HA400

ABAP Programming on SAP HANA

SAP HANA

Date _____

Training Center _____

Instructors _____

_____

Education Website _____

Participant Handbook

Course Version: 98
Course Duration: 2 Day(s)
Material Number: 50117467

An SAP course - use it to learn, reference it for work

Copyright

Copyright © 2013 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Trademarks

- Microsoft®, WINDOWS®, NT®, EXCEL®, Word®, PowerPoint® and SQL Server® are registered trademarks of Microsoft Corporation.
- IBM®, DB2®, OS/2®, DB2/6000®, Parallel Sysplex®, MVS/ESA®, RS/6000®, AIX®, S/390®, AS/400®, OS/390®, and OS/400® are registered trademarks of IBM Corporation.
- ORACLE® is a registered trademark of ORACLE Corporation.
- INFORMIX®-OnLine for SAP and INFORMIX® Dynamic ServerTM are registered trademarks of Informix Software Incorporated.
- UNIX®, X/Open®, OSF/1®, and Motif® are registered trademarks of the Open Group.
- Citrix®, the Citrix logo, ICA®, Program Neighborhood®, MetaFrame®, WinFrame®, VideoFrame®, MultiWin® and other Citrix product names referenced herein are trademarks of Citrix Systems, Inc.
- HTML, DHTML, XML, XHTML are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.
- JAVA® is a registered trademark of Sun Microsystems, Inc.
- JAVASCRIPT® is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.
- SAP, SAP Logo, R/2, RIVA, R/3, SAP ArchiveLink, SAP Business Workflow, WebFlow, SAP EarlyWatch, BAPI, SAPPHIRE, Management Cockpit, mySAP.com Logo and mySAP.com are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other products mentioned are trademarks or registered trademarks of their respective companies.

Disclaimer

THESE MATERIALS ARE PROVIDED BY SAP ON AN "AS IS" BASIS, AND SAP EXPRESSLY DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR APPLIED, INCLUDING WITHOUT LIMITATION WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WITH RESPECT TO THESE MATERIALS AND THE SERVICE, INFORMATION, TEXT, GRAPHICS, LINKS, OR ANY OTHER MATERIALS AND PRODUCTS CONTAINED HEREIN. IN NO EVENT SHALL SAP BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR PUNITIVE DAMAGES OF ANY KIND WHATSOEVER, INCLUDING WITHOUT LIMITATION LOST REVENUES OR LOST PROFITS, WHICH MAY RESULT FROM THE USE OF THESE MATERIALS OR INCLUDED SOFTWARE COMPONENTS.

About This Handbook

This handbook is intended to complement the instructor-led presentation of this course, and serve as a source of reference. It is not suitable for self-study.

Typographic Conventions

American English is the standard used in this handbook. The following typographic conventions are also used.

Type Style	Description
Example text	Words or characters that appear on the screen. These include field names, screen titles, pushbuttons as well as menu names, paths, and options. Also used for cross-references to other documentation both internal and external.
Example text	Emphasized words or phrases in body text, titles of graphics, and tables
EXAMPLE TEXT	Names of elements in the system. These include report names, program names, transaction codes, table names, and individual key words of a programming language, when surrounded by body text, for example SELECT and INCLUDE.
`Example text`	Screen output. This includes file and directory names and their paths, messages, names of variables and parameters, and passages of the source text of a program.
`Example text`	Exact user entry. These are words and characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Pointed brackets indicate that you replace these words and characters with appropriate entries.

© 2013 SAP AG. All rights reserved.

Icons in Body Text

The following icons are used in this handbook.

Icon	Meaning
	For more information, tips, or background
	Note or further explanation of previous point
	Exception or caution
	Procedures
	Indicates that the item is displayed in the instructor's presentation.

© 2013 SAP AG. All rights reserved.

Contents

© 2013 SAP AG. All rights reserved.

© 2013 SAP AG. All rights reserved.

Course Overview

Target Audience

This course is intended for the following audiences:

- Primary target audiences for this training are developers and development consultants but also other roles involved with implementing or reviewing program code to optimize ABAP based applications for SAP HANA.

Course Prerequisites

Required Knowledge

- HA100 HANA overview
- BC400 ABAP Workbench
- BC401 ABAP Objects
- BC402 Advanced ABAP
- Experience ABAP Programing (procedure + object oriented)
- Open SQL (joins, view, aggregations)

Recommended Knowledge

- HA150 SQL Basics for SAP HANA

Course Goals

This course will prepare you to:

- Develop and optimize ABAP applications that access data stored in the SAP HANA Database.

Course Objectives

After completing this course, you will be able to:

- Understand the Technical SAP HANA concepts
- Understand Optimization of classical ABAP in HANA Context
- Describe the usage of Analysis Tools (Runtime Analysis, Code Inspector, SQL Trace)
- Understand SQL Performance Rules of ABAP for HANA
- Explain Implementing ABAP report on HANA using ADBC (ABAP Data Base Connectivity)
- Explain Consuming HANA views in ABAP

© 2013 SAP AG. All rights reserved.

- Explain Creating HANA views and Consuming in ABAP

© 2013 SAP AG. All rights reserved.

Unit 1

Introduction and Technical Concepts

Unit Overview

This Unit is designed to teach the following topics:

- Introduction
- HANA Technical concepts
- Introduction to HANA Studio

Unit Objectives

After completing this unit, you will be able to:

- Describe the HANA evolution, architecture, and direction
- Understand the fundamental technical concepts of SAP HANA
- Explain the central functions of HANA Studio
- Choose different perspectives
- Set up a connection to a SAP HANA Database

Unit Contents

© 2013 SAP AG. All rights reserved.

Lesson: Introduction

Lesson Overview

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the HANA evolution, architecture, and direction

Business Example

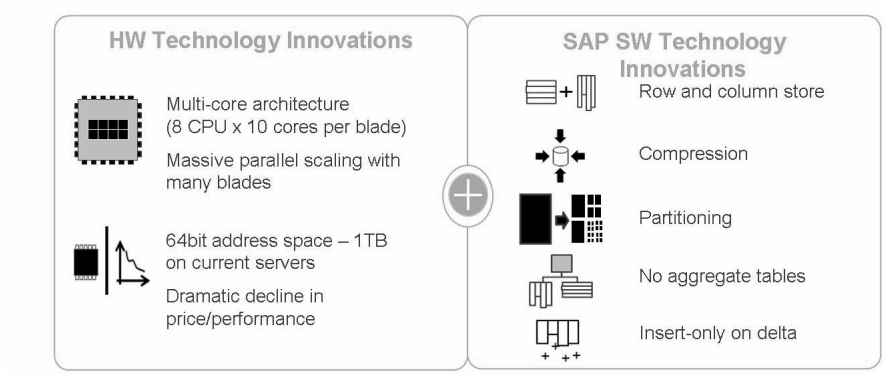


Figure 1: Technology Innovations

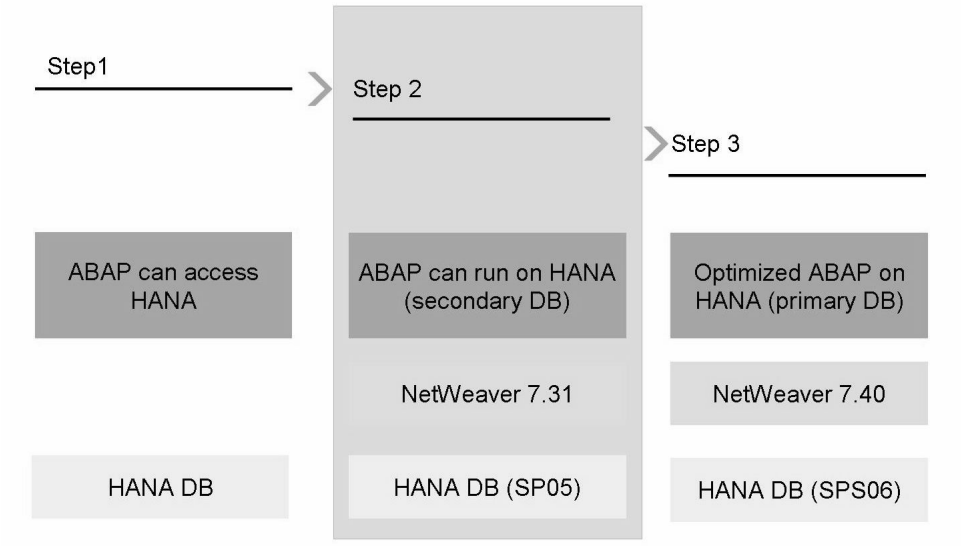


Figure 2: HANA Evolution

© 2013 SAP AG. All rights reserved.

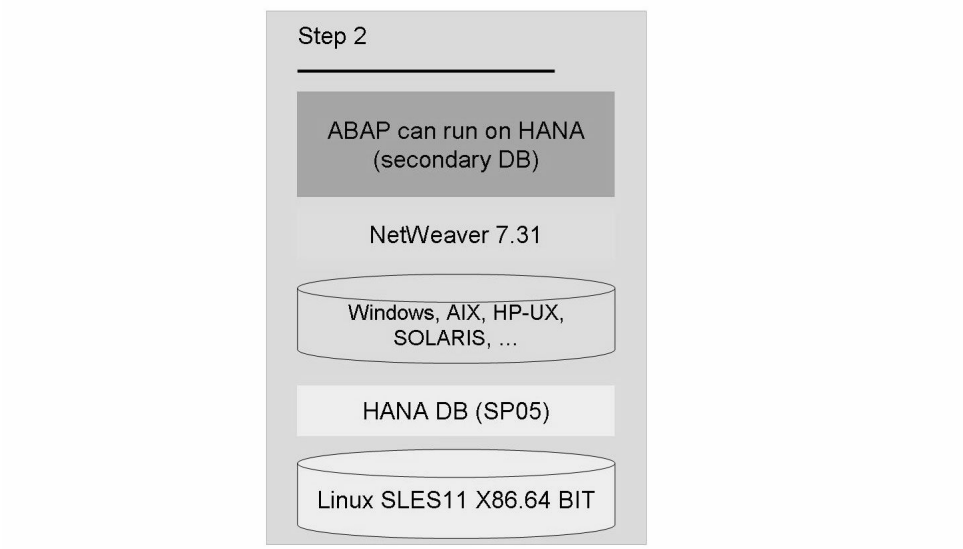


Figure 3: HANA Architecture

- NetWeaver 64 BIT is mandatory
- NetWeaver dual stack is not supported anymore

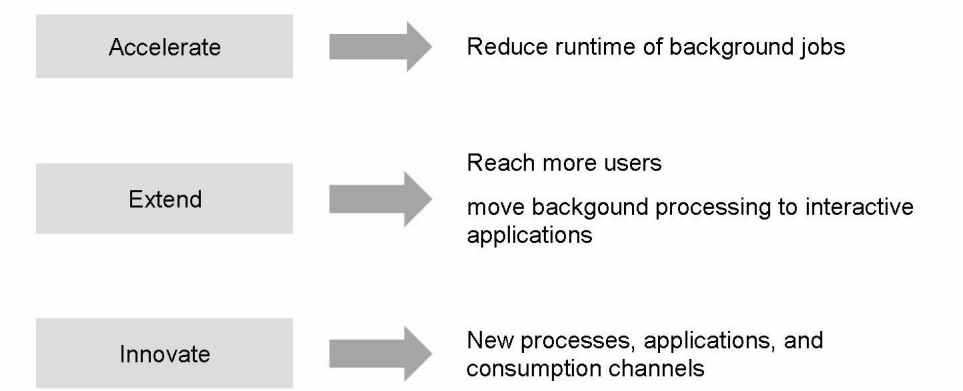


Figure 4: HANA Direction

© 2013 SAP AG. All rights reserved.

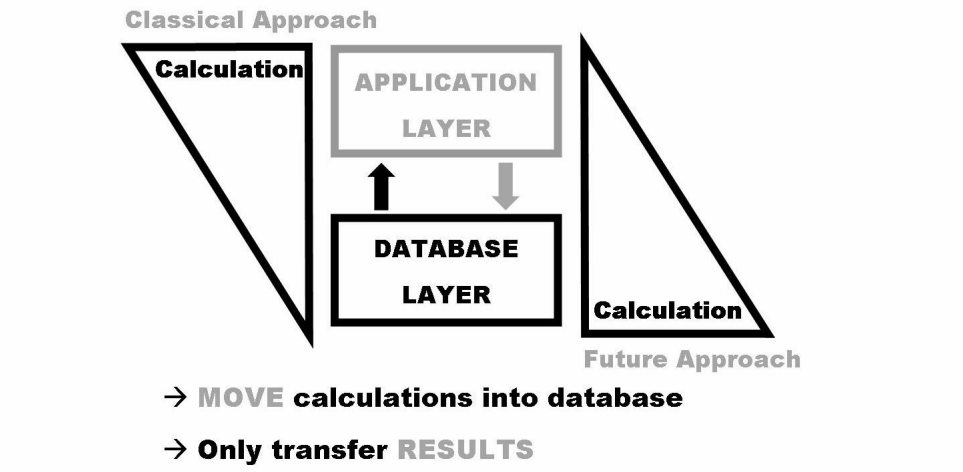


Figure 5: Avoid Bottlenecks

- How can I detect HANA optimization potential?
- How can I optimize ABAP Code for HANA?
- Do I have to rewrite everything?

Figure 6: SAP HANA: ABAP Developers Questions

© 2013 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Describe the HANA evolution, architecture, and direction

© 2013 SAP AG. All rights reserved.

Lesson: HANA Technical Concepts

Lesson Overview

Lesson Objectives

After completing this lesson, you will be able to:

- Understand the fundamental technical concepts of SAP HANA

Business Example

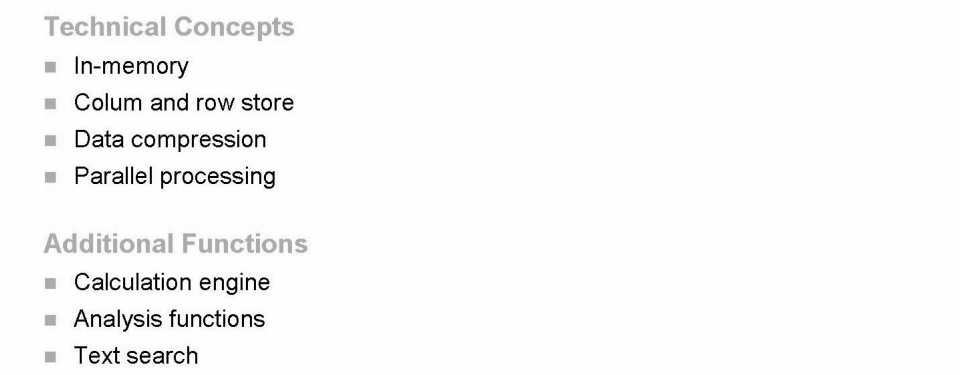


Figure 7: SAP HANA Facts

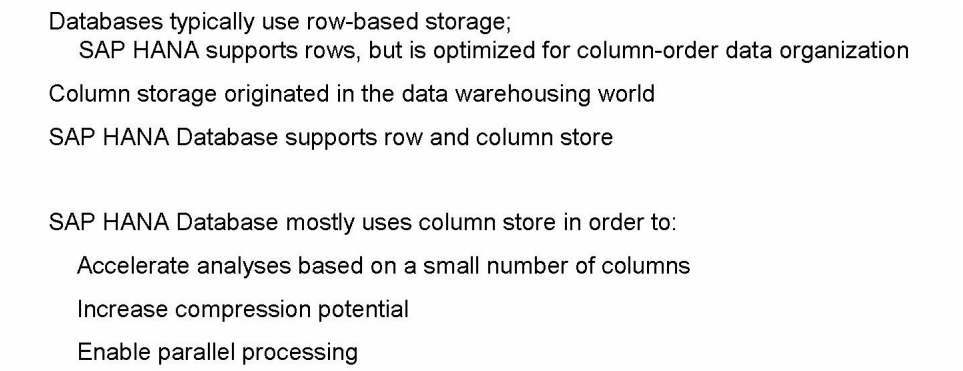


Figure 8: Column and Row Store

© 2013 SAP AG. All rights reserved.

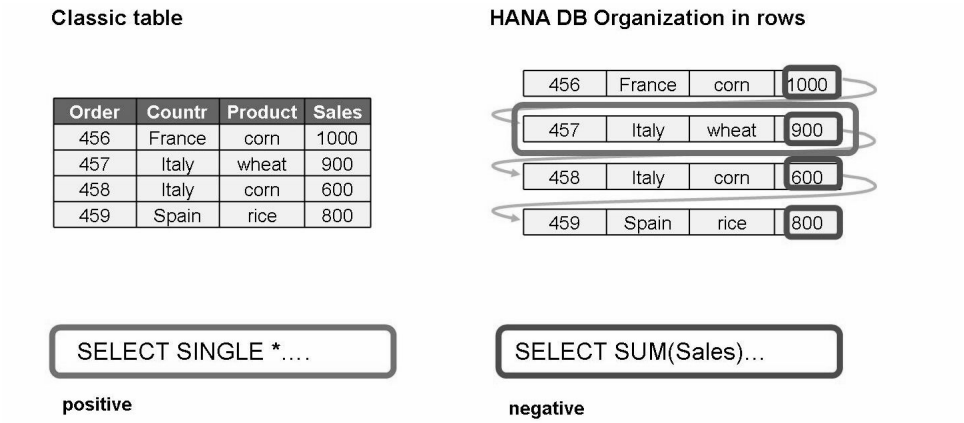


Figure 9: Row Store

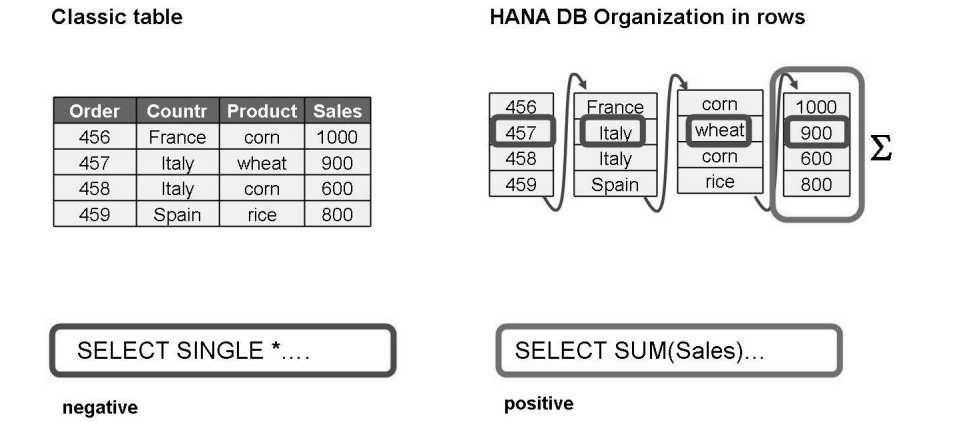


Figure 10: Column Store

By accessing data in column-store order, you benefit immensely from simplified table-scan and data pre-caching.

This can make all the difference in performance.

© 2013 SAP AG. All rights reserved.

- Column store is best for:
  - Column-based operations on large numbers of rows
  - Large number of columns, especialy when many are unused
  - For aggregations and intensive search (e.g. fuzzy text search)
- Row store is more suitable when:
  - Tables contain mainly distinct values
  - All colums of the tables are relevant
  - No aggregation or search by non index columns is required
  - Table is fully buffered

Figure 11: Column and Row Store

- Compress repeated values in column memory
- Works best on sorted columns with few distinct values
- Other encodings in other cases

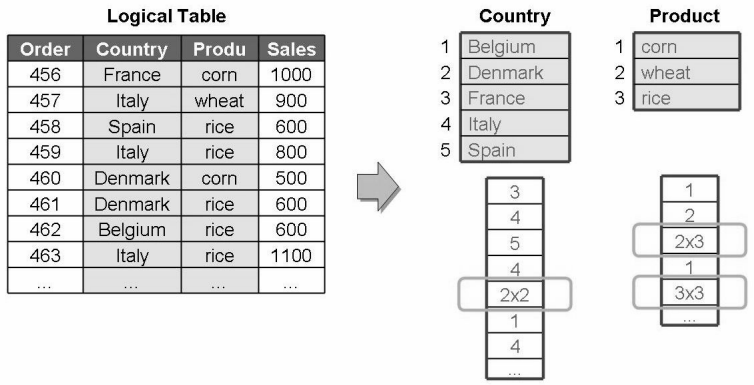


Figure 12: Example: Compression of Column

Prefix encoding, sparse encoding, delta encoding

- Efficient compression methods (run length, bit vector, dictionary, etc.)
- Compression works well with columns and can speedup operations on columns
- Because of compression, write changes into less compressed delta storage (uses also a dictionary)
- Needs to be merged into columns from time to time or when a certain size is exceeded
- Delta merge can be done asynchronously
- Trade-off between compression ratio and delta merge runtime

© 2013 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Understand the fundamental technical concepts of SAP HANA

© 2013 SAP AG. All rights reserved.

Lesson: Introduction to HANA Studio

Lesson Overview

Lesson Objectives

After completing this lesson, you will be able to:

- Explain the central functions of HANA Studio
- Choose different perspectives
- Set up a connection to a SAP HANA Database

Business Example

Eclipse-Based

Widely-accepted development environment

Tools based on existing Eclipse frameworks

Common development environment for SAP HANA and ABAP (ADT) going forward

Several Perspectives

A perspective is a window within the Eclipse workbench that provides functions related to a particular task:

- Administration
- Modeling

Views

Each perspective is made up of a number of panes called views

- Navigator
- Table Editor...

Figure 13: Central Facts

© 2013 SAP AG. All rights reserved.

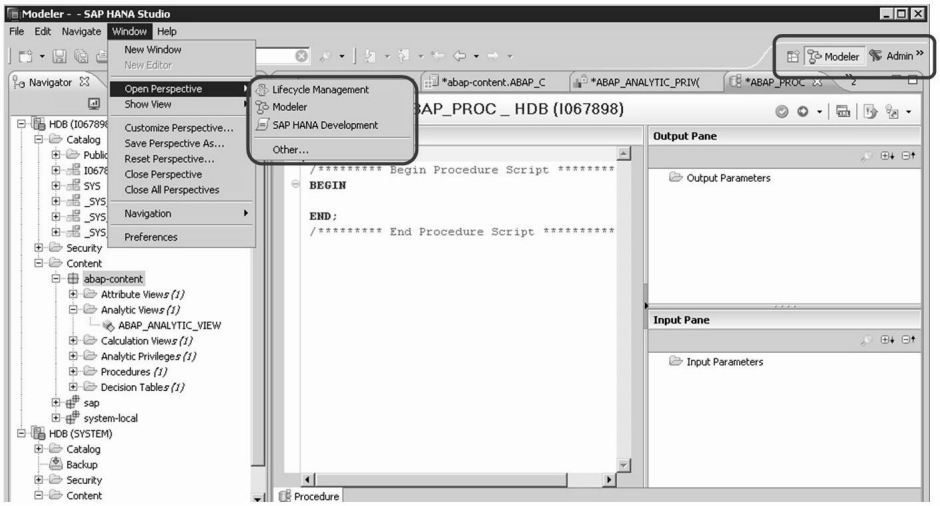


Figure 14: Perspectives

- When creating views you want to use the Modeler perspective.

Folder

A structural device

Contains connections to one or more systems

System

A connection to a given system under a particular user

Each system contains the following:

Catalog: A list of database artefacts, each belonging to a particular schema

Security: Information regarding users and roles in the database

Content: Views and procedures at HANA database level. Content is arranged in packages.

Figure 15: Getting Started – The Modeler Perspective

© 2013 SAP AG. All rights reserved.

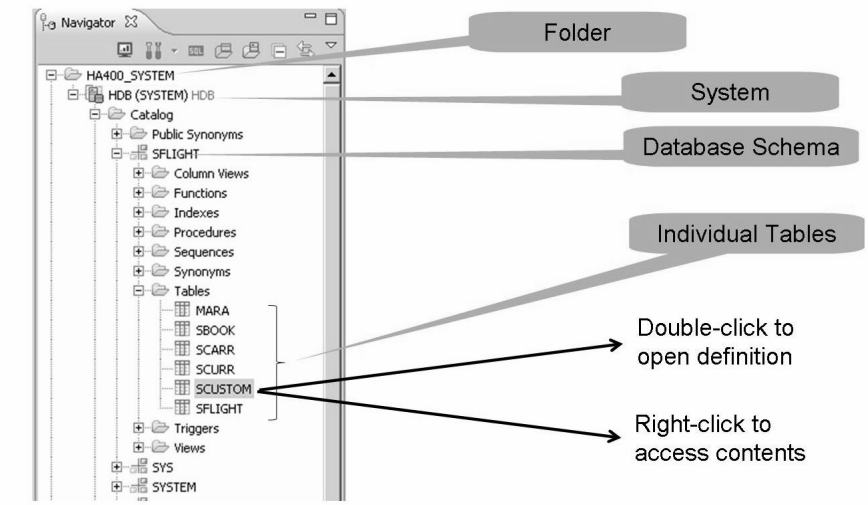


Figure 16: The Modeler Perspective

To connect to a system you need

Hostname
Instance Number
Description
User
Password
(Port is automatically generated)

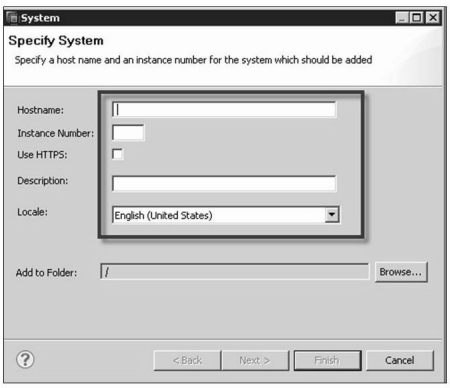


Figure 17: System

© 2013 SAP AG. All rights reserved.

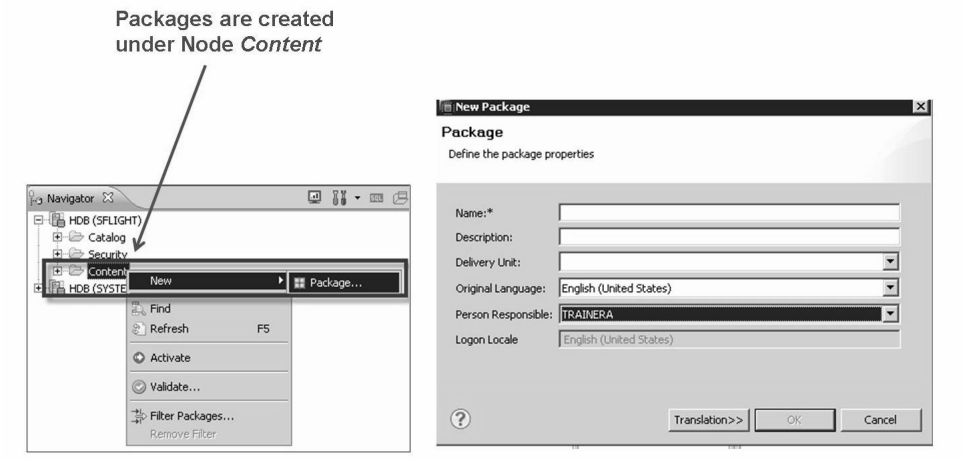


Figure 18: Creating Content Package

© 2013 SAP AG. All rights reserved.

© 2013 SAP AG. All rights reserved.

Exercise 1: Log on to Systems and Create Packages

Exercise Objectives

After completing this exercise, you will be able to:

- log on to SAP Net Weaver Application Server ABAP
- open SAP HANA Studio and connect to the SAP HANA Server
- create a development package in SAP NetWeaver AS ABAP
- create a content package in SAP HANA

Business Example

Task 1: Log on to Application Server ABAP

Log on to the SAP NetWeaver Application Server ABAP with the user and password provided by your instructor. Open the ABAP Workbench (SE80) and create development package ZHA400_##, where ## stands for your group ID.

1. Start SAPlogon and log on to the system.
2. Start transaction SE80 and create the package. Assign the package to application component “CA” and software component “HOME”. Use the workbench request already created by your instructor (press button *Own Requests* to find and use this request).

Task 2: Log on to SAP HANA

Open SAP HANA Studio and establish a connection to the SAP HANA Server told to you by your instructor. Use the user ID and password provided by your instructor. Create content package zha400_##, where ## stands for your group ID.

1. Open SAP HANA Studio.
2. Add a new system. Use the server ID, user ID and Password provided by your instructor.
3. Create the content package.

© 2013 SAP AG. All rights reserved.

Solution 1: Log on to Systems and Create Packages

Task 1: Log on to Application Server ABAP

Log on to the SAP NetWeaver Application Server ABAP with the user and password provided by your instructor. Open the ABAP Workbench (SE80) and create development package ZHA400_##, where ## stands for your group ID.

1. Start SAPlogon and log on to the system.
   a) Perform this step as you have (hopefully) done before.
2. Start transaction SE80 and create the package. Assign the package to application component “CA” and software component “HOME”. Use the workbench request already created by your instructor (press button *Own Requests* to find and use this request).
   a) Perform this step as you have (hopefully) done before.

