Automated Tools for SAP HANA® Performance Optimization

Webinar Presenters:

Albrecht Gass, Chris Hanshew

Agenda

1. Introduction smartShift Technologies
2. SAP HANA Overview
3. SAP HANA Enhancements
4. HANA Code Compatibility
5. HANA Performance Optimization (HPO)
   - ABAP
   - Code Pushdown
6. Q & A

Note: You can also continue the discussion via Twitter @smartShiftTech with **#smartHANA**

About smartShift Technologies

About smartShift Tech

smartShift TECHNOLOGIES

RED HERRING EUROPE FINALIST 100

SAP® Certified
Powered by SAP NetWeaver®

Gartner 2012 CoolVendor

Select Customers

We Partner with the Best of the Breed!

Presenters

Albrecht Gass

Chief Architect

smartShift Technologies

Chris Hanshew

Senior Solution Architect

smartShift Technologies

SAP HANA Overview

HANA “Objectives”

- Does not impact transaction processing
- Enables “reporting without fear” by increasing reporting speeds dramatically
- Eliminates SAP as Access Loader
- Keeps processing within SAP and not Excel
- Avoid using old and/or partial data
- Convert batch processes to real-time operations
- Enable new big data processes

HANA Implementation Considerations

- Side-car vs. Primary DB
- Instance Size
  - Memory
  - CPU
- Table Partitioning
- Consider Changes in Query Result Ordering
- Incompatible Native SQL Code
- Unicode Requirement
- On-premise or Cloud Deployment

Why Code Optimization

- Data intensive programming logic in application server
  - Reversal of complete HW/SW abstraction, everything in ABAP
- Change in data architecture forces changes to application structure
  - SELECT * …
  - SORT vs. ORDER BY
  - CHECK within SELECT / END SELECT
- New HANA / NetWeaver framework functionality
  - List Display
  - Code Maintainability

smartScale for HANA Offering

smartScale for HANA will consist of the appropriate components with newly added HANA code optimization rules.

- Analyze
  - SaaS Analysis
- Upgrade & Unicode
  - Unicode Enablement
  - Change Impact Analysis
  - Automated Code Changes
- Develop
  - On-going ABAP Code Remediation
- Management
  - Migration Support
  - Monitor and Manage of Cloud Environment

Benefits for Automated Solution

- Get benefits quickly
- Resource Requirements
- Low Hanging Fruit
- Accuracy
- Repeatability
- Coverage
- Cost

