without ORDER BY

New Code inspector checks – Search problematic SELECT * statements

- **Search problematic SELECT * statements** - This check searches for *SELECT * FROM dbtab* statements where less than a specified percentage of the fields that are selected from the database are used in the code later.

- The default value for the percentage is 20 percent. If less than 20 percent of the fields are used it is more adequate to select only the used fields.



New Code inspector checks – Search problematic SELECT * statements

Check parameters:

- **Minimum table size category**: With this parameter we can restrict the search on SELECT-statements that access database tables with a size category that is equal or larger than the parameter.
- **Minimum number of table fields**: with this parameter you can restrict the search on SELECT-statements that access database tables with equal or more columns than the parameter.
- **Only search for these tables**: with this parameter it is possible to restrict the search to accesses to the selected tables.
- **Percentage of fields**: The percentage factor of the fields of the database table
- **Evaluation level**: With this parameter you can specify the evaluation depth.
- **Exclude buffered tables**: Normally accesses to buffered tables are analyzed as well. With this parameter you can exclude it.
- **Restrict to object set:** Only calls in procedures being contained in the object set of the inspection are analyzed
- **No transformable selects**: Display also information about SELECT-statements which cannot be transformed.
- **Display detail information**: If we switch on this parameter you get very detailed information about the access to the result of the select-* statements and the evaluation

New Code inspector checks – Search problematic SELECT * statements

Check Results:

- **FEW SELECT**-Statement can be transformed, because less than the specified percentage of fields is used.
- **MANY SELECT-** Statement should not be transformed, because more than the specified percentage of fields is used.
- **EXISTS SELECT-** Statement can be transformed very easily, because it is used just for an existence check. The work area or any of its components are not used.
- **UNCLEAR -** The situation for the select statement is unclear because of one of two reasons:
  - The result of the SELECT-statement cannot be analyzed further on.
  - The evaluation was aborted because the evaluation level has been exceeded.

New Code inspector checks – Search problematic SELECT * statements

Errors	6	0	0
Message Code FEW	5	0	0
Class ZSAPLINK Method GETPLUGINS Row 39 Column 2	1	0	0
Select-Statement can be transformed. 11.1% of fields used			
Program Z_FLIGHT_REPORT Include Z_FLIGHT_REPORT Row 25 Column 0	1	0	0
Select-Statement can be transformed. 16.7% of fields used			
Program Z_FLIGHT_REPORT Include Z_FLIGHT_REPORT Row 42 Column 6	1	0	0
Select-Statement can be transformed. 18.8% of fields used			
Program Z_FLIGHT_REPORT1 Include Z_FLIGHT_REPORT1 Row 18 Column 0	1	0	0
Select-Statement can be transformed. 16.7% of fields used			
Program Z_FLIGHT_REPORT1 Include Z_FLIGHT_REPORT1 Row 35 Column 6	1	0	0
Select-Statement can be transformed. 18.8% of fields used			
==> Select-Statement can be transformed. ... of fields used			
Message Code EXISTS	1	0	0
Program Z_HEUR_PPS Include Z_HEUR_PPS Row 24 Column 2	1	0	0
Existence check. No fields used			
==> Existence check. No fields used			

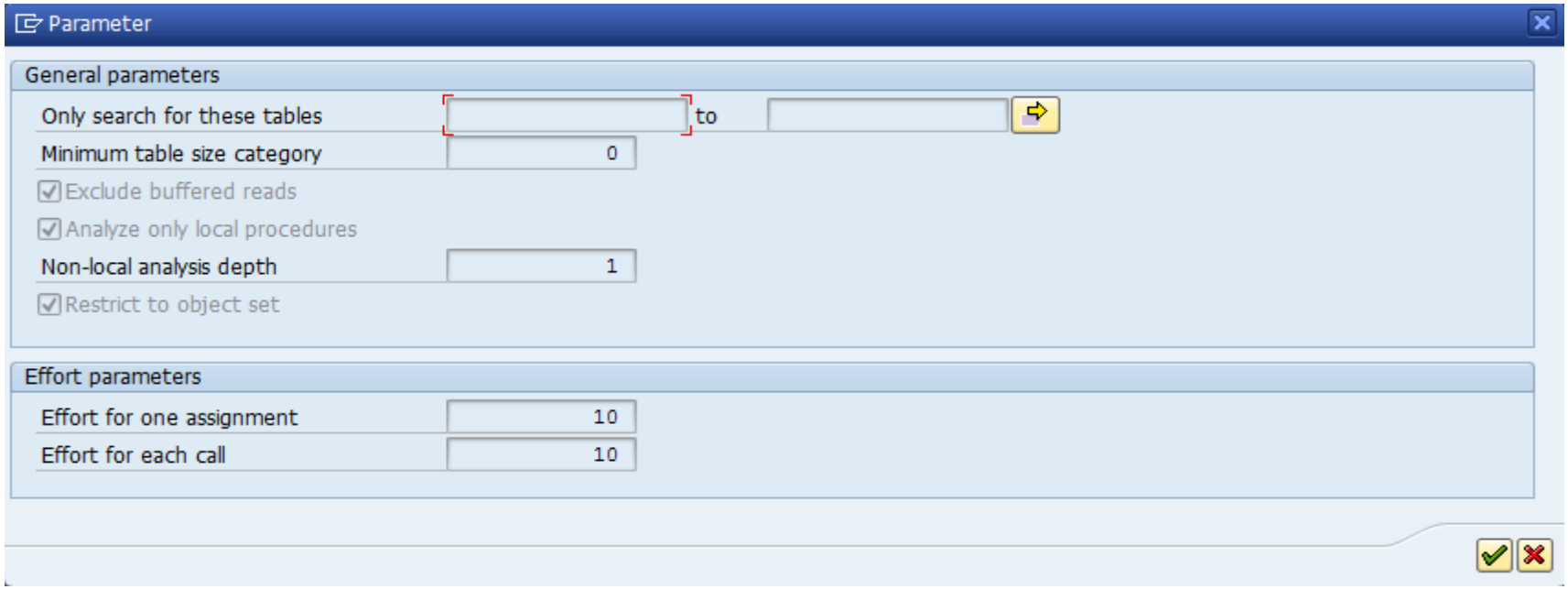
Message Text	Include	Line
SELECT * FROM SBOOK	Z_FLIGHT_REPORT1	18
Component CARRID used	Z_FLIGHT_REPORT1	21
Component CONNID used	Z_FLIGHT_REPORT1	21
Component CUSTOMID used	Z_FLIGHT_REPORT1	43
Component FLDATE used	Z_FLIGHT_REPORT1	21

```
25  SELECT * FROM sbook
26     WHERE agencynum = p_agency.
27
28  SELECT * FROM sflight
29     WHERE carrid = sbook-carrid
30     AND connid = sbook-connid
31     AND fldate = sbook-fldate
32     AND planetype = p_plane.
33
```

New Code inspector checks – Search SELECT .. FOR ALL ENTRIES-clauses to be transformed

- **Search SELECT .. FOR ALL ENTRIES-clauses to be transformed**- This check searches for SELECT ... FOR ALL ENTRIES where the input for the *for all entries* table is the result of another SELECT statement.

- The two SELECT statements can be in different subroutines.



New Code inspector checks – Search SELECT .. FOR ALL ENTRIES-clauses to be transformed

Check Results:

- **TRANSFORM -** The SELECT-FOR-ALL-ENTRIES statement can be joined with another SELECT-statement. The position of the second statement is mentioned in the message text.

New Code inspector checks – Search SELECT .. FOR ALL ENTRIES-clauses to be transformed

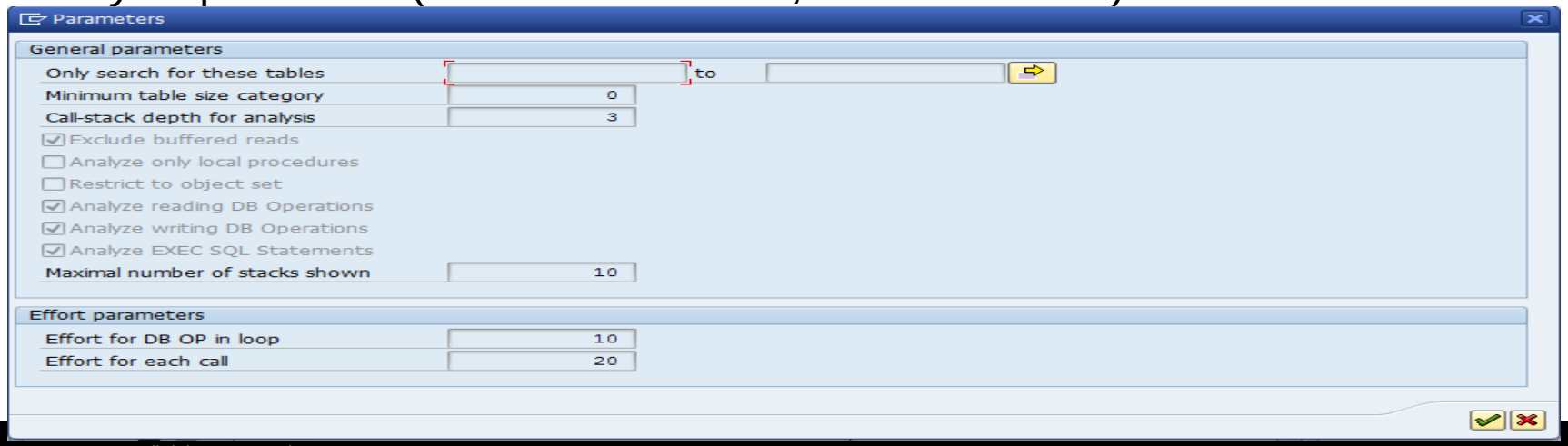
Search SELECT .. FOR ALL ENTRIES-clauses to be transformed	6	0	1
Errors	6	0	0
Message Code TRANSFORM	6	0	0
Program Z_TEST_JOIN1 Include Z_TEST_JOIN1 Row 23 Column 2 SELECT * FOR ALL statement can be joined with SELECT statement at Include Z_TEST_JOIN1 line 21	1	0	0
Program Z_TEST_JOIN1 Include Z_TEST_JOIN1 Row 40 Column 2 SELECT * FOR ALL statement can be joined with SELECT statement at Include Z_TEST_JOIN1 line 38	1	0	0
Program Z_TEST_JOIN1 Include Z_TEST_JOIN1 Row 53 Column 2 SELECT * FOR ALL statement can be joined with SELECT statement at Include Z_TEST_JOIN1 line 51	1	0	0
Program Z_TEST_JOIN1 Include Z_TEST_JOIN1 Row 68 Column 2 SELECT * FOR ALL statement can be joined with SELECT statement at Include Z_TEST_JOIN1 line 66	1	0	0
Program Z_TEST_JOIN1 Include Z_TEST_JOIN1 Row 82 Column 2 SELECT * FOR ALL statement can be joined with SELECT statement at Include Z_TEST_JOIN1 line 80	1	0	0
Program Z_TEST_JOIN1 Include Z_TEST_JOIN1 Row 115 Column 2 SELECT * FOR ALL statement can be joined with SELECT statement at Include Z_TEST_JOIN1 line 113	1	0	0