Task 2: Log on to SAP HANA

Open SAP HANA Studio and establish a connection to the SAP HANA Server told to you by your instructor. Use the user ID and password provided by your instructor. Create content package zha400_##, where ## stands for your group ID.

1. Open SAP HANA Studio.
   a) Choose *hdbstudio* from the Windows Start Menu
2. Add a new system. Use the server ID, user ID and Password provided by your instructor.
   a) Make sure you are in the *Modeler* perspective. If not, change perspective using the pushbuttons in the upper right-hand corner.
   b) Right-click anywhere In the *Navigator* window on the left, and choose *Add System...*.
   c) Enter host name and instance number provided by your instructor and a description, then press button *Next >*.
   d) Enter user ID and password provided by your instructor, then choose *Finish*.
3. Create the content package.
   a) In the *Navigator* window, open node *Content*. Right click on this node and choose *New -> Package...*.
   b) Enter the package name and a description, then press button *OK*.

© 2013 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Explain the central functions of HANA Studio
- Choose different perspectives
- Set up a connection to a SAP HANA Database

© 2013 SAP AG. All rights reserved.

Unit Summary

You should now be able to:

- Describe the HANA evolution, architecture, and direction
- Understand the fundamental technical concepts of SAP HANA
- Explain the central functions of HANA Studio
- Choose different perspectives
- Set up a connection to a SAP HANA Database

© 2013 SAP AG. All rights reserved.

Unit 2

Taking ABAP to HANA

Unit Overview

This Unit is designed to teach the following topics:

- Optimize conventional code
- Access the SAP HANA Database using Open SQL and secondary connection
- Tables in SAP HANA Studio
- Performance rules and guidelines for SAP HANA

Unit Objectives

After completing this unit, you will be able to:

- Understand the Runtime Analysis (SE30)
- Understand the ABAP Trace (SAT)
- Understand the Code Inspector (SCI)
- Understand the SQL Trace (ST05)
- Describe how to access the SAP HANA Database by using Open SQL and a secondary database connection
- Understand Schemata and Tables in SAP HANA Studio
- Analyze the Definition of Tables in SAP HANA Studio
- Understand the performance rules and guidelines for HANA

Unit Contents

© 2013 SAP AG. All rights reserved.

© 2013 SAP AG. All rights reserved.

Lesson: Optimize Conventional Code

Lesson Overview

Lesson Objectives

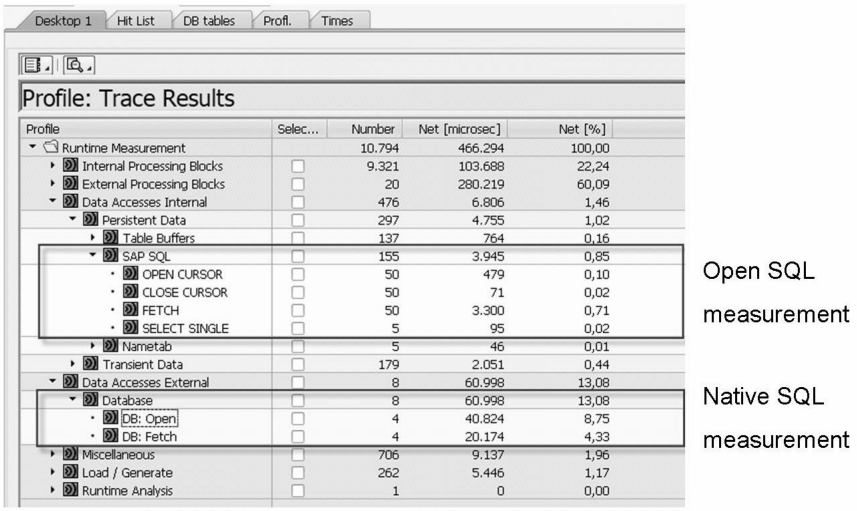
After completing this lesson, you will be able to:

- Understand the Runtime Analysis (SE30)
- Understand the ABAP Trace (SAT)
- Understand the Code Inspector (SCI)
- Understand the SQL Trace (ST05)

Business Example

- The functions of the classic Runtime Analysis (SE30) and the new ABAP Trace (SAT) can be completely used in the HANA Context
- The new ABAP Trace replaces the classical Runtime Analysis
- Both tools measure runtime of processing blocks (methods, functions, subroutines) or single statements
- No database-specific functions are available

Figure 19: Classic Runtime Analysis (SE30) & ABAP Trace (SAT)



Profile	Selec...	Number	Net [microsec]	Net [%]
Runtime Measurement		10.794	466.294	100,00
Internal Processing Blocks		9.321	103.688	22,24
External Processing Blocks		20	280.219	60,09
Data Accesses Internal		476	6.806	1,46
Persistent Data		297	4.755	1,02
Table Buffers		137	764	0,16
SAP SQL		155	3.945	0,85
OPEN CURSOR		50	479	0,10
CLOSE CURSOR		50	71	0,02
FETCH		50	3.300	0,71
SELECT SINGLE		5	95	0,02
Nametab		5	46	0,01
Transient Data		179	2.051	0,44
Data Accesses External		8	60.998	13,08
Database		8	60.998	13,08
DB: Open		4	40.824	8,75
DB: Fetch		4	20.174	4,33
Miscellaneous		706	9.137	1,96
Load / Generate		262	5.446	1,17
Runtime Analysis		1	0	0,00

Figure 20: ABAP Trace (SAT)

© 2013 SAP AG. All rights reserved.

The existing rules and guidelines checked by Code Inspector are still valid in the context of HANA

For NW 7.40 the following additional HANA checks are planned:

SELECTS in LOOPs across modularization units

Problematic SELECT * statements (where less than a certain percentage of the fields are subsequently used)

Find SELECT…FOR ALL ENTRIES statements that can be replaced by a join

Figure 21: Code Inspector

Use SQL Trace (ST05) to

- Display record of all database access (including SQL calls over a secondary database connection)
- Detect redundant / identical SELECT statements
- Locate database-related performance issues
- Display database execution plan for statements (SAP HANA execution plans planned for NW 7.40)

Figure 22: SQL Trace (ST05)

© 2013 SAP AG. All rights reserved.

Exercise 2: Improve Open SQL Statements by Using Field Lists instead of SELECT *

Exercise Objectives

After completing this exercise, you will be able to:

- analyze a program with ABAP Trace (SAT) to locate performance leaks
- decrease the runtime for database selects by using field lists instead of SELECT *
- quantify the improvement with ABAP Trace (SAT)

Business Example

Template:

Report YHA400_OPTIMIZE_OSQL_**T1**

Solution:

Report YHA400_OPTIMIZE_OSQL_**S1**

Task 1: Copy and understand template

Create a copy of report YHA400_OPTIMIZE_OSQL_T1 in your package ZHA400_## (suggested name: ZHA400_##_OSQL_1, where ## is your group number). Analyze the program code to get an idea its functionality. Activate and execute the program.

1. Create a copy of the report. Place it in your package ZHA400_00 and assign it to your workbench task.

Continued on next page

 © 2013 SAP AG. All rights reserved.

2. Analyze the source code of the program. Which are the two main parts of the data processing. In which processing blocks are they encapsulated?

_____
_____
_____
_____

3. The data come from database tables _____ and _____. All columns but one come from database table _____. The remaining column is the result of a calculation. As input, this calculation uses two fields of table _____. These fields are _____ and _____. The calculated column returns the _____

Fill in the blanks to complete the sentence.

4. Activate and execute the program.

Task 2: Analyze

Analyze the program in ABAP Trace (SAT) and identify the processing block and the individual statement with the largest improvement potential.

1. Analyze the program in ABAP Trace (SAT). Use the default variant in both cases.
2. Analyze the trace result. Look for the most time consuming processing block (internal or external).
3. The most expensive processing block is _____.

Fill in the blanks to complete the sentence.

4. Look for the most expensive data access (internal or external).
5. The most expensive data access is _____.

Fill in the blanks to complete the sentence.

Continued on next page

 © 2013 SAP AG. All rights reserved.

Task 3: Improve

Improve the program. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*. In subroutine *get_data_solution*, make sure only those columns are read from the database which are actually needed.

1. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*

2. Which fields of table SCUSTOM are actually needed?

3. Which fields of table SBOOK are actually needed?

4. Define two local structure types with only the required fields (suggested names: *lty_s_book* and *lty_s_cust*. Use these types for the data objects *ls_scustom* and *ls_sbook*.
5. In the two SELECT--Statements, replace "*" with a list of the required fields.
6. Activate and test your program. Make sure the two subroutines deliver exactly the same data.

Task 4: Quantify improvement

Repeat your runtime measurement with ABAP Trace (SAT). Compare the runtime consumption of subroutine *get_data_template* and subroutine *get_data_solution*

1. Perform a runtime measurement with ABAP Trace (SAT) and compare the gross runtime of the two subroutines.

2. The gross runtime of subroutine *get_data_template* is: _____ micro seconds. The gross runtime of subroutine *get_data_solution* is: _____ micro seconds. This is a reduction of the runtime by _____ %.
Fill in the blanks to complete the sentence.

© 2013 SAP AG. All rights reserved.

Solution 2: Improve Open SQL Statements by Using Field Lists instead of SELECT *

Task 1: Copy and understand template

Create a copy of report YHA400_OPTIMIZE_OSQL_T1 in your package ZHA400_## (suggested name: ZHA400_##_OSQL_1, where ## is your group number). Analyze the program code to get an idea its functionality. Activate and execute the program.

1. Create a copy of the report. Place it in your package ZHA400_00 and assign it to your workbench task.

   a) Complete this step as you learned to do in previous classes.

2. Analyze the source code of the program. Which are the two main parts of the data processing. In which processing blocks are they encapsulated?

   Answer:

   1. Reading from the data base and computing a list of customers, subroutine *get_data_template*.
   2. displaying the list of customers, subroutine *display*.

3. The data come from database tables SCUSTOM and SBOOK. All columns but one come from database table SCUSTOM. The remaining column is the result of a calculation. As input, this calculation uses two fields of table SBOOK. These fields are ORDER_DATE and FLDATE. The calculated column returns the average number of days between booking date and flight date.

   Answer: SCUSTOM, SBOOK, SCUSTOM, SBOOK, ORDER_DATE, FLDATE, average number of days between booking date and flight date

4. Activate and execute the program.

   a) Complete this step as you learned to do in previous classes.

Continued on next page

 © 2013 SAP AG. All rights reserved.

Task 2: Analyze

Analyze the program in ABAP Trace (SAT) and identify the processing block and the individual statement with the largest improvement potential.

1. Analyze the program in ABAP Trace (SAT). Use the default variant in both cases.
   a) Start transaction SAT .
   b) Enter the name of your program and, if necessary, the name of the variant (DEFAULT).
   c) Press *Execute* and wait.
2. Analyze the trace result. Look for the most time consuming processing block (internal or external).
   a) On Tab *Desktop 1*, double click on *Internal Processing Blocks*.
   b) Sort the *Hitlist* on the right by column *Net [microsec]* und look for the topmost entry.
   c) Repeat with entry *External Processing Blocks*.
   d) Alternative: Select the checkbox for both entries. From the tool bar above *Profile: trace Result* choose: *Display Subarea in Hitlist tool*.
3. The most expensive processing block is subroutine *GET_DATA_TEMPLATE*.

   Answer: subroutine *GET_DATA_TEMPLATE*
4. Look for the most expensive data access (internal or external).
   a) Repeat the previous step with *Data Acess Internal* and *Data Access External*.
5. The most expensive data access is FETCH from DB table SBOOK.

   Answer: FETCH from DB table SBOOK

Task 3: Improve

Improve the program. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*. In subroutine *get_data_solution*, make sure only those columns are read from the database which are actually needed.

1. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*
   a) Use *Copy & Paste*-function of the ABAP Editor.
2. Which fields of table SCUSTOM are actually needed?

   Answer: ID, NAME, POSTCODE, CITY, COUNTRY

Continued on next page

 © 2013 SAP AG. All rights reserved.

3. Which fields of table SBOOK are actually needed?

   Answer: FLDATE, ORDER_DATE

4. Define two local structure types with only the required fields (suggested names: *lty_s_book* and *lty_s_cust*. Use these types for the data objects *ls_scustom* and *ls_sbook*.

   a) See source code extract from model solution.

5. In the two SELECT--Statements, replace "*" with a list of the required fields.

   a) See source code extract from model solution.

6. Activate and test your program. Make sure the two subroutines deliver exactly the same data.

   a) Complete this step as you learned to do in previous classes.

Task 4: Quantify improvement

Repeat your runtime measurement with ABAP Trace (SAT). Compare the runtime consumption of subroutine *get_data_template* and subroutine *get_data_solution*

1. Perform a runtime measurement with ABAP Trace (SAT) and compare the gross runtime of the two subroutines.

   a) Perform this step as before.

2. The gross runtime of subroutine *get_data_template* is: _____ micro seconds. The gross runtime of subroutine *get_data_solution* is: _____ micro seconds. This is a reduction of the runtime by _____ %.

   Answer:

Result

Source code extract from model solution (Program YHA400_OPTIMIZE_OSQL_S1)

```
*&---------------------------------------------------------------------*
*&      Form  get_data_template
*&---------------------------------------------------------------------*

FORM get_data_template CHANGING ct_customers TYPE ty_t_customers.

* Declarations
****************

* Work Area for Result
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
  DATA ls_customer LIKE LINE OF ct_customers.

* Targets for Select
  DATA: ls_scustom TYPE scustom,
        ls_sbook   TYPE sbook.

* help variables
  DATA lv_count TYPE i.

* Processing
*****************

  CLEAR ct_customers.

  SELECT * FROM scustom
     INTO ls_scustom.

    ls_customer-id       = ls_scustom-id.
    ls_customer-name     = ls_scustom-name.
    ls_customer-postcode = ls_scustom-postcode.
    ls_customer-city     = ls_scustom-city.
    ls_customer-country  = ls_scustom-country.

    CLEAR ls_customer-days_ahead.
    CLEAR lv_count.

    SELECT * FROM sbook
       INTO ls_sbook
            WHERE customid = ls_scustom-id
              AND cancelled = space.

      ls_customer-days_ahead = ls_customer-days_ahead + ( ls_sbook-fldate - ls_sbook-order_
      lv_count = lv_count + 1.

    ENDSELECT.
    IF lv_count <> 0.
      ls_customer-days_ahead = ls_customer-days_ahead / lv_count.
      INSERT ls_customer INTO TABLE ct_customers.
    ENDIF.

  ENDSELECT.

  SORT ct_customers BY id.

ENDFORM.                    "
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
*&---------------------------------------------------------------------*
*&      Form  get_data_solution
*&---------------------------------------------------------------------*
FORM get_data_solution CHANGING ct_customers TYPE ty_t_customers.

* Declarations
****************

* Types for target fields

  TYPES: BEGIN OF lty_s_cust,
           id          TYPE scustom-id,
           name        TYPE scustom-name,
           postcode    TYPE scustom-postcode,
           city        TYPE scustom-city,
           country     TYPE scustom-country,
         END OF lty_s_cust.

  TYPES: BEGIN OF lty_s_book,
             fldate      TYPE sbook-fldate,
            order_date   TYPE sbook-order_date,
        END OF lty_s_book.



* Work Area for Result
  DATA ls_customer LIKE LINE OF ct_customers.

* Targets for Select
  DATA: ls_scustom TYPE lty_s_cust,
        ls_sbook   TYPE lty_s_book.

* help variables
  DATA lv_count TYPE i.

* Processing
*****************

  CLEAR ct_customers.

  SELECT id name postcode city country
    FROM scustom
    INTO ls_scustom.
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
    ls_customer-id       = ls_scustom-id.
    ls_customer-name     = ls_scustom-name.
    ls_customer-postcode = ls_scustom-postcode.
    ls_customer-city     = ls_scustom-city.
    ls_customer-country  = ls_scustom-country.

    CLEAR ls_customer-days_ahead.
    CLEAR lv_count.

    SELECT fldate order_date
      FROM sbook
      INTO ls_sbook
     WHERE customid = ls_scustom-id
       AND cancelled = space.

      ls_customer-days_ahead = ls_customer-days_ahead + ( ls_sbook-fldate - ls_sbook-order_
      lv_count = lv_count + 1.

    ENDSELECT.

    IF lv_count <> 0.
      ls_customer-days_ahead = ls_customer-days_ahead / lv_count.
      INSERT ls_customer INTO TABLE ct_customers.

    ENDIF.

  ENDSELECT.

  SORT ct_customers BY id.

ENDFORM.                    "
```


© 2013 SAP AG. All rights reserved.

© 2013 SAP AG. All rights reserved.

Exercise 3: Improve Open SQL Statements by Using a Join instead of nested SELECT-statements

Exercise Objectives

After completing this exercise, you will be able to:

- analyze a program with Code Inspector (SCI) to detect potential performance problems
- decrease the runtime for data base selects by using a join instead of nested SELECT-Statements
- quantify the improvement with ABAP Trace (SAT)

Business Example

Template:

Report YHA400_OPTIMIZE_OSQL_**T2**

Solution:

Report YHA400_OPTIMIZE_OSQL_**S2**

Task 1: Copy and understand template

Create a copy of report YHA400_OPTIMIZE_OSQL_T2 in your package ZHA400_## (suggested name: ZHA400_##_OSQL_2, where ## is your group number). Activate and execute the program.

1. Create a copy of the report. Place it in your package ZHA400_00 and assign it to your workbench task.
2. Activate and execute the program.

Task 2: Analyze

Analyze the program with Code Inspector (SCI) to identify potential performance problems.

1. Create a local check variant for Code Inspector (suggested name: *HA400_PERF##*, where ## is your group number). In the check variant activate all performance checks.
2. Perform an Inspection of your program based on your new check variant (suggested name for the inspection: *HA400_OSQL##*)..
3. Analyze the inspection result. Navigate to the ABAP Source Code. What can be done about the reported issue?

Continued on next page

 © 2013 SAP AG. All rights reserved.

Task 3: Improve

Improve the program. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*. In subroutine *get_data_solution*, replace the two nested SELECT-loops by one SELECT on both database tables (Join).

1. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*
2. Define one local structure type with all required fields from both tables (suggested name: *lty_s_cust_book*. Declare a structure and an internal table based on this type (suggested name: *ls_cust_book*).
3. Remove the two SELECT-statements, by just one SELECT-Statement. Combine the data from SBOOK and SCUSTOM through an inner join with a suitable join condition. Adjust the field list and use internal table *lt_cust_book* as target for the select statement.
4. Optional: Use aliases for the two tables to improve readability and add the table names in teh field list and WHERE-clause.
5. In a loop over *lt_cust_book*, implement the calculation of the average days between order date and flight date and fill table *ct_customers* with the result.

Hint: First make sure the data is sorted by column *ID*. Whenever you reach a new customer, finish summing up the day differences, divide by the number of bookings and insert a new line in *ct_customers*. Then start with the new customer.

Caution: Don’t forget to insert the last customer into *ct_customers*!

6. Activate and test your program. Make sure the two subroutines deliver exactly the same data.

Task 4: Quantify improvement

Perform a runtime measurement with ABAP Trace (SAT). Compare the runtime consumption of subroutine *get_data_template* and subroutine *get_data_solution*

1. Perform a runtime measurement with ABAP Trace (SAT) and compare the gross runtime of the two subroutines.

2. The gross runtime of subroutine *get_data_template* is: _____ micro seconds. The gross runtime of subroutine *get_data_solution* is: _____ micro seconds. This is a reduction of the runtime by _____ %.
Fill in the blanks to complete the sentence.

© 2013 SAP AG. All rights reserved.

Solution 3: Improve Open SQL Statements by Using a Join instead of nested SELECT-statements

Task 1: Copy and understand template

Create a copy of report YHA400_OPTIMIZE_OSQL_T2 in your package ZHA400_## (suggested name: ZHA400_##_OSQL_2, where ## is your group number). Activate and execute the program.

1. Create a copy of the report. Place it in your package ZHA400_00 and assign it to your workbench task.
   a) Complete this step as you learned to do in previous classes.
2. Activate and execute the program.
   a) Complete this step as you learned to do in previous classes.

Task 2: Analyze

Analyze the program with Code Inspector (SCI) to identify potential performance problems.

1. Create a local check variant for Code Inspector (suggested name: *HA400_PERF##*, where ## is your group number). In the check variant activate all performance checks.
   a) Start transaction SCI .
   b) In Frame *Check Variant*, enter the check variant name and click on button *Create*.
   c) In the check variant, make sure all *Performance Checks* are selected. Save the check variant and go back to the start screen of transaction SCI.
2. Perform an Inspection of your program based on your new check variant (suggested name for the inspection: *HA400_OSQL##*)..
   a) Start transaction SCI.
   b) In Frame *Inspection*, enter the name for the inspection and click on button *Create*.
   c) In frame *Object Selection*, enter *Single*, *Program* and type in the name of your program.
   d) In frame *Check Variant*, enter the name of your check variant.
   e) On the toolbar, click on the button labelled *Execute (F8)*.

Continued on next page

 © 2013 SAP AG. All rights reserved.

3. Analyze the inspection result. Navigate to the ABAP Source Code. What can be done about the reported issue?

   a) On the toolbar, click on the button labelled *Results (Shift F6)* to see the inspection result

   b) Check the reported problems. Double click a message to navigate to the ABAP source code.

   c) Read the check documentation to get hints on how to improve your program.

Task 3: Improve

Improve the program. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*. In subroutine *get_data_solution*, replace the two nested SELECT-loops by one SELECT on both database tables (Join).

1. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*

   a) Use *Copy & Paste*-function of the ABAP Editor.

2. Define one local structure type with all required fields from both tables (suggested name: *lty_s_cust_book*. Declare a structure and an internal table based on this type (suggested name: *ls_cust_book*).

   a) See source code extract from model solution.

3. Remove the two SELECT-statements, by just one SELECT-Statement. Combine the data from SBOOK and SCUSTOM through an inner join with a suitable join condition. Adjust the field list and use internal table *lt_cust_book* as target for the select statement.

   a) See source code extract from model solution.

4. Optional: Use aliases for the two tables to improve readability and add the table names in teh field list and WHERE-clause.

   a) See source code extract from model solution.

5. In a loop over *lt_cust_book*, implement the calculation of the average days between order date and flight date and fill table *ct_customers* with the result.

Hint: First make sure the data is sorted by column *ID*. Whenever you reach a new customer, finish summing up the day differences, divide by the number of bookings and insert a new line in *ct_customers*. Then start with the new customer.

Caution: Don’t forget to insert the last customer into *ct_customers*!

Continued on next page

© 2013 SAP AG. All rights reserved.

6. Activate and test your program. Make sure the two subroutines deliver exactly the same data.

   a) Complete this step as you learned to do in previous classes.

Task 4: Quantify improvement

Perform a runtime measurement with ABAP Trace (SAT). Compare the runtime consumption of subroutine *get_data_template* and subroutine *get_data_solution*

1. Perform a runtime measurement with ABAP Trace (SAT) and compare the gross runtime of the two subroutines.

   a) Perform this step as before.

2. The gross runtime of subroutine *get_data_template* is: _____ micro seconds. The gross runtime of subroutine *get_data_solution* is: _____ micro seconds. This is a reduction of the runtime by _____ %.

   Answer:

Result

Source code extract from model solution (Program YHA400_OPTIMIZE_OSQL_S2)

```
*&---------------------------------------------------------------------*
*&      Form  get_data_template
*&---------------------------------------------------------------------*

FORM get_data_template CHANGING ct_customers TYPE ty_t_customers.


* Declarations
****************

* Types for target fields

  TYPES: BEGIN OF lty_s_cust,
           id          TYPE scustom-id,
           name        TYPE scustom-name,
           postcode    TYPE scustom-postcode,
           city        TYPE scustom-city,
           country     TYPE scustom-country,
         END OF lty_s_cust.

  TYPES: BEGIN OF lty_s_book,
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
             fldate      TYPE sbook-fldate,
            order_date   TYPE sbook-order_date,
        END OF lty_s_book.



* Work Area for Result
  DATA ls_customer LIKE LINE OF ct_customers.

* Targets for Select
  DATA: ls_scustom TYPE lty_s_cust,
        ls_sbook   TYPE lty_s_book.

* help variables
  DATA lv_count TYPE i.

* Processing
*****************

  CLEAR ct_customers.

  SELECT id name postcode city country
    FROM scustom
    INTO ls_scustom.

    ls_customer-id       = ls_scustom-id.
    ls_customer-name     = ls_scustom-name.
    ls_customer-postcode = ls_scustom-postcode.
    ls_customer-city     = ls_scustom-city.
    ls_customer-country  = ls_scustom-country.

    CLEAR ls_customer-days_ahead.
    CLEAR lv_count.

    SELECT fldate order_date
      FROM sbook
      INTO ls_sbook
     WHERE customid = ls_scustom-id
       AND cancelled = space.

      ls_customer-days_ahead = ls_customer-days_ahead + ( ls_sbook-fldate - ls_sbook-order_date
      lv_count = lv_count + 1.

    ENDSELECT.
```

Continued on next page

© 2013 SAP AG. All rights reserved.

```
    IF lv_count <> 0.
      ls_customer-days_ahead = ls_customer-days_ahead / lv_count.
      INSERT ls_customer INTO TABLE ct_customers.

    ENDIF.

  ENDSELECT.

  SORT ct_customers BY id.

ENDFORM.                    "

*&---------------------------------------------------------------------*
*&      Form  get_data_solution
*&---------------------------------------------------------------------*
FORM get_data_solution CHANGING ct_customers TYPE ty_t_customers.

* Declarations
****************

* Types for target fields

  TYPES: BEGIN OF lty_s_cust_book,
           id          TYPE scustom-id,
           name        TYPE scustom-name,
           postcode    TYPE scustom-postcode,
           city        TYPE scustom-city,
           country     TYPE scustom-country,
           fldate      TYPE sbook-fldate,
           order_date  TYPE sbook-order_date,
         END OF lty_s_cust_book.

* Work Area for Result
  DATA ls_customer LIKE LINE OF ct_customers.

* Targets for Select
  DATA: lt_cust_book TYPE SORTED TABLE OF lty_s_cust_book WITH NON-UNIQUE KEY id,
        ls_cust_book TYPE lty_s_cust_book.

* help variables
  DATA lv_count TYPE i.

* Processing
*****************
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
  CLEAR ct_customers.

  SELECT c~id c~name c~postcode c~city c~country b~fldate b~order_date
    FROM scustom AS c INNER JOIN sbook AS b
      ON c~id = b~customid
    INTO TABLE lt_cust_book
         WHERE b~cancelled = space.

  LOOP AT lt_cust_book INTO ls_cust_book.

    IF sy-tabix = 1.                                    "first booking of first customer

      ls_customer-id       = ls_cust_book-id.
      ls_customer-name     = ls_cust_book-name.
      ls_customer-postcode = ls_cust_book-postcode.
      ls_customer-city     = ls_cust_book-city.
      ls_customer-country  = ls_cust_book-country.

      CLEAR ls_customer-days_ahead.

    ELSEIF ls_cust_book-id <> ls_customer-id.          "first booking of new customer

      ls_customer-days_ahead = ls_customer-days_ahead / lv_count.
      INSERT ls_customer INTO TABLE ct_customers.

      ls_customer-id       = ls_cust_book-id.
      ls_customer-name     = ls_cust_book-name.
      ls_customer-postcode = ls_cust_book-postcode.
      ls_customer-city     = ls_cust_book-city.
      ls_customer-country  = ls_cust_book-country.

      CLEAR ls_customer-days_ahead.
      CLEAR lv_count.
    ENDIF.

    ls_customer-days_ahead = ls_customer-days_ahead + ( ls_cust_book-fldate - ls_cust_book-orde
    lv_count = lv_count + 1.

  ENDLOOP.

* Store last entry in result table
  ls_customer-days_ahead = ls_customer-days_ahead / lv_count.
  INSERT ls_customer INTO TABLE ct_customers.

*  SORT ct_customers BY id.   " not needed, already sorted
```

Continued on next page

© 2013 SAP AG. All rights reserved.

```
ENDFORM.                    "
```


© 2013 SAP AG. All rights reserved.

© 2013 SAP AG. All rights reserved.

Exercise 4: Improve Open SQL Statements by buffering all data you need

Exercise Objectives

After completing this exercise, you will be able to:

- analyze a program with SQL Trace (ST05) to detect repeated selects
- decrease the runtime for data base selects by buffering data in internal tables
- quantify the improvement with ABAP Trace (SAT)

Business Example

Template:

Report YHA400_OPTIMIZE_OSQL_**T3**

Solution:

Report YHA400_OPTIMIZE_OSQL_**S3**

Task 1: Copy and understand template

Create a copy of report YHA400_OPTIMIZE_OSQL_T3 in your package ZHA400_## (suggested name: ZHA400_##_OSQL_3, where ## is your group number). Activate and execute the program.

1. Create a copy of the report. Place it in your package ZHA400_00 and assign it to your workbench task.
2. Activate and execute the program.

Task 2: Analyze

Analyze the program with SQL Trace (ST05) to detect repeated selects

1. In a separate session (window), switch on SQL Trace with a filter for your user name and your program.
2. Execute your program and switch off SQL Trace as soon as possible.
3. Analyze the trace result and identify the most expensive statement and repeated SELECTs.