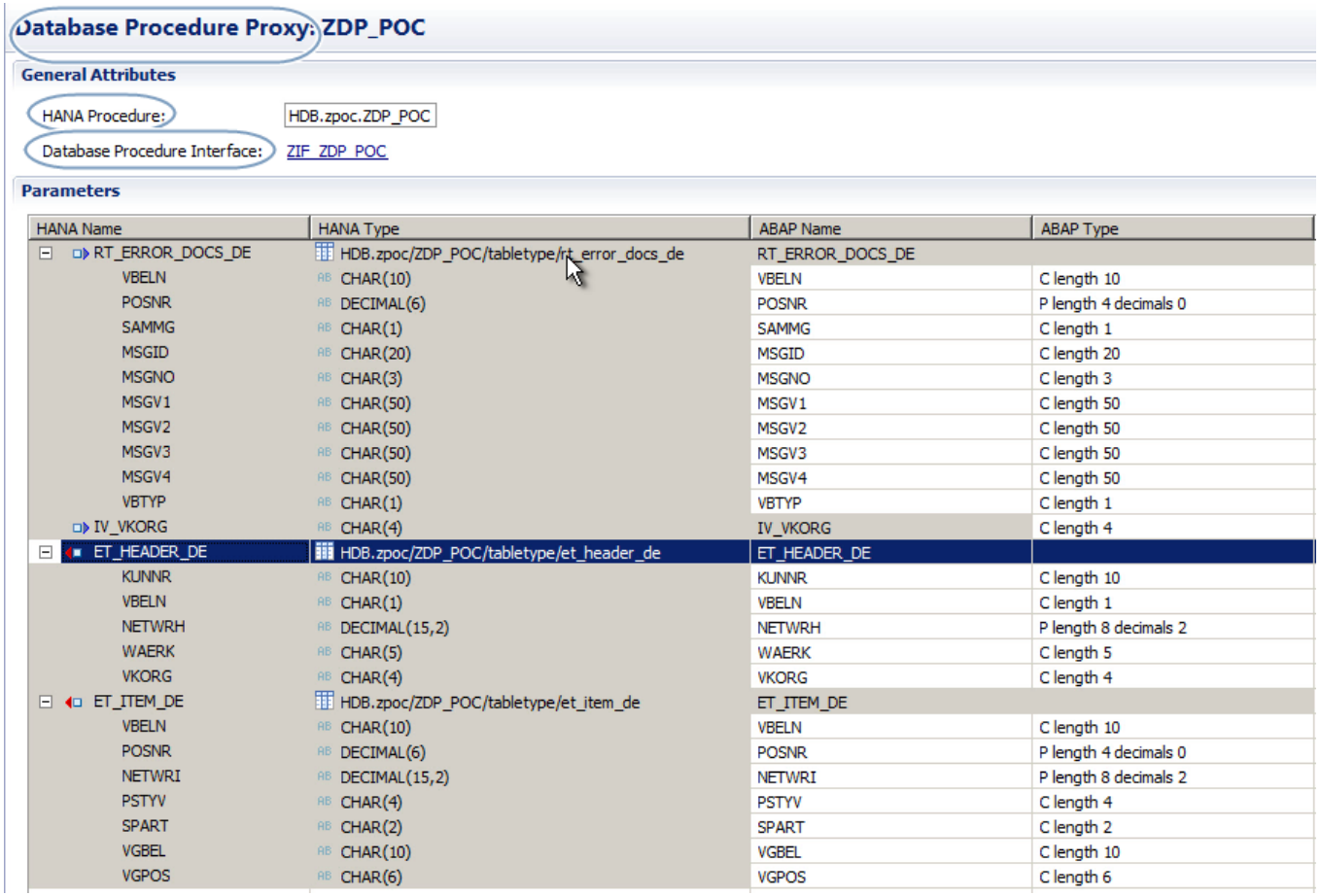
SAP HANA Enhancements

NetWeaver 7.4 ABAP Enhancements

- ABAP Managed Database Procedures (AMDP)
- Integrated Data Access List Grid (IDA-ALV)
- Inline Declarations
- Table Expressions
- New Internal Table Functions
- ABAP Objects (Exporting, Importing and Changing, Partially Implemented Interfaces for Testing)
- Reference Operator
- Value Operator
- Constructor Operator
- Conditional Operators
- Conversion Operators
- Lossless Operator EXACT
- Internal Tables with Empty Keys
- Expressions