Message Text	Include	Line
Select filling ITAB1	Z_TEST_JOIN1	66
Select for all entries of ITAB1	Z_TEST_JOIN1	68

```
select * from snwd_bpa into corresponding FIELDS OF TABLE itab3 where company_name = 'HEPA Tec'.
GET RUN TIME FIELD t.
 Select gross_amount snwd_so~currency_code snwd_so~NODE_KEY from snwd_so into cORRESPONDING FIELDS OF table itab1 for all entries in itab3 where buyer_guid = itab3~node_key.
   GET RUN TIME FIELD t1.
```

New Code inspector checks – Search DB Operations in loops across modularization units

- Search DB Operations in loops across modularization units - This check finds database operations in nested loops across modularization units
- If in the loop there is a call to a modularization unit (PERFORM, CALL FUNCTION, CALL METHOD), then this call is tracked and it will be analyzed, whether there is a SELECT-statement in the called unit.
- Several levels of the call-stack can be analyzed based on *Call-stack depth for analysis* parameter (maximum value 10, default value 3)



New Code inspector checks – Search DB Operations in loops across modularization units

Check Results:

- **DBREAD_LOC -** A reading database operation was found inside a local loop
- **DBREAD -** A reading database operation was found inside a non local loop.
  DBREAD_S - A reading database operation was found as only statement inside a loop, this is easy to be transformed into an array operation.
- **DBWRT_LOC -** A writing database operation was found inside a local loop.
- **DBWRITE A -** writing database operation was found inside a non local loop.
- **DBWRITE_S -** A writing database operation was found as only statement inside a loop, this is easy to be transformed into an array operation.
- **EXEC_LOC -** An exec sql operation was found inside a local loop.
- **EXEC** - An exec sql operation was found inside a non local loop.

New Code inspector checks – Search DB Operations in loops across modularization units

Code Inspector: Results for ZBCV_SIN_SEARCH (PROG)

Inspection Version 1 Person Responsible

Messages

	D..	...	E...	Tests	Error	Warn...	Infor...
				List of Checks	2	0	1
				Performance Checks	2	0	1
				Search SELECTs in loops across modularization units	2	0	1
				Errors	2	0	0
				Message Code LOOP	2	0	0
				Class CL_ABAP_DATFM Method GET_DELIMITER Row 11 Column 6	1	0	0
				NonLocal Nested SELECT found			
				Function CONVERSION_EXIT_PERI7_INPUT Row 26 Column 4	1	0	0
				NonLocal Nested SELECT found			
				==> NonLocal Nested SELECT found			
				Information	0	0	1

Message Text	Include	Line
DO	ZBCV_SIN_SEARCH	267
Call Function 'CONVERSION_EXIT_PERI7_INPUT'	ZBCV_SIN_SEARCH	318
Call Method CL_ABAP_DATFM=>GET_DELIMITER	LRRKEBU03	47
SELECT USR01	CL_ABAP_DATFM==========CM007	11

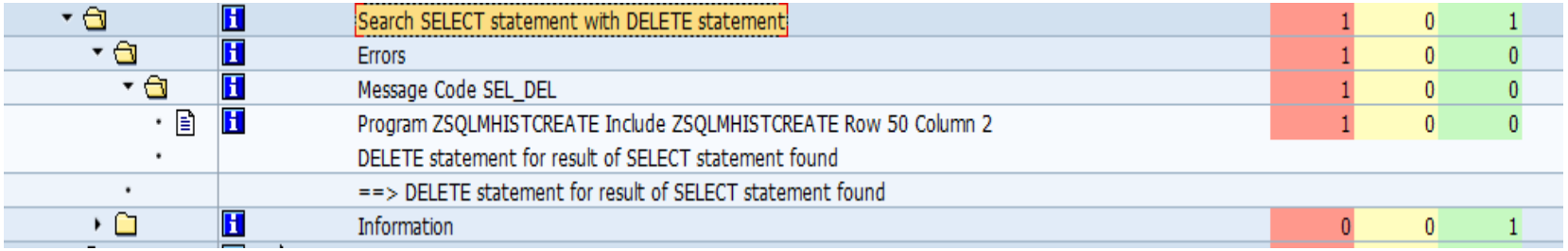
```
if l_datfm = ' '.                                    "#EC DATFM
  SELECT SINGLE datfm FROM USR01 INTO l_datfm
     WHERE BNAME = sy-uname.
endif.
```

New Code inspector checks – Search SELECT statement with DELETE statement

- **Search SELECT statement with DELETE statement**- Search for SELECT .. INTO TABLE tab statements which are followed by a DELETE statement for the result table tab .

- Check Results:
  - **Severity –** This is always 1
  - **Effort -** The effort is the distance in lines between the SELECT and the DELETE statement

New Code inspector checks – Search SELECT statement with DELETE statement



Search SELECT statement with DELETE statement	1	0	1
Errors	1	0	0
Message Code SEL_DEL	1	0	0
Program ZSQLMHISTCREATE Include ZSQLMHISTCREATE Row 50 Column 2	1	0	0
DELETE statement for result of SELECT statement found			
==> DELETE statement for result of SELECT statement found			
Information	0	0	1

Message Text	Include	Line
SELECT	ZSQLMHISTCREATE	50
DELETE	ZSQLMHISTCREATE	67

```
67    DELETE ADJACENT DUPLICATES FROM lt_toprecs COMPARING  progname progkind proctype
68      procname procline runlevel roottype rootname stmtkind tablename.
69
```

Depooling/Declustering: Search SELECT for Pool/Cluster-Tables w/o ORDER BY

- OPEN SQL SELECT statement without explicit ORDER BY clause retrieves the selected lines in unpredictable sequence.
- The current implementation of the OPEN SQL SELECT on pool/cluster tables returns the result set in the primary key order without explicit ORDER BY clause.
- If the ABAP code requires the result set in the primary key order, we need an OPEN SQL SELECT statement with explicit ORDER BY clause. Otherwise the ABAP code may fail after the conversion of pool/cluster tables to transparent tables.
- In case of such situations, we should add the needed ORDER BY clause in the ABAP code.

Depooling/Declustering: Search SELECT for Pool/Cluster-Tables w/o ORDER BY

Robust Programming	9	4	4
Depooling/Declustering: Search SELECT for Pool/Cluster-Tables w/o ORDER BY	1	0	1
Errors	1	0	0
Message Code SEL_CLUST	1	0	0
Program Z_HANA_PROBLEMATIC_STMT_EX1 Include Z_HANA_PROBLEMATIC_STMT_EX1 Row 26 Column 0	1	0	0
SELECT ... FOR cluster table CDPOS without ORDER BY found			
==> SELECT ... FOR cluster table CDPOS without ORDER BY found			

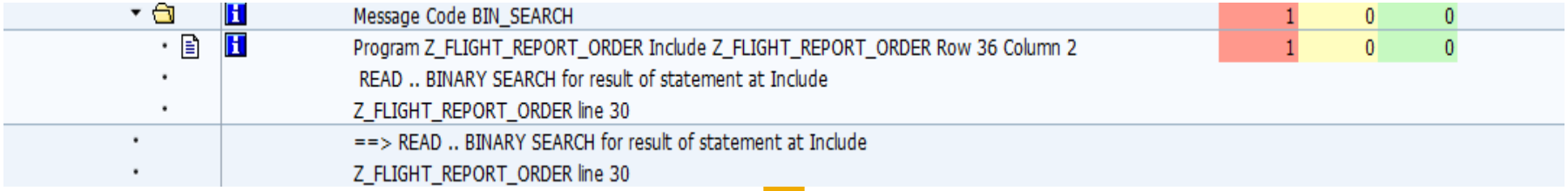
```
26  select objectclas objectid
27     from cdpos
28      into ls_cdpos
29        where objectclas = 'BUPA_BUP'.
30  endselect.
```

Search problematic statements for result of SELECT/OPEN CURSOR without ORDER BY

- This check searches for SELECT and OPEN CURSOR statements where no ORDER BY clause is specified.