Continued on next page

 © 2013 SAP AG. All rights reserved.

Task 3: Improve

Improve the program. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*. In subroutine *get_data_solution*, replace the two nested SELECT-loops by two array fetches on SCUSTOM and SBOOK and two nested loops over the internal tables.

1. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*
2. Define two internal tables (suggested names: *lt_scustom* and *lt_sbook*, with line types *lty_s_book* and *lty_s_cust*. Make sure you define the tables as sorted tables with table key *ID* or *CUSTOMID*
3. Why is it important to define the two tables as sorted tables?

   _____
   _____
   _____
   _____
4. Before the two SELECT-loops, implement two SELECT-statements to read all customers and all non-cancelled bookings into the internal tables. Replace the SELECT-loops with LOOPs over the internal tables.
5. Activate and test your program. Make sure the two subroutines deliver exactly the same data.

Task 4: Quantify Improvement

Perform a runtime measurement with ABAP Trace (SAT). Compare the runtime consumption of subroutine *get_data_template* and subroutine *get_data_solution*

1. Perform a runtime measurement with ABAP Trace (SAT) and compare the gross runtime of the two subroutines.
2. The gross runtime of subroutine *get_data_template* is: _____ micro seconds. The gross runtime of subroutine *get_data_solution* is: _____ micro seconds. This is a reduction of the runtime by _____ %.
   Fill in the blanks to complete the sentence.
3. Optional: Change the declaration of *lt_sbook* from sorted table to standard table and repeat the measurement.

© 2013 SAP AG. All rights reserved.

Solution 4: Improve Open SQL Statements by buffering all data you need

Task 1: Copy and understand template

Create a copy of report YHA400_OPTIMIZE_OSQL_T3 in your package ZHA400_## (suggested name: ZHA400_##_OSQL_3, where ## is your group number). Activate and execute the program.

1. Create a copy of the report. Place it in your package ZHA400_00 and assign it to your workbench task.
   a) Complete this step as you learned to do in previous classes.
2. Activate and execute the program.
   a) Complete this step as you learned to do in previous classes.

Task 2: Analyze

Analyze the program with SQL Trace (ST05) to detect repeated selects

1. In a separate session (window), switch on SQL Trace with a filter for your user name and your program.
   a) Start transaction ST05 in a new session (enter /oST05 in the command field).
   b) Choose *Activate Trace with Filter*.
   c) Enter your user and program name and choose *Enter*.
2. Execute your program and switch off SQL Trace as soon as possible.
   a) Return to the first session (window) and execute your program.
   b) As soon as possible, return to the SQL Trace and choose *Deactivate*.

Continued on next page

 © 2013 SAP AG. All rights reserved.

3. Analyze the trace result and identify the most expensive statement and repeated SELECTs.

   a) In SQL Trace, choose *Display Trace* and *Execute (F8)* to see the trace result.

   b) From the menu, choose *Trace List* → *Summarize Trace by SQL Statement*.

   c) Sort the list by column *Duration* and identify the most expensive statement.

   d) Sort the list by column *Executions* and identify the statement that has been repeated most often.

   e) Click on the statement and choose *Display call positions in ABAP program* from the tool bar.

Task 3: Improve

Improve the program. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*. In subroutine *get_data_solution*, replace the two nested SELECT-loops by two array fetches on SCUSTOM and SBOOK and two nested loops over the internal tables.

1. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*

   a) Use *Copy & Paste*-function of the ABAP Editor.

2. Define two internal tables (suggested names: *lt_scustom* and *lt_sbook*, with line types *lty_s_book* and *lty_s_cust*. Make sure you define the tables as sorted tables with table key *ID* or *CUSTOMID*

   a) See source code extract from model solution.

3. Why is it important to define the two tables as sorted tables?

   Answer: Otherwise the runtime environment would not be able to optimize the loops over these tables and we would loose a lot of runtime by searching for the necessary entries.

4. Before the two SELECT-loops, implement two SELECT-statements to read all customers and all non-cancelled bookings into the internal tables. Replace the SELECT-loops with LOOPs over the internal tables.

   a) See source code extract from model solution.

5. Activate and test your program. Make sure the two subroutines deliver exactly the same data.

   a) Complete this step as you learned to do in previous classes.

Continued on next page

 © 2013 SAP AG. All rights reserved.

Task 4: Quantify Improvement

Perform a runtime measurement with ABAP Trace (SAT). Compare the runtime consumption of subroutine *get_data_template* and subroutine *get_data_solution*

1. Perform a runtime measurement with ABAP Trace (SAT) and compare the gross runtime of the two subroutines.

   a) Perform this step as before.

2. The gross runtime of subroutine *get_data_template* is: _____ micro seconds. The gross runtime of subroutine *get_data_solution* is: _____ micro seconds. This is a reduction of the runtime by _____ %.

   Answer:

3. Optional: Change the declaration of *lt_sbook* from sorted table to standard table and repeat the measurement.

   a) Perform this step as before.

Result

Source code extract from model solution (Program YHA400_OPTIMIZE_OSQL_S3)

```
*&---------------------------------------------------------------------*
*&      Form  get_data_template
*&---------------------------------------------------------------------*

FORM get_data_template CHANGING ct_customers TYPE ty_t_customers.

* Declarations
****************

* Types for target fields

  TYPES: BEGIN OF lty_s_cust,
           id          TYPE scustom-id,
           name        TYPE scustom-name,
           postcode    TYPE scustom-postcode,
           city        TYPE scustom-city,
           country     TYPE scustom-country,
         END OF lty_s_cust.

  TYPES: BEGIN OF lty_s_book,
             fldate      TYPE sbook-fldate,
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
             order_date   TYPE sbook-order_date,
         END OF lty_s_book.



* Work Area for Result
  DATA ls_customer LIKE LINE OF ct_customers.

* Targets for Select
  DATA: ls_scustom TYPE lty_s_cust,
        ls_sbook   TYPE lty_s_book.

* help variables
  DATA lv_count TYPE i.

* Processing
*****************

  CLEAR ct_customers.

  SELECT id name postcode city country
    FROM scustom
    INTO ls_scustom.

    ls_customer-id       = ls_scustom-id.
    ls_customer-name     = ls_scustom-name.
    ls_customer-postcode = ls_scustom-postcode.
    ls_customer-city     = ls_scustom-city.
    ls_customer-country  = ls_scustom-country.

    CLEAR ls_customer-days_ahead.
    CLEAR lv_count.

    SELECT fldate order_date
      FROM sbook
      INTO ls_sbook
     WHERE customid = ls_scustom-id
       AND cancelled = space.

      ls_customer-days_ahead = ls_customer-days_ahead + ( ls_sbook-fldate - ls_sbook-order_date
      lv_count = lv_count + 1.

    ENDSELECT.

    IF lv_count <> 0.
```

Continued on next page

© 2013 SAP AG. All rights reserved.

```
      ls_customer-days_ahead = ls_customer-days_ahead / lv_count.
      INSERT ls_customer INTO TABLE ct_customers.

    ENDIF.

  ENDSELECT.

  SORT ct_customers BY id.
ENDFORM.                    "

*&---------------------------------------------------------------------*
*&      Form  get_data_solution
*&---------------------------------------------------------------------*
FORM get_data_solution CHANGING ct_customers TYPE ty_t_customers.

* Declarations
****************

* Types for target fields

  TYPES: BEGIN OF lty_s_cust,
           id          TYPE scustom-id,
           name        TYPE scustom-name,
           postcode    TYPE scustom-postcode,
           city        TYPE scustom-city,
           country     TYPE scustom-country,
         END OF lty_s_cust.

  TYPES: BEGIN OF lty_s_book,
             customid    TYPE sbook-customid,
             fldate      TYPE sbook-fldate,
            order_date   TYPE sbook-order_date,
        END OF lty_s_book.



* Work Area for Result
  DATA ls_customer LIKE LINE OF ct_customers.

* Targets for Select
  DATA: lt_scustom TYPE SORTED TABLE OF lty_s_cust WITH NON-UNIQUE KEY id,
        ls_scustom TYPE lty_s_cust,
        lt_sbook   TYPE SORTED TABLE OF lty_s_book WITH NON-UNIQUE KEY customid,
        ls_sbook   TYPE lty_s_book.
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
* help variables
  DATA lv_count TYPE i.

* Processing
*****************

  CLEAR ct_customers.

  SELECT id name postcode city country
    FROM scustom
    INTO TABLE lt_scustom.
*    ORDER BY id            no improvement, sorting on Appl. server more efficient
  SELECT customid fldate order_date
    FROM sbook
    INTO TABLE lt_sbook
   WHERE cancelled = space.
*    ORDER BY customid       no improvement, sorting on Appl. server more efficient

  LOOP AT lt_scustom INTO ls_scustom.

    ls_customer-id       = ls_scustom-id.
    ls_customer-name     = ls_scustom-name.
    ls_customer-postcode = ls_scustom-postcode.
    ls_customer-city     = ls_scustom-city.
    ls_customer-country  = ls_scustom-country.

    CLEAR ls_customer-days_ahead.
    CLEAR lv_count.

    LOOP AT lt_sbook INTO ls_sbook
                    WHERE customid = ls_scustom-id.
      ls_customer-days_ahead = ls_customer-days_ahead + ( ls_sbook-fldate - ls_sbook-order_date
      lv_count = lv_count + 1.
    ENDLOOP.

    IF lv_count > 0.
      ls_customer-days_ahead = ls_customer-days_ahead / lv_count.
      INSERT ls_customer INTO TABLE ct_customers.
    ENDIF.

  ENDLOOP.

*  SORT ct_customers BY id. " already sorted

ENDFORM.                    "
```


© 2013 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Understand the Runtime Analysis (SE30)
- Understand the ABAP Trace (SAT)
- Understand the Code Inspector (SCI)
- Understand the SQL Trace (ST05)

© 2013 SAP AG. All rights reserved.

Lesson: Access the SAP HANA Database Using Open SQL and Secondary Connection

Lesson Overview

Lesson Objectives

After completing this lesson, you will be able to:

- Describe how to access the SAP HANA Database by using Open SQL and a secondary database connection

Business Example

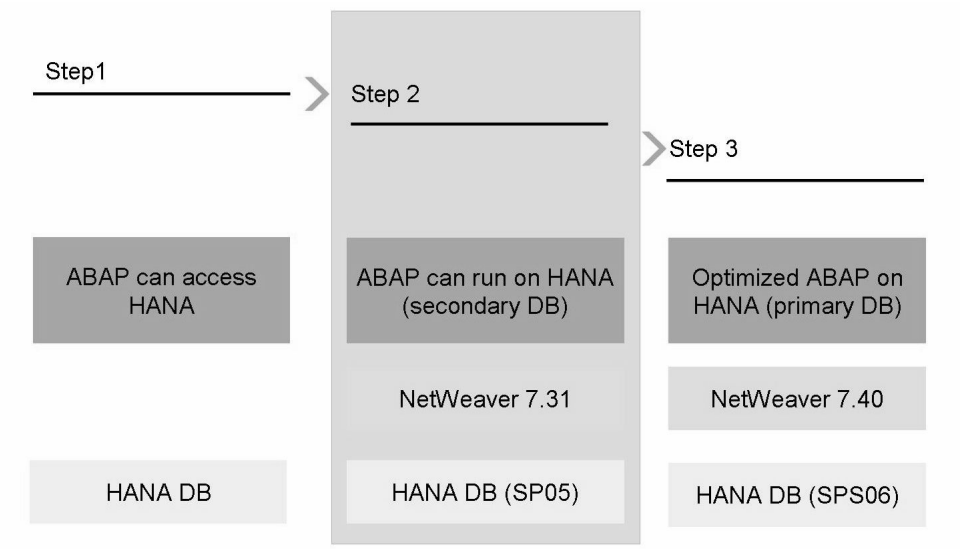


Figure 23: Recap: HANA Evolution

© 2013 SAP AG. All rights reserved.

Can be used to access local or remote database systems

are maintained in DB table DBCON (using Transaction SM30)

Require specification of connection data including DB user (= DB schema) and password

Service note 1597627 describes the prerequisites and procedure for setting up a secondary SB connection to HANA.

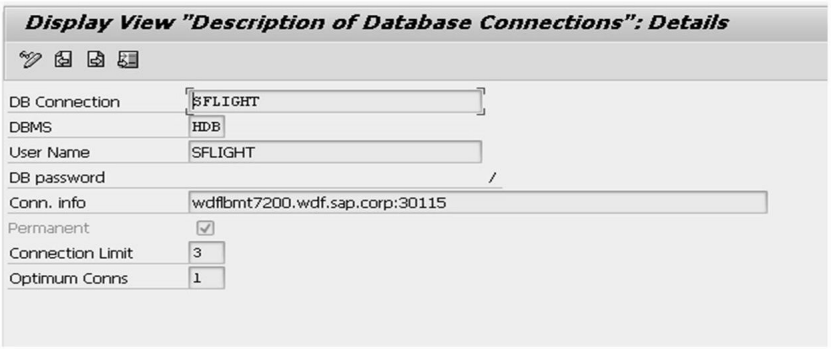


Figure 24: Secondary Database Connections

Maintain, monitor and test secondary database connections

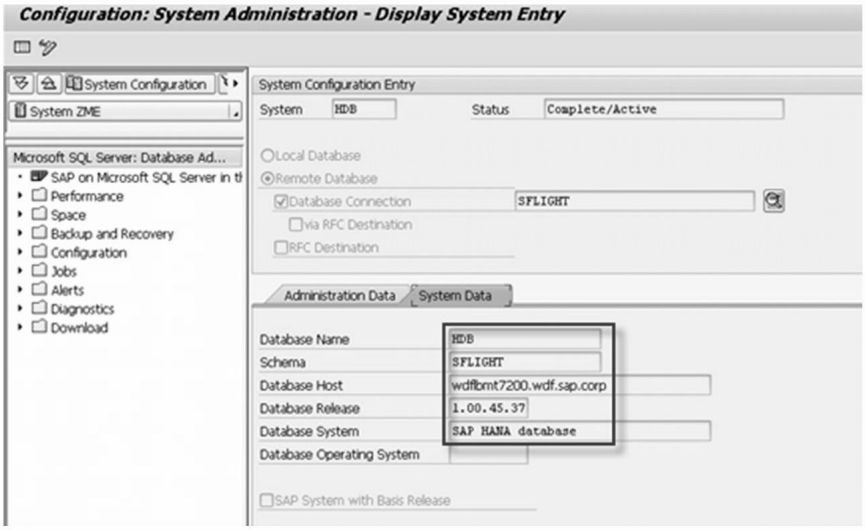


Figure 25: Transaction DBACOCKPIT

© 2013 SAP AG. All rights reserved.

```
SELECT ... FROM <table> CONNECTION <connection> INTO ...
```

Prerequisite:

- HANA DB Connection exists in DB Table DBCON

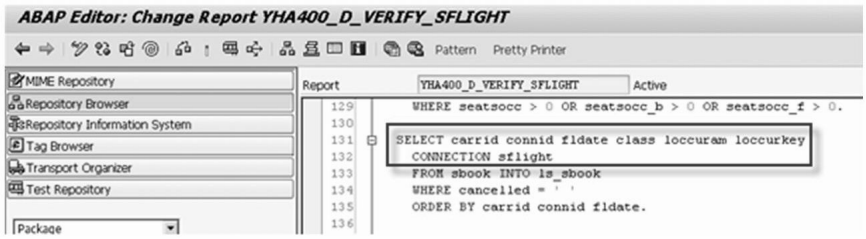


Figure 26: Open SQL Using Secondary Database Connection

Only tables, no HANA specific objects (e.g. views)

Only tables and fields that are also defined in the local ABAP Dictionary

Only tables in the schema of the user specified in the connection

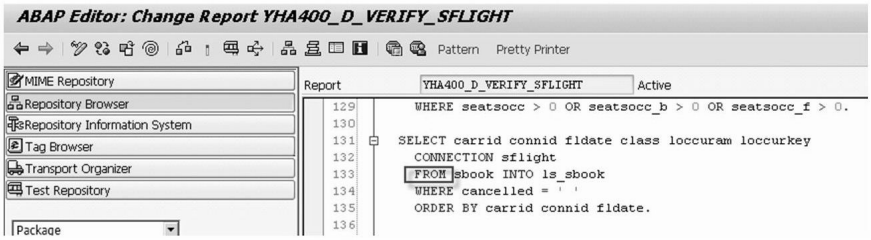


Figure 27: Restriction

© 2013 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Describe how to access the SAP HANA Database by using Open SQL and a secondary database connection

© 2013 SAP AG. All rights reserved.

Lesson: Tables in SAP HANA Studio

Lesson Overview

Lesson Objectives

After completing this lesson, you will be able to:

- Understand Schemata and Tables in SAP HANA Studio
- Analyze the Definition of Tables in SAP HANA Studio

Business Example

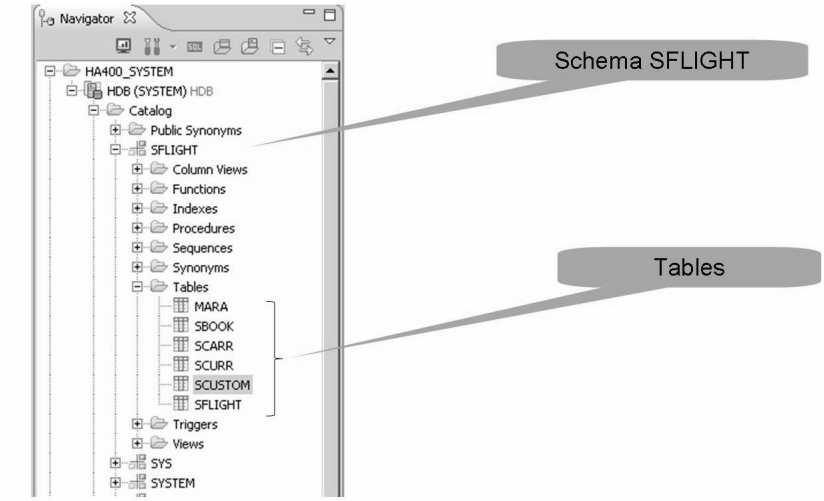


Figure 28: SAP HANA Studio: Schema and Table

© 2013 SAP AG. All rights reserved.

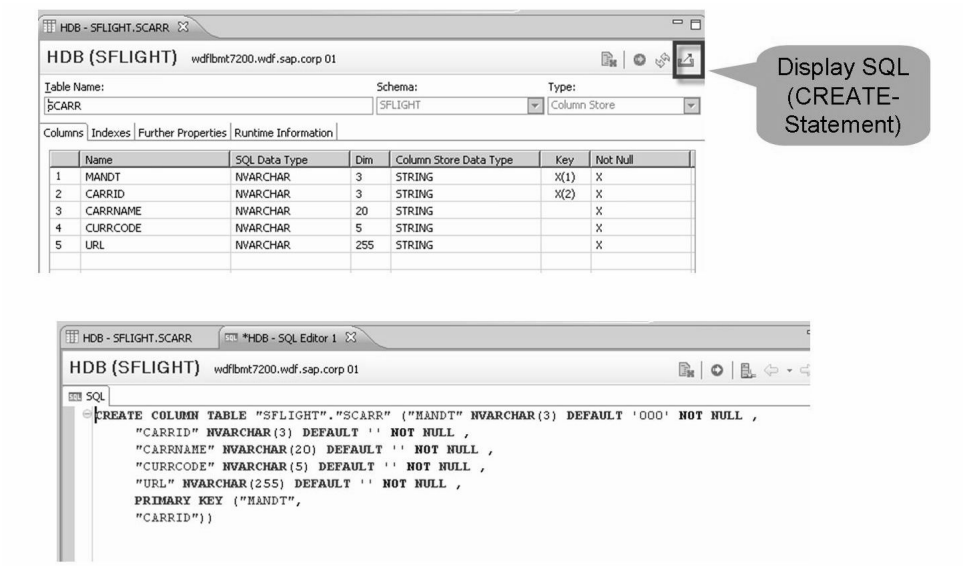


Figure 29: Table Definition

© 2013 SAP AG. All rights reserved.

© 2013 SAP AG. All rights reserved.

Exercise 5: Access SAP HANA via a secondary Database Connection

Exercise Objectives

After completing this exercise, you will be able to:

- understand how secondary database connections are maintained tested
- understand the prerequisites to access tables of the secondary database via open SQL
- quantify various improvements of database accesses for SAP HANA

Business Example

Template:

Report YHA400_OPTIMIZE_OSQL_**S1**

Report YHA400_OPTIMIZE_OSQL_**S2**

Report YHA400_OPTIMIZE_OSQL_**S3**

Solution:

Report YHA400_SEC_DB_CON_**S1**

Report YHA400_SEC_DB_CON_**S2**

Report YHA400_SEC_DB_CON_**S3**

Task 1: Copy and understand templates

Create copies of reports YHA400_OPTIMIZE_OSQL_S1,YHA400_OPTIMIZE_OSQL_S2 and YHA400_OPTIMIZE_OSQL_S3 in your package ZHA400_## (suggested names: ZHA400_##_SEC_DB_1, ZHA400_##_SEC_DB_2, ZHA400_##_SEC_DB_3, where ## is your group number). Activate the programs.

Hint: Alternatively, you may use your own solutions from the previous three exercises as templates for this exercise.

1. Create copies of the reports. Place them in your package ZHA400_00 and assign them to your workbench task.
2. Activate the programs.

Continued on next page

 © 2013 SAP AG. All rights reserved.

Task 2: Analyze

Look for a secondary database connection that points to the SAP HANA server. Find out which tables can be accessed via this connection using open SQL.

1. Look up the secondary database connections which have been maintained in your AS ABAP.
2. Find a database connection that points to your SAP HANA server and note down user and connection ID.
3. The connection ID is _____ and the user for the SAP HANA server is _____
   Fill in the blanks to complete the sentence.
4. Go to SAP HANA Studio and open the schema of the same name as the user in the database connection.
5. Look for the tables you find in this schema. Those are the ones you can access via the secondary database connection using open SQL.
6. The tables are:

   _____
   _____
   _____
   _____

7. Verify for at least one table that the definition on SAP HANA is identical to the definition on the primary database.

Task 3: Access SAP HANA

Change your three programs. In all select statements (subroutines *get_data_solution* and subroutine *get_data_template*) read from SAP HANA rather than the primar database.

1. Edit your program ZHA400_##_SEC_DB_1. Search for all SELECT-statments and use addition `CONNECTION` to access the secondary database rather than the primary database.

   Hint: Instead of hard coding the connection it is recommended to define a global constant of type string.

2. Repeat with programs ZHA400_##_SEC_DB_2 and ZHA400_##_SEC_DB_3.

Continued on next page

 © 2013 SAP AG. All rights reserved.

3. Activate and test your programs.

Task 4: Quantify improvements when using SAP HANA as database

Perform runtime measurements with ABAP Trace (SAT) for all three programs . In each case, compare the runtime consumption of subroutine *get_data_template* and subroutine *get_data_solution*

1. Perform a runtime measurement with ABAP Trace (SAT) for program ZHA400_##_SEC_DB_1 and compare the gross runtime of the two subroutines.

2. The gross runtime of subroutine *get_data_template* is: _____ micro seconds. The gross runtime of subroutine *get_data_solution* is: _____ micro seconds. This is a reduction of the runtime by _____ % through the usage of field lists
   Fill in the blanks to complete the sentence.

3. Perform a runtime measurement with ABAP Trace (SAT) for program ZHA400_##_SEC_DB_2 and compare the gross runtime of the two subroutines.

4. The gross runtime of subroutine *get_data_template* is: _____ micro seconds. The gross runtime of subroutine *get_data_solution* is: _____ micro seconds. This is a reduction of the runtime by _____ % through the usage of a join instead of nested selects.
   Fill in the blanks to complete the sentence.

5. Perform a runtime measurement with ABAP Trace (SAT) for program ZHA400_##_SEC_DB_3 and compare the gross runtime of the two subroutines.

6. The gross runtime of subroutine *get_data_template* is: _____ micro seconds. The gross runtime of subroutine *get_data_solution* is: _____ micro seconds. This is a reduction of the runtime by _____ % through the usage of full buffering instead of nested selects.
   Fill in the blanks to complete the sentence.

© 2013 SAP AG. All rights reserved.

Solution 5: Access SAP HANA via a secondary Database Connection

Task 1: Copy and understand templates

Create copies of reports YHA400_OPTIMIZE_OSQL_S1,YHA400_OPTIMIZE_OSQL_S2 and YHA400_OPTIMIZE_OSQL_S3 in your package ZHA400_## (suggested names: ZHA400_##_SEC_DB_1, ZHA400_##_SEC_DB_2, ZHA400_##_SEC_DB_3, where ## is your group number). Activate the programs.

Hint: Alternatively, you may use your own solutions from the previous three exercises as templates for this exercise.

1. Create copies of the reports. Place them in your package ZHA400_00 and assign them to your workbench task.

   a) Complete this step as you learned to do in previous classes.

2. Activate the programs.

   a) Complete this step as you learned to do in previous classes.

Task 2: Analyze

Look for a secondary database connection that points to the SAP HANA server. Find out which tables can be accessed via this connection using open SQL.

1. Look up the secondary database connections which have been maintained in your AS ABAP.

   a) Start transaction SM30.

   b) Enter *DBCON* as table name and choose *Display*.

2. Find a database connection that points to your SAP HANA server and note down user and connection ID.

   a) On the list, look for a connection with *HDB* in column *DBS*.

   b) Make sure the server in column *connection information* is right.

3. The connection ID is SFLIGHT and the user for the SAP HANA server is SFLIGHT

   Answer: SFLIGHT, SFLIGHT

Continued on next page

 © 2013 SAP AG. All rights reserved.

4. Go to SAP HANA Studio and open the schema of the same name as the user in the database connection.

   a) In SAP HANA Studio, open perspective *Modeler*.

   b) In the Navigation window on the left, open *Catalog -> SFLIGHT*.

5. Look for the tables you find in this schema. Those are the ones you can access via the secondary database connection using open SQL.

   a) Under schema *SFLIGHT*, open node *tables*.

6. The tables are:

   Answer: MARA, SBOOK, SARR, SCUSTOM, SFLIGHT, SPFLI, TCURF, TCURR, TCURV, TCURX

7. Verify for at least one table that the definition on SAP HANA is identical to the definition on the primary database.

   a) Double click on one of the tables to see the list of fields, their SQL data types and dimensions.

   b) On the AS ABAP, open the definition of this table in the ABAP Dictionary (transaction SE11 or SE80).

   c) From the menu choose *Utilities -> Database Object -> Display* to see the definition of this table on the primary database.

Task 3: Access SAP HANA

Change your three programs. In all select statements (subroutines *get_data_solution* and subroutine *get_data_template*) read from SAP HANA rather than the primar database.

1. Edit your program ZHA400_##_SEC_DB_1. Search for all SELECT-statments and use addition `CONNECTION` to access the secondary database rather than the primary database.

   Hint: Instead of hard coding the connection it is recommended to define a global constant of type string.

   a) See source code extract from model solution.

2. Repeat with programs ZHA400_##_SEC_DB_2 and ZHA400_##_SEC_DB_3.

   a) See source code extract from model solution.

3. Activate and test your programs.

   a) Complete this step as you learned to do in previous classes.

Continued on next page

 © 2013 SAP AG. All rights reserved.

Task 4: Quantify improvements when using SAP HANA as database

Perform runtime measurements with ABAP Trace (SAT) for all three programs . In each case, compare the runtime consumption of subroutine *get_data_template* and subroutine *get_data_solution*

1. Perform a runtime measurement with ABAP Trace (SAT) for program ZHA400_##_SEC_DB_1 and compare the gross runtime of the two subroutines.

   a) Perform this step as before.

2. The gross runtime of subroutine *get_data_template* is: _____ micro seconds. The gross runtime of subroutine *get_data_solution* is: _____ micro seconds. This is a reduction of the runtime by _____ % through the usage of field lists

   Answer:

3. Perform a runtime measurement with ABAP Trace (SAT) for program ZHA400_##_SEC_DB_2 and compare the gross runtime of the two subroutines.

   a) Perform this step as before.

4. The gross runtime of subroutine *get_data_template* is: _____ micro seconds. The gross runtime of subroutine *get_data_solution* is: _____ micro seconds. This is a reduction of the runtime by _____ % through the usage of a join instead of nested selects.

   Answer:

5. Perform a runtime measurement with ABAP Trace (SAT) for program ZHA400_##_SEC_DB_3 and compare the gross runtime of the two subroutines.

   a) Perform this step as before.

6. The gross runtime of subroutine *get_data_template* is: _____ micro seconds. The gross runtime of subroutine *get_data_solution* is: _____ micro seconds. This is a reduction of the runtime by _____ % through the usage of full buffering instead of nested selects.

   Answer:

Continued on next page

 © 2013 SAP AG. All rights reserved.