Pushing code to DB layer - Step 1

- Define procedure



Database Procedure Proxy: ZDP_POC

General Attributes

HANA Procedure: HDB.zpoc.ZDP_POC

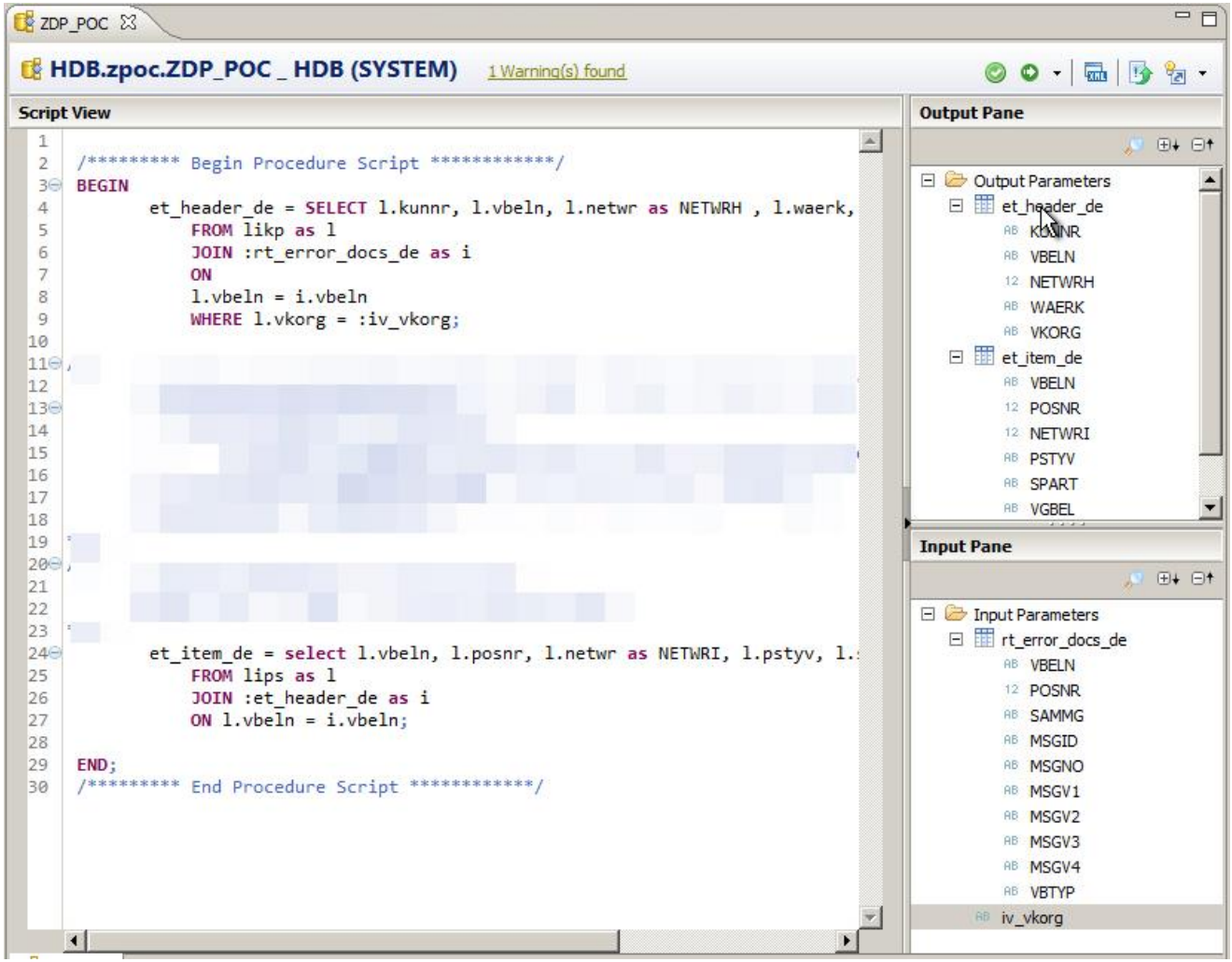
Database Procedure Interface: ZIF_ZDP_POC

Parameters

HANA Name	HANA Type	ABAP Name	ABAP Type
RT_ERROR_DOCS_DE	HDB.zpoc/ZDP_POC/tabletype/rt_error_docs_de	RT_ERROR_DOCS_DE	
VBELN	CHAR(10)	VBELN	C length 10
POSNR	DECIMAL(6)	POSNR	P length 4 decimals 0
SAMMG	CHAR(1)	SAMMG	C length 1
MSGID	CHAR(20)	MSGID	C length 20
MSGNO	CHAR(3)	MSGNO	C length 3
MSGV1	CHAR(50)	MSGV1	C length 50
MSGV2	CHAR(50)	MSGV2	C length 50
MSGV3	CHAR(50)	MSGV3	C length 50
MSGV4	CHAR(50)	MSGV4	C length 50
VBTYP	CHAR(1)	VBTYP	C length 1
IV_VKORG	CHAR(4)	IV_VKORG	C length 4
ET_HEADER_DE	HDB.zpoc/ZDP_POC/tabletype/et_header_de	ET_HEADER_DE	
KUNNR	CHAR(10)	KUNNR	C length 10
VBELN	CHAR(1)	VBELN	C length 1
NETWRH	DECIMAL(15,2)	NETWRH	P length 8 decimals 2
WAERK	CHAR(5)	WAERK	C length 5
VKORG	CHAR(4)	VKORG	C length 4
ET_ITEM_DE	HDB.zpoc/ZDP_POC/tabletype/et_item_de	ET_ITEM_DE	
VBELN	CHAR(10)	VBELN	C length 10
POSNR	DECIMAL(6)	POSNR	P length 4 decimals 0
NETWRI	DECIMAL(15,2)	NETWRI	P length 8 decimals 2
PSTYV	CHAR(4)	PSTYV	C length 4
SPART	CHAR(2)	SPART	C length 2
VGBEL	CHAR(10)	VGBEL	C length 10
VGPOS	CHAR(6)	VGPOS	C length 6