- Afterwards problematic statements are searched which use the results of these SELECT or OPEN CURSOR statements. Problematic statements are statements which depend on some sort of default sorting provided by the DB based on the DB index that will be used for the Select.
- E.g.
  - READ TABLE itab ... BINARY SEARCH
  - DELETE ADJECENT DUPLICATES FROM itab

Search problematic statements for result of SELECT/OPEN CURSOR without ORDER BY



Message Text	Include	Line
Select/Fetch filling internal table/workarea	Z_FLIGHT_REPORT_ORDER	30
read table LT_BOOK binary search	Z_FLIGHT_REPORT_ORDER	36

```
    SELECT         * FROM  sbook INTO TABLE lt_book
           WHERE   carrid  = ls_flight-carrid
           AND     connid  = ls_flight-connid.
  ENDAT.

* Find first booking for this flight to later on loop from the right position.
  READ TABLE lt_book WITH KEY carrid = ls_flight-carrid
                              connid = ls_flight-connid
                              fldate = ls_flight-fldate
                              BINARY SEARCH
                              TRANSPORTING NO FIELDS.
```

SQL Monitor

Suite on HANA Migration – Expectations v/s Reality

Customer : After the migration I must run my business as before! None of the main business processes shall be slowed down noticeably

Reality : Currently HANA can slow down some ABAP programs since HANA is e.g. small frequent selects are not so good for HANA. So some ABAP codes have to be adapted.

SQLM – Motivation and Need

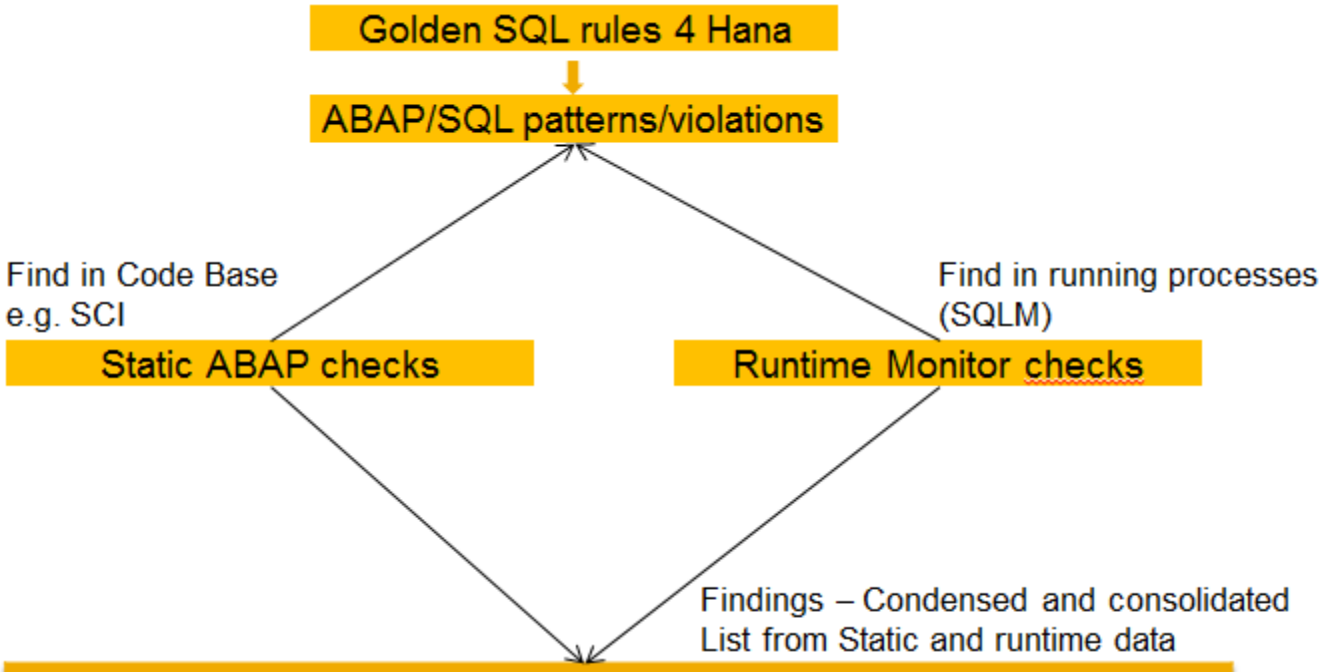
- Most of the customers have many custom codes and exits. On migrating to HANA customer is always interested to know which of these custom codes and exits need to be adapted/optimized for HANA and how to find them?

- Customer is always interested to know how to find HANA potential in their ABAP code?

Guided ABAP for HANA Performance Optimization

Step	Action
1	Preparation at SAP: Define the ABAP SQL patterns which must be optimized for HANA.
2	Automatically find the ABAP SQL patterns in the code base using static checks or runtime checks in the productive system.
3	Automatically monitor the performance characteristic of the running business processes in the productive customer system (execution time, number of executions …). SQLM (SQL Monitor)
4	Automatically correlate the findings of the scans (static checks) with the monitoring data of the productive system in order to allow business experts to rank and filter the work list according to business relevance and performance criticality.
5	Developers optimize the ABAP snippets recorded in the condensed work list (SWLT) using solution proposals.
6	The upgrade and DB migration to Suite on HANA is done
7	Developers and consultants solve the remaining issues after the HANA migration using optimized monitoring and ABAP performance analysis tools. (SQL Monitor)

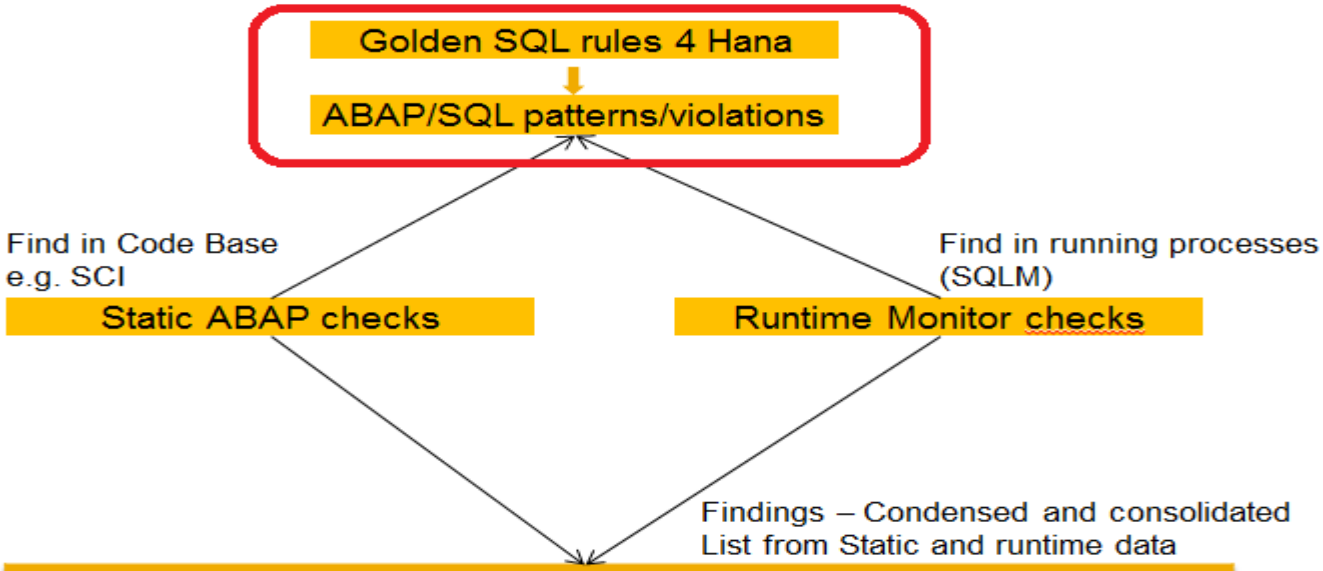
Guided ABAP for HANA Performance Optimization



Program	Include	Line	SQL finding	Priority
SAPMV45A	Lxyz12345	4	SELECT *	2
…				

Step1 - Define the ABAP SQL patterns

- Define and identify ABAP SQL patterns which must be optimized for HANA (Golden rules, Patterns and Anti-Patterns)



Program	Include	Line	SQL finding	Priority
SAPMV45A	Lxyz12345	4	SELECT *	2
…				

Step1 - Define the ABAP SQL patterns

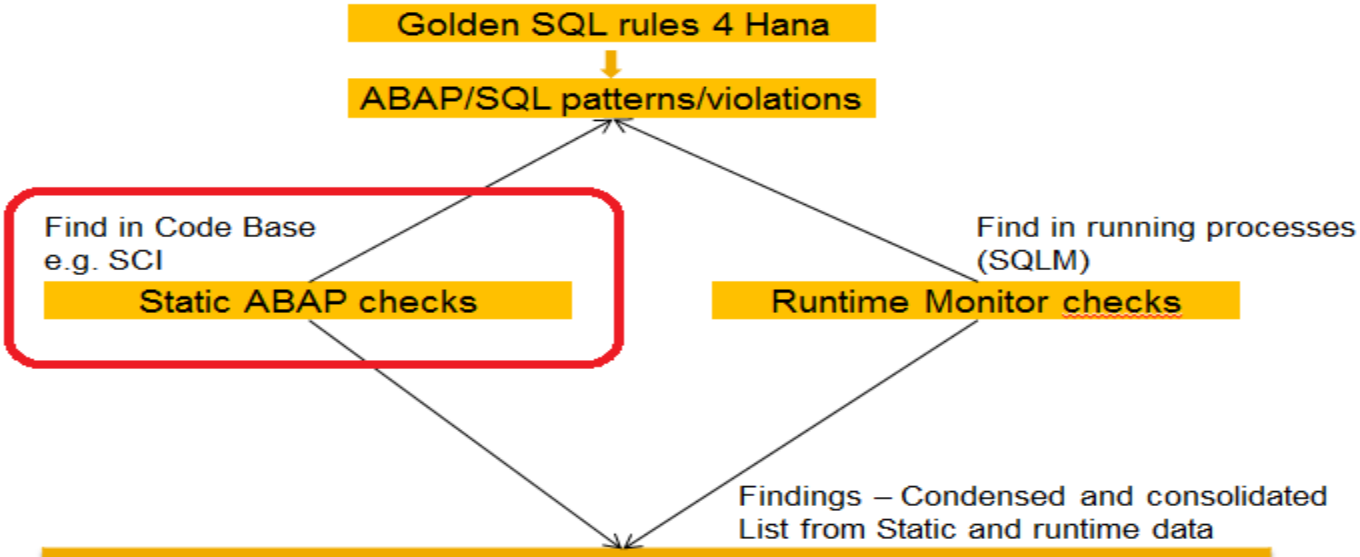
- All known "golden SQL rules" are still valid with HANA. There is only a shift in priority for certain rules:

  - In a column store and in-memory DB the following SQL statements are accelerated:
    - All statements that involve physical I/O on traditional databases
    - All statements that scan a large dataset and provide a small result set

  - In a column store the following SQL "mistakes" are even more expensive
    - Frequent single row based access (e.g. SELECT SINGLE in a LOOP, nested SELECTs)
    - Retrieving unneeded columns (SELECT *)

Step2 – New Static checks in Code Inspector for HANA

- New checks have been made in SCI to identify ABAP SQL patterns which need to be optimized for HANA.