Result

Source code extract from model solution (Program YHA400_SEC_DB_CON_S1)

```
REPORT  yha400_sec_db_con_s1.

...

* Database connection
CONSTANTS c_con TYPE string VALUE 'SFLIGHT'.

...

*&---------------------------------------------------------------------*
*&      Form  get_data_template
*&---------------------------------------------------------------------*

FORM get_data_template CHANGING ct_customers TYPE ty_t_customers.

* Declarations
****************

* Work Area for Result
  DATA ls_customer LIKE LINE OF ct_customers.

* Targets for Select
  DATA: ls_scustom TYPE scustom,
        ls_sbook   TYPE sbook.

* help variables
  DATA lv_count TYPE i.

* Processing
*****************

  CLEAR ct_customers.

  SELECT * FROM scustom
     CONNECTION (c_con)
     INTO ls_scustom.

    ls_customer-id       = ls_scustom-id.
    ls_customer-name     = ls_scustom-name.
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
    ls_customer-postcode = ls_scustom-postcode.
    ls_customer-city     = ls_scustom-city.
    ls_customer-country  = ls_scustom-country.

    CLEAR ls_customer-days_ahead.
    CLEAR lv_count.

    SELECT * FROM sbook
       CONNECTION (c_con)
       INTO ls_sbook
            WHERE customid = ls_scustom-id
              AND cancelled = space.

      ls_customer-days_ahead = ls_customer-days_ahead + ( ls_sbook-fldate - ls_sbook-order_date
      lv_count = lv_count + 1.

    ENDSELECT.
    IF lv_count <> 0.
      ls_customer-days_ahead = ls_customer-days_ahead / lv_count.
      INSERT ls_customer INTO TABLE ct_customers.
    ENDIF.

  ENDSELECT.

  SORT ct_customers BY id.

ENDFORM.                    "

*&---------------------------------------------------------------------*
*&      Form  get_data_solution
*&---------------------------------------------------------------------*
FORM get_data_solution CHANGING ct_customers TYPE ty_t_customers.

* Declarations
****************

* Types for target fields

  TYPES: BEGIN OF lty_s_cust,
           id          TYPE scustom-id,
           name        TYPE scustom-name,
           postcode    TYPE scustom-postcode,
           city        TYPE scustom-city,
           country     TYPE scustom-country,
         END OF lty_s_cust.
```

Continued on next page

© 2013 SAP AG. All rights reserved.

```
  TYPES: BEGIN OF lty_s_book,
              fldate      TYPE sbook-fldate,
             order_date   TYPE sbook-order_date,
         END OF lty_s_book.



* Work Area for Result
  DATA ls_customer LIKE LINE OF ct_customers.

* Targets for Select
  DATA: ls_scustom TYPE lty_s_cust,
        ls_sbook   TYPE lty_s_book.

* help variables
  DATA lv_count TYPE i.

* Processing
*****************

  CLEAR ct_customers.

  SELECT id name postcode city country
    FROM scustom
    CONNECTION (c_con)
    INTO ls_scustom.

    ls_customer-id       = ls_scustom-id.
    ls_customer-name     = ls_scustom-name.
    ls_customer-postcode = ls_scustom-postcode.
    ls_customer-city     = ls_scustom-city.
    ls_customer-country  = ls_scustom-country.

    CLEAR ls_customer-days_ahead.
    CLEAR lv_count.

    SELECT fldate order_date
      FROM sbook
      CONNECTION (c_con)
      INTO ls_sbook
     WHERE customid = ls_scustom-id
       AND cancelled = space.

      ls_customer-days_ahead = ls_customer-days_ahead + ( ls_sbook-fldate - ls_sbook-order_
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
      lv_count = lv_count + 1.

    ENDSELECT.

    IF lv_count <> 0.
      ls_customer-days_ahead = ls_customer-days_ahead / lv_count.
      INSERT ls_customer INTO TABLE ct_customers.

    ENDIF.

  ENDSELECT.

  SORT ct_customers BY id.


ENDFORM.                    "
```


© 2013 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Understand Schemata and Tables in SAP HANA Studio
- Analyze the Definition of Tables in SAP HANA Studio

© 2013 SAP AG. All rights reserved.

Lesson: Performance Rules and Guidelines for SAP HANA

Lesson Overview

Lesson Objectives

After completing this lesson, you will be able to:

- Understand the performance rules and guidelines for HANA

Business Example

Most guidelines remain valid

Reduce number of database accesses

Use sorting, aggregation, sub-selects, etc.

Use joins instead of nested SELECT statements

Use the SAP table buffer

Buffer data in programs using internal tables

Reduce the amount of data transferred between the database and the application server

Avoid SELECT *

Make WHERE clauses as specific as possible

Figure 30: Conclusion I

Some guidelines are even more important

- Only read as much data as you really need
- The more fields you use in the field list, the more columns have to be unpacked (column store)
- Use mass processing wherever possible
- All rows and columns that are needed should be selected in one SQL statement (avoid nested SELECTS and SELECTS in loops)

Figure 31: Conclusion II

© 2013 SAP AG. All rights reserved.

Some guidelines are new

- WHERE clause containing non-indexed fields is not bad anymore (in-memory full table scan is fast)
- Calculation and aggregation in the database are efficient (fast calculation in the database, minimized result set)

Figure 32: Conclusion III

© 2013 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Understand the performance rules and guidelines for HANA

© 2013 SAP AG. All rights reserved.

Unit Summary

You should now be able to:

- Understand the Runtime Analysis (SE30)
- Understand the ABAP Trace (SAT)
- Understand the Code Inspector (SCI)
- Understand the SQL Trace (ST05)
- Describe how to access the SAP HANA Database by using Open SQL and a secondary database connection
- Understand Schemata and Tables in SAP HANA Studio
- Analyze the Definition of Tables in SAP HANA Studio
- Understand the performance rules and guidelines for HANA

© 2013 SAP AG. All rights reserved.

© 2013 SAP AG. All rights reserved.

Unit 3

Implementing ABAP Report on HANA Using ADBC

Unit Overview

This Unit is designed to teach the following topics:

- Why are we using native SQL?
- ABAP Database Connectivity (ADBC)
- Native SQL Syntax

Unit Objectives

After completing this unit, you will be able to:

- Describe the use of native SQL in the context of SAP HANA
- Understand ABAP Database Connectivity (ADBC)
- Use ADBC to execute native SQL statements
- Understand the main difference between native SQL Syntax and Open SQL Syntax
- Write syntactically correct Native SQL Statements

Unit Contents

© 2013 SAP AG. All rights reserved.

Lesson: Why are we using native SQL?

Lesson Overview

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the use of native SQL in the context of SAP HANA

Business Example

SQL (Structured Query Language) is a largely standardized language for accessing relational databases. It can be divided into three areas:

Data Manipulation Language (DML)
Statements for reading and changing data in database tables.
E.g. SELECT; INSERT; UPDATE; DELETE

Data Definition Language (DDL)
Statements for creating and administering database tables.
E.g. CREATE TABLE; DROP TABLE

Data Control Language (DCL)
Statements for authorization and consistency checks.
E.g. GRANT SELECT ON <..> TO <…>; REVOKE SELECT

Figure 33: Reminder: SQL In A Nutshell

Open SQL in a nutshell

Open SQL provides a uniform syntax and semantics for all of the database systems supported by SAP.

ABAP programs that only use Open SQL statements will work in any SAP system, regardless of the database system in use.

Open SQL statements can only work with database tables that have been created in the ABAP Dictionary.

Open SQL can be used via secondary database connections

Relation of Open SQL to the DML/DDL/DCL aspects of SQL

Open SQL covers the DML aspects

The ABAP dictionary tools control the DDL aspects

The DCL aspects are not reflected in standard ABAP; instead the data access is managed by the ABAP authorization concept.

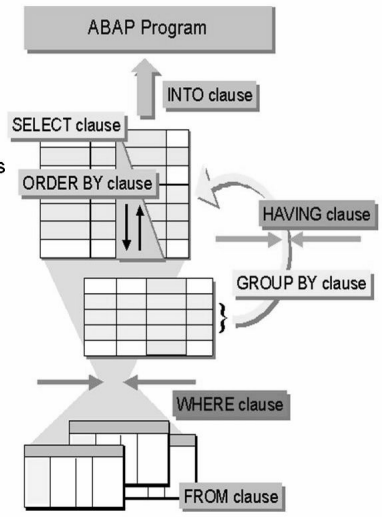
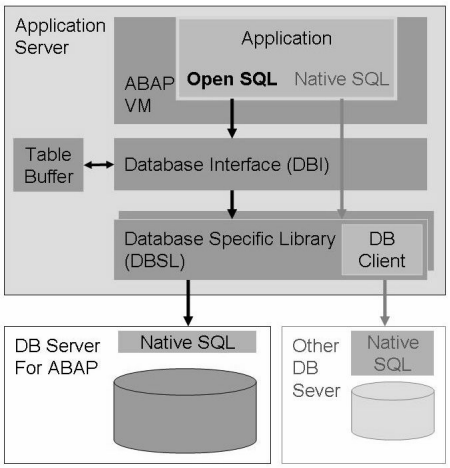


Figure 34: Open SQL in ABAP

© 2013 SAP AG. All rights reserved.



Open SQL statement

```
SELECT carrid connid fldate
       FROM sflight
       INTO CORRESPONDING FIELDS OF TABLE sflight_tab
       FOR ALL ENTRIES IN entry_tab
       WHERE carrid = entry_tab-carrid AND
             connid = entry_tab-connid.
```

Translated to native SQL (taken from SQL trace; ST05)

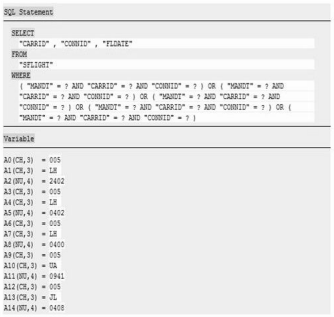


Figure 35: Example: Translation of Open SQL to Native SQL

Open SQL does not support everything possible with standard SQL DML or HANA SQL

Standard SQL DML:
- Fixed-value and computed columns
- Sub-queries in the SELECT or FROM clauses
- UNIONs

HANA SQL:
- Using HANA SQL's CASE construct
- Using HANA built-in functions, e.g. DAYS_BETWEEN( )…
- Accessing HANA Views

If you need these features, you need to use Native SQL

Figure 36: Restrictions of Open SQL

© 2013 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Describe the use of native SQL in the context of SAP HANA

© 2013 SAP AG. All rights reserved.

Lesson: ABAP Database Connectivity (ADBC)

Lesson Overview

Lesson Objectives

After completing this lesson, you will be able to:

- Understand ABAP Database Connectivity (ADBC)
- Use ADBC to execute native SQL statements

Business Example

ADBC

***A**BAP **D**atab**a**se **C**onnectivity*(ADBC) is an Object-based API

ADBC allows native SQL access providing:

Flexibility

Where used list

Error handling

Main classes are:

CL_SQL_CONNECTION

CL_SQL_STATEMENT

CL_SQL_RESULT_SET

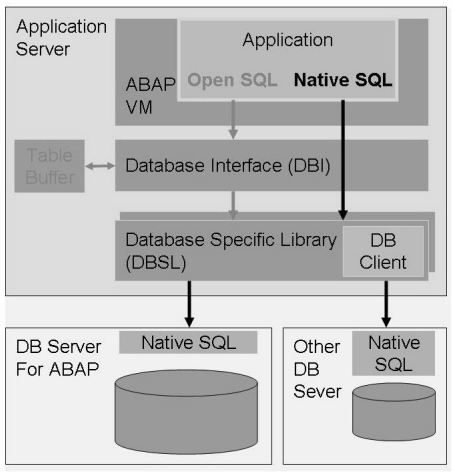


Figure 37: Basic Features of ADBC

© 2013 SAP AG. All rights reserved.

1.	Choose database connection (only when accessing secondary DB)	Call method *get_connection()* of class CL_SQL_CONNECTION
2.	Create a statement object	Instantiation of class CL_SQL_STATEMENT
3.	Fill string variable with SQL syntax	Use either CONCATENATE or string templates/string expressions
4.	Issue native SQL call	Call method *execute_query()* of class CL_SQL_STATEMENT
5.	Assign target variable for result set	Call method *set_param()* or *set_param_table()* of class CL_SQL_RESULT_SET
6.	Retrieve result set	Call method *next_package()* of class CL_SQL_RESULT_SET
7.	Close query and release resources	Method *close()* of class CL_SQL_RESULT_SET

Figure 38: Sequence for Reading Data with ADBC

```
DATA: lo_con    TYPE REF TO cl_sql_connection,
      lo_sql    TYPE REF TO cl_sql_statement,
      lv_sql    TYPE string,
      lo_result TYPE REF TO cl_sql_result_set,
      lr_data   TYPE REF TO data,
      lt_flight TYPE STANDARD TABLE OF sflight.
TRY.
  lo_con = cl_sql_connection=>get_connection( 'HANA' ).

  CREATE OBJECT lo_sql
       EXPORTING
         con_ref = lo_con
         table_name_for_trace = 'SFLIGHT'.

  lv_sql = `SELECT ...        `.

  lo_result = lo_sql->execute_query( lv_sql ).

  GET REFERENCE OF lt_flight INTO lr_flight.
  lo_result->set_param_table( lr_flight ).
  lo_result->next_package( ).

  lo_result->close( ).
CATCH cx_sql_exception INTO ... .
    ...
ENDTRY.
```

Prepare native SQL call
- Specify secondary DB connection
- And info for SQL trace

Define native SQL syntax

Issue native SQL call

Retrieve result of native SQL call – in packages if needed

Figure 39: Coding Example: ABAP Database Connectivity (ADBC)

© 2013 SAP AG. All rights reserved.

No syntax check for native SQL statements

Make sure to handle exception cx_sql_exception

No hashed or sorted tables allowed as target

Use standard table (probably with hashed or sorted secondary key)

No automatic client handling

Do not forget to specify the client explicitly in the WHERE-clause, join conditions, etc.

No guaranteed release of allocated resources on DB

Do not forget to close the query

Figure 40: ADBC: Important Things to Keep in Mind

© 2013 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Understand ABAP Database Connectivity (ADBC)
- Use ADBC to execute native SQL statements

© 2013 SAP AG. All rights reserved.

Lesson: Native SQL Syntax

Lesson Overview

Lesson Objectives

After completing this lesson, you will be able to:

- Understand the main difference between native SQL Syntax and Open SQL Syntax
- Write syntactically correct Native SQL Statements

Business Example

Native SQL in a nutshell

Loosely integrated into ABAP, but allows access to all functions contained in the programming interface of the respective database system.

Syntax of native SQL statements is not checked. Statements are directly sent to the database system.

All tables, in all schemas can be accessed

No automatic client handling, no table buffering.

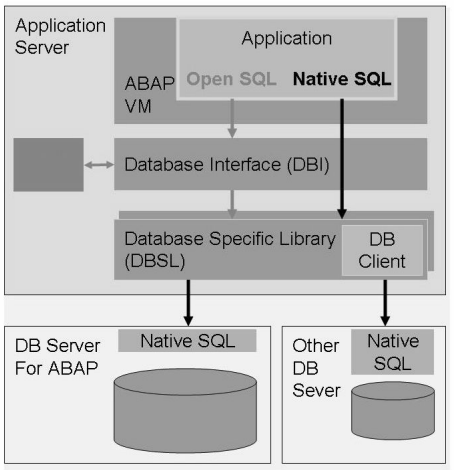


Figure 41: Native SQL in ABAP

Common ground between native HANA SQL and Open SQL:

- Key words are case insensitive
- As seem table & column names – but beware! HANA converts them to upper case unless quoted

Important syntax differences:

- Column lists are comma separated
- Table and column name qualifiers are separated using "."
- There is no "FOR ALL ENTRIES IN" or "INTO CORRESPONDING FIELDS OF" – these are only known to ABAP / Open SQL

Figure 42: Native SQL Syntax → Open SQL Syntax

© 2013 SAP AG. All rights reserved.

Open SQL:

```
SELECT carrid connid cityfrom cityto
    FROM spfli
    INTO ...
    WHERE carrid = 'LH'
      AND connid = '0400'
    ORDER BY carrid.
```

Native SQL:

```
SELECT carrid, connid, cityfrom, cityto
       FROM sflight.spfli
       INTO ...
        WHERE carrid = 'LH'
          AND connid = '0400'
          AND  mandt = '800'
       ORDER BY mandt, carrid
```

Comma-separated field list

Schema has to be specified (if not default schema of user)

No automatic client handling
Client is 'just a key field '

Note that there will be no INTO-clause when ADBC is used

Figure 43: Example 1: Open SQL Syntax and Native SQL Syntax

Open SQL:

```
SELECT b~carrid a~carrname b~connid b~cityfrom b~cityto
    FROM scarr AS a INNER JOIN spfli AS B
      ON a~carrid = b~carrid
    INTO ...
    WHERE b~carrid = 'LH'
    AND   b~connid = '0400'
    ORDER BY b~carrid.
```

Native SQL:

```
SELECT b.carrid, b.connid, a.carrname, b.cityfrom, b.cityto
    FROM sflight.scarr AS a INNER JOIN sflight.spfli AS b
      ON a.mandt  = b.mandt
     AND a.carrid = b.carrid
    INTO ...
       WHERE b.carrid = 'LH'
          AND b.connid = '0400'
          AND  b.mandt = '800'
      ORDER BY mandt, carrid
```

Figure 44: Example 2: Joins in Open SQL and Native SQL

© 2013 SAP AG. All rights reserved.

Exercise 6: Use ABAP Database Connectivity (ADBC) to issue a native SQL SELECT statement

Exercise Objectives

After completing this exercise, you will be able to:

- use ABAP Database Connectivity (ADBC) to read data from a Database
- understand the main differences between the syntax of native SQL and open SQL

Business Example

Template:

Report YHA400_ADBC_**T1**

Solution:

Report YHA400_ADBC_**S1**

Task 1: Copy and understand template

Create a copy of report YHA400_ADBC_T1 in your package ZHA400_## (suggested name: ZHA400_##_ADBC_1, where ## is your group number). Activate and execute the program.

1. Create a copy of the report. Place it in your package ZHA400_00 and assign it to your workbench task.
2. Activate and execute the program.

Task 2: Use ADBC to access SAP HANA via native SQL

Edit your program. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*. In subroutine *get_data_solution*, replace the open SQL SELECT on SCUSTOM with a native SQL SELECT issued by means of ADBC. Make sure you read exactly the same data.

1. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*
2. In subroutine *get_data_solution*, create an instance of class CL_SQL_CONNECTION.

Hint: Use public static method *get_connection* of that class.

Continued on next page

 © 2013 SAP AG. All rights reserved.

3. Create an instance of class CL_SQL_STATEMENT, handing over the instance of CL_SQL_CONNECTION to the constructor
4. Define a string variable (suggested name: *lv_sql*) and fill it with the native SQL syntax that corresponds to the open SQL statement found in the template. Use either a classical CONCATENATE or a more modern string-template/-expression approach.

Caution: Make sure to use "," as separator in the field list and the ORDER BY-addition and don't forget to add a WHERE-clause for the client.

5. Create an instance of CL_SQL_RESULT_SET by executing the query.

Hint: Use public instance method *execute_query* of class CL_SQL_STATEMENT.

6. Define a reference variable of type REF TO DATA, let it point to the target data object and hand this reference over to the instance of CL_SQL_RESULT_SET.

Hint: Use method *set_param_table* of class CL_SQL_RESULT_SET.

7. Retrieve the result and close the query.

Hint: Use methods *next_package* and *close* of class CL_SQL_RESULT_SET.

8. Implement an exception handling for all class based exceptions that might be raised by the called ADBC-methods.
9. Activate and test your program. Make sure the two subroutines deliver exactly the same data.

© 2013 SAP AG. All rights reserved.

Solution 6: Use ABAP Database Connectivity (ADBC) to issue a native SQL SELECT statement

Task 1: Copy and understand template

Create a copy of report YHA400_ADBC_T1 in your package ZHA400_## (suggested name: ZHA400_##_ADBC_1, where ## is your group number). Activate and execute the program.

1. Create a copy of the report. Place it in your package ZHA400_00 and assign it to your workbench task.
   a) Complete this step as you learned to do in previous classes.
2. Activate and execute the program.
   a) Complete this step as you learned to do in previous classes.

Task 2: Use ADBC to access SAP HANA via native SQL

Edit your program. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*. In subroutine *get_data_solution*, replace the open SQL SELECT on SCUSTOM with a native SQL SELECT issued by means of ADBC. Make sure you read exactly the same data.

1. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*
   a) Use *Copy & Paste*-function of the ABAP Editor.
2. In subroutine *get_data_solution*, create an instance of class CL_SQL_CONNECTION.

Hint: Use public static method *get_connection* of that class.

   a) See source code extract from model solution.
3. Create an instance of class CL_SQL_STATEMENT, handing over the instance of CL_SQL_CONNECTION to the constructor
   a) See source code extract from model solution.
4. Define a string variable (suggested name: *lv_sql*) and fill it with the native SQL syntax that corresponds to the open SQL statement found in the template. Use either a classical CONCATENATE or a more modern string-template/-expression approach.

Continued on next page

 © 2013 SAP AG. All rights reserved.

Caution: Make sure to use "," as separator in the field list and the ORDER BY-addition and don't forget to add a WHERE-clause for the client.

a) See source code extract from model solution.

5. Create an instance of CL_SQL_RESULT_SET by executing the query.

Hint: Use public instance method *execute_query* of class CL_SQL_STATEMENT.

a) See source code extract from model solution.

6. Define a reference variable of type REF TO DATA, let it point to the target data object and hand this reference over to the instance of CL_SQL_RESULT_SET.

Hint: Use method *set_param_table* of class CL_SQL_RESULT_SET.

a) See source code extract from model solution.

7. Retrieve the result and close the query.

Hint: Use methods *next_package* and *close* of class CL_SQL_RESULT_SET.

a) See source code extract from model solution.

8. Implement an exception handling for all class based exceptions that might be raised by the called ADBC-methods.

a) See source code extract from model solution.

9. Activate and test your program. Make sure the two subroutines deliver exactly the same data.

a) Complete this step as you learned to do in previous classes.

Result

Source code extract from model solution (Program YHA400_ADBC_S1)

```
*&---------------------------------------------------------------------*
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
*&      Form  get_data_template
*&---------------------------------------------------------------------*

FORM get_data_template CHANGING ct_customers TYPE ty_t_customers.

  CLEAR ct_customers.

  SELECT id name postcode city country
            FROM scustom
      CONNECTION (c_con)
      INTO TABLE ct_customers
      ORDER BY id.

ENDFORM.                    "

*&---------------------------------------------------------------------*
*&      Form  get_data_solution
*&---------------------------------------------------------------------*
FORM get_data_solution CHANGING ct_customers TYPE ty_t_customers.


* Declarations
****************

* ADBC Objects and Variables

  DATA: lo_con     TYPE REF TO cl_sql_connection,
        lo_sql     TYPE REF TO cl_sql_statement,
        lo_result  TYPE REF TO cl_sql_result_set,
        lv_sql     TYPE string,
        lr_data    TYPE REF TO data.

* Exception Handling

  DATA:   lx_sql_exc TYPE REF TO cx_sql_exception,
          lv_text TYPE string.

* Processing
*****************************

  CLEAR ct_customers.

*  SELECT id name postcode city country
*            FROM scustom
*      CONNECTION sflight
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
*      INTO TABLE ct_customers.

  TRY.

* Get secondary DB Connection
      lo_con = cl_sql_connection=>get_connection( c_con ).

* Create statement
      CREATE OBJECT lo_sql
        EXPORTING
          con_ref = lo_con.

* create SQL statement

      CONCATENATE `SELECT ID, NAME, POSTCODE, CITY, COUNTRY`
                  `FROM SFLIGHT.SCUSTOM`
                  `WHERE MANDT = `
                   sy-mandt
                  `ORDER BY MANDT, ID `
            INTO lv_sql
            SEPARATED BY space.

* Alternative: Use string templates and string expressions
*      lv_sql = |SELECT ID, NAME, POSTCODE, CITY, COUNTRY | &&
*               |FROM SFLIGHT.SCUSTOM | &&
*               |WHERE MANDT = { sy-mandt } | &&
*               |ORDER BY MANDT, ID |.

* Execute Query

      lo_result = lo_sql->execute_query( lv_sql ).

* Read result into internal Table
      GET REFERENCE OF ct_customers INTO lr_data.
      lo_result->set_param_table( lr_data ).
      lo_result->next_package( ).
      lo_result->close( ).

    CATCH cx_sql_exception INTO lx_sql_exc.  " Exception Class for SQL Error
      lv_text = lx_sql_exc->get_text( ).
      MESSAGE lv_text TYPE 'E'.
  ENDTRY.

ENDFORM.                    "
```


© 2013 SAP AG. All rights reserved.

Exercise 7: Issue a native SQL Join via ABAP Database Connectivity (ADBC)

Exercise Objectives

After completing this exercise, you will be able to:

- use ABAP Database Connectivity (ADBC) to read data from a database
- understand the syntax of joins in native SQL

Business Example

Template:

Report YHA400_ADBC_**T2**

Solution:

Report YHA400_ADBC_**S2**

Task 1: Copy and understand template

Create a copy of report YHA400_ADBC_T2 in your package ZHA400_## (suggested name: ZHA400_##_ADBC_2, where ## is your group number). Activate and execute the program.

1. Create a copy of the report. Place it in your package ZHA400_00 and assign it to your workbench task.
2. Activate and execute the program.

Task 2: Use ADBC to send a native SQL-SELECT with a join to the SAP HANA database

Edit your program. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*. In subroutine *get_data_solution*, replace the open SQL SELECT on SCUSTOM and SBOOK with a native SQL SELECT issued by means of ADBC. Make sure you read exactly the same data.

1. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*

Continued on next page

 © 2013 SAP AG. All rights reserved.

2. In subroutine *get_data_solution*, create an instance of class CL_SQL_CONNECTION. Create an instance of class CL_SQL_STATEMENT, handing over the instance of CL_SQL_CONNECTION to the constructor

Hint: As of Netweaver 7.02, you may chain these two methods together, thus omitting the reference variable for the connection object.

3. Define a string variable (suggested name: *lv_sql*) and fill it with the native SQL syntax that corresponds to the open SQL statement found in the template. Use either a classical CONCATENATE or a more modern string-template/-expression approach.

Caution: Make sure to use "." instead of "~" as field selector.

4. Execute the query, assign the target data object, retrieve the data and close the query.

Hint: Check the definition of the target data object carefully.

5. What do you have to change to make the data object work as target for ADBC?

6. Implement an exception handling for all class based exceptions that might be raised by the called ADBC-methods.

7. Activate and test your program. Make sure the two subroutines deliver exactly the same data.

© 2013 SAP AG. All rights reserved.

Solution 7: Issue a native SQL Join via ABAP Database Connectivity (ADBC)

Task 1: Copy and understand template

Create a copy of report YHA400_ADBC_T2 in your package ZHA400_## (suggested name: ZHA400_##_ADBC_2, where ## is your group number). Activate and execute the program.

1. Create a copy of the report. Place it in your package ZHA400_00 and assign it to your workbench task.

   a) Complete this step as you learned to do in previous classes.

2. Activate and execute the program.

   a) Complete this step as you learned to do in previous classes.

Task 2: Use ADBC to send a native SQL-SELECT with a join to the SAP HANA database

Edit your program. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*. In subroutine *get_data_solution*, replace the open SQL SELECT on SCUSTOM and SBOOK with a native SQL SELECT issued by means of ADBC. Make sure you read exactly the same data.

1. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*

   a) Use *Copy & Paste*-function of the ABAP Editor.

2. In subroutine *get_data_solution*, create an instance of class CL_SQL_CONNECTION. Create an instance of class CL_SQL_STATEMENT, handing over the instance of CL_SQL_CONNECTION to the constructor

   Hint: As of Netweaver 7.02, you may chain these two methods together, thus omitting the reference variable for the connection object.

   a) See source code extract from model solution.

3. Define a string variable (suggested name: *lv_sql*) and fill it with the native SQL syntax that corresponds to the open SQL statement found in the template. Use either a classical CONCATENATE or a more modern string-template/-expression approach.

Continued on next page

 © 2013 SAP AG. All rights reserved.

Caution: Make sure to use "." instead of "~" as field selector.

a) See source code extract from model solution.

4. Execute the query, assign the target data object, retrieve the data and close the query.

Hint: Check the definition of the target data object carefully.

a) See source code extract from model solution.

5. What do you have to change to make the data object work as target for ADBC?

Answer: Sorted tables as targets are not supported by ADBC.

6. Implement an exception handling for all class based exceptions that might be raised by the called ADBC-methods.

a) See source code extract from model solution.

7. Activate and test your program. Make sure the two subroutines deliver exactly the same data.

a) Complete this step as you learned to do in previous classes.