Pushing code to DB layer – Step 2

- SQL script that implements reading from relevant tables



Pushing code to DB layer – Step 3

- Generated interface object, to be used in application logic

```
interface ZIF_ZDP_POC public.

" This interface pool has been generated.
" It contains the type definitions for
" database procedure proxy ZDP_POC
" representing db procedure _SYS_BIC.HDB.zpoc/ZDP_POC

types: begin of rt_error_docs_de,
         vbeln                          type c length 10,
         posnr                          type p length 4 decimals 0,
         sammg                          type c length 1,
         msgid                          type c length 20,
         msgno                          type c length 3,
         msgv1                          type c length 50,
         msgv2                          type c length 50,
         msgv3                          type c length 50,
         msgv4                          type c length 50,
         vbtyp                          type c length 1,
       end of rt_error_docs_de.
types: iv_vkorg                        type c length 4.
types: begin of et_header_de,
         kunnr                          type c length 10,
         vbeln                          type c length 1,
         netwrh                         type p length 8 decimals 2,
         waerk                          type c length 5,
         vkorg                          type c length 4,
       end of et_header_de.
types: begin of et_item_de,
         vbeln                          type c length 10,
         posnr                          type p length 4 decimals 0,
         netwri                         type p length 8 decimals 2,
         pstyv                          type c length 4,
         spart                          type c length 2,
         vgbel                          type c length 10,
         vgpos                          type c length 6,
       end of et_item_de.

endinterface .
```

Pushing code to DB layer - Step 4

- Modify code to use interface / call procedure

```
58    DATA: lt_delivery_data TYPE TABLE OF zv_delivery_data.
59    DATA: ls_delivery_data LIKE LINE OF lt_delivery_data.
60    DATA: ls_item_de LIKE LINE OF rt_item_de.
61
62    DATA: lv_vkorg TYPE zif_zdp_poc=>iv_vkorg VALUE '1000'.
63
64    DATA: ls_error_docs_de LIKE LINE OF rt_error_docs_de.
65    DATA: ls_header_de LIKE LINE OF rt_header_de.
66
67    DATA: lt_error_docs_de_db TYPE STANDARD TABLE OF zif_zdp_poc=>rt_error_docs_de.
68    DATA: ls_error_docs_de_db TYPE zif_zdp_poc=>rt_error_docs_de.
69
70    DATA: ls_header_de_db TYPE zif_zdp_poc=>et_header_de.
71    DATA: lt_header_de_db TYPE STANDARD TABLE OF zif_zdp_poc=>et_header_de.
72
73    DATA: ls_item_de_db TYPE zif_zdp_poc=>et_item_de.
74    DATA: lt_item_de_db TYPE STANDARD TABLE OF zif_zdp_poc=>et_item_de.
75
76  IF NOT rt_error_docs_de[] IS INITIAL.
77
78
79
80
81
82
83
84
85
86
87
88    CALL DATABASE PROCEDURE zdp_poc
89          EXPORTING
90              iv_vkorg = lv_vkorg
91              rt_error_docs_de = lt_error_docs_de_db
92          IMPORTING
93              et_header_de = lt_header_de_db
94              et_item_de = lt_item_de_db
95              .
96    IF sy-subrc <> 0.
97        MESSAGE I034 WITH 'Failure'(015).
98    ELSE.
```

New ABAP language feature!

ABAP Managed Database Procedures (AMDP)

```
CLASS zcl_amdp DEFINITION.
    INTERFACES if_amdp_marker_hdb.
    METHODS: pushdown_code
            IMPORTING VALUE(iv_client) TYPE mandt
            EXPORTING VALUE(et_result) TYPE tt_result.
ENDCLASS.

CLASS zcl_amdp IMPLEMENTATION.
    METHOD pushdown_code BY DATABASE PROCEDURE
            FOR HDB LANGUAGE SQLSCRIPT
            USING snwd_so_i snwd_so_sl snwd_pd.
                    -- SQLScript goes here

    ENDMETHOD.
ENDCLASS.
```