Program	Include	Line	SQL finding	Priority
SAPMV45A	Lxyz12345	4	SELECT *	2
…				

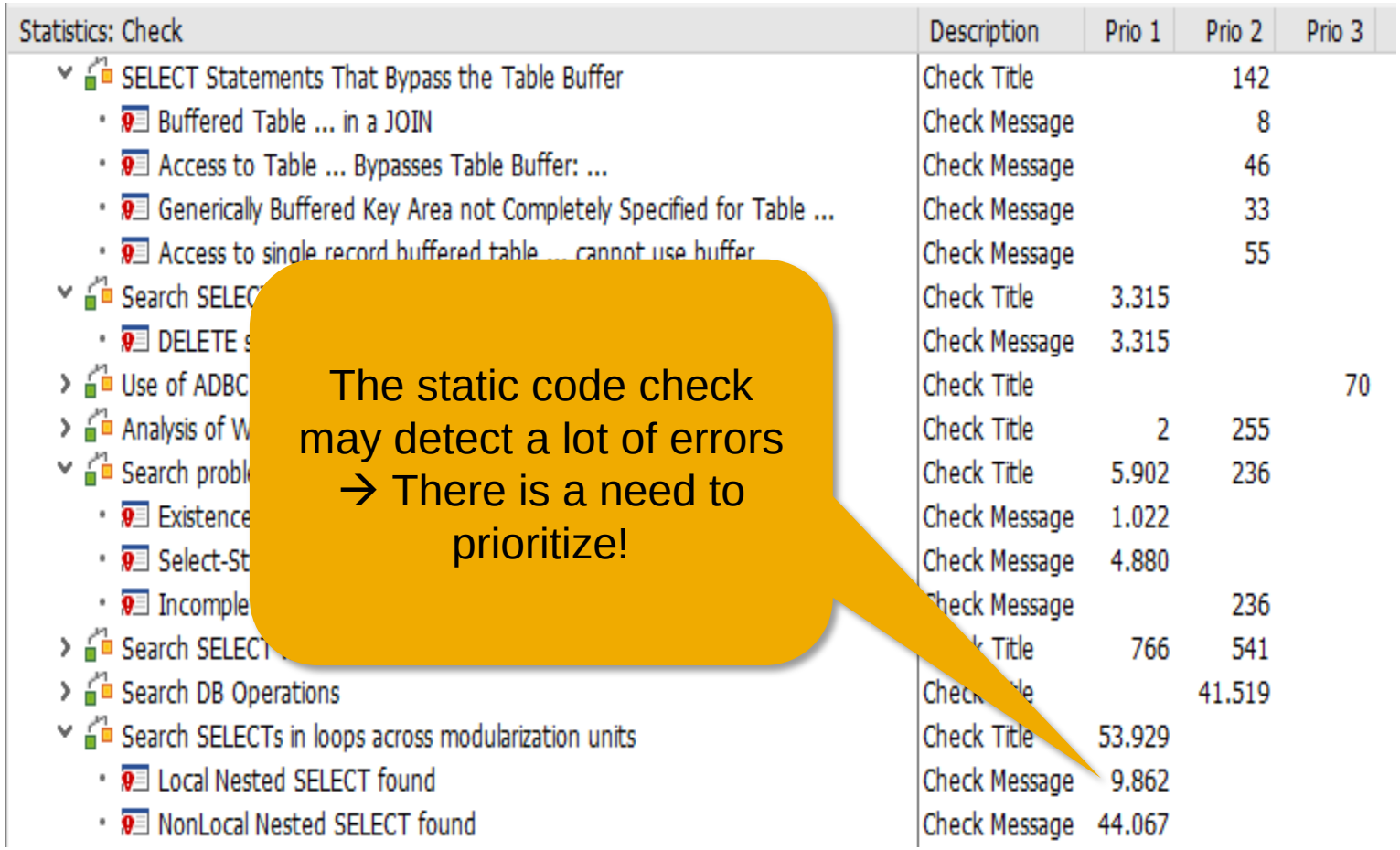
Step2 – New Static checks in Code Inspector for HANA

- But these checks are STATIC and may not give the real picture
  e.g. the static check may identify a select in loop as a problem but the business logic may be such that the loop is executed just once* and thus in reality (productive use), the select is executed just once.

```
LOOP AT itab into wa.
…….
…….
  SELECT … FROM dbtab INTO WHERE …. =
     wa-feild1 and ….. = wa-field2.
……
……
ENDLOOP.
```

**Number of entries in itab is always 1.*

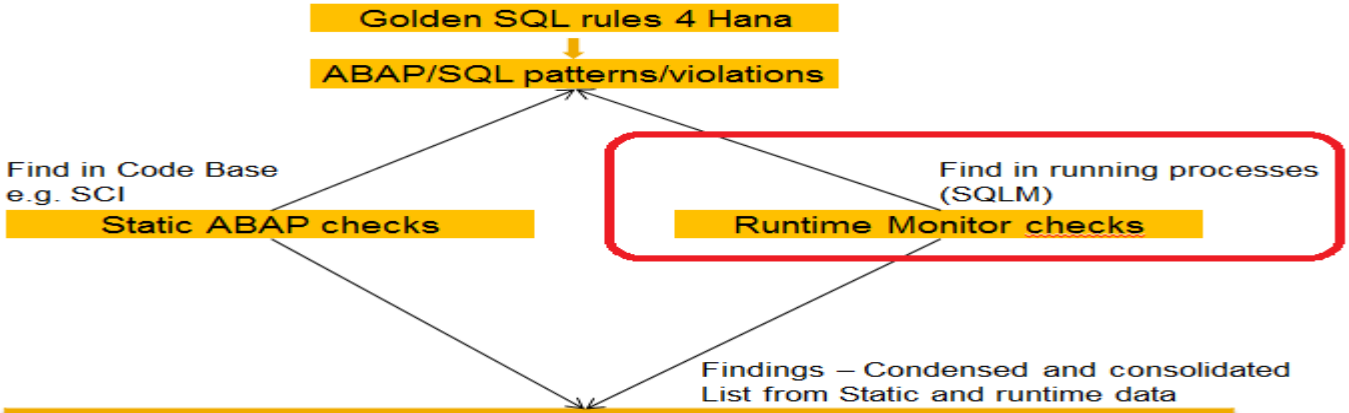
Step2 – New Static checks in Code Inspector for HANA



Step 3 – SQL Monitor (TA – SQLM)

- The SQL monitor is a tool which monitors the performance characteristic of the running business processes in the productive customer system (execution time, number of executions etc..).

- It provides real time data and more '*productive usage*' picture of ABAP codes that need to adapted or optimized for HANA.



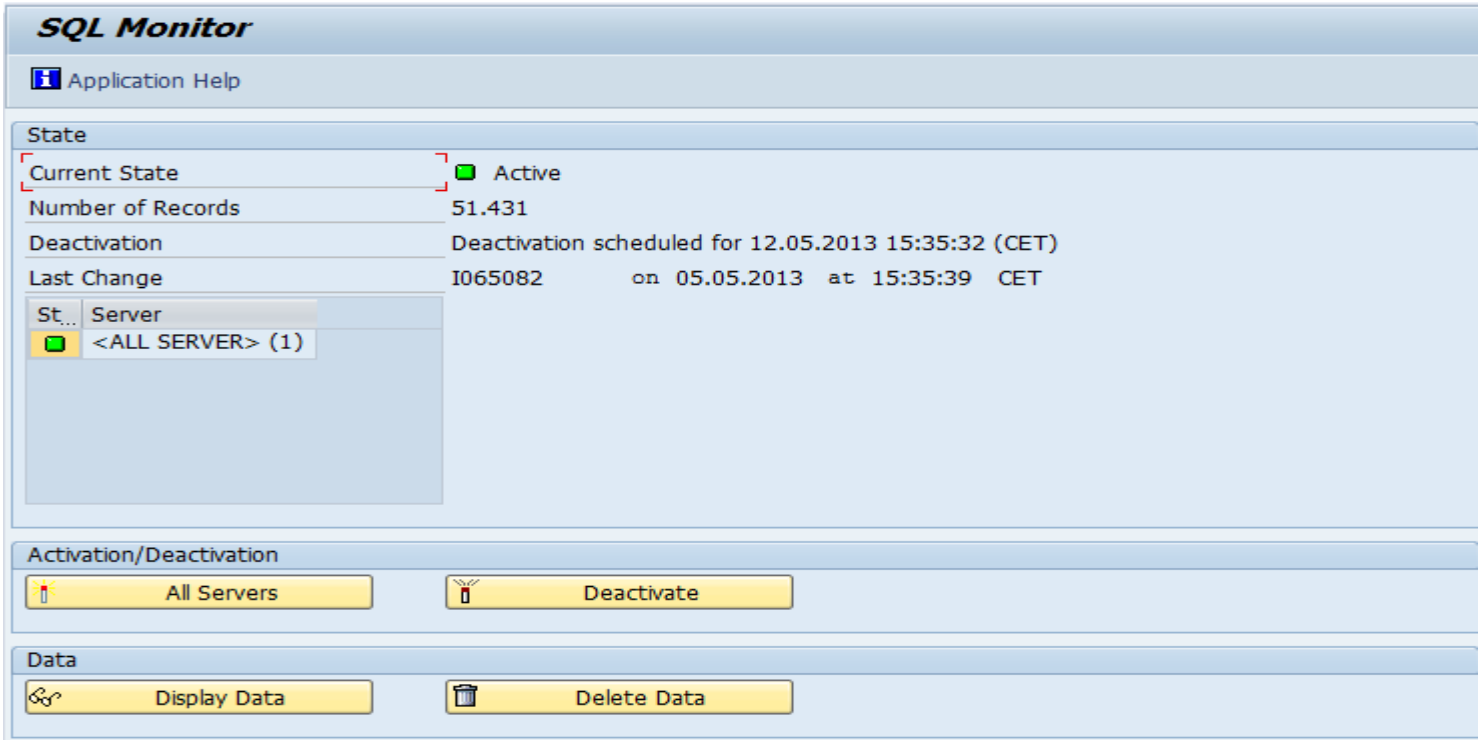
Program	Include	Line	SQL finding	Priority
SAPMV45A	Lxyz12345	4	SELECT *	2
…				