Result

Source code extract from model solution (Program YHA400_ADBC_S2)

```
*&---------------------------------------------------------------------*
*&      Form  get_data_template
*&---------------------------------------------------------------------*

FORM get_data_template CHANGING ct_customers TYPE ty_t_customers.

* Declarations
****************

* Types for target fields

  TYPES: BEGIN OF lty_s_cust_book,
           id          TYPE scustom-id,
           name        TYPE scustom-name,
           postcode    TYPE scustom-postcode,
```

Continued on next page

© 2013 SAP AG. All rights reserved.

```
           city        TYPE scustom-city,
           country     TYPE scustom-country,
           fldate      TYPE sbook-fldate,
           order_date  TYPE sbook-order_date,
         END OF lty_s_cust_book.

* Work Area for Result
  DATA ls_customer LIKE LINE OF ct_customers.


* Targets for Select

  DATA: lt_cust_book   TYPE SORTED TABLE OF lty_s_cust_book WITH NON-UNIQUE KEY id,
        ls_cust_book TYPE lty_s_cust_book.

* help variables
  DATA lv_count TYPE i.

* Processing
*****************

  CLEAR ct_customers.

  SELECT c~id c~name c~postcode c~city c~country b~fldate b~order_date
    FROM scustom AS c INNER JOIN sbook   AS b
    ON   c~id = b~customid
    CONNECTION (c_con)
    INTO TABLE lt_cust_book
           WHERE b~cancelled = space
    ORDER BY c~id.

  LOOP AT lt_cust_book INTO ls_cust_book.

    IF sy-tabix = 1.
      ls_customer-id       = ls_cust_book-id.
      ls_customer-name     = ls_cust_book-name.
      ls_customer-postcode = ls_cust_book-postcode.
      ls_customer-city     = ls_cust_book-city.
      ls_customer-country  = ls_cust_book-country.

    ELSEIF ls_cust_book-id <> ls_customer-id.

      ls_customer-days_ahead = ls_customer-days_ahead / lv_count.
      INSERT ls_customer INTO TABLE ct_customers.
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
      CLEAR ls_customer.
      CLEAR lv_count.

      ls_customer-id       = ls_cust_book-id.
      ls_customer-name     = ls_cust_book-name.
      ls_customer-postcode = ls_cust_book-postcode.
      ls_customer-city     = ls_cust_book-city.
      ls_customer-country  = ls_cust_book-country.

    ENDIF.

    lv_count = lv_count + 1.
    ls_customer-days_ahead = ls_customer-days_ahead + ls_cust_book-fldate - ls_cust_book-order_

  ENDLOOP.

  ls_customer-days_ahead = ls_customer-days_ahead / lv_count.
  INSERT ls_customer INTO TABLE ct_customers.

*  SORT ct_customers BY id. "already sorted


ENDFORM.                    "

*&---------------------------------------------------------------------*
*&      Form  get_data_solution
*&---------------------------------------------------------------------*
FORM get_data_solution CHANGING ct_customers TYPE ty_t_customers.

* Declarations
****************

* Types for target fields

  TYPES: BEGIN OF lty_s_cust_book,
           id          TYPE scustom-id,
           name        TYPE scustom-name,
           postcode    TYPE scustom-postcode,
           city        TYPE scustom-city,
           country     TYPE scustom-country,
           fldate      TYPE sbook-fldate,
           order_date  TYPE sbook-order_date,
         END OF lty_s_cust_book.

* Work Area for Result
```

Continued on next page

© 2013 SAP AG. All rights reserved.

```
  DATA ls_customer LIKE LINE OF ct_customers.


* Targets for Select
*  DATA: lt_cust_book   TYPE SORTED TABLE OF lty_s_cust_book WITH NON-UNIQUE KEY id, " sore
  DATA: lt_cust_book   TYPE STANDARD TABLE OF lty_s_cust_book WITH NON-UNIQUE KEY id,
        ls_cust_book TYPE lty_s_cust_book.

* help variables
  DATA lv_count TYPE i.

* ADBC Objects and variables
  DATA:  lo_con     TYPE REF TO cl_sql_connection,
         lo_sql     TYPE REF TO cl_sql_statement,
         lo_result  TYPE REF TO cl_sql_result_set,
         lv_sql     TYPE string,
         lr_data    TYPE REF TO data.

* Eception handling
  DATA:   lx_sql_exc TYPE REF TO cx_sql_exception,
          lv_text TYPE string.

* Processing
****************************************
  CLEAR ct_customers.

*  SELECT c~id c~name c~postcode c~city c~country b~fldate b~order_date
*    FROM scustom AS c INNER JOIN sbook   AS b
*    ON   c~id = b~customid
*    CONNECTION (c_con)
*    INTO TABLE lt_cust_book
*           WHERE b~cancelled = space.

  TRY.

* Get secondary DB Connection and Create statement object

      lo_con = cl_sql_connection=>get_connection( c_con ).

      CREATE OBJECT lo_sql
        EXPORTING
          con_ref = lo_con.

* optional as of NW 7.02
*
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
*      CREATE OBJECT lo_sql
*        EXPORTING
*          con_ref = cl_sql_connection=>get_connection( c_con ).

* create SQL statement

      CONCATENATE `SELECT C.ID, C.NAME, C.POSTCODE, C.CITY, C.COUNTRY, B.FLDATE, B.ORDER_DATE
                  `FROM SFLIGHT.SCUSTOM AS C INNER JOIN SFLIGHT.SBOOK AS B`
                  `ON C.MANDT = B.MANDT`
                  `AND C.ID = B.CUSTOMID`
                  `WHERE C.MANDT = `
                  sy-mandt
                  `AND B.CANCELLED = ' '`
                  `ORDER BY C.MANDT, C.ID`
            INTO lv_sql
            SEPARATED BY space.

* Alternative: Use string templates and string expressions
*      lv_sql = |SELECT C.ID, C.NAME, C.POSTCODE, C.CITY, C.COUNTRY | &&
*               |FROM SFLIGHT.SCUSTOM AS C INNER JOIN SFLIGHT.SBOOK AS B | &&
*               |ON C.MANDT = B.MANDT AND C.ID = B.CUSTOMID | &&
*               |WHERE C.MANDT = { sy-mandt } | &&
*               |AND B.CANCELLED = { SPACE } | &&
*               |ORDER BY C.MANDT, C.ID |.

* Execute Query

      lo_result = lo_sql->execute_query( lv_sql ).

* Read result into internal Table
      GET REFERENCE OF lt_cust_book INTO lr_data.
      lo_result->set_param_table( lr_data ).
      lo_result->next_package( ).
      lo_result->close( ).

    CATCH cx_sql_exception INTO lx_sql_exc.  " Exception Class for SQL Error
      lv_text = lx_sql_exc->get_text( ).
      MESSAGE lv_text TYPE 'E'.

  ENDTRY.

  SORT lt_cust_book BY id.

  LOOP AT lt_cust_book INTO ls_cust_book.
```

Continued on next page

© 2013 SAP AG. All rights reserved.

```
    IF sy-tabix = 1.
      ls_customer-id       = ls_cust_book-id.
      ls_customer-name     = ls_cust_book-name.
      ls_customer-postcode = ls_cust_book-postcode.
      ls_customer-city     = ls_cust_book-city.
      ls_customer-country  = ls_cust_book-country.

    ELSEIF ls_cust_book-id <> ls_customer-id.

      ls_customer-days_ahead = ls_customer-days_ahead / lv_count.
      INSERT ls_customer INTO TABLE ct_customers.

      CLEAR ls_customer.
      CLEAR lv_count.

      ls_customer-id       = ls_cust_book-id.
      ls_customer-name     = ls_cust_book-name.
      ls_customer-postcode = ls_cust_book-postcode.
      ls_customer-city     = ls_cust_book-city.
      ls_customer-country  = ls_cust_book-country.

    ENDIF.

    lv_count = lv_count + 1.
    ls_customer-days_ahead = ls_customer-days_ahead + ls_cust_book-fldate - ls_cust_book-or

  ENDLOOP.

  ls_customer-days_ahead = ls_customer-days_ahead / lv_count.
  INSERT ls_customer INTO TABLE ct_customers.

*  SORT ct_customers BY id. "already sorted

ENDFORM.                    "
```


© 2013 SAP AG. All rights reserved.

© 2013 SAP AG. All rights reserved.

Exercise 8: Make use of native SQL to access database specific functions

Exercise Objectives

After completing this exercise, you will be able to:

- use ABAP Database Connectivity (ADBC) to read data from a database
- understand the syntax of native SQL

Business Example

Template:

Report YHA400_ADBC_**T3**

Solution:

Report YHA400_ADBC_**S3**

Task 1: Copy and understand template

Create a copy of report YHA400_ADBC_T3 in your package ZHA400_## (suggested name: ZHA400_##_ADBC_3, where ## is your group number). Activate and execute the program.

1. Create a copy of the report. Place it in your package ZHA400_00 and assign it to your workbench task.
2. Activate and execute the program.

Task 2: Use ADBC and native SQL to replace the complete data retrieval by just one SELECT

Edit your program. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*. In subroutine *get_data_solution*. Implement a native SQL SELECT that fills internal table *ct_customers* directly without any further processing. Make use of the fact that within the field list of native SQL SELECTs calculations are possible. Also use aggregation and ordering techniques of SQL. Make sure you read exactly the same data.

1. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*

Continued on next page

 © 2013 SAP AG. All rights reserved.

2. In subroutine *get_data_solution*, change the native SQL Statement. Eliminate fields B.ORDER_DATE and B.FLDATE from the field list and replace them by the aggregated difference.

Hint: The correct syntax is `AVG( DAYS_BETWEEN(B.ORDER_DATE,B.FLDATE) ) AS DAYS_AHEAD`

3. In the native SQL Syntax, add a GROUP BY addition that lists the client and all fields found in the field list.
4. Replace target data object *lt_cust_book* with *ct_customers* .
5. Remove the now superfluous loop over lt_cust_book with the calculation and aggregation.
6. Activate and test your program. Make sure the two subroutines deliver exactly the same data.

© 2013 SAP AG. All rights reserved.

Solution 8: Make use of native SQL to access database specific functions

Task 1: Copy and understand template

Create a copy of report YHA400_ADBC_T3 in your package ZHA400_## (suggested name: ZHA400_##_ADBC_3, where ## is your group number). Activate and execute the program.

1. Create a copy of the report. Place it in your package ZHA400_00 and assign it to your workbench task.

   a) Complete this step as you learned to do in previous classes.

2. Activate and execute the program.

   a) Complete this step as you learned to do in previous classes.

Task 2: Use ADBC and native SQL to replace the complete data retrieval by just one SELECT

Edit your program. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*. In subroutine *get_data_solution*. Implement a native SQL SELECT that fills internal table *ct_customers* directly without any further processing. Make use of the fact that within the field list of native SQL SELECTs calculations are possible. Also use aggregation and ordering techniques of SQL. Make sure you read exactly the same data.

1. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*

   a) Use *Copy & Paste*-function of the ABAP Editor.

2. In subroutine *get_data_solution*, change the native SQL Statement. Eliminate fields B.ORDER_DATE and B.FLDATE from the field list and replace them by the aggregated difference.

   Hint: The correct syntax is `AVG( DAYS_BETWEEN(B.ORDER_DATE,B.FLDATE) ) AS DAYS_AHEAD`

   a) See source code extract from model solution.

3. In the native SQL Syntax, add a GROUP BY addition that lists the client and all fields found in the field list.

   a) See source code extract from model solution.

Continued on next page

 © 2013 SAP AG. All rights reserved.

4. Replace target data object *lt_cust_book* with *ct_customers* .
   a) See source code extract from model solution.
5. Remove the now superfluous loop over lt_cust_book with the calculation and aggregation.
   a) See source code extract from model solution.
6. Activate and test your program. Make sure the two subroutines deliver exactly the same data.
   a) Complete this step as you learned to do in previous classes.

Result

Source code extract from model solution (Program YHA400_ADBC_S3)

```
*&---------------------------------------------------------------------*
*&      Form  get_data_template
*&---------------------------------------------------------------------*

FORM get_data_template CHANGING ct_customers TYPE ty_t_customers.

* Declarations
****************

* Types for target fields

  TYPES: BEGIN OF lty_s_cust_book,
           id          TYPE scustom-id,
           name        TYPE scustom-name,
           postcode    TYPE scustom-postcode,
           city        TYPE scustom-city,
           country     TYPE scustom-country,
           fldate      TYPE sbook-fldate,
           order_date  TYPE sbook-order_date,
         END OF lty_s_cust_book.

* Work Area for Result
  DATA ls_customer LIKE LINE OF ct_customers.


* Targets for Select
*  DATA: lt_cust_book   TYPE SORTED TABLE OF lty_s_cust_book WITH NON-UNIQUE KEY id, " soreted
```

Continued on next page

© 2013 SAP AG. All rights reserved.

```
  DATA: lt_cust_book   TYPE STANDARD TABLE OF lty_s_cust_book WITH NON-UNIQUE KEY id,
        ls_cust_book TYPE lty_s_cust_book.

* help variables
  DATA lv_count TYPE i.

* ADBC Objects and variables
  DATA:  lo_con     TYPE REF TO cl_sql_connection,
         lo_sql     TYPE REF TO cl_sql_statement,
         lo_result  TYPE REF TO cl_sql_result_set,
         lv_sql     TYPE string,
         lr_data    TYPE REF TO data.

* Eception handling
  DATA:   lx_sql_exc TYPE REF TO cx_sql_exception,
          lv_text TYPE string.

* Processing
****************************************
  CLEAR ct_customers.

*  SELECT c~id c~name c~postcode c~city c~country b~fldate b~order_date
*    FROM scustom AS c INNER JOIN sbook   AS b
*    ON   c~id = b~customid
*    CONNECTION sflight
*    INTO TABLE lt_cust_book
*           WHERE b~cancelled = space.

  TRY.

* Get secondary DB Connection
      lo_con = cl_sql_connection=>get_connection( c_con ).

* Create statement
      CREATE OBJECT lo_sql
        EXPORTING
          con_ref = lo_con.

* create SQL statement
      CONCATENATE `SELECT C.ID, C.NAME, C.POSTCODE, C.CITY, C.COUNTRY, B.FLDATE, B.ORDER_DA
                  `FROM SFLIGHT.SCUSTOM AS C INNER JOIN SFLIGHT.SBOOK AS B`
                  `ON C.MANDT = B.MANDT`
                  `AND C.ID = B.CUSTOMID`
                  `WHERE C.MANDT = `
                  sy-mandt
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
                `AND B.CANCELLED = ' '`
                `ORDER BY C.MANDT, C.ID`
          INTO lv_sql
          SEPARATED BY space.

* Alternative: Use string templates and string expressions
*      lv_sql = |SELECT C.ID, C.NAME, C.POSTCODE, C.CITY, C.COUNTRY | &&
*               |FROM SFLIGHT.SCUSTOM AS C INNER JOIN SFLIGHT.SBOOK AS B | &&
*               |ON C.MANDT = B.MANDT AND C.ID = B.CUSTOMID | &&
*               |WHERE C.MANDT = { sy-mandt } | &&
*               |AND B.CANCELLED = { SPACE } | &&
*               |ORDER BY C.MANDT, C.ID |.

* Execute Query
     lo_result = lo_sql->execute_query( lv_sql ).

* Read result into internal Table
     GET REFERENCE OF lt_cust_book INTO lr_data.
     lo_result->set_param_table( lr_data ).
     lo_result->next_package( ).
     lo_result->close( ).

   CATCH cx_sql_exception INTO lx_sql_exc.  " Exception Class for SQL Error
     lv_text = lx_sql_exc->get_text( ).
     MESSAGE lv_text TYPE 'E'.

  ENDTRY.

  SORT lt_cust_book BY id.

  LOOP AT lt_cust_book INTO ls_cust_book.

    IF sy-tabix = 1.
      ls_customer-id       = ls_cust_book-id.
      ls_customer-name     = ls_cust_book-name.
      ls_customer-postcode = ls_cust_book-postcode.
      ls_customer-city     = ls_cust_book-city.
      ls_customer-country  = ls_cust_book-country.

    ELSEIF ls_cust_book-id <> ls_customer-id.

      ls_customer-days_ahead = round( val = ls_customer-days_ahead / lv_count dec = 2 mode = cl
      INSERT ls_customer INTO TABLE ct_customers.

      CLEAR ls_customer.
```

Continued on next page

© 2013 SAP AG. All rights reserved.

```
      CLEAR lv_count.

      ls_customer-id       = ls_cust_book-id.
      ls_customer-name     = ls_cust_book-name.
      ls_customer-postcode = ls_cust_book-postcode.
      ls_customer-city     = ls_cust_book-city.
      ls_customer-country  = ls_cust_book-country.

    ENDIF.

    lv_count = lv_count + 1.
    ls_customer-days_ahead = ls_customer-days_ahead + ls_cust_book-fldate - ls_cust_book-or

  ENDLOOP.

  ls_customer-days_ahead = round( val = ls_customer-days_ahead / lv_count dec = 2 mode = cl
  INSERT ls_customer INTO TABLE ct_customers.

*  SORT ct_customers BY id. "already sorted


ENDFORM.                    "

*&---------------------------------------------------------------------*
*&      Form  get_data_solution
*&---------------------------------------------------------------------*
FORM get_data_solution CHANGING ct_customers TYPE ty_t_customers.

* Declarations
****************

** Types for target fields
*
*  TYPES: BEGIN OF lty_s_cust_book,
*           id          TYPE scustom-id,
*           name        TYPE scustom-name,
*           postcode    TYPE scustom-postcode,
*           city        TYPE scustom-city,
*           country     TYPE scustom-country,
*           fldate      TYPE sbook-fldate,
*           order_date  TYPE sbook-order_date,
*         END OF lty_s_cust_book.
*
** Work Area for Result
*  DATA ls_customer LIKE LINE OF ct_customers.
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
*
*
** Targets for Select
*  DATA: lt_cust_book   TYPE STANDARD TABLE OF lty_s_cust_book WITH NON-UNIQUE KEY id,
*        ls_cust_book TYPE lty_s_cust_book.
*
** help variables
*  DATA lv_count TYPE i.

* ADBC Objects and variables
  DATA:  lo_con     TYPE REF TO cl_sql_connection,
         lo_sql     TYPE REF TO cl_sql_statement,
         lo_result  TYPE REF TO cl_sql_result_set,
         lv_sql     TYPE string,
         lr_data    TYPE REF TO data.

* Eception handling
  DATA:   lx_sql_exc TYPE REF TO cx_sql_exception,
          lv_text TYPE string.

* Processing
****************************************
  CLEAR ct_customers.

  TRY.

* Get secondary DB Connection
      lo_con = cl_sql_connection=>get_connection( c_con ).

* Create statement
      CREATE OBJECT lo_sql
        EXPORTING
          con_ref = lo_con.

* create SQL statement
      CONCATENATE `SELECT C.ID, C.NAME, C.POSTCODE, C.CITY, C.COUNTRY, AVG( DAYS_BETWEEN(B.ORDE
                  `FROM SFLIGHT.SCUSTOM AS C INNER JOIN SFLIGHT.SBOOK AS B`
                  `ON C.MANDT = B.MANDT`
                  `AND C.ID = B.CUSTOMID`
                  `WHERE C.MANDT = `
                   sy-mandt
                  `AND B.CANCELLED = ' '`
                  `GROUP BY C.MANDT, C.ID, C.NAME, C.POSTCODE, C.CITY, C.COUNTRY`
                  `ORDER BY C.MANDT, C.ID `
             INTO lv_sql
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
            SEPARATED BY space.

* Alternative: Use string templates and string expressions
*      lv_sql = |SELECT C.ID, C.NAME, C.POSTCODE, C.CITY, C.COUNTRY AVG( DAYS_BETWEEN(B.ORD
*               |FROM SFLIGHT.SCUSTOM AS C INNER JOIN SFLIGHT.SBOOK AS B | &&
*               |ON C.MANDT = B.MANDT AND C.ID = B.CUSTOMID | &&
*               |WHERE C.MANDT = { sy-mandt } | &&
*               |AND B.CANCELLED = { SPACE } | &&
*               |GROUP BY C.MANDT, C.ID, C.NAME, C.POSTCODE, C.CITY, C.COUNTRY | &&
*               |ORDER BY C.MANDT, C.ID | .

* Execute Query
      lo_result = lo_sql->execute_query( lv_sql ).

* Read result into internal Table
*      GET REFERENCE OF lt_cust_book INTO lr_data.
      GET REFERENCE OF ct_customers INTO lr_data.
      lo_result->set_param_table( lr_data ).
      lo_result->next_package( ).
      lo_result->close( ).

    CATCH cx_sql_exception INTO lx_sql_exc.  " Exception Class for SQL Error
      lv_text = lx_sql_exc->get_text( ).
      MESSAGE lv_text TYPE 'E'.

  ENDTRY.

ENDFORM.                    "
```


© 2013 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Understand the main difference between native SQL Syntax and Open SQL Syntax
- Write syntactically correct Native SQL Statements

© 2013 SAP AG. All rights reserved.

Unit Summary

You should now be able to:

- Describe the use of native SQL in the context of SAP HANA
- Understand ABAP Database Connectivity (ADBC)
- Use ADBC to execute native SQL statements
- Understand the main difference between native SQL Syntax and Open SQL Syntax
- Write syntactically correct Native SQL Statements

© 2013 SAP AG. All rights reserved.

© 2013 SAP AG. All rights reserved.

Unit 4

Consuming HANA Views and Procedures in ABAP

Unit Overview

This Unit is designed to teach the following topics:

- Working with views in SAP HANA Studio
- Overview of different Types of SAP HANA views
- Consuming SAP HANA views in ABAP
- Calling SAP HANA procedures in ABAP

Unit Objectives

After completing this unit, you will be able to:

- Work with views in the SAP HANA Studio
- Describe the different Types of SAP HANA views
- Consume HANA views in ABAP
- Call HANA procedures in ABAP

Unit Contents

© 2013 SAP AG. All rights reserved.

Lesson: Working with Views in SAP HANA Studio

Lesson Overview

Lesson Objectives

After completing this lesson, you will be able to:

- Work with views in the SAP HANA Studio

Business Example

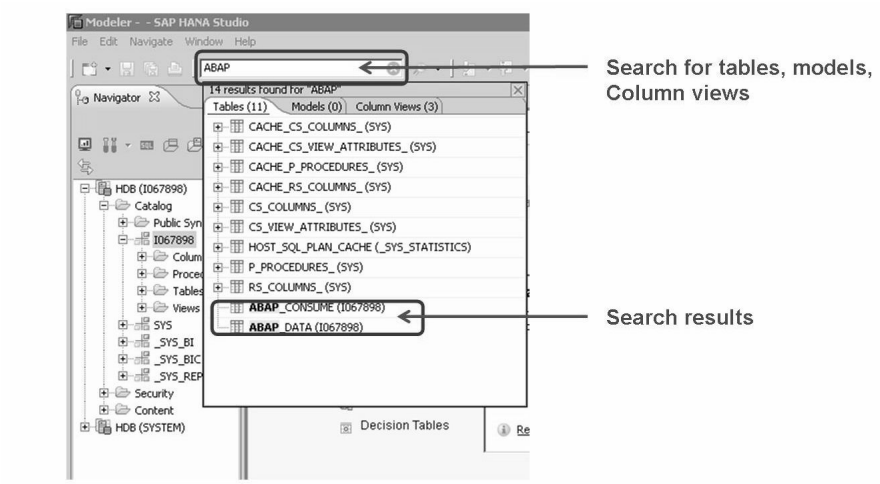


Figure 45: Search Functions – Global Search

© 2013 SAP AG. All rights reserved.

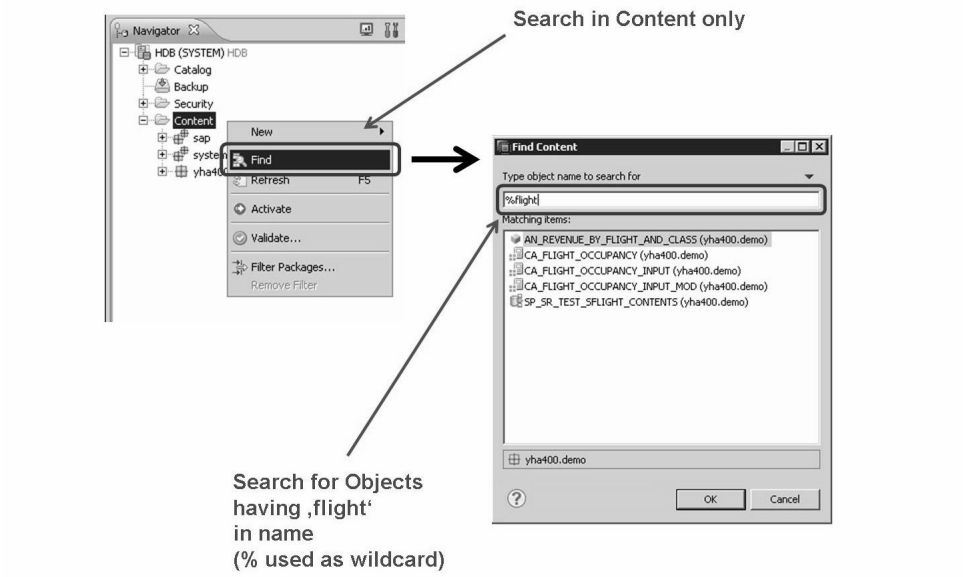


Figure 46: Search Functions – Local Search in Content

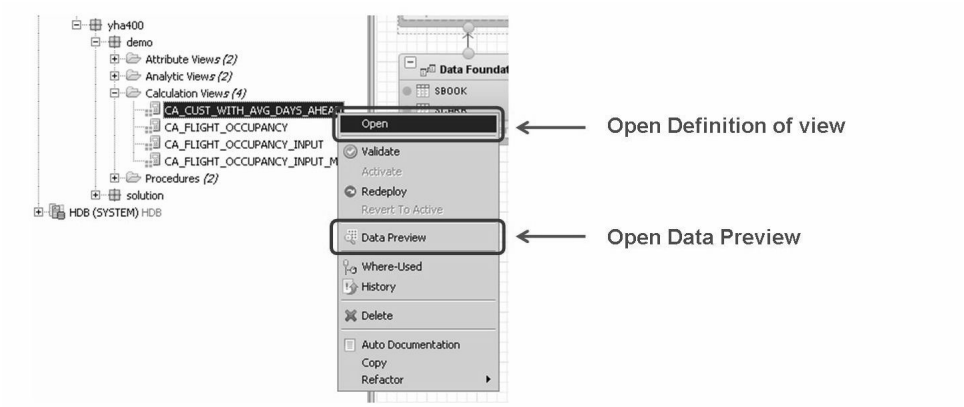


Figure 47: Working with Views – Open Definition and Data Preview

© 2013 SAP AG. All rights reserved.

Data Preview

Raw data (table display)

Number of distinct values per column

Interactive graphical analysis

Analysis | Distinct values | Raw Data

Filter pattern | 8 rows retrieved - 247 ms

MANDT	CARRID	CARRNAME	CONVAMNT_ECO	CONVAMNT_BUS	CONVAMNT_FST
800	AZ	Alitalia	111187850.2875 USD	19523996.0243 USD	19242729.4151 USD
800	AA	American Airlines	48810544.1982 USD	8312726.4284 USD	8026642.6461 USD
800	DL	Delta Airlines	66089854.345 USD	8932122.0856 USD	8986421.0585 USD
800	JL	Japan Airlines	54479530.8959 USD	9432761.2096 USD	8634445.8227 USD
800	LH	Lufthansa	164736733.9073 USD	25709558.7107 USD	25922882.1112 USD
800	QF	Qantas Airways	71528886.2674 USD	12216067.5458 USD	12360422.261 USD
800	SQ	Singapore Airlines	226780018.9817 USD	35765019.7321 USD	35909180.2172 USD
800	UA	United Airlines	135974402.4184 USD	19712323.9753 USD	19039672.5495 USD

Figure 48: Working with Views – Preview of Raw Data

Data Preview

Raw data (table display)

Number of distinct values per column

Interactive graphical analysis

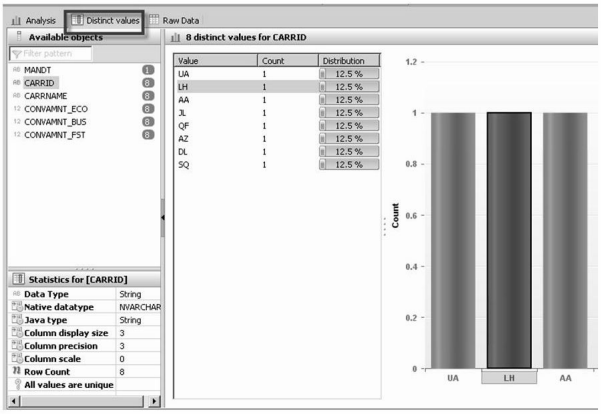


Figure 49: Working with Views – Analysis of Distinct Values

© 2013 SAP AG. All rights reserved.