Use of Integrated Data Access List Grid (IDA-ALV)

```
MODULE pbo OUTPUT.
    IF g_custom_container IS INITIAL.
        CREATE OBJECT g_custom_container
            EXPORTING
                container_name = g_container.

*       Instantiate IDA_ALV (ALV with IDA=Integrated Data Access)
        data(lo_alv_display) = cl_salv_gui_table_ida=>create( iv_table_name = 'ZV_DELIVERY_DATA'
                                                              io_gui_container = g_custom_container ).

        data(lo_collector) = new cl_salv_range_tab_collector( ).  "Helper Class
        lo_collector->add_ranges_for_name( iv_name = 'VKORG' it_ranges = o_vkorg[] ).
        lo_collector->get_collected_ranges( IMPORTING et_named_ranges = data(lt_name_range_pairs) ).
        lo_alv_display->set_select_options( it_ranges = lt_name_range_pairs ).


*       Initial Grouping
        lo_alv_display->default_layout( )->set_sort_order( value #( ( field_name = 'VKORG' is_grouped = abap_true )
                                                                    ( field_name = 'SPART' is_grouped = abap_true )
                                                                    ( field_name = 'VBELN' is_grouped = abap_true ) ) ).

    ENDIF.
ENDMODULE.
```

Access a view

Add applicable ranges

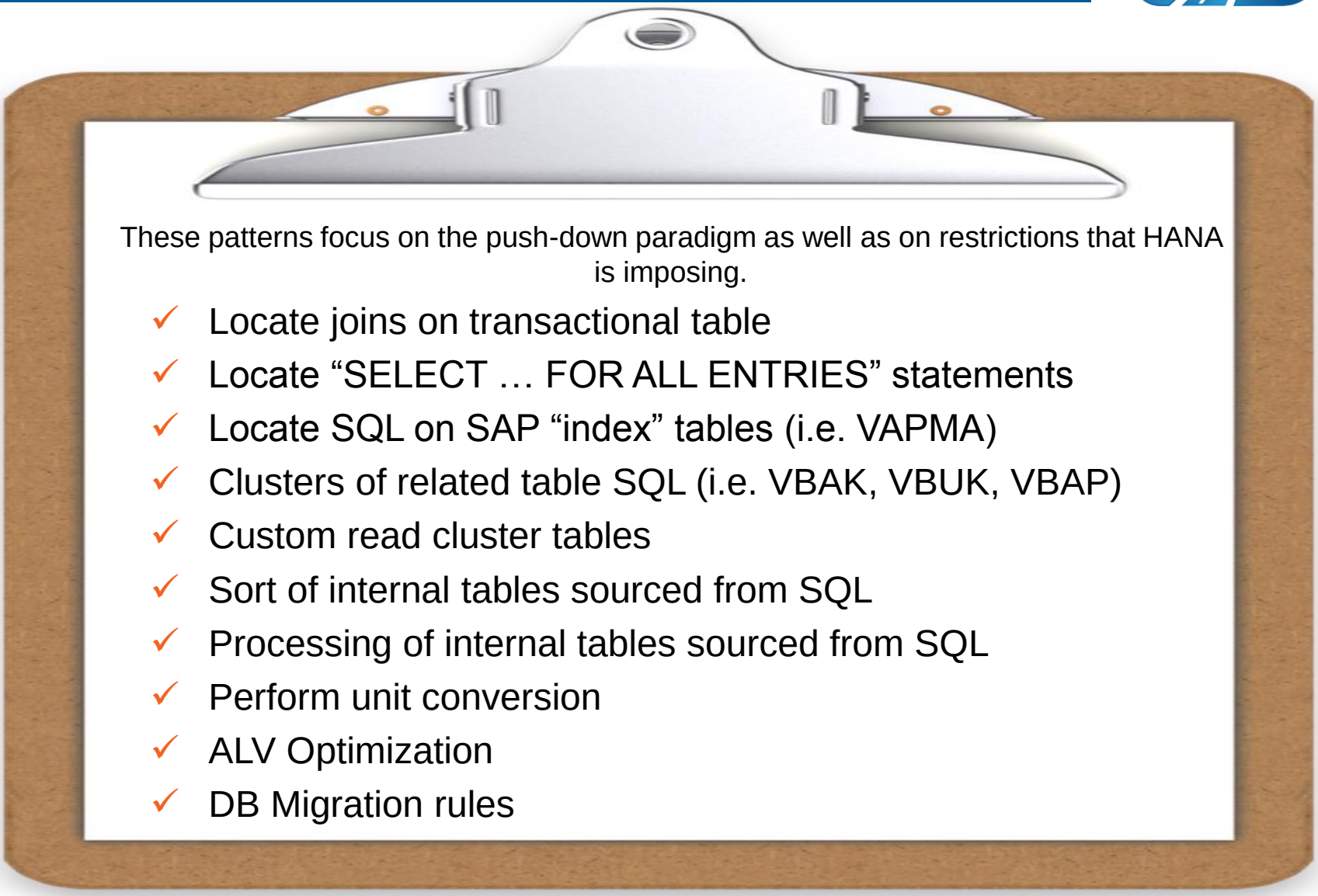
Define sort order

Performance Guidelines and Rules

HANA Optimization Guidelines

- Reduce result set
- Reduce amount of data transfer
- Reduce number of DB round trips
- Index/Query optimization
- Text search for F4 help and type-ahead search
- Avoid native SQL
- Use buffering on application server
- Existing best practices still apply
- Don’t overload HANA server

New HANA Rule Concepts



These patterns focus on the push-down paradigm as well as on restrictions that HANA is imposing.

- ✓ Locate joins on transactional table
- ✓ Locate “SELECT … FOR ALL ENTRIES” statements
- ✓ Locate SQL on SAP “index” tables (i.e. VAPMA)
- ✓ Clusters of related table SQL (i.e. VBAK, VBUK, VBAP)
- ✓ Custom read cluster tables
- ✓ Sort of internal tables sourced from SQL