Step 3 – SQL Monitor (TA – SQLM)

- The SQL Monitor traces all SQL statements executed by running ABAP applications and all SQL statements that are passed to DB
- The SQL Monitor trace contains aggregated performance indicators (number of executions, execution time, number of effected rows etc.) for all executed SQLs.
- The data is aggregated by:
  - Code Line (ABAP include name + include line)
  - Involved Tables
  - Request Entry Point (e.g. Transaction Code, Report, RFC, URL)

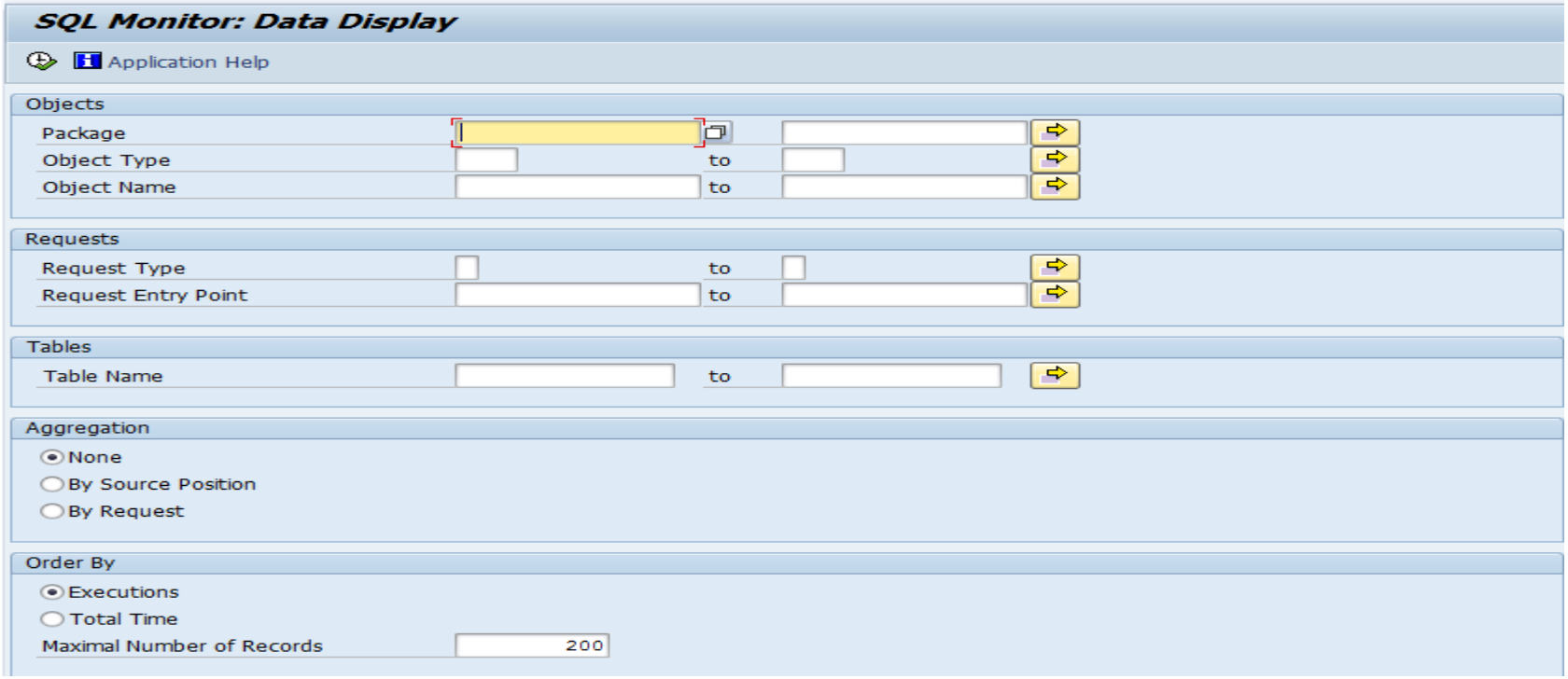
Step 3 – SQL Monitor (TA – SQLM)

- SQLM can be activated on ALL or individual servers.
- At the time of activation, deactivation is also scheduled (be default its 7 days from the day of activation)



Step 3 – SQL Monitor (Runtime Monitor, TA – SQLM)

- To display the data collected by the monitor, use TA SQLMD or *Display Data* button on TA SQLM.



- Display can be restricted to particular object, packages or requests. Display can also be restricted for particular tables.

Step 3 – SQL Monitor (Runtime Monitor, TA – SQLM)

- The result of the SQL monitor shows runtime information collected.

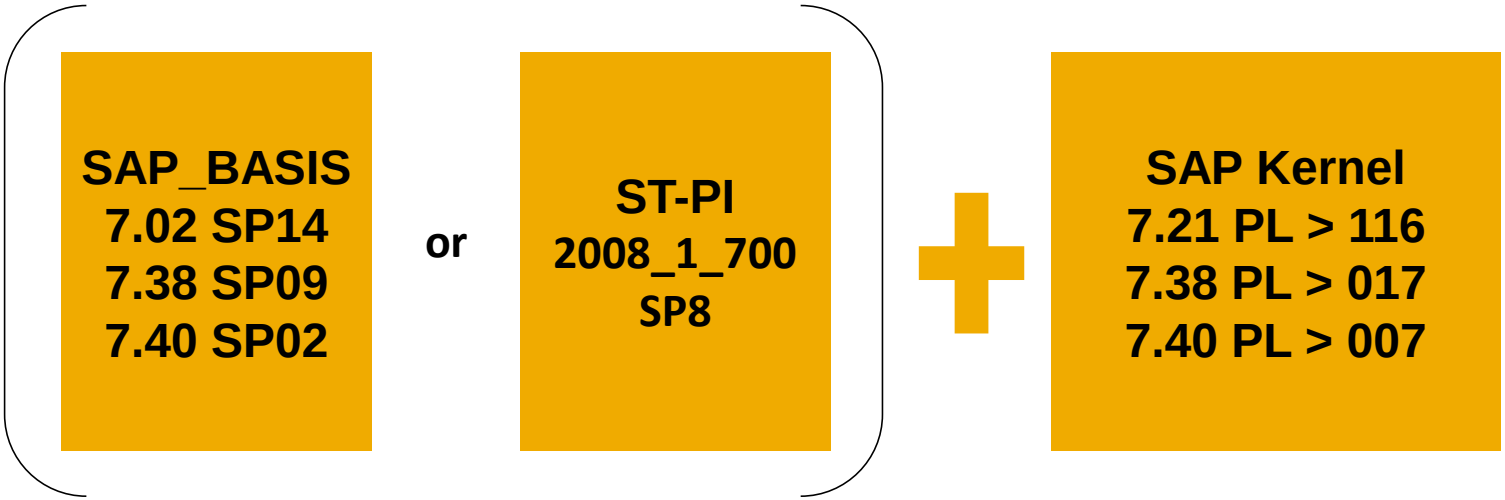
Executions	Total Time	Mean Ti...	Mean R...	Table Names	SQL Operation	Obj.	Object Name	Include Name	Include Li	Request Type	Entry Point	Int. Sess.
321.845.130	154.975,...	0,482	0,248	SFLIGHT	SELECT (Open S...	PROG	Z_FLIGHT_REPORT	Z_FLIGHT_REPORT	28	Submit Report	Z_FLIGHT_REPO...	11.573
53.131.643	17.858,6...	0,336	1,000	SCARR	SELECT (Open S...	PROG	Z_FLIGHT_REPORT	Z_FLIGHT_REPORT	47	Submit Report	Z_FLIGHT_REPO...	11.573
53.131.643	27.164,3...	0,511	1,000	SPFLI	SELECT (Open S...	PROG	Z_FLIGHT_REPORT	Z_FLIGHT_REPORT	42	Submit Report	Z_FLIGHT_REPO...	11.573
5.734.058	197.215,...	0,034	0,335	REPOSRC	System (Kernel)	CLAS	CL_RIS_HANA_INDEX_...	CL_RIS_HANA_INDEX_HANDLER==...	98	Submit Report	SRIS_PROG_TAD...	1
1.756.381	199.928,...	0,114	0,354	REPOSRC	System (Kernel)	CLAS	CL_ABAP_COMPILER	[UNRESOLVABLE_SOURCE_POSITIO...	0	Remote Func...	O2_GENERATE_I...	1
1.688.807	522.970,...	0,310	0,980	D010SINF	SELECT (Open S...	FUGR	SEDA	LSEDAU25	80	Submit Report	RSEUINC_RESET	23
1.526.052	583.684,...	0,382	1,000	TST03	SELECT (Open S...	PROG	Z_FLIGHT_REPORT	Z_FLIGHT_REPORT	47	Submit Report	Z_FLIGHT_REPO...	11.561