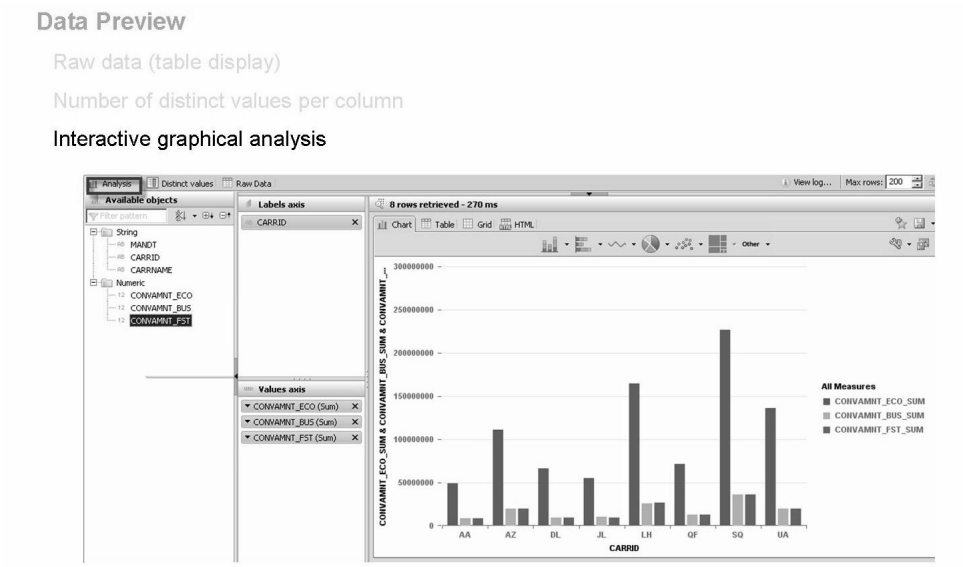


Figure 50: Working with Views – Graphical Analysis

© 2013 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Work with views in the SAP HANA Studio

© 2013 SAP AG. All rights reserved.

Lesson: Overview of Different Types of SAP HANA Views

Lesson Overview

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the different Types of SAP HANA views

Business Example

What is an Attribute View?

Used to join tables (e.g. text tables to master data tables)

Used as a projection (subset of fields)

May contain filters

May have columns with simple calculations

May be used in analytic or calculation views

Figure 51: Attribute View

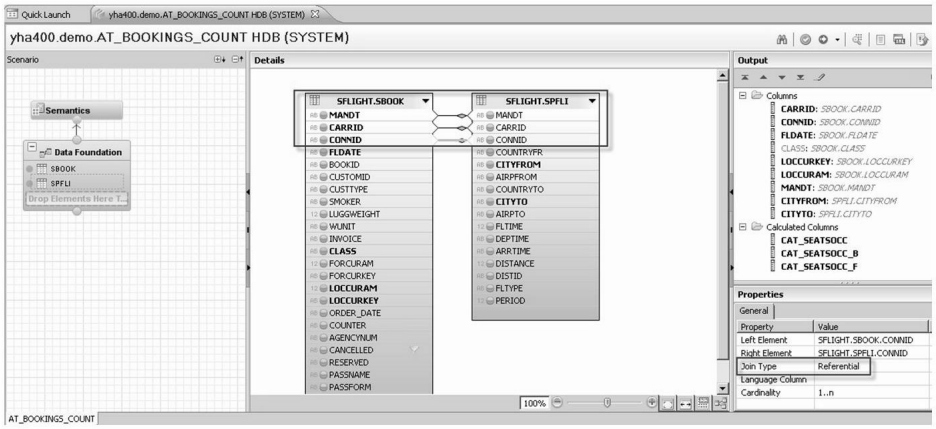


Figure 52: Example: Attribute View with Referential Join

© 2013 SAP AG. All rights reserved.

What is an Analytic View?

Used to join tables

Joins fact tables (data foundation) with attribute views

Data is read from the joined database tables

Joins and calculated measures are evaluated at run time

May contain more sophisticated calculations (e.g. currency conversion)

Figure 53: Analytic View

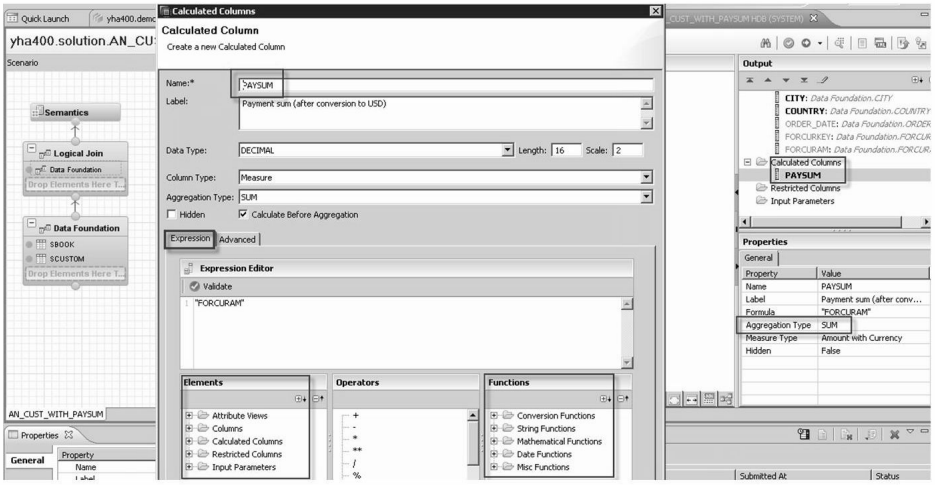


Figure 54: Example: Analytic View with Calculated Column

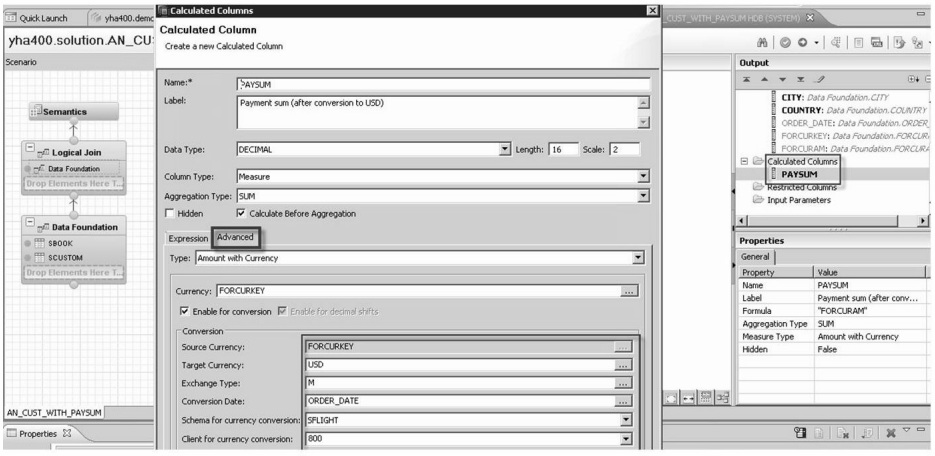


Figure 55: Example: Analytic View with Currency Conversion

© 2013 SAP AG. All rights reserved.

What is a calculation view?

can be created graphically or based on a script (SQL Script, CE Functions)

is designed for sophisticated analysis based on:

Functions defined in the HANA-specific language 'SQL Script'

Functions that contain SQL commands

SELECT <FIELDS> FROM <TABLE, VIEW or COLUMN VIEW> ...

SQL in functions must be 'read only' (no insert, update, delete, drop, ...)

Functions that call other functions

Modularize the logic within the calculation view

HANA offers pre-defined functions, e.g. for creating a join or union of tables

Figure 56: Calculation View

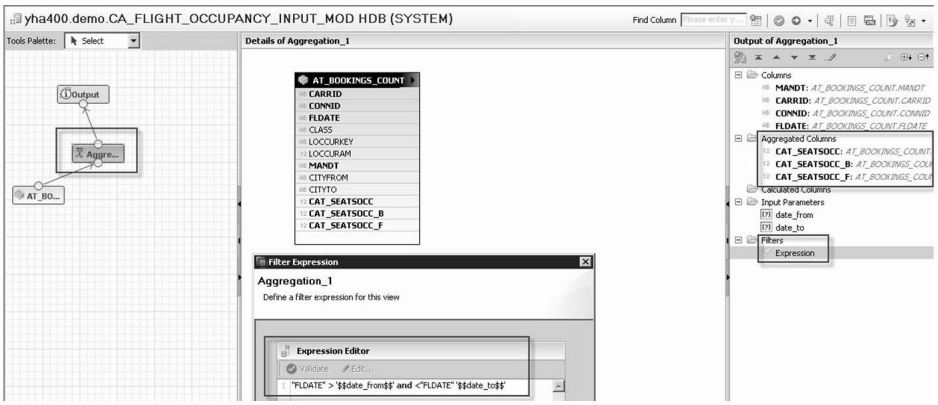


Figure 57: Example: Graphical Calculation View with Aggregated Columns and Filter

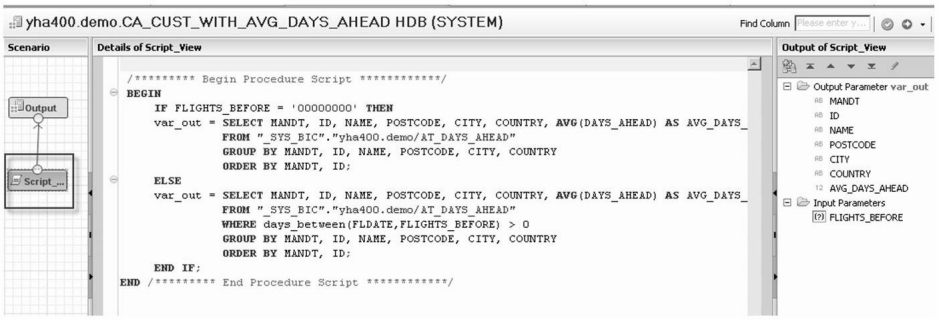


Figure 58: Example: Calculation View based on SQL Script

© 2013 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Describe the different Types of SAP HANA views

© 2013 SAP AG. All rights reserved.

Lesson: Consuming SAP HANA Views in ABAP

Lesson Overview

Lesson Objectives

After completing this lesson, you will be able to:

- Consume HANA views in ABAP

Business Example

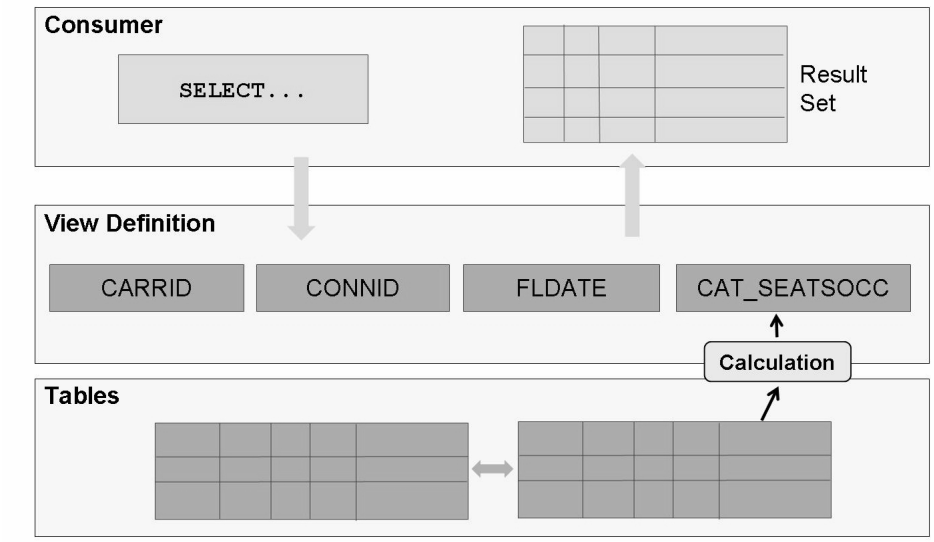


Figure 59: Consuming HANA View – Logical Picture

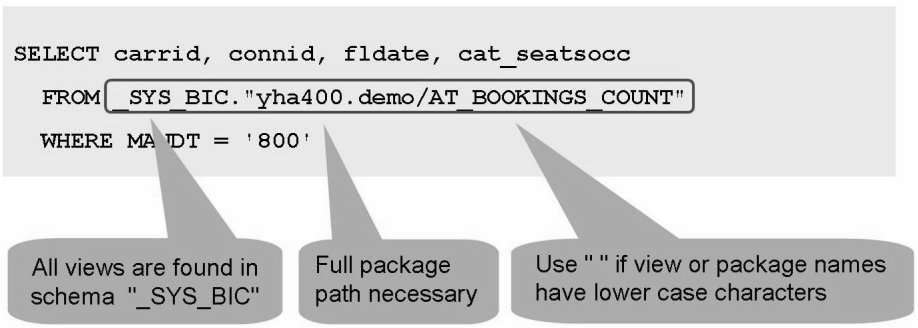


Figure 60: Consuming HANA Views – Native SQL Syntax

© 2013 SAP AG. All rights reserved.

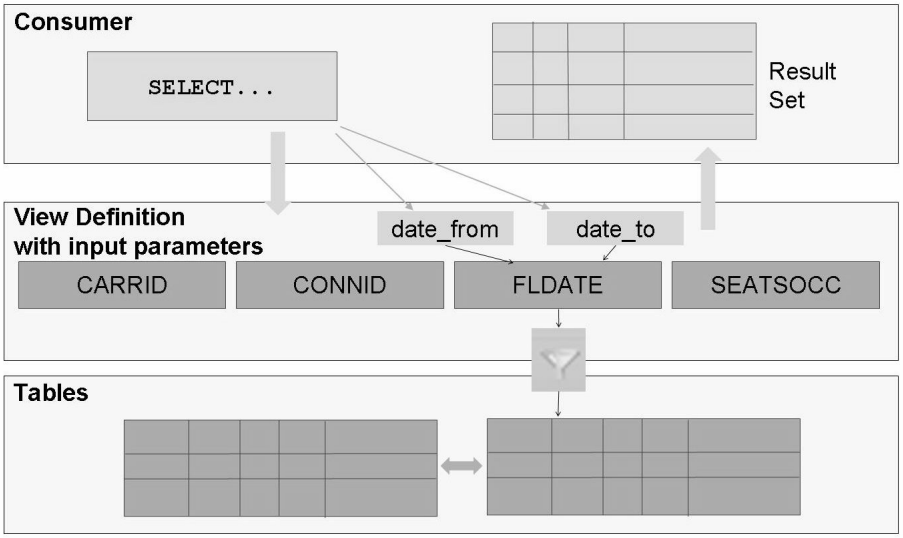


Figure 61: Consuming HANA Views with Input Parameters – Logical View

An Analytical View or Calculation View may have input parameters

These can be used for

- Setting a filter on a column or columns
- Specifying a reference currency or unit of measurement
- Passing other values to be used in calculations

Parameters can be optional, but may be mandatory

Parameters can have a default value, but will not necessarily have one

You pass parameter values to the HANA view immediately after the view name in the FROM clause

Figure 62: Input Parameters in HANA Views – The Facts

```
SELECT carrid, connid, fldate, seatsocc, seatsocc_b
  FROM _SYS_BIC."yha400.demo/CA_FLIGHT_OCUPANCY_INPUT_MOD"
  ('PLACEHOLDER'=('$$from_in$$','20121201')
   'PLACEHOLDER'=('$$to_in$$', '20130130'))
  WHERE MANDT = '800'
```

Parameter name in quotes and double dollar signs

Value

Figure 63: Consuming HANA Views with Input Parameters – Native SQL Syntax

© 2013 SAP AG. All rights reserved.

Exercise 9: Analyze a view defined on SAP HANA and use it in a native SQL SELECT

Exercise Objectives

After completing this exercise, you will be able to:

- analyze and understand views defined on SAP HANA
- use SAP HANA views in the FROM-clause of native SQL SELECTs

Business Example

Template:

Report YHA400_USE_HANA_VIEW**T1**

Solution:

Report YHA400_USE_HANA_VIEW_**S1**

Task 1: Copy and understand template

Create a copy of report YHA400_HANA_VIEW_T1 in your package ZHA400_## (suggested name: ZHA400_##_HVIEW_1, where ## is your group number). Activate and execute the program.

1. Create a copy of the report. Place it in your package ZHA400_00 and assign it to your workbench task.
2. Activate and execute the program.

Task 2: Find and analyze a SAP HANA View

In the SAP HANA Studio, search for a view that contains the relevant information from tables SCUSTOM and SBOOK, including a calculated field for the number of days between order date and flight date. The view should not provide any aggregation, yet. Check all relevant properties of the view, display the raw data and do a little analysis of the view's content with the data preview tool.

1. Go to the SAP HANA Studio and search the Content for views having "days" in their names.

Hint: Keep in mind that SAP HANA uses "%" as wildcard symbol (as common in SQL) rather than the "*" character known from SAP GUI selection screens.

Continued on next page

 © 2013 SAP AG. All rights reserved.

2. The list contains the following views:

3. Both views are located in package

4. According to naming conventions, which of the views can you expect to be an attribute view?

5. Choose the attribute view and analyze its definition.

6. View _____ is based on database tables _____ and _____. The tables are joined by a _____ join of cardinality _____. The view consists of __ fields from the left table, __ fields from the right table and 1 calculated field. The calculated field's name is _____. It's calculation expression reads as follows:
_____
The view contains one filter condition: Only such entries are selected where the field _____ contains the value ____. The views first __ fields are key fields. The field with name _____ is hidden.
Fill in the blanks to complete the sentence.

Continued on next page

© 2013 SAP AG. All rights reserved.

Task 3: Use the SAP HANA View in a native SQL Select

Edit your program. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*. In subroutine *get_data_solution*. Implement a native SQL SELECT that fills internal table *ct_customers* directly without any further processing. Select from the view you just found and implement the aggregation (calculation of average) in the SELECT statement.

1. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*
2. In subroutine *get_data_solution*, change the native SQL Statement. In the FROM-clause, replace the join of tables SCUSTOM and SBOOK with the view you just found.

Hint: Remember to state the view's name with it's complete path, i.e. _SYS_BIC/<package name>.<subpackage name> ... /<view name>

Caution: Names on SAP HANA are case sensitive. But everything not included in " " will be translated to upper case!

3. Adjust the field list of your SQL statement to make the field names correspond to the field names in the view. Define an aggregation of field *DAYS_AHEAD*, using aggregation function AVG( ) and add a GROUP BY addition that lists the client and all fields from the field list.
4. Remove field *CANCELLED* from the WHERE-clause as this filter condition is already part of the view's definition.
5. Replace target data object *lt_cust_book* with *ct_customers* .
6. Remove the now superfluous loop over lt_cust_book with the calculation and aggregation.
7. Activate and test your program. Make sure the two subroutines deliver exactly the same data.

© 2013 SAP AG. All rights reserved.

Solution 9: Analyze a view defined on SAP HANA and use it in a native SQL SELECT

Task 1: Copy and understand template

Create a copy of report YHA400_HANA_VIEW_T1 in your package ZHA400_## (suggested name: ZHA400_##_HVIEW_1, where ## is your group number). Activate and execute the program.

1. Create a copy of the report. Place it in your package ZHA400_00 and assign it to your workbench task.

   a) Complete this step as you learned to do in previous classes.

2. Activate and execute the program.

   a) Complete this step as you learned to do in previous classes.

Task 2: Find and analyze a SAP HANA View

In the SAP HANA Studio, search for a view that contains the relevant information from tables SCUSTOM and SBOOK, including a calculated field for the number of days between order date and flight date. The view should not provide any aggregation, yet. Check all relevant properties of the view, display the raw data and do a little analysis of the view's content with the data preview tool.

1. Go to the SAP HANA Studio and search the Content for views having "days" in their names.

   Hint: Keep in mind that SAP HANA uses "%" as wildcard symbol (as common in SQL) rather than the "*" character known from SAP GUI selection screens.

   a) In the Navigator window on the left side, right-click on node *Content* and choose *Find*.

   b) Enter the search string (e.g. "%days").

2. The list contains the following views:

   Answer:

   AT_DAYS_AHEAD

   CA_CUST_WITH_AVG_DAYS_AHEAD

3. Both views are located in package

   Answer: yha400.demo

Continued on next page

 © 2013 SAP AG. All rights reserved.

4. According to naming conventions, which of the views can you expect to be an attribute view?

   Answer: AT_DAYS_AHEAD, as AT stands for attribute view.

5. Choose the attribute view and analyze its definition.

   a) Double click the view in the search result or click on it and choose *OK*.

   b) Double click on *Data Foundation* to see a graphical display of the database tables, the join conditions, the fields of the output and their origin. Filter icons indicate fields where filters are in place.

   c) Double click on one of the join lines to see the join type and the cardinality.

   d) In the output-frame on the right, double click on the calculated field *DAYS_AHEAD* to see the calculation rule.

   e) Double click on *Semantics* to see a list of the output fields with their visibility and their classification as key attributes or normal attributes.

6. View AT_DAYS_AHEAD is based on database tables SBOOK and SCUSTOM. The tables are joined by a referential join of cardinality n..1. The view consists of 6 fields from the left table, 5 fields from the right table and 1 calculated field. The calculated field's name is days_ahead. It's calculation expression reads as follows: daysbetween(date("ORDER_DATE"), date("FLDATE")). The view contains one filter condition: Only such entries are selected where the field CANCELLED contains the value ' '. The views first 5 fields are key fields. The field with name ORDER_DATE is hidden.

   Answer: AT_DAYS_AHEAD, SBOOK, SCUSTOM, referential, n..1, 6, 5, days_ahead, daysbetween(date("ORDER_DATE"), date("FLDATE")), CANCELLED, ' ', 5, ORDER_DATE

Task 3: Use the SAP HANA View in a native SQL Select

Edit your program. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*. In subroutine *get_data_solution*. Implement a native SQL SELECT that fills internal table *ct_customers* directly without any further processing. Select from the view you just found and implement the aggregation (calculation of average) in the SELECT statement.

1. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*

   a) Use *Copy & Paste*-function of the ABAP Editor.

Continued on next page

 © 2013 SAP AG. All rights reserved.

2. In subroutine *get_data_solution*, change the native SQL Statement. In the FROM-clause, replace the join of tables SCUSTOM and SBOOK with the view you just found.

Hint: Remember to state the view's name with it's complete path, i.e. _SYS_BIC/<package name>.<subpackage name> ... /<view name>

Caution: Names on SAP HANA are case sensitive. But everything not included in " " will be translated to upper case!

   a) See source code extract from model solution.

3. Adjust the field list of your SQL statement to make the field names correspond to the field names in the view. Define an aggregation of field *DAYS_AHEAD*, using aggregation function AVG( ) and add a GROUP BY addition that lists the client and all fields from the field list.

   a) See source code extract from model solution.

4. Remove field *CANCELLED* from the WHERE-clause as this filter condition is already part of the view's definition.

   a) See source code extract from model solution.

5. Replace target data object *lt_cust_book* with *ct_customers* .

   a) See source code extract from model solution.

6. Remove the now superfluous loop over lt_cust_book with the calculation and aggregation.

   a) See source code extract from model solution.

7. Activate and test your program. Make sure the two subroutines deliver exactly the same data.

   a) Complete this step as you learned to do in previous classes.

Result

Source code extract from model solution (Program YHA400_USE_HANA_VIEW_S1)

```
*&---------------------------------------------------------------------*
*&      Form  get_data_template
*&---------------------------------------------------------------------*
FORM get_data_template CHANGING ct_customers TYPE ty_t_customers.
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
* Declarations
****************

* Types for target fields

  TYPES: BEGIN OF lty_s_cust_book,
           id          TYPE scustom-id,
           name        TYPE scustom-name,
           postcode    TYPE scustom-postcode,
           city        TYPE scustom-city,
           country     TYPE scustom-country,
           fldate      TYPE sbook-fldate,
           order_date  TYPE sbook-order_date,
         END OF lty_s_cust_book.

* Work Area for Result
  DATA ls_customer LIKE LINE OF ct_customers.


* Targets for Select
  DATA: lt_cust_book   TYPE STANDARD TABLE OF lty_s_cust_book WITH NON-UNIQUE KEY id,
        ls_cust_book TYPE lty_s_cust_book.

* help variables
  DATA lv_count TYPE i.

* ADBC Objects and variables
  DATA:  lo_con     TYPE REF TO cl_sql_connection,
         lo_sql     TYPE REF TO cl_sql_statement,
         lo_result  TYPE REF TO cl_sql_result_set,
         lv_sql     TYPE string,
         lr_data    TYPE REF TO data.

* Eception handling
  DATA:   lx_sql_exc TYPE REF TO cx_sql_exception,
          lv_text TYPE string.

* Processing
*******************************

  CLEAR ct_customers.

  TRY.
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
* Get secondary DB Connection
      lo_con = cl_sql_connection=>get_connection( c_con ).

* Create statement
      CREATE OBJECT lo_sql
        EXPORTING
          con_ref = lo_con.

* create SQL statement
      CONCATENATE `SELECT C.ID, C.NAME, C.POSTCODE, C.CITY, C.COUNTRY, B.FLDATE, B.ORDER_DATE
                  `FROM SFLIGHT.SCUSTOM AS C INNER JOIN SFLIGHT.SBOOK AS B`
                  `ON C.MANDT = B.MANDT`
                  `AND C.ID = B.CUSTOMID`
                  `WHERE C.MANDT = `
                  sy-mandt
                  `AND B.CANCELLED = ' '`
                  `ORDER BY C.MANDT, C.ID`
            INTO lv_sql
            SEPARATED BY space.

* Alternative: Use string templates and string expressions
*      lv_sql = |SELECT C.ID, C.NAME, C.POSTCODE, C.CITY, C.COUNTRY | &&
*               |FROM SFLIGHT.SCUSTOM AS C INNER JOIN SFLIGHT.SBOOK AS B | &&
*               |ON C.MANDT = B.MANDT AND C.ID = B.CUSTOMID | &&
*               |WHERE C.MANDT = { sy-mandt } | &&
*               |AND B.CANCELLED = { SPACE } | &&
*               |ORDER BY C.MANDT, C.ID |.

* Execute Query
      lo_result = lo_sql->execute_query( lv_sql ).

* Read result into internal Table
      GET REFERENCE OF lt_cust_book INTO lr_data.
      lo_result->set_param_table( lr_data ).
      lo_result->next_package( ).
      lo_result->close( ).

    CATCH cx_sql_exception INTO lx_sql_exc.  " Exception Class for SQL Error
      lv_text = lx_sql_exc->get_text( ).
      MESSAGE lv_text TYPE 'E'.

  ENDTRY.

  SORT lt_cust_book BY id.
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
  LOOP AT lt_cust_book INTO ls_cust_book.

    IF sy-tabix = 1.
      ls_customer-id       = ls_cust_book-id.
      ls_customer-name     = ls_cust_book-name.
      ls_customer-postcode = ls_cust_book-postcode.
      ls_customer-city     = ls_cust_book-city.
      ls_customer-country  = ls_cust_book-country.

    ELSEIF ls_cust_book-id <> ls_customer-id.

      ls_customer-days_ahead = round( val = ls_customer-days_ahead / lv_count dec = 2 mode
      INSERT ls_customer INTO TABLE ct_customers.

      CLEAR ls_customer.
      CLEAR lv_count.

      ls_customer-id       = ls_cust_book-id.
      ls_customer-name     = ls_cust_book-name.
      ls_customer-postcode = ls_cust_book-postcode.
      ls_customer-city     = ls_cust_book-city.
      ls_customer-country  = ls_cust_book-country.

    ENDIF.

    lv_count = lv_count + 1.
    ls_customer-days_ahead = ls_customer-days_ahead + ls_cust_book-fldate - ls_cust_book-or

  ENDLOOP.

  ls_customer-days_ahead = round( val = ls_customer-days_ahead / lv_count dec = 2 mode = cl
  INSERT ls_customer INTO TABLE ct_customers.

*  SORT ct_customers BY id. "already sorted

ENDFORM.                    "

*&---------------------------------------------------------------------*
*&      Form  get_data_solution
*&---------------------------------------------------------------------*
FORM get_data_solution CHANGING ct_customers TYPE ty_t_customers.

* Declarations
*****************************
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
* Work Area for Result
  DATA ls_customer LIKE LINE OF ct_customers.

* ADBC Objects and variables
  DATA: lo_con     TYPE REF TO cl_sql_connection,
        lo_sql     TYPE REF TO cl_sql_statement,
        lo_result  TYPE REF TO cl_sql_result_set,
        lv_sql     TYPE string,
        lr_data    TYPE REF TO data.

* Eception handling
  DATA:   lx_sql_exc TYPE REF TO cx_sql_exception,
          lv_text TYPE string.

* Processing
*******************************

  CLEAR ct_customers.

  TRY.

      lo_con = cl_sql_connection=>get_connection( c_con ).

      CREATE OBJECT lo_sql
        EXPORTING
          con_ref = lo_con.

* create SQL statement
      lv_sql = |SELECT ID, NAME, POSTCODE, CITY, COUNTRY, AVG(DAYS_AHEAD) AS DAYS_AHEAD | &&
               |FROM _SYS_BIC."yha400.demo/AT_DAYS_AHEAD"  | &&
               |WHERE MANDT = { sy-mandt } | &&
               |GROUP BY MANDT, ID, NAME, POSTCODE, CITY, COUNTRY | &&
               |ORDER BY MANDT, ID |.

      lo_result = lo_sql->execute_query( lv_sql ).

      GET REFERENCE OF ct_customers INTO lr_data.
      lo_result->set_param_table( lr_data ).
      lo_result->next_package( ).
      lo_result->close( ).

    CATCH cx_sql_exception INTO lx_sql_exc.  " Exception Class for SQL Error
      lv_text = lx_sql_exc->get_text( ).
      MESSAGE lv_text TYPE 'E'.
  ENDTRY.
```

Continued on next page

© 2013 SAP AG. All rights reserved.

```
ENDFORM.                    "
```


© 2013 SAP AG. All rights reserved.

