HA400

ABAP Programming for SAP HANA

EXERCISES AND SOLUTIONS

Course Version: 18
Course Duration: 5 Hours
Material Number: 50160487

SAP Copyrights, Trademarks and Disclaimers

Typographic Conventions

American English is the standard used in this handbook.

The following typographic conventions are also used.

This information is displayed in the instructor's presentation	🖥
Demonstration	➤
Procedure	▼1 ▼2 ▼3
Warning or Caution	⚠
Hint	💡
Related or Additional Information	»
Facilitated Discussion	💬
User interface control	*Example text*
Window title	*Example text*

Contents

Unit 1
Exercise 1

Log On to ABAP Systems and Create Packages

Business Example

Your company has identified SAP HANA as an important strategic topic and has asked you to refresh your ABAP skills, paying particular attention to those required to develop or review code that leverages the strengths of SAP HANA.

Task 1: Log On to Application Server ABAP with SAP HANA as the Primary Database
Using SAP Logon, log on to the Application Server ABAP that uses SAP HANA as a primary database. Create a development package ZHA400_##, where ## stands for your group ID.

1. Log on to system T8N, which uses SAP HANA as its primary database.
2. Open system status and check the system's release and database.
3. Create ABAP package ZHA400_##.

Task 2: Log On to Other Application Server ABAP with Sybase as the Primary Database
Using SAP Logon, log on to the Application Server ABAP that does not have SAP HANA as a primary database. Use the system ID, user, and password provided by your instructor. Create a development package ZHA400_##, where ## stands for your group ID.

1. Log on to system ZME, which uses Sybase as its primary database.
2. Open the system status and check the system's release and database.
3. 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 (choose *Own Requests* to find and use this request).

Unit 1
Solution 1

Log On to ABAP Systems and Create Packages

Business Example

Your company has identified SAP HANA as an important strategic topic and has asked you to refresh your ABAP skills, paying particular attention to those required to develop or review code that leverages the strengths of SAP HANA.

Task 1: Log On to Application Server ABAP with SAP HANA as the Primary Database
Using SAP Logon, log on to the Application Server ABAP that uses SAP HANA as a primary database. Create a development package ZHA400_##, where ## stands for your group ID.

1. Log on to system T8N, which uses SAP HANA as its primary database.
   a) Choose *Start* → *SAP Logon* from the Microsoft Windows *Start* menu.
   b) Double-click the connection named *T8N[SPACE]*.
   c) Enter your user name and password for the ABAP system(s), as provided by your instructor, and select *Enter*.
2. Open system status and check the system's release and database.
   a) From the main menu bar, open *System* → → *Status*.
   b) In the *SAP System data* frame, choose *Details* to see the installed components and their versions.
   c) In the *Database data* frame, check the content of the *Database System* field.
3. Create ABAP package ZHA400_##.
   a) Start transaction `SE80`.
   b) Ensure the *Repository Browser* is selected in the left-hand frame, and that *Package* is chosen in the first list box.
   c) Enter the package name in the second input field and select *Return*.
   d) Confirm that you want to create the non-existing package.
   e) Enter a short description, application component **`CA`**, and software component **`HOME`**, and select *Enter*.
   f) When prompted for a transportable workbench request, use the workbench request already created by your instructor (choose *Own Requests* to find and use this request).

Task 2: Log On to Other Application Server ABAP with Sybase as the Primary Database
Using SAP Logon, log on to the Application Server ABAP that does not have SAP HANA as a primary database. Use the system ID, user, and password provided by your instructor. Create a development package ZHA400_##, where ## stands for your group ID.

1. Log on to system ZME, which uses Sybase as its primary database.
   a) Choose *Start* → *SAP Logon* from the Microsoft Windows *Start* menu.
   b) Double-click the connection named *ZME[SPACE]*.
   c) Enter your user name and password for the ABAP system(s) as provided by your instructor and select *Enter*.
2. Open the system status and check the system's release and database.
   a) Perform this step as before.
3. 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 (choose *Own Requests* to find and use this request).
   a) Perform this step as before.

Unit 1
Exercise 2

Log On to SAP HANA

Business Example

Your company has identified SAP HANA as an important strategic topic and has asked you to refresh your ABAP skills, paying particular attention to those required to develop or review code that leverages the strengths of SAP HANA.

Task 1: Log On to SAP HANA
Open the *SAP HANA Modeler* perspective that is included in *Eclipse.exe* and establish a connection to the SAP HANA server.

1. Open the *SAP HANA Modeler* perspective that is included in *Eclipse.exe*.
2. Add the SAP HANA system to the *Systems* view. Use the server name, user ID and password provided by your instructor.

Task 2: Open the Definition of a Table in SAP HANA Studio
Using the SAP HANA tools in *Eclipse*, open the definition of table SBOOK in schema SAPT8N and display its content.

1. Open *Eclipse*, switch to the *SAP HANA Modeler* perspective and log on to the SAP HANA system if you haven't already done so yet.
2. Expand schema SAPT8N and filter the tables of this schema to locate table SBOOK.
3. Open the definition of table SBOOK.

Task 3: Find and Open the Definition of an SAP HANA View
Using the SAP HANA tools in *Eclipse*, search for view CA_DAYS_AHEAD.

1. Search the SAP HANA content for the string "CA_DAYS".

   The search result contains the following view:

   ____

   ____

   ____

   The view is located in package:

   ____

   ____

   ____

2. Open the definition of the view.

Log On to SAP HANA

Business Example

Your company has identified SAP HANA as an important strategic topic and has asked you to refresh your ABAP skills, paying particular attention to those required to develop or review code that leverages the strengths of SAP HANA.

Task 1: Log On to SAP HANA

Open the *SAP HANA Modeler* perspective that is included in *Eclipse.exe* and establish a connection to the SAP HANA server.

1. Open the *SAP HANA Modeler* perspective that is included in *Eclipse.exe*.
   a) Choose *Start* → *All Programs* → → *Eclipse.exe* from the Microsoft Windows *Start* menu. Alternatively you can choose *Eclipse.exe* from the windows task bar.
   b) If the *Welcome* page is open, close the page or choose the *Workbench* link.
   c) Choose the *Open Perspective* button or menu item *Window* → *Perspective* → *Open Perspective* → *Other ...* , select item *SAP HANA Modeler* and choose *Open*.
2. Add the SAP HANA system to the *Systems* view. Use the server name, user ID and password provided by your instructor.
   a) Make sure you are in the *SAP HANA Modeler* perspective. If not, change the perspective using the buttons in the upper right-hand corner.
   b) Right-click the area in the *Systems* window on the left and choose *Add System*.
   c) Enter the host name and instance number provided by your instructor, and the description **`SAP HANA`**.
   d) In the *Locale* dropdown list, choose *English (United States)*.
   e) Choose *Next*.
   f) Enter the user ID and password provided by your instructor, then choose *Finish*.

Task 2: Open the Definition of a Table in SAP HANA Studio

Using the SAP HANA tools in *Eclipse*, open the definition of table SBOOK in schema SAPT8N and display its content.

1. Open *Eclipse*, switch to the *SAP HANA Modeler* perspective and log on to the SAP HANA system if you haven't already done so yet.
   a) Open *Eclipse* if it is not open yet.
   b) Switch to the *SAP HANA Modeler* perspective.
   c) Log on to the SAP HANA system using the connection prepared earlier in this exercise.

2. Expand schema SAPT8N and filter the tables of this schema to locate table SBOOK.
   a) In the *Navigator* window on the left side, expand *Catalog* → → *SAPT8N*.
   b) Right-click subnode *Tables* and choose *Filters ...*.
   c) Enter the search string **`SBOOK`** and choose *Okay*.
3. Open the definition of table SBOOK.
   a) Expand node *Tables* and double-click table *SBOOK*.

Task 3: Find and Open the Definition of an SAP HANA View
Using the SAP HANA tools in *Eclipse*, search for view CA_DAYS_AHEAD.

1. Search the SAP HANA content for the string "CA_DAYS".
   a) In the Navigator window on the left side, right-click the connection's node *Content* and choose *Find*.
   b) Enter the search string "CA_DAYS".

   The search result contains the following view:

   CA_DAYS_AHEAD

   The view is located in package:

   ha400.primdb
2. Open the definition of the view.
   a) Double-click the view *CA_DAYS_AHEAD (ha400.primdb)* in the search results, or select it and choose *OK*.

Analyze SAP HANA-specific settings in ABAP Dictionary

Business Example

Your company has recognized SAP HANA as an important strategic topic and has asked you to refresh your ABAP skills, paying particular attention to those required to develop or review code that leverages the strengths of SAP HANA.

Analyze Transparent Table SBOOK

On application server T8N, which uses SAP HANA as a primary database, analyze the DB-specific settings and secondary indexes of transparent table SBOOK.

1. If you are not already logged on, log on to system T8N, which uses SAP HANA as its primary database.
2. In the *Object Browser* (transaction `SE80`), open the definition of transparent table SBOOK.
3. Open the DB-specific properties of this transparent table.

   Which of SAP HANA's table storage types