Exe./Sess.	Max. Time	Min. Time	Dev. Ti...	Records	Max. Re...	Min. Rec...	DevRec...	Run Mode	Program Name	Type	Processing Block Name	Proc...	Package
27.810,000	3.487,348	0,362	0,829	79.784,...	4	0	0,432	ABAP Exe...	Z_FLIGHT_REPORT	SSEL	START-OF-SELECTION:00	6	ZMADEMO
4.591,000	2.048,034	0,241	0,721	53.131,...	1	1	0,000	ABAP Exe...	Z_FLIGHT_REPORT	SSEL	START-OF-SELECTION:00	25	ZMADEMO
4.591,000	1.908,296	0,399	0,805	53.131,...	3	1	0,000	ABAP Exe...	Z_FLIGHT_REPORT	SSEL	START-OF-SELECTION:00	20	ZMADEMO
5.734.058,000	150,449	0,002	0,456	1.921.2...	2.788	0	30,560	ABAP Exe...	CL_RIS_HANA_INDEX_...	METH	LCL_DATABASE_ABSTRACTION_RE...	4	SRIS_SEARCH
1.756.381,000	1.144,320	0,002	0,886	621.962	1	0	0,478	ABAP Com...	CL_ABAP_COMPILER==...			0	SABP_COMPILER
73.426,391	45,887	0,216	0,064	1.655.1...	2	0	0,140	ABAP Exe...	SAPLSEDA	FUNC	RS_RESET_RSEUINC	45	SEDI
132,000	692,650	0,275	0,822	1.526.0...	1	1	0,000	ABAP Exe...	Z_FLIGHT_REPORT	SSEL	START-OF-SELECTION:00	25	ZMADEMO

- It is simple to identify the candidates for adaptation as per HANA guidelines and also relate the code to the business process which will benefit from adaptation/optimization (Entry point*)

SQLM -
Availability and Prerequisites

- SQLM has been made available with the following software versions:

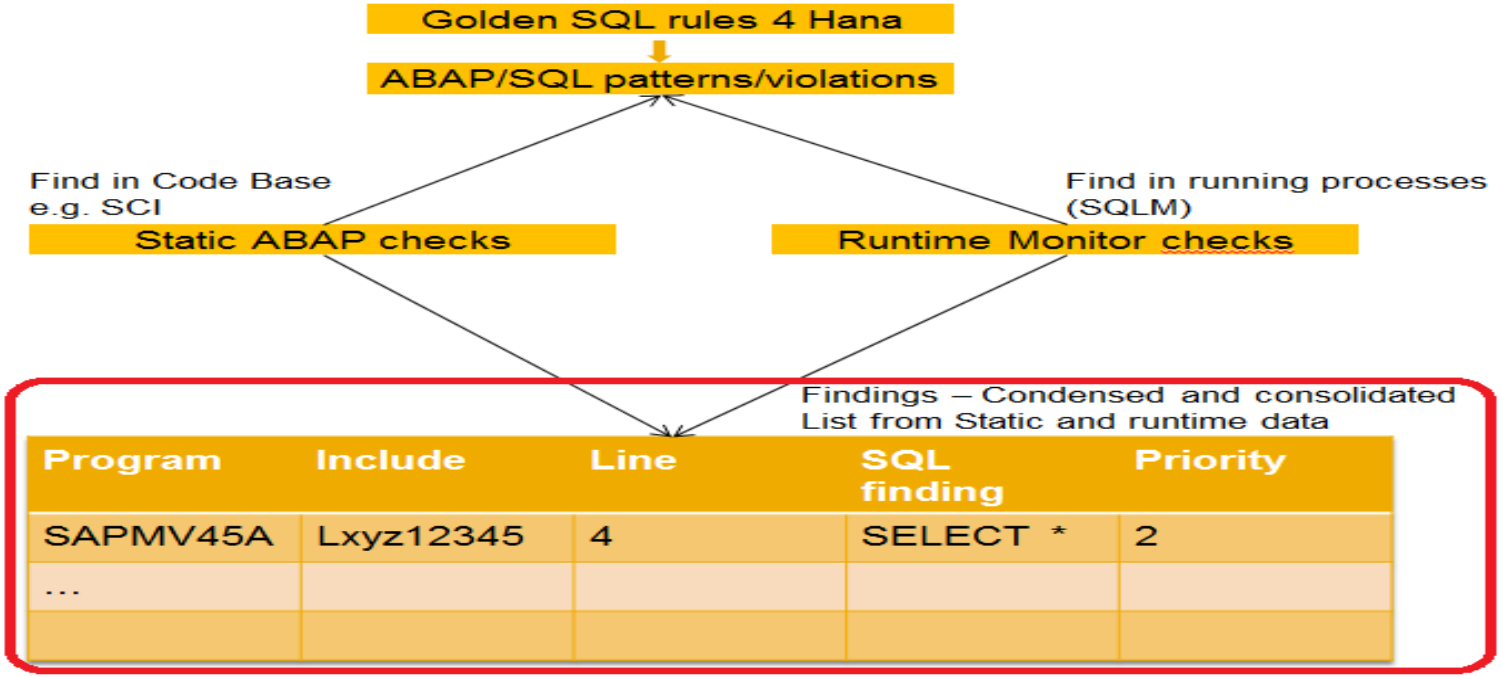


- For SAP_BASIS ≥ 7.0 the SQL Monitor has been downported with ST-PI 2008_1_700 SP8 within the /SDF/-namespace. Therefore you have to call transaction /SDF/SQLM instead of SQLM.

- Depending on the used version, some SAP notes might be required additionally. For details see SAP note 1885926.

Step 4 – SQL Performance Tuning Worklist (TA – SWLT)

- Combine the result from Static checks (step2) and SQL monitor (step 3) to generate a condensed work list to identify hotspots (candidates for adaptation and code pushdown).



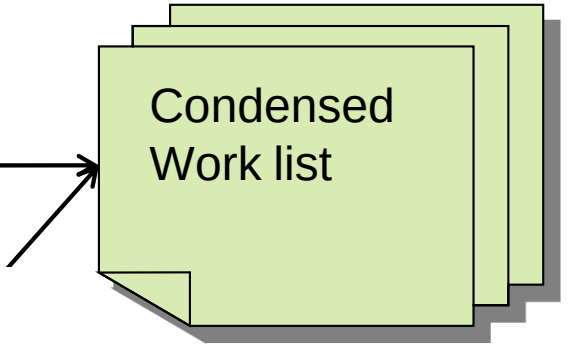
Program	Include	Line	SQL finding	Priority
SAPMV45A	Lxyz12345	4	SELECT *	2
…				

- The result allows us to rank the work list according to specific performance issues and business relevance.

Step 4 – SQL Performance Tuning Worklist (TA – SWLT)

Findings in ATC or Code inspector

Program	Include	Line	SQL finding	Priority
SAPMV45A	Lxyz12345	4	SELECT *	2



Filter/Sort by

- business relevance,
  performance criticality,
  - # executions,
  - execution time...

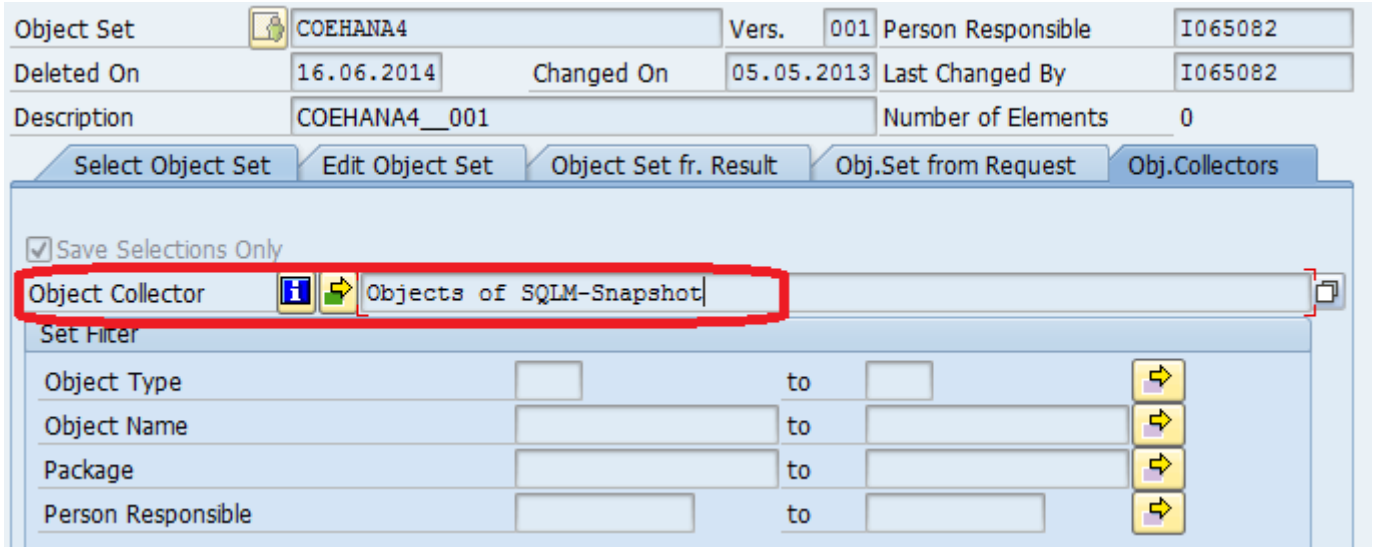
Performance data measured with SQLM

Root node (Process)	Program	Include	Event	Line	Data	Performance data
TA VA01	SAPMV45A	Lxyz12345		4	SELECT *	# Executions, time (sum)...
TA VA01	SAPMV45A	lu1sdsdsd	Form get_xyz			# Executions, time (sum)...