© 2013 SAP AG. All rights reserved.

Exercise 10: Call a SAP HANA view with an input parameter in a native SQL SELECT

Exercise Objectives

After completing this exercise, you will be able to:

- analyze and understand scripted calculation views defined on SAP HANA
- know how to provide values for input parameters of SAP HANA views

Business Example

Template:

Report YHA400_USE_HANA_VIEW**T2**

Solution:

Report YHA400_USE_HANA_VIEW_**S2**

Task 1: Copy and understand template

Create a copy of report YHA400_HANA_VIEW_T2 in your package ZHA400_## (suggested name: ZHA400_##_HVIEW_2, where ## is your group number). Activate and execute the program.

1. Create a copy of the report. Place it in your package ZHA400_00 and assign it to your workbench task.
2. Analyze the source code of subroutine *get_data_template*. Pay special attention to the WHERE-clause of the SELECT statement.
3. What is different to the programs used in the previous exercises?

   _____
   _____
   _____
   _____

4. Activate and execute the program.

Continued on next page

 © 2013 SAP AG. All rights reserved.

Task 2: Analyze a scripted calculation view

In the SAP HANA Studio, analyze the definition of the scripted calculation view CA_CUST_WITH_AVG_DAYS_AHEAD. Check all relevant properties of the view, display the raw data and do a little analysis of the view's content with the data preview tool.

1. Go to the SAP HANA Studio, search for view CA_CUST_WITH_AVG_DAYS_AHEAD and open its definition.
2. Analyze the views definition.
3. View CA_CUST_WITH_AVG_DAYS_AHEAD is a scripted calculation view. The script has ___ input parameter named _____. The script's output parameter *var_out* has ___ components. The script first decides, _____
   All data are selected from _____.
   The Views output consists of ___ fields, all of which are mapped to the components of _____..
   One of the view fields is classified as _____, the rest are _____.
   Fill in the blanks to complete the sentence.

Task 3: Use the SAP HANA View in a native SQL Select

Edit your program. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*. In subroutine *get_data_solution*. Replace the native SQL SELECT from AT_DAYS_AHEAD with a select from calculation view CA_CUST_WITH_AVG_DAYS_AHEAD. Make sure to supply the view's input parameter *FLIGHTS_BEFORE* with the current date.

1. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*
2. In subroutine *get_data_solution*, change the native SQL Statement. In the FROM-clause, replace the attribute view with the calculation view you just found.
3. Adjust the field list of your SQL statement to match the field names of the calculation view. Because the aggregation is already done in the view's script, remove the aggregation function AVG( ) and the GROUP BY addition. Also remove the ORDER BY addition as it too is part of the script inside the view.

Continued on next page

 © 2013 SAP AG. All rights reserved.

4. Supply the views input parameter with the current date from system field (*sy-datum*)

Hint: The HANA native SQL syntax is ('PLACE-HOLDER'=('$$FLIGHTS_BEFORE$$','<value>')), where <value> stands for the current date in internal format.

5. Activate and test your program. Make sure the two subroutines deliver exactly the same data.

© 2013 SAP AG. All rights reserved.

Solution 10: Call a SAP HANA view with an input parameter in a native SQL SELECT

Task 1: Copy and understand template

Create a copy of report YHA400_HANA_VIEW_T2 in your package ZHA400_## (suggested name: ZHA400_##_HVIEW_2, where ## is your group number). Activate and execute the program.

1. Create a copy of the report. Place it in your package ZHA400_00 and assign it to your workbench task.

   a) Complete this step as you learned to do in previous classes.

2. Analyze the source code of subroutine *get_data_template*. Pay special attention to the WHERE-clause of the SELECT statement.

   a) Complete this step as you learned to do in previous classes.

3. What is different to the programs used in the previous exercises?

   Answer: Only such bookings are taken into account where the flight date lies in the past.

4. Activate and execute the program.

   a) Complete this step as you learned to do in previous classes.

Task 2: Analyze a scripted calculation view

In the SAP HANA Studio, analyze the definition of the scripted calculation view CA_CUST_WITH_AVG_DAYS_AHEAD. Check all relevant properties of the view, display the raw data and do a little analysis of the view's content with the data preview tool.

1. Go to the SAP HANA Studio, search for view CA_CUST_WITH_AVG_DAYS_AHEAD and open its definition.

   a) Perform this step as before.

2. Analyze the views definition.

   a) Double click on *Script_ ...* to see the SQL script inside the view, the output parameter of the script and its input parameters.

   b) Double click on *Output* to see how the components of the script's input parameter are mapped to fields of the calculation view.

Continued on next page

 © 2013 SAP AG. All rights reserved.

3. View CA_CUST_WITH_AVG_DAYS_AHEAD is a scripted calculation view. The script has 1 input parameter named FLIGHTS_BEFORE. The script's output parameter *var_out* has 7 components. The script first decides, whether the input parameter contains a non-initial value or not. All data are selected from attribute view AT_DAYS_AHEAD. The Views output consists of 7 fields, all of which are mapped to the components of the scripts output parameter *var_out*.. One of the view fields is classified as Measure, the rest are Attributes.

   Answer: 1, FLIGHTS_BEFORE, 7, whether the input parameter contains a non-initial value or not, attribute view AT_DAYS_AHEAD, 7, the scripts output parameter *var_out*, Measure, Attributes

Task 3: Use the SAP HANA View in a native SQL Select

Edit your program. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*. In subroutine *get_data_solution*. Replace the native SQL SELECT from AT_DAYS_AHEAD with a select from calculation view CA_CUST_WITH_AVG_DAYS_AHEAD. Make sure to supply the view's input parameter *FLIGHTS_BEFORE* with the current date.

1. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*

   a) Use *Copy & Paste*-function of the ABAP Editor.

2. In subroutine *get_data_solution*, change the native SQL Statement. In the FROM-clause, replace the attribute view with the calculation view you just found.

   a) See source code extract from model solution.

3. Adjust the field list of your SQL statement to match the field names of the calculation view. Because the aggregation is already done in the view's script, remove the aggregation function AVG( ) and the GROUP BY addition. Also remove the ORDER BY addition as it too is part of the script inside the view.

   a) See source code extract from model solution.

4. Supply the views input parameter with the current date from system field (*sy-datum*)

   Hint: The HANA native SQL syntax is ('PLACE-HOLDER'=('$$FLIGHTS_BEFORE$$','<value>')), where <value> stands for the current date in internal format.

   a) See source code extract from model solution.

Continued on next page

 © 2013 SAP AG. All rights reserved.

5. Activate and test your program. Make sure the two subroutines deliver exactly the same data.

   a) Complete this step as you learned to do in previous classes.

Result

Source code extract from model solution (Program YHA400_USE_HANA_VIEW_S2)

```
*&---------------------------------------------------------------------*
*&      Form  get_data_template
*&---------------------------------------------------------------------*

FORM get_data_template CHANGING ct_customers TYPE ty_t_customers.

* Declarations
****************

* Work Area for Result
  DATA ls_customer LIKE LINE OF ct_customers.

* ADBC Objects and variables
  DATA: lo_con     TYPE REF TO cl_sql_connection,
        lo_sql     TYPE REF TO cl_sql_statement,
        lo_result  TYPE REF TO cl_sql_result_set,
        lv_sql     TYPE string,
        lr_data    TYPE REF TO data.

* Eception handling
  DATA:   lx_sql_exc TYPE REF TO cx_sql_exception,
          lv_text TYPE string.

* Processing
*******************************

  CLEAR ct_customers.

  TRY.

      lo_con = cl_sql_connection=>get_connection( c_con ).

      CREATE OBJECT lo_sql
        EXPORTING
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
          con_ref = lo_con.

* create SQL statement
      lv_sql = |SELECT ID, NAME, POSTCODE, CITY, COUNTRY, AVG(DAYS_AHEAD) AS DAYS_AHEAD | &
               |FROM _SYS_BIC."yha400.demo/AT_DAYS_AHEAD"  | &&
               |WHERE MANDT  = { sy-mandt } | &&
               |AND   FLDATE < { sy-datum } | &&
               |GROUP BY MANDT, ID, NAME, POSTCODE, CITY, COUNTRY | &&
               |ORDER BY MANDT, ID |.

      lo_result = lo_sql->execute_query( lv_sql ).

      GET REFERENCE OF ct_customers INTO lr_data.
      lo_result->set_param_table( lr_data ).
      lo_result->next_package( ).
      lo_result->close( ).

    CATCH cx_sql_exception INTO lx_sql_exc.  " Exception Class for SQL Error
      lv_text = lx_sql_exc->get_text( ).
      MESSAGE lv_text TYPE 'E'.
  ENDTRY.

ENDFORM.                    "

*&---------------------------------------------------------------------*
*&      Form  get_data_solution
*&---------------------------------------------------------------------*
FORM get_data_solution CHANGING ct_customers TYPE ty_t_customers.


* Declarations
****************

* Work Area for Result
  DATA ls_customer LIKE LINE OF ct_customers.

* ADBC Objects and variables
  DATA: lo_con     TYPE REF TO cl_sql_connection,
        lo_sql     TYPE REF TO cl_sql_statement,
        lo_result  TYPE REF TO cl_sql_result_set,
        lv_sql     TYPE string,
        lr_data    TYPE REF TO data.

* Eception handling
  DATA:   lx_sql_exc TYPE REF TO cx_sql_exception,
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
          lv_text TYPE string.

* Processing
*******************************

  CLEAR ct_customers.

  TRY.

      lo_con = cl_sql_connection=>get_connection( c_con ).

      CREATE OBJECT lo_sql
        EXPORTING
          con_ref = lo_con.

* create SQL statement
      lv_sql = |SELECT ID, NAME, POSTCODE, CITY, COUNTRY, AVG_DAYS_AHEAD AS DAYS_AHEAD | &&
               |FROM _SYS_BIC."yha400.demo/CA_CUST_WITH_AVG_DAYS_AHEAD"  | &&
               |('PLACEHOLDER'=('$$FLIGHTS_BEFORE$$','{ sy-datum }')) | &&
               |WHERE MANDT = { sy-mandt } |.

      lo_result = lo_sql->execute_query( lv_sql ).

      GET REFERENCE OF ct_customers INTO lr_data.
      lo_result->set_param_table( lr_data ).
      lo_result->next_package( ).
      lo_result->close( ).

    CATCH cx_sql_exception INTO lx_sql_exc.  " Exception Class for SQL Error
      lv_text = lx_sql_exc->get_text( ).
      MESSAGE lv_text TYPE 'E'.
  ENDTRY.

ENDFORM.                    "
```


© 2013 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Consume HANA views in ABAP

© 2013 SAP AG. All rights reserved.

Lesson: Calling SAP HANA Procedures in ABAP

Lesson Overview

Lesson Objectives

After completing this lesson, you will be able to:

- Call HANA procedures in ABAP

Business Example

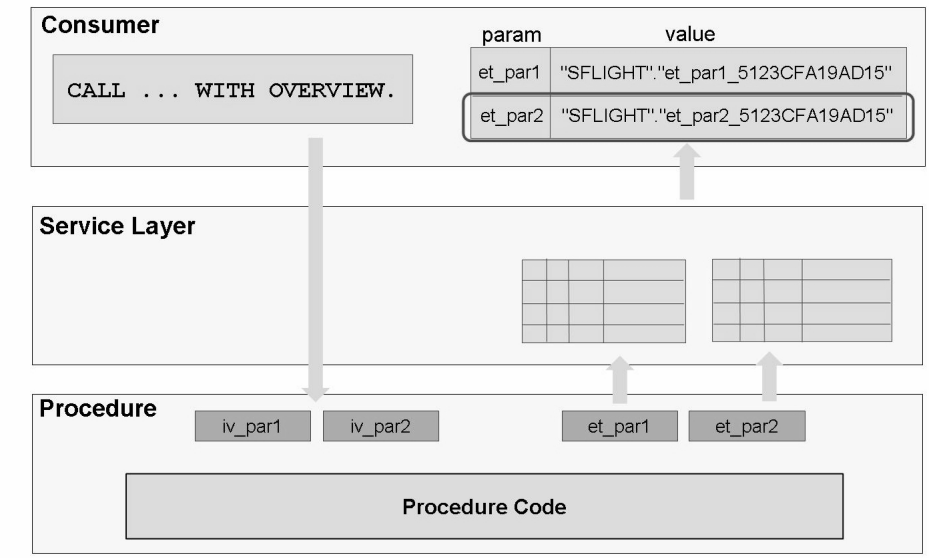


Figure 64: Calling HANA Procedures with Overview

© 2013 SAP AG. All rights reserved.

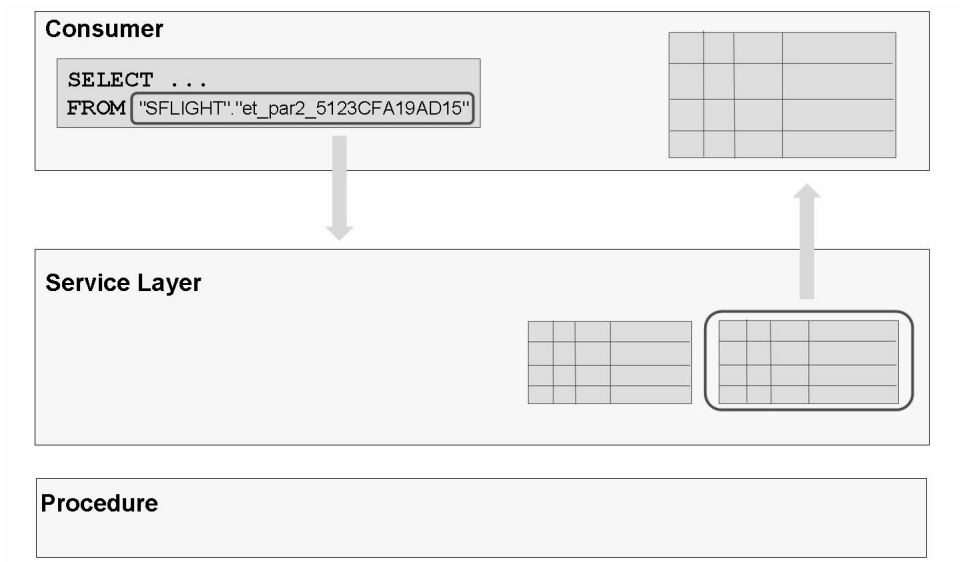


Figure 65: Calling HANA Procedures – Retrieve one Result Set

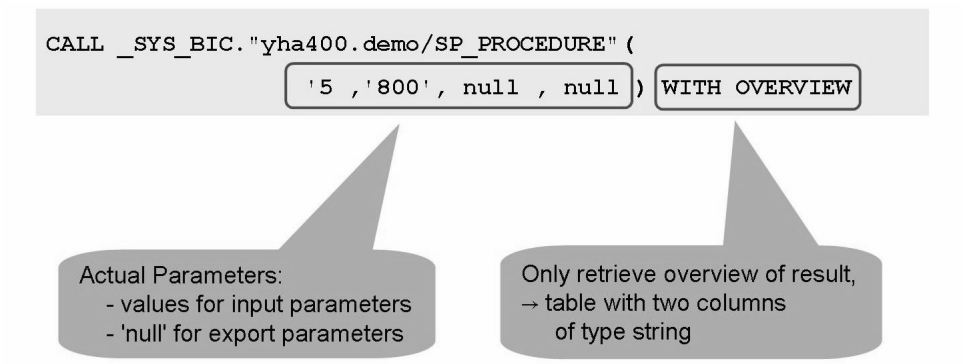


Figure 66: Calling HANA Procedures with Overview – Native SQL Syntax

© 2013 SAP AG. All rights reserved.

```
* type for result overview
  TYPES: BEGIN OF lty_s_overview,
           param   TYPE string,
           value   TYPE string,
         END OF lty_s_overview.


* Data objects for result overview
  DATA: lt_overview TYPE TABLE OF lty_s_overview,
        ls_overview TYPE lty_s_overview.


...


 lv_sql = `CALL _SYS_BIC."yha400.demo/SP_PROCEDURE( ` &&
          ` '5', '800', null, null ) WITH OVERVIEW `.

  lo_result = lo_sql->execute_query( lv_sql ).

  GET REFERENCE OF lt_overview INTO lr_flight.
  lo_result->set_param_table( lr_flight ).
  lo_result->next_package( ).

  lo_result->close( ).

   ...
```

Define internal table with two columns of type string for result overview

Call the procedure with overview in a query

Use overview table for query result

Figure 67: Coding Example: Calling HANA Procedure with Overview

```
...


  READ TABLE lt_overview INTO ls_overview
             WITH KEY param = 'ET_PAR2'.


  lv_sql =  `SELECT * FROM ` && ls_overview-value.


  lo_result = lo_sql->execute_query( lv_sql ).


  GET REFERENCE OF lt_par2 INTO lr_data.
  lo_result->set_param_table( lr_data ).
  lo_result->next_package( ).
  lo_result->close( ).
```

Ride single line from overview table

Use content of column *value* in FROM-clause of a query

Use data object matching the parameter definition

Figure 68: Coding Example: Retrieving Result from Overview

© 2013 SAP AG. All rights reserved.

Exercise 11: Analyze a stored procedure defined on SAP HANA and call it in a native SQL statement

Exercise Objectives

After completing this exercise, you will be able to:

- analyze and understand stored procedures defined on SAP HANA
- call SAP HANA stored procedures via native SQL

Business Example

Template:

Report YHA400_CALL_PROCEDURE_**T1**

Solution:

Report YHA400_CALL_PROCEDURE_**S1**

Task 1: Copy and understand template

Create a copy of report YHA400_CALL_PROCEDURE_T1 in your package ZHA400_## (suggested name: ZHA400_##_HPROC_1, where ## is your group number). Analyze the source code, activate and execute the program.

1. Create a copy of the report. Place it in your package ZHA400_00 and assign it to your workbench task.
2. Analyze the source code of subroutine *get_data_template*. Pay special attention to the changing parameters of the routine and how they are filled with data.
3. What is the main difference between this program and the ones we used before?

   _____
   _____
   _____
   _____

4. Activate and execute the program.

Continued on next page

 © 2013 SAP AG. All rights reserved.

Task 2: Analyze a SAP HANA stored procedure

In the SAP HANA Studio, analyze the definition of the stored procedure SP_EARLY_BIRD_AND_LAST_MINUTE. Check all relevant properties of the procedure.

1. Go to the SAP HANA Studio, search for procedure SP_EARLY_BIRD_AND_LAST_MINUTE and open its definition.
2. Analyze the procedure's definition.
3. Stored procedure SP_EARLY_BIRD_AND_LAST_MINUTE has ___ input parameters of name _____. They are used to _____
   The script has ___ output parameters named _____ and *ET_LAST_MINUTE*. The script contains two select statements which look almost identical. The only difference is _____
   Fill in the blanks to complete the sentence.

Task 3: Call the SAP HANA Procedure in a native SQL Statement

Edit your program. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*. In subroutine *get_data_solution*. Implement a native SQL Statement that calls the stored procedure. Use addition *WITH OVERVIEW* to receive first a list of name/value pairs for the export parameters. Using this list, retrieve the content of the export parameters via native SQL Selects from the database.

1. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*
2. In subroutine *get_data_solution*, replace the string with the native SQL SELECT with call of the stored procedure. Use addition *WITH OVERVIEW*.

Hint: The procedure expects four parameters separated by colons. The first two are input parameters. Provide suitable values for them (content of data objects *pv_number* and *sy-mandt*). The last two parameters correspond to the output parameters. they can be supplied with "null" when using addition *WITH OVERVIEW*.

Continued on next page

 © 2013 SAP AG. All rights reserved.

3. Define an internal table with two columns of type string. (suggested name for the table: *lt_overview*, suggested names for the columns: *param* and *value*). Replace target data object *ct_cust_late* with this new internal table..
4. After you closed the first query with the call of the procedure, use statement `READ TABLE` to read the line from *lt_overview* where column *param* contains the name of procedure parameter *ET_EARLY_BIRDS*. Use the content of column *value* in the FROM-clause of a native SQL SELECT. By that you read the content of the procedures export parameter from the database. Use the subroutines parameter *ct_cust_early* as target for this SELECT-statement.
5. After you closed this second query, use statement `READ TABLE` to read the line from *lt_overview* where column *param* contains the name of procedure parameter *ET_LAST_MINUTE*. Again, use the content of column *value* in the FROM-clause of a native SQL SELECT. But this time use the subroutines parameter *ct_cust_late* as the target for the SELECT-statement.
6. Activate and test your program. Use the debugger to analyze the content of the overview table and the retrieval of the procedures results. Make sure the two subroutines deliver exactly the same data.

© 2013 SAP AG. All rights reserved.

Solution 11: Analyze a stored procedure defined on SAP HANA and call it in a native SQL statement

Task 1: Copy and understand template

Create a copy of report YHA400_CALL_PROCEDURE_T1 in your package ZHA400_## (suggested name: ZHA400_##_HPROC_1, where ## is your group number). Analyze the source code, activate and execute the program.

1. Create a copy of the report. Place it in your package ZHA400_00 and assign it to your workbench task.

   a) Complete this step as you learned to do in previous classes.

2. Analyze the source code of subroutine *get_data_template*. Pay special attention to the changing parameters of the routine and how they are filled with data.

   a) Complete this step as you learned to do in previous classes.

3. What is the main difference between this program and the ones we used before?

   Answer: Instead of just one list of all customers two lists are gathered: One list of customers with the longest average time between order date and flight date and list of customers with the shortest average time between order date and flight date.

4. Activate and execute the program.

   a) Complete this step as you learned to do in previous classes.

Task 2: Analyze a SAP HANA stored procedure

In the SAP HANA Studio, analyze the definition of the stored procedure SP_EARLY_BIRD_AND_LAST_MINUTE. Check all relevant properties of the procedure.

1. Go to the SAP HANA Studio, search for procedure SP_EARLY_BIRD_AND_LAST_MINUTE and open its definition.

   a) Perform this step as before.

2. Analyze the procedure's definition.

   a) All relevant information is directly visible in the procedure definition.

Continued on next page

 © 2013 SAP AG. All rights reserved.

3. Stored procedure SP_EARLY_BIRD_AND_LAST_MINUTE has 2 input parameters of name *iv_number* and *iv_mandt*. They are used to restrict the client in the select statements of the procedure and to restrict the number of datasets that are selected. The script has 2 output parameters named *ET_EARLY_BIRDS* and *ET_LAST_MINUTE*. The script contains two select statements which look almost identical. The only difference is the sorting order (one sorts ascending, the other descending)

   Answer: 2, *iv_number* and *iv_mandt*, restrict the client in the select statements of the procedure and to restrict the number of datasets that are selected, 2, *ET_EARLY_BIRDS* , the sorting order (one sorts ascending, the other descending)

Task 3: Call the SAP HANA Procedure in a native SQL Statement

Edit your program. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*. In subroutine *get_data_solution*. Implement a native SQL Statement that calls the stored procedure. Use addition *WITH OVERVIEW* to receive first a list of name/value pairs for the export parameters. Using this list, retrieve the content of the export parameters via native SQL Selects from the database.

1. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*

   a) Use *Copy & Paste*-function of the ABAP Editor.

2. In subroutine *get_data_solution*, replace the string with the native SQL SELECT with call of the stored procedure. Use addition *WITH OVERVIEW*.

   Hint: The procedure expects four parameters separated by colons. The first two are input parameters. Provide suitable values for them (content of data objects *pv_number* and *sy-mandt*). The last two parameters correspond to the output parameters. they can be supplied with “null” when using addition *WITH OVERVIEW*.

   a) See source code extract from model solution.

3. Define an internal table with two columns of type string. (suggested name for the table: *lt_overview*, suggested names for the columns: *param* and *value*). Replace target data object *ct_cust_late* with this new internal table..

   a) See source code extract from model solution.

4. After you closed the first query with the call of the procedure, use statement `READ TABLE` to read the line from *lt_overview* where column *param* contains the name of procedure parameter *ET_EARLY_BIRDS*. Use the

Continued on next page

 © 2013 SAP AG. All rights reserved.

content of column *value* in the FROM-clause of a native SQL SELECT. By that you read the content of the procedures export parameter from the database. Use the subroutines parameter *ct_cust_early* as target for this SELECT-statement.

a) See source code extract from model solution.

5. After you closed this second query, use statement `READ TABLE` to read the line from *lt_overview* where column *param* contains the name of procedure parameter *ET_LAST_MINUTE*. Again, use the content of column *value* in the FROM-clause of a native SQL SELECT. But this time use the subroutines parameter *ct_cust_late* as the target for the SELECT-statement.

   a) See source code extract from model solution.

6. Activate and test your program. Use the debugger to analyze the content of the overview table and the retrieval of the procedures results. Make sure the two subroutines deliver exactly the same data.

   a) Complete this step as you learned to do in previous classes.

Result

Source code extract from model solution (Program YHA400_CALL_PROCEDURE_S1)

```
*&---------------------------------------------------------------------*
*&      Form  get_data_template
*&---------------------------------------------------------------------*

FORM get_data_template USING    pv_number TYPE i
                      CHANGING ct_cust_early TYPE ty_t_customers
                               ct_cust_late  TYPE ty_t_customers.

* Declarations
****************

* help variables
  DATA lv_count TYPE i.

* ADBC Objects and variables
  DATA:  lo_con     TYPE REF TO cl_sql_connection,
         lo_sql     TYPE REF TO cl_sql_statement,
         lo_result  TYPE REF TO cl_sql_result_set,
         lv_sql     TYPE string,
         lr_data    TYPE REF TO data.
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
* Eception handling
  DATA:   lx_sql_exc TYPE REF TO cx_sql_exception,
          lv_text TYPE string.

* Processing
*******************************

  CLEAR: ct_cust_early,
         ct_cust_late.

  TRY.

* Get secondary DB Connection
      lo_con = cl_sql_connection=>get_connection( c_con ).

* Create statement
      CREATE OBJECT lo_sql
        EXPORTING
          con_ref = lo_con.

* create SQL statement

      lv_sql =   |SELECT TOP { pv_number } | &&
                 |ID, NAME, POSTCODE, CITY, COUNTRY, AVG(DAYS_AHEAD) AS DAYS_AHEAD | &&
                 |FROM _SYS_BIC."yha400.demo/AT_DAYS_AHEAD"  | &&
                 |WHERE MANDT = { sy-mandt } | &&
                 |GROUP BY MANDT, ID, NAME, POSTCODE, CITY, COUNTRY | &&
                 |ORDER BY MANDT, DAYS_AHEAD DESC |.

* Execute Query
      lo_result = lo_sql->execute_query( lv_sql ).

* Read result into internal Table
      GET REFERENCE OF ct_cust_early INTO lr_data.
      lo_result->set_param_table( lr_data ).
      lo_result->next_package( ).
      lo_result->close( ).

* create second SQL statement
      lv_sql =   |SELECT TOP { pv_number } | &&
                 |ID, NAME, POSTCODE, CITY, COUNTRY, AVG(DAYS_AHEAD) AS DAYS_AHEAD | &&
                 |FROM _SYS_BIC."yha400.demo/AT_DAYS_AHEAD"  | &&
                 |WHERE MANDT = { sy-mandt } | &&
                 |GROUP BY MANDT, ID, NAME, POSTCODE, CITY, COUNTRY | &&
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
                |ORDER BY MANDT, DAYS_AHEAD ASC |.

* Execute Query
      lo_result = lo_sql->execute_query( lv_sql ).

* Read result into internal Table
      GET REFERENCE OF ct_cust_late INTO lr_data.
      lo_result->set_param_table( lr_data ).
      lo_result->next_package( ).
      lo_result->close( ).

    CATCH cx_sql_exception INTO lx_sql_exc.  " Exception Class for SQL Error
      lv_text = lx_sql_exc->get_text( ).
      MESSAGE lv_text TYPE 'E'.

  ENDTRY.

ENDFORM.                    "

*&---------------------------------------------------------------------*
*&      Form  get_data_solution
*&---------------------------------------------------------------------*
FORM get_data_solution USING    pv_number TYPE i
                       CHANGING ct_cust_early TYPE ty_t_customers
                                ct_cust_late TYPE ty_t_customers.

* Declarations
****************

* type for result overview
  TYPES: BEGIN OF lty_s_overview,
           param   TYPE string,
           value   TYPE string,
         END OF lty_s_overview.

* Data objects for result overview
  DATA: lt_overview TYPE TABLE OF lty_s_overview,
        ls_overview TYPE lty_s_overview.

* help variables
  DATA lv_count TYPE i.

* ADBC Objects and variables
  DATA:  lo_con     TYPE REF TO cl_sql_connection,
         lo_sql     TYPE REF TO cl_sql_statement,
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
        lo_result  TYPE REF TO cl_sql_result_set,
        lv_sql     TYPE string,
        lr_data    TYPE REF TO data.

* Eception handling
  DATA:   lx_sql_exc TYPE REF TO cx_sql_exception,
          lv_text TYPE string.

* Processing
*******************************

  CLEAR: ct_cust_early,
         ct_cust_late.

  TRY.

* Get secondary DB Connection
      lo_con = cl_sql_connection=>get_connection( c_con ).

* Create statement
      CREATE OBJECT lo_sql
        EXPORTING
          con_ref = lo_con.

* create SQL statement
      lv_sql =  |CALL _SYS_BIC."yha400.demo/SP_EARLY_BIRD_AND_LAST_MINUTE"( { pv_number } ,

* Execute Query
      lo_result = lo_sql->execute_query( lv_sql ).

* Read result overview into internal Table
      GET REFERENCE OF lt_overview INTO lr_data.
      lo_result->set_param_table( lr_data ).
      lo_result->next_package( ).
      lo_result->close( ).

* Retrieve the results

* 1. Parameter et_early_birds

      READ TABLE lt_overview INTO ls_overview WITH KEY param = 'ET_EARLY_BIRDS'.
      lv_sql = |SELECT * FROM { ls_overview-value } |.
      lo_result = lo_sql->execute_query( lv_sql ).
      GET REFERENCE OF ct_cust_early INTO lr_data.
      lo_result->set_param_table( lr_data ).
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
      lo_result->next_package( ).
      lo_result->close( ).


* 2. Parameter et_last_minute

      READ TABLE lt_overview INTO ls_overview WITH KEY param = 'ET_LAST_MINUTE'.
      lv_sql = |SELECT * FROM { ls_overview-value } |.
      lo_result = lo_sql->execute_query( lv_sql ).
      GET REFERENCE OF ct_cust_late INTO lr_data.
      lo_result->set_param_table( lr_data ).
      lo_result->next_package( ).
      lo_result->close( ).

    CATCH cx_sql_exception INTO lx_sql_exc.  " Exception Class for SQL Error
      lv_text = lx_sql_exc->get_text( ).
      MESSAGE lv_text TYPE 'E'.

  ENDTRY.

ENDFORM.                    "
```


© 2013 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Call HANA procedures in ABAP

© 2013 SAP AG. All rights reserved.