- ✓ Processing of internal tables sourced from SQL
- ✓ Perform unit conversion
- ✓ ALV Optimization
- ✓ DB Migration rules

Optimized ABAP Code Examples

New HANA Rules

Only a subset of rules are referenced here

- HCC
  - #600 SELECT on former cluster/pool tables
- HPO
  - #601 SELECT with PACKAGE SIZE option
  - #602 SELECT/ENDSELECT are indicators for code optimization
  - #610 HANA Performance Optimization using code push-down
  - #712 SELECT * --> Reduce selected column set
  - #719 SELECT with IF/CHECK
  - #723 OpenSQL modifications inside LOOPS
  - #727 SELECT followed by DELETE ADJACENT DUPLICATES
  - #728 Index Table Access

Rule #600 - SELECT on former cluster/pool tables

- Cluster and Pool tables return data implicitly sorted by the primary key
- In HANA cluster/pool tables are converted into transparent tables – this might changed in future SPs
- Not all cases need adjustment, only where ordering is important
- Customer case (via Code Inspector)
  - 4,500 instances
  - Interestingly there are 1,500 instance where no ordering is specified but the code might rely on order

Rule #600 - SELECT on former cluster/pool tables

- Code Comparison

Rule #712 SELECT * --> Reduce column set

- In row based RDBMS the penalty for not restricting the selected column set is small.
- For column tables in HANA, there are multiple reasons why this becomes important
  - Reduced processing on DB server
  - Reduced data transfer
  - Reduced overhead in result set realization on app server
- Needs the ability to detect access to columns in result set and related data structures.
- Many pit falls and conditions that cannot be easily investigated manually.

Rule #712 SELECT * --> Reduce column set

- Code Comparison

Rule #719 SELECT/LOOP with IF/CHECK

- Checks that filter out data that has been sourced from the DB should be pushed to the DB
- Applies to many constructs
  - SELECT
  - SELECT/ENDSELECT
  - LOOP
  - OPEN CURSOR
- Existing restrictions for OpenSQL must be considered
- HANA benefits
  - Reduced data transfer
  - Reduced overhead in result set realization on app server

Rule #719 SELECT/LOOP with IF/CHECK

- Code Comparison

Rule #723 OpenSQL modifications inside LOOPS

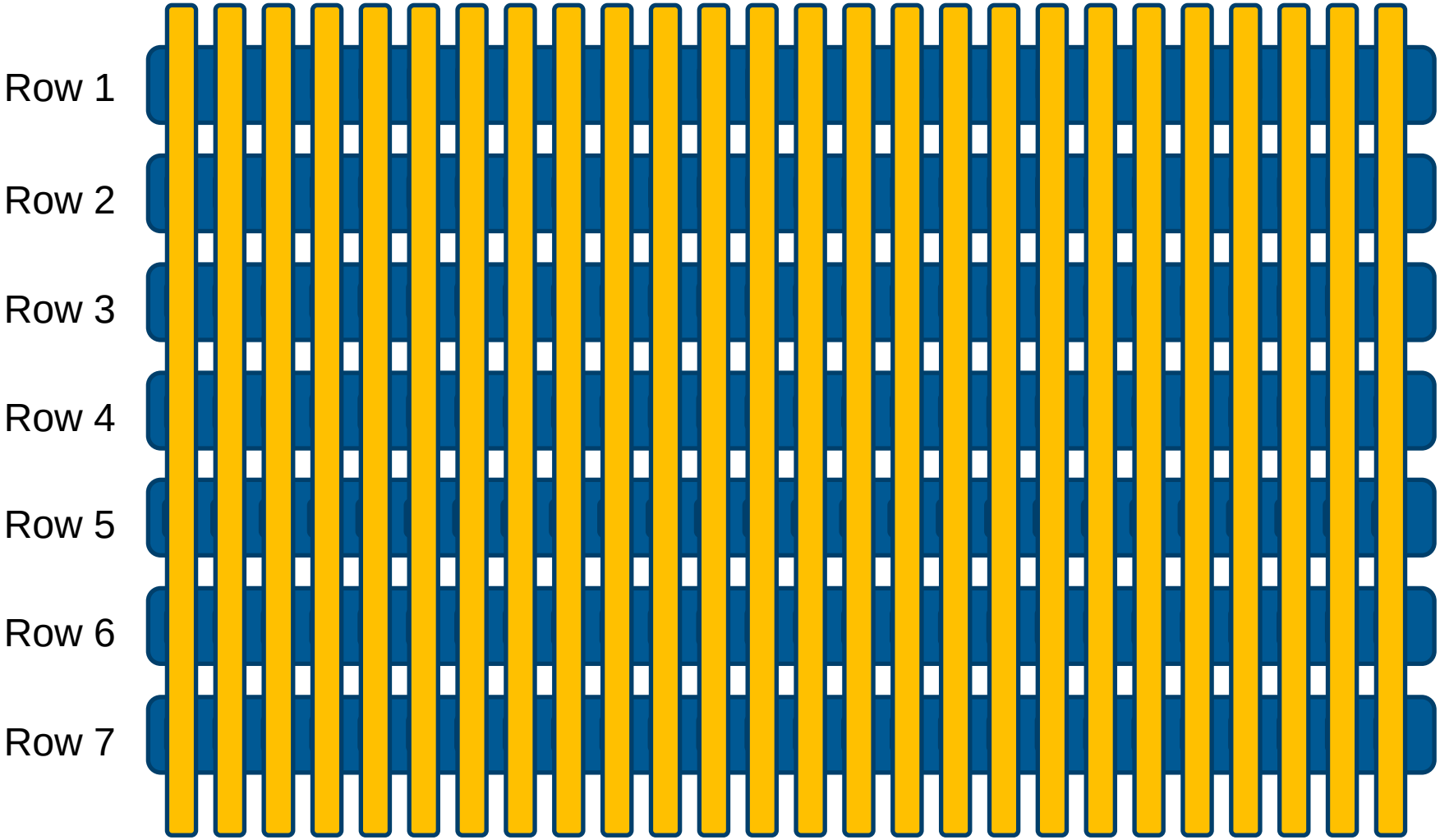
- Data modifications that are done record by record are very expensive in.
- Approach is to gather up the data modifications in the loop and then execute a single OpenSQL statement to perform bulk processing.

Rule #723 OpenSQL modifications inside LOOPS

- Code Comparison

Rule #610 HANA Performance Optimization using code push-down

Column based, parallelized processing in HANA with SQLScript



Rule #610 HANA Performance Optimization using code push-down

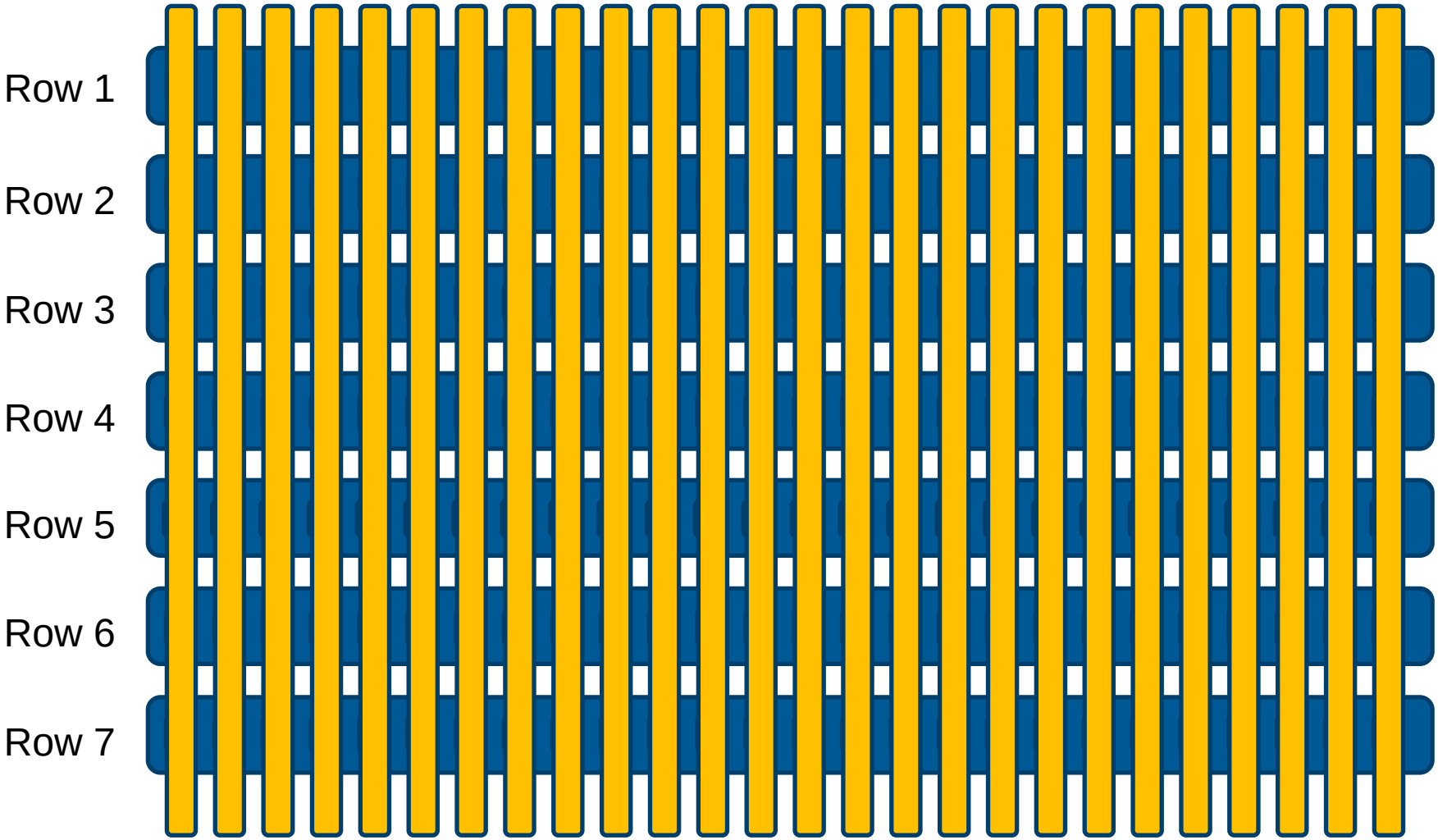
- Large ABAP code segments to be moved to DB
- Funnel use case
  - A lot of data coming in, only small amount of data is the result
- Take advantage of HANA capabilities and strengths
  - Parallization
  - Fast in memory operations
  - Reduce data transfer
- Very resource intensive if attempted manually
- Highest level of difficulty

Rule #610 HANA Performance Optimization using code push-down

- Code Comparison

Rule #610 HANA Performance Optimization using code push-down

Column based, parallelized processing in HANA



smartScale for HANA

Automation is your friend!

Decide your HANA needs, and choose the right tools at the right steps:

- Analyze the scope
- Achieve HANA pre-requisites - Upgrade & Unicode
- Develop new code for HPO
- Manage in the Cloud

Request a Demo

Request a Custom Demo

http://info.smartshifttech.com/demo-automated-abap-code-remediation-tool-by-smartshift