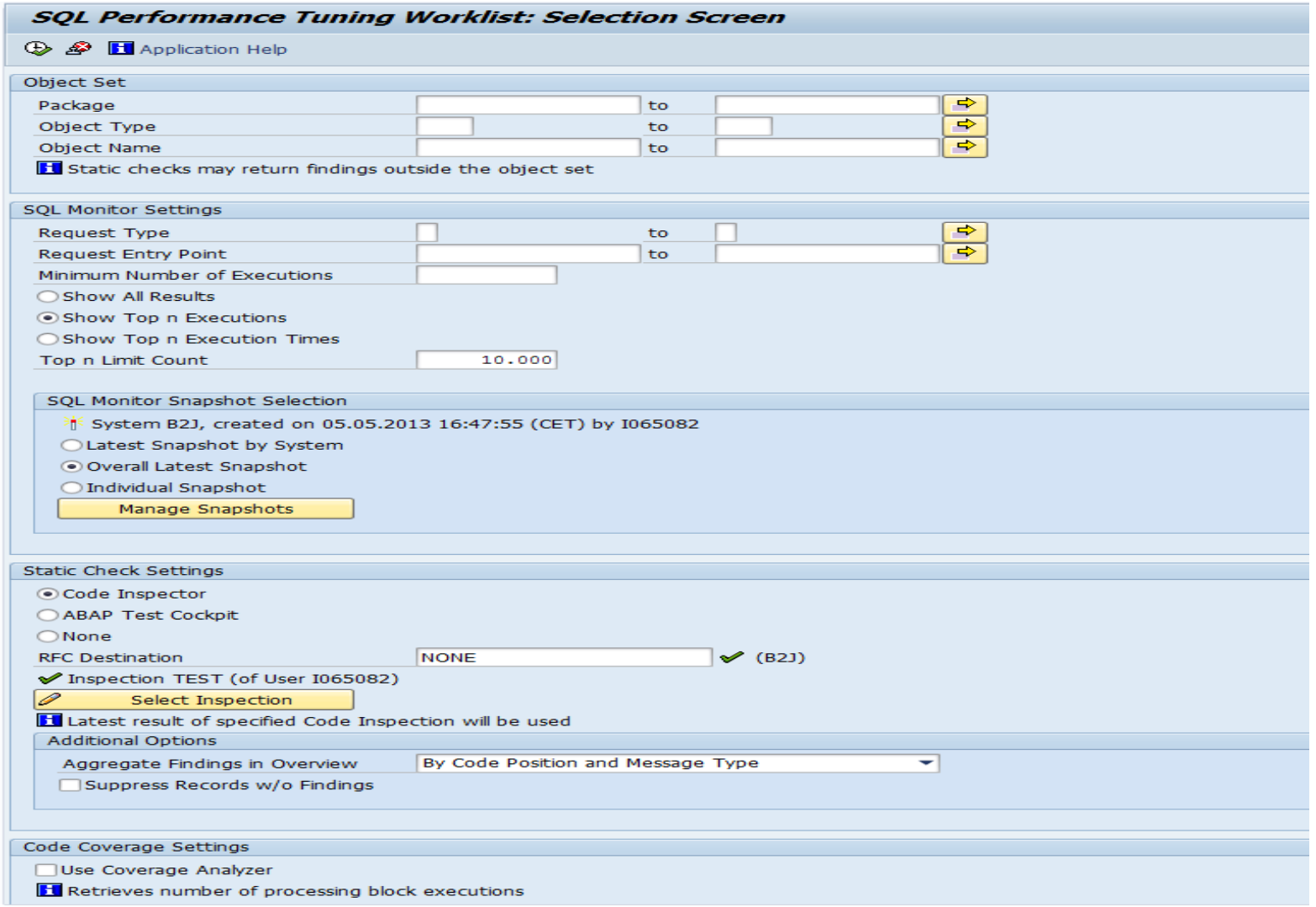
Step 4 – SQL Performance Tuning Worklist (TA – SWLT)

- In case we are only interested in static check results for SQL statements that have been recorded by the SQL Monitor, SAP advises to use the Code Inspector object collector ("Objects of SQLM-Snapshot").

- For the execution of static checks (Code Inspector) , SAP recommends using the pre-defined Code Inspector variant PERFORMANCE_DB.



Step 4 – SQL Performance Tuning Worklist (TA – SWLT)



Step 4 – SQL Performance Tuning Worklist (TA – SWLT)

- According to the selection options, we get a list of results (findings) that, in the standard version, are sorted according to the number of executions or the execution time.

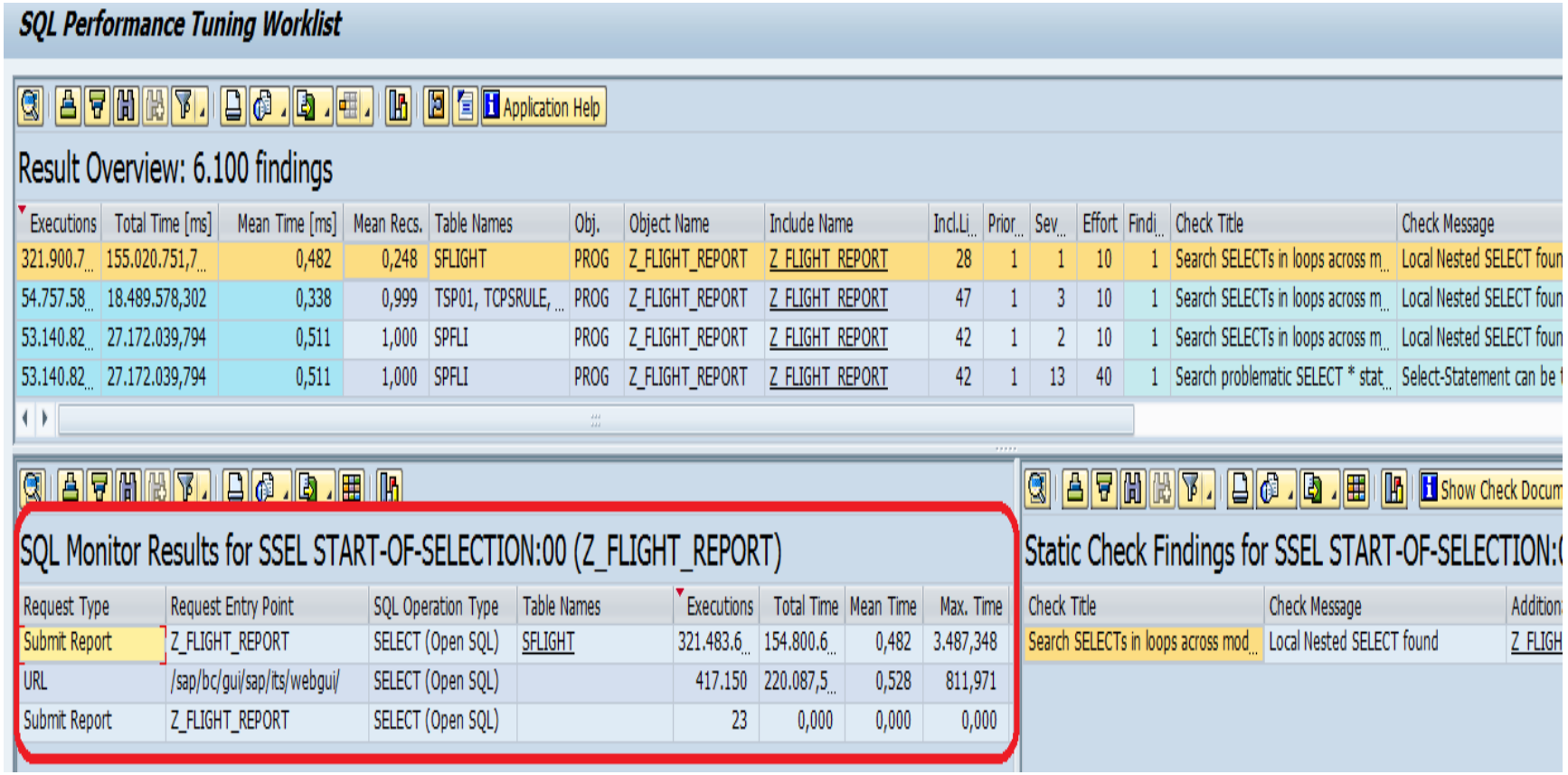
- From here we have the possibility to detect the performance hotspots and also double click on the line to get more details.

Executions	Total Time [ms]	Mean Time [ms]	Mean Recs.	Table Names	Obj.	Object Name	Include Name	Incl.Li...	Prior...	Sev...	Effort	Findi...	Check Title	Check Message
321.900.7...	155.020.751,7...	0,482	0,248	SFLIGHT	PROG	Z_FLIGHT_REPORT	Z_FLIGHT_REPORT	28	1	1	10	1	Search SELECTs in loops across m...	Local Nested SELECT found
54.757.58...	18.489.578,302	0,338	0,999	TSP01, TCPSRULE, ...	PROG	Z_FLIGHT_REPORT	Z_FLIGHT_REPORT	47	1	3	10	1	Search SELECTs in loops across m...	Local Nested SELECT found
53.140.82...	27.172.039,794	0,511	1,000	SPFLI	PROG	Z_FLIGHT_REPORT	Z_FLIGHT_REPORT	42	1	2	10	1	Search SELECTs in loops across m...	Local Nested SELECT found
53.140.82...	27.172.039,794	0,511	1,000	SPFLI	PROG	Z_FLIGHT_REPORT	Z_FLIGHT_REPORT	42	1	13	40	1	Search problematic SELECT * stat...	Select-Statement can be transfor...