Unit Summary

You should now be able to:

- Work with views in the SAP HANA Studio
- Describe the different Types of SAP HANA views
- Consume HANA views in ABAP
- Call HANA procedures in ABAP

© 2013 SAP AG. All rights reserved.

Unit 5

Creating Analytical View

Unit Overview

This Unit is designed to teach the creation of Analytic Views

Unit Objectives

After completing this unit, you will be able to:

- Create an Analytic view

Unit Contents

© 2013 SAP AG. All rights reserved.

Lesson: Creating Analytic View

Lesson Overview

Lesson Objectives

After completing this lesson, you will be able to:

- Create an Analytic view

Business Example

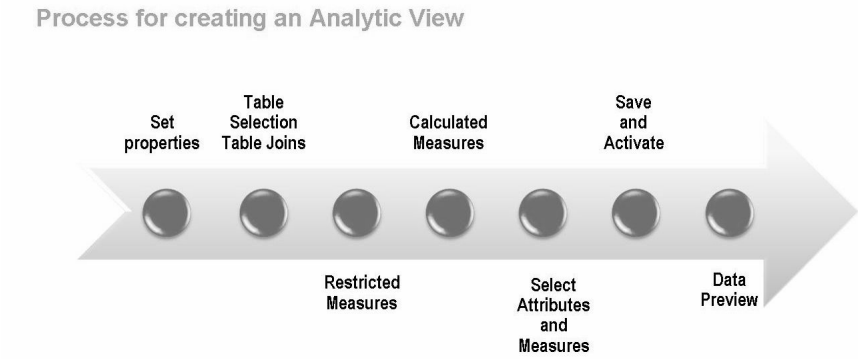


Figure 69: Creating Analytic View

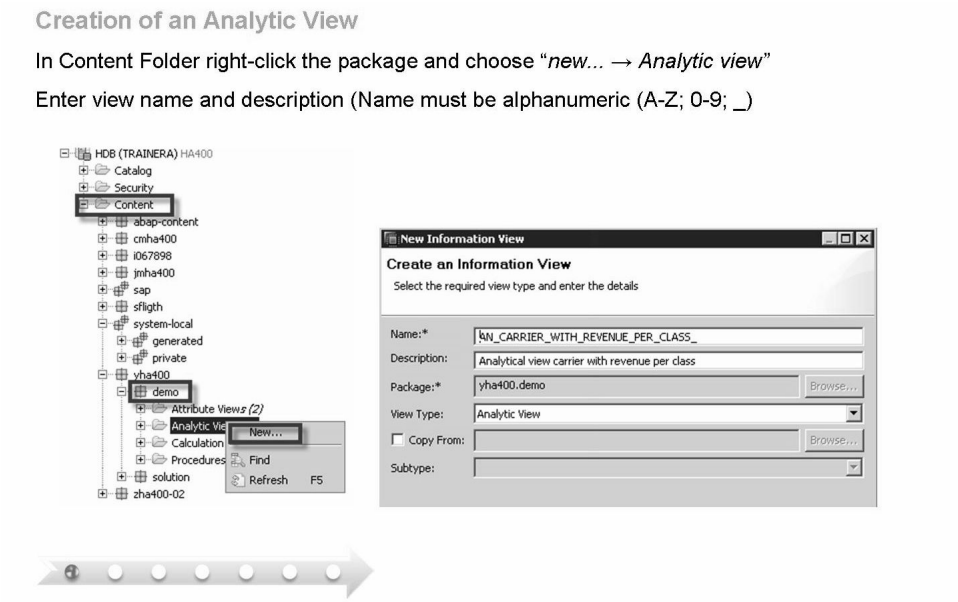


Figure 70: Creating Analytic View – Set Properties

© 2013 SAP AG. All rights reserved.

Data Foundation – Table Join

Select the relevant tables by adding them to the data foundation

Join the tables by connecting the relevant fields

Choose join type

Referential (default)

Inner, Left outer, Right Outer, Text Join

Adjust cardinality

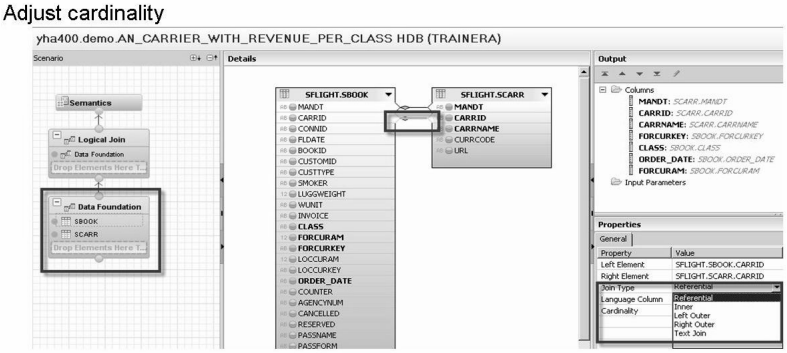


Figure 71: Creating Analytic View – Data Foundation – Table Join

Logical Join

join attribute views to the data foundation

Define restricted and calculated columns

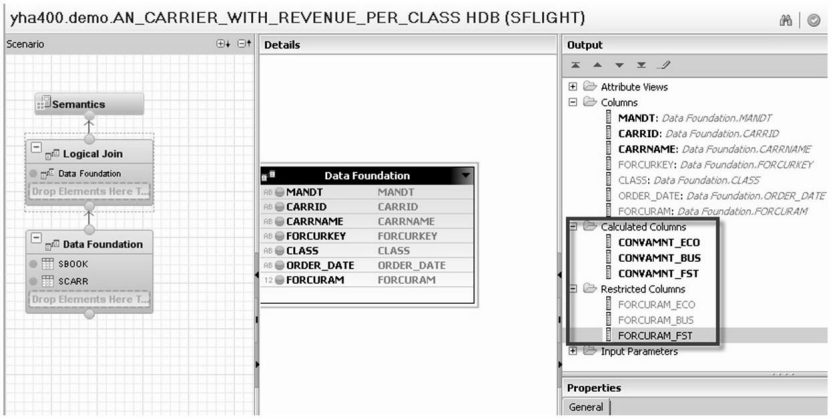


Figure 72: Creating Analytic View – Logical Join

© 2013 SAP AG. All rights reserved.

Create new restricted column

Choose source column

Add restriction(s)

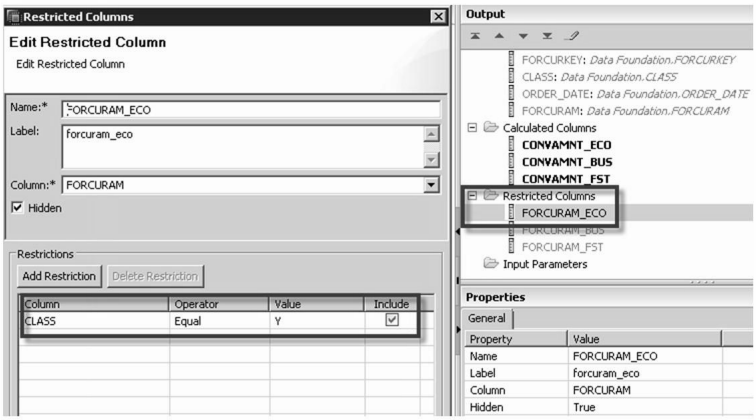


Figure 73: Example: Define Restricted Column

Create calculated column

Select Column Type Measure

Enter calculation expression

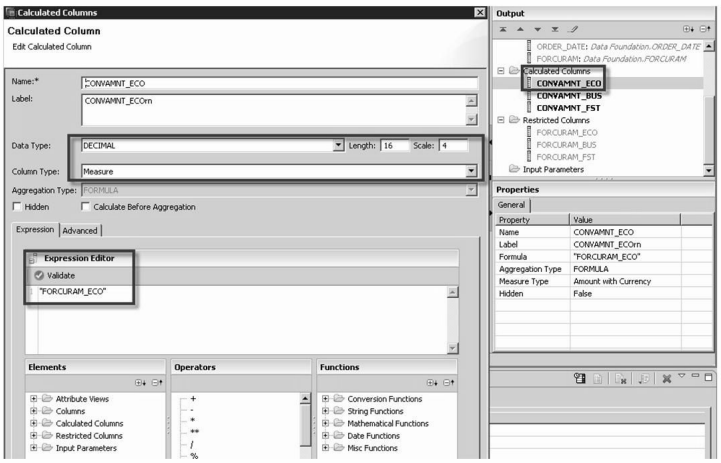


Figure 74: Example: Define Calculated Measure

© 2013 SAP AG. All rights reserved.

Semantics

Classify columns and calculated columns as attributes or measures

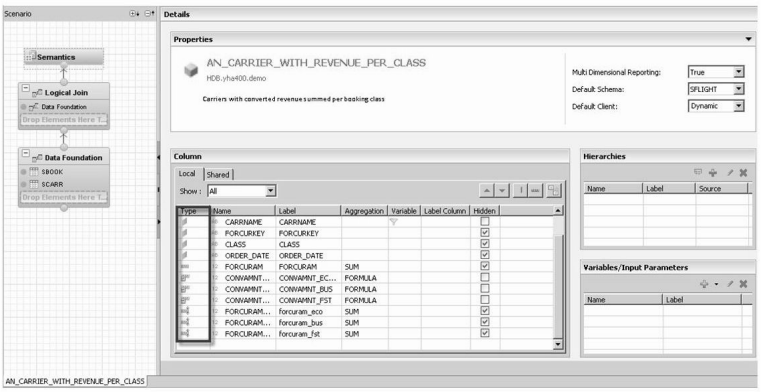


Figure 75: Creating Analytic View – Semantics

Save and Activate Analytic view

Check Status via Job Log

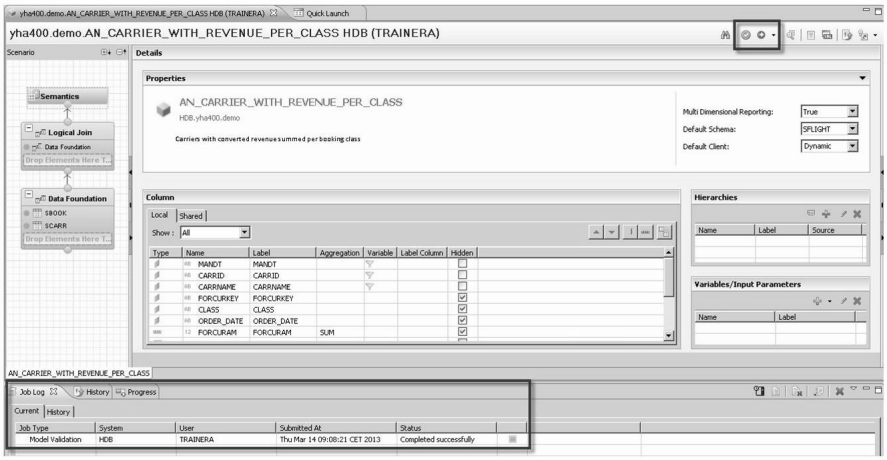


Figure 76: Creating Analytic View – Save and Activate

© 2013 SAP AG. All rights reserved.

Raw Data

Distinct values analysis

Graphical analysis

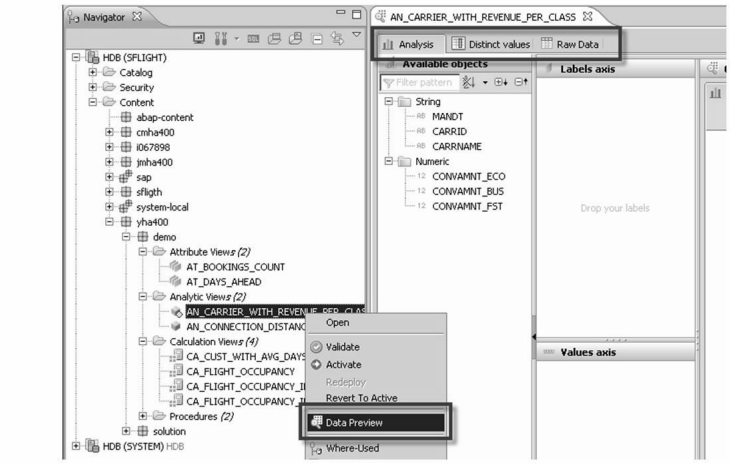


Figure 77: Creating Analytic View – Data Preview

© 2013 SAP AG. All rights reserved.

Exercise 12: Create a SAP HANA analytic view and use it in ABAP via a native SQL SELECT

Exercise Objectives

After completing this exercise, you will be able to:

- create an analytic view on SAP HANA
- understand and use the SAP HANA built-in currency conversion
- use SAP HANA analytic views in ABAP using ADBC and a native SQL SELECT

Business Example

Template:

Report YHA400_CREATE_HANA_VIEW**T1**

Solution:

Report YHA400_CREATE_HANA_VIEW_**S1**

Task 1: Copy and understand template

Create a copy of report YHA400_CREATE_HANA_VIEW_T1 in your package ZHA400_## (suggested name: ZHA400_##_CVIEW_1, where ## is your group number). Activate, analyze and execute the program.

1. Create a copy of the report. Place it in your package ZHA400_00 and assign it to your workbench task.
2. Analyze the source code of subroutine *get_data_template*.
3. What is calculated in the loop over the customers flight bookings? How is this done?

4. Activate and execute the program.

Continued on next page

 © 2013 SAP AG. All rights reserved.

Task 2: Create an analytic view

In the SAP HANA Studio, create a view that contains the complete analysis found in subroutine *get_data_template*. Make use of the fact that SAP HANA supports currency conversions and aggregations in analytic views.

1. Go to the SAP HANA Studio. In the Navigator, search for the content package you created in Exercise 1.
2. In your package, create a new analytic view (suggested name: *AN_CUST_WITH_PAYSUM*).
3. In the *Data Foundation*, define a referential join of tables SCUSTOM and SBOOK with a suitable join condition.
4. Add all required fields to the views output (See template program). Set a filter on column *CANCELLED* to make sure only non-cancelled bookings are considered.
5. In the *Logical Join* of the view, define a calculated column (suggested name: *PAYSUM_CONV*) of suitable type that converts the content of column *FORCURAM* to currency USD. Use the same exchange type (= *TYPE_OF_RATE*) and conversion date used for the function module in the template program.
6. In the *Semantics* of the view, declare the calculated column *PAYSUM* and the currency amount *FORCURAM* to be of type *Measure* and all other columns to be of type *Attribute*.
7. Back in the of the view, hide all columns from the view's output that are only needed as input for the currency conversion.
8. Save and Activate your view.

Task 3: Use the SAP HANA View in a native SQL Select

Edit your program. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*. In subroutine *get_data_solution*. Replace the two SELECTs, the nested LOOPs, the currency conversion and the aggregation by just one select from your analytic view.

1. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*
2. In subroutine *get_data_solution*, implement an ADBC call of a native SQL SELECT that reads data from the view you just created. Make the select read into data objects *ct_customers* directly.
3. Activate and test your program. Make sure the two subroutines deliver exactly the same data.
4. Perform a Runtime Measurement and Quantify the improvement you achieved.

© 2013 SAP AG. All rights reserved.

Solution 12: Create a SAP HANA analytic view and use it in ABAP via a native SQL SELECT

Task 1: Copy and understand template

Create a copy of report YHA400_CREATE_HANA_VIEW_T1 in your package ZHA400_## (suggested name: ZHA400_##_CVIEW_1, where ## is your group number). Activate, analyze and execute the program.

1. Create a copy of the report. Place it in your package ZHA400_00 and assign it to your workbench task.

   a) Complete this step as you learned to do in previous classes.

2. Analyze the source code of subroutine *get_data_template*.

   a) Complete this step as you learned to do in previous classes.

3. What is calculated in the loop over the customers flight bookings? How is this done?

   Answer: The payment in local currency is converted to US Dollar. This is done via function module CONVERT_TO_LOCAL_CURRENCY. The payments are then summed up per customer.

4. Activate and execute the program.

   a) Complete this step as you learned to do in previous classes.

Task 2: Create an analytic view

In the SAP HANA Studio, create a view that contains the complete analysis found in subroutine *get_data_template*. Make use of the fact that SAP HANA supports currency conversions and aggregations in analytic views.

1. Go to the SAP HANA Studio. In the Navigator, search for the content package you created in Exercise 1.

   a) Perform this step as before.

2. In your package, create a new analytic view (suggested name: *AN_CUST_WITH_PAYSUM*).

   a) Right click on the package and choose *New -> Analytic View...*.

   b) Enter the name of the view and a description, then press *Finish*

Continued on next page

 © 2013 SAP AG. All rights reserved.

3. In the *Data Foundation*, define a referential join of tables SCUSTOM and SBOOK with a suitable join condition.

   a) In the Navigator, open the schema and the node with the two tables.

   b) Use the *Drag & Drop* -function to add the tables to the data foundation.

   c) Use the *Drag & Drop*-function to join field *SCUSTOM.MANDT* with *SBOOK.MANDT* and *SCUSTOM.ID* with *SBOOK.CUSTOMID*.

   d) Click on one of the join lines and, on the lower right, check the join type and cardinality. Adjust this information if necessary.

4. Add all required fields to the views output (See template program). Set a filter on column *CANCELLED* to make sure only non-cancelled bookings are considered.

   a) Click on the grey bullet next to a table field to add it to the output (the bullet's color will then change to orange). The required fields are:

      - *SCUSTOM.MANDT*
      - *SCUSTOM.ID*
      - *SCUSTOM.NAME*
      - *SCUSTOM.POSTCODE*
      - *SCUSTOM.CITY*
      - *SCUSTOM.COUNTRY*
      - *SBOOK.ORDER_DATE*
      - *SBOOK.FORCURAM*
      - *SBOOK.FORCURKEY*

   b) Right click on field *SBOOK.CANCELLED* and choose *Apply filter* to set the filter.

Continued on next page

 © 2013 SAP AG. All rights reserved.

5. In the *Logical Join* of the view, define a calculated column (suggested name: *PAYSUM_CONV*) of suitable type that converts the content of column *FORCURAM* to currency USD. Use the same exchange type (= *TYPE_OF_RATE*) and conversion date used for the function module in the template program.

   a) In the *Scenario* window on the left, click on *Logical Join* to switch to the definition of the view logic.

   b) In the *Output* window on the right, right-click on node *Calculated Columns* and Choose *New...*

   c) Enter a name and a description for the calculated column and set the data type to *DECIMAL* with dimension = 16 and scale=4

   d) Change the column type from *Attribute* to *Measure* to allow conversion logic and aggregation.

   e) In the *Expression Editor*, just add the column to be converted (*FORCURAM*).

   f) Switch to tab *Advanced* to define the currency conversion rule. Set the type to *Amount with Currency* and choose *Enable for Conversion*.

   g) Left-click on the value-help button next to input field *Currency*, choose *Column* and assign the column with the source currency code (*FORCURKEY*).

   h) Enter Target Currency “USD” and Exchange Type “M”. For Conversion Date assign column *ODER_DATE*. Finally Set Schema for Conversion to *SFLIGHT* and Client for Currency Conversion to *Dynamic Client*.

   i) Finally, choose *Calculate before Aggregation* and set the Aggregation type to *SUM*. Then close the definition of the calculated column by clicking on *OK*.

6. In the *Semantics* of the view, declare the calculated column *PAYSUM* and the currency amount *FORCURAM* to be of type *Measure* and all other columns to be of type *Attribute*.

   a) In the *Scenario* window on the left, click on *Semantics*.

   b) On the list of columns edit the type of the columns.

7. Back in the of the view, hide all columns from the view’s output that are only needed as input for the currency conversion.

   a) Go back to the *Logical Join* of the view. In the *Output* window, double click on the column. In the *Properties* window below, change the value of property *Hidden* from *False* to *True*.

Continued on next page

 © 2013 SAP AG. All rights reserved.

8. Save and Activate your view.

   a) On the tool bar with the name of the view, choose button *Save and Activate*. Pay attention to the *Job Log* window on the bottom.

Task 3: Use the SAP HANA View in a native SQL Select

Edit your program. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*. In subroutine *get_data_solution*. Replace the two SELECTs, the nested LOOPs, the currency conversion and the aggregation by just one select from your analytic view.

1. Copy the source code of subroutine *get_data_template* to subroutine *get_data_solution*

   a) Use *Copy & Paste*-function of the ABAP Editor.

2. In subroutine *get_data_solution*, implement an ADBC call of a native SQL SELECT that reads data from the view you just created. Make the select read into data objects *ct_customers* directly.

   a) See source code extract from model solution.

3. Activate and test your program. Make sure the two subroutines deliver exactly the same data.

   a) Complete this step as you learned to do in previous classes.

4. Perform a Runtime Measurement and Quantify the improvement you achieved.

   a) Complete this step as you learned to do in previous classes.

Result

Source code extract from model solution (Program YHA400_CREATE_HANA_VIEW_S1)

```
*&---------------------------------------------------------------------*
*&      Form  get_data_template
*&---------------------------------------------------------------------*

FORM get_data_template CHANGING ct_customers TYPE ty_t_customers.

* Declarations
****************

* Types for target fields
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
  TYPES: BEGIN OF lty_s_cust,
           id          TYPE scustom-id,
           name        TYPE scustom-name,
           postcode    TYPE scustom-postcode,
           city        TYPE scustom-city,
           country     TYPE scustom-country,
         END OF lty_s_cust.

  TYPES: BEGIN OF lty_s_book,
              customid    TYPE sbook-customid,
              order_date  TYPE sbook-order_date,
              forcuram    TYPE sbook-forcuram,
             forcurkey    TYPE sbook-forcurkey,
        END OF lty_s_book.


* Work Area for Result
  DATA ls_customer LIKE LINE OF ct_customers.

* Targets for Select
  DATA: lt_scustom TYPE SORTED TABLE OF lty_s_cust WITH NON-UNIQUE KEY id,
        ls_scustom TYPE lty_s_cust,
        lt_sbook   TYPE SORTED TABLE OF lty_s_book WITH NON-UNIQUE KEY customid,
        ls_sbook   TYPE lty_s_book.

* help variables

  DATA lv_amnt_conv TYPE p LENGTH 16 DECIMALS 4.

* Processing
*****************

  CLEAR ct_customers.

  SELECT id name postcode city country
    FROM scustom
    CONNECTION (c_con)
    INTO TABLE lt_scustom
    ORDER BY id.

  SELECT customid order_date forcuram forcurkey
    FROM sbook
    CONNECTION (c_con)
    INTO TABLE lt_sbook
   WHERE cancelled = space
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
    ORDER BY customid.

  LOOP AT lt_scustom INTO ls_scustom.

    ls_customer-id       = ls_scustom-id.
    ls_customer-name     = ls_scustom-name.
    ls_customer-postcode = ls_scustom-postcode.
    ls_customer-city     = ls_scustom-city.
    ls_customer-country  = ls_scustom-country.

    CLEAR ls_customer-paysum.

    LOOP AT lt_sbook INTO ls_sbook
                   WHERE customid = ls_scustom-id.

      lv_amnt_conv = ls_sbook-forcuram.

      CALL FUNCTION 'CONVERT_TO_LOCAL_CURRENCY'
        EXPORTING
          date             = ls_sbook-order_date
          foreign_amount   = lv_amnt_conv
          foreign_currency = ls_sbook-forcurkey
          local_currency   = 'USD'
*         TYPE_OF_RATE     = 'M'
          read_tcurr       = 'X'
        IMPORTING
          local_amount     = lv_amnt_conv
        EXCEPTIONS
          no_rate_found    = 1
          overflow         = 2
          no_factors_found = 3
          no_spread_found  = 4
          derived_2_times  = 5.
      IF sy-subrc <> 0.
        MESSAGE 'Currency conversion failed' TYPE 'E'.
      ENDIF.
      ls_customer-paysum = ls_customer-paysum + lv_amnt_conv.

    ENDLOOP.

    IF ls_customer-paysum <> 0.
      INSERT ls_customer INTO TABLE ct_customers.
    ENDIF.
  ENDLOOP.
```

Continued on next page

© 2013 SAP AG. All rights reserved.

```
ENDFORM.                    "

*&---------------------------------------------------------------------*
*&      Form  get_data_solution
*&---------------------------------------------------------------------*
FORM get_data_solution CHANGING ct_customers TYPE ty_t_customers.

* ADBC Objects and variables
  DATA:  lo_con     TYPE REF TO cl_sql_connection,
         lo_sql     TYPE REF TO cl_sql_statement,
         lo_result  TYPE REF TO cl_sql_result_set,
         lv_sql     TYPE string,
         lr_data    TYPE REF TO data.

* Eception handling
  DATA:   lx_sql_exc TYPE REF TO cx_sql_exception,
          lv_text TYPE string.

* Processing
*****************

  CLEAR ct_customers.

  TRY.

* Get secondary DB Connection
      lo_con = cl_sql_connection=>get_connection( c_con ).

* Create statement
      CREATE OBJECT lo_sql
        EXPORTING
          con_ref = lo_con.

* create SQL statement
      CONCATENATE `SELECT ID, NAME, POSTCODE, CITY, COUNTRY, PAYSUM`
                  `FROM _SYS_BIC."yha400.solution/AN_CUST_WITH_PAYSUM"`
                  `WHERE MANDT = `
                  sy-mandt
                  'ORDER BY MANDT, ID '
            INTO lv_sql
            SEPARATED BY space.

* Alternative: Use string templates and string expressions
*      lv_sql = |SELECT ID, NAME, POSTCODE, CITY, COUNTRY, PAYSUMM | &&
*               |FROM _SYS_BIC."yha400.solution/AN_CUST_WITH_PAYSUM | &&
```

Continued on next page

 © 2013 SAP AG. All rights reserved.

```
*               |WHERE MANDT = { sy-mandt } | &&
*               |ORDER BY MANDT, ID |.

* Execute Query
      lo_result = lo_sql->execute_query( lv_sql ).

* Read result into internal Table
      GET REFERENCE OF ct_customers INTO lr_data.
      lo_result->set_param_table( lr_data ).
      lo_result->next_package( ).
      lo_result->close( ).

    CATCH cx_sql_exception INTO lx_sql_exc.  " Exception Class for SQL Error
      lv_text = lx_sql_exc->get_text( ).
      MESSAGE lv_text TYPE 'E'.

  ENDTRY.

ENDFORM.                    "
```


© 2013 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Create an Analytic view

© 2013 SAP AG. All rights reserved.

Unit Summary

You should now be able to:

- Create an Analytic view

© 2013 SAP AG. All rights reserved.

Course Summary

You should now be able to:

- Understand the Technical SAP HANA concepts
- Understand Optimization of classical ABAP in HANA Context
- Describe the usage of Analysis Tools (Runtime Analysis, Code Inspector, SQL Trace)
- Understand SQL Performance Rules of ABAP for HANA
- Explain Implementing ABAP report on HANA using ADBC (ABAP Data Base Connectivity)
- Explain Consuming HANA views in ABAP
- Explain Creating HANA views and Consuming in ABAP

© 2013 SAP AG. All rights reserved.

© 2013 SAP AG. All rights reserved.

Feedback

SAP AG has made every effort in the preparation of this course to ensure the accuracy and completeness of the materials. If you have any corrections or suggestions for improvement, please record them in the appropriate place in the course evaluation.

© 2013 SAP AG. All rights reserved.