Max. Time	Min. Time	Dev. Time	Records	Max. Re...	Min. Rec...	DevRec...	Int. Sess.	Exe./Se...	Buffering Type	Colu...	Key ...	Width ...	Store Type	Size ...	Table Class	Type	Processing Block Name
3.487,348	0,000	0,830	79.798,...	4	0	0,432	11.598	27.754,...	No Buffering	14	4	112	Column Store...	0	Transparent table	SSEL	START-OF-SELECTION:00
2.048,034	0,188	0,726	54.723,...	8	0	0,025	69.323	789,891	Unknown	0	0	0				SSEL	START-OF-SELECTION:00
1.908,296	0,399	0,805	53.140,...	3	1	0,000	11.575	4.591,0...	Table is completely tran...	16	3	168	Column Store...	1	Transparent table	SSEL	START-OF-SELECTION:00
1.908,296	0,399	0,805	53.140,...	3	1	0,000	11.575	4.591,0...	Table is completely tran...	16	3	168	Column Store...	1	Transparent table	SSEL	START-OF-SELECTION:00

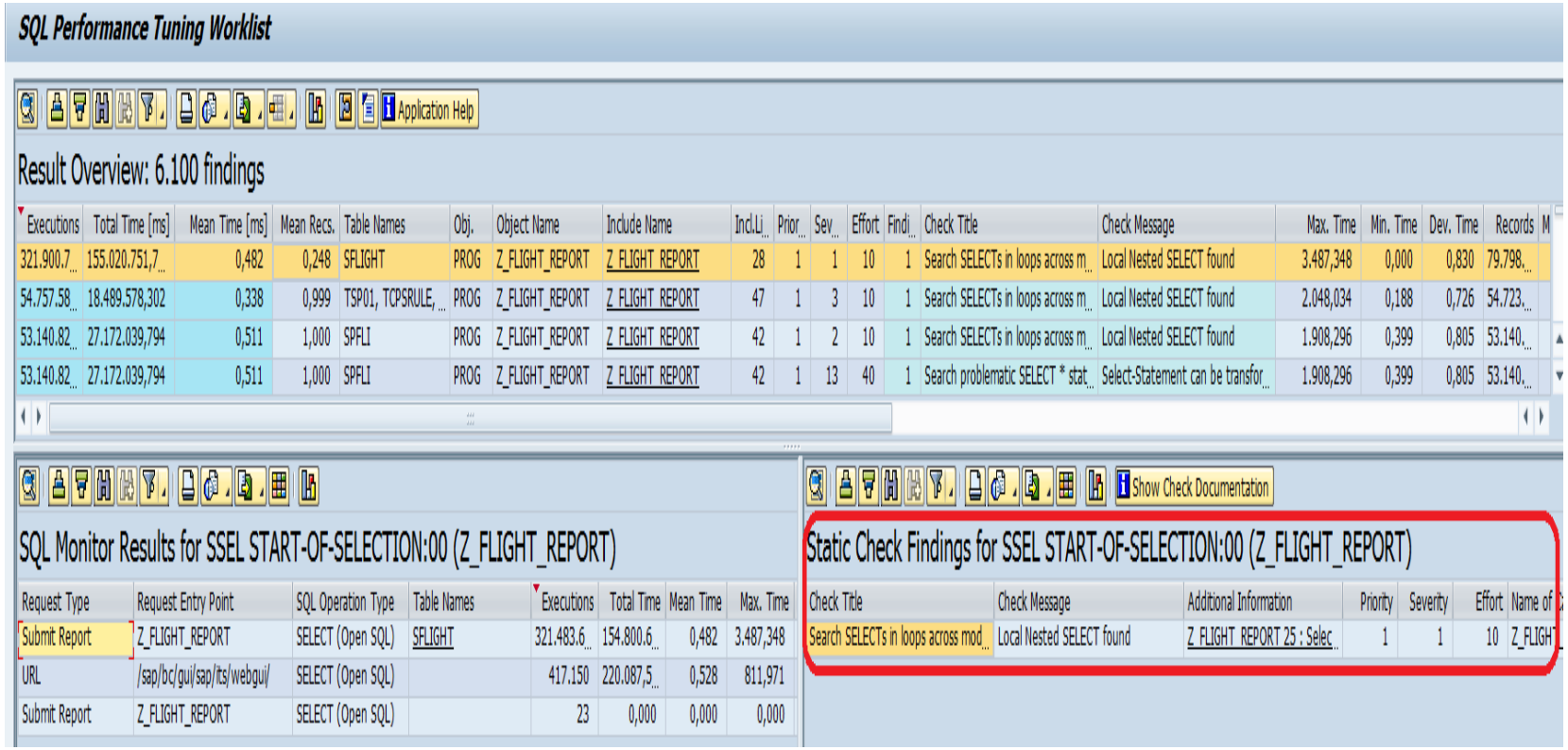
Step 4 – SQL Performance Tuning Worklist (TA – SWLT)

- **Detailed View with SQL Monitoring Data –** We can get all the SQLM data and determine all the entry points to the DB operation or process.



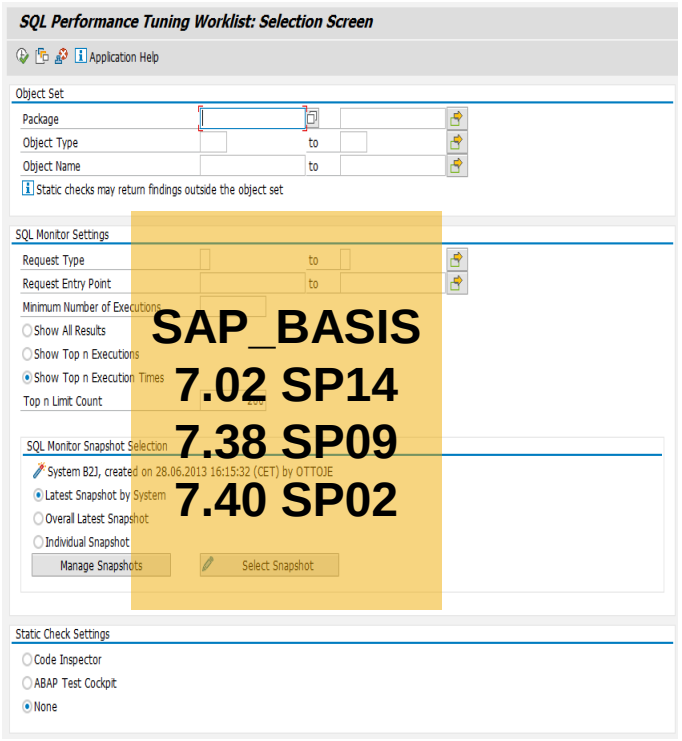
Step 4 – SQL Performance Tuning Worklist (TA – SWLT)

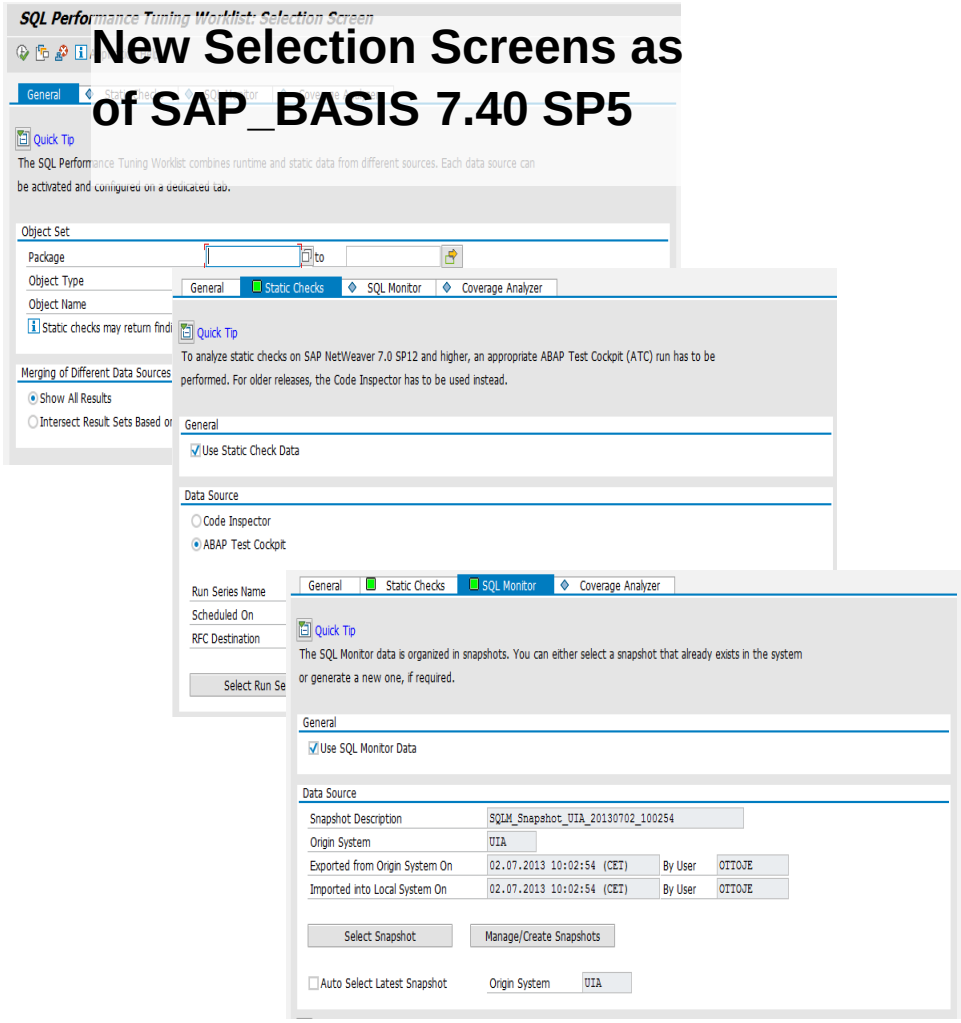
- **Detailed View with Results from Static Checks –** We can get the results from Static checks that provide the possible code optimization methods/ways. We get details like priority, severity, and estimated work effort etc.



SQL Performance Tuning Worklist
Transaction SWLT – Availability

SWLT has been made available with the following SAP_BASIS versions:





Thank you

ABAP on SAP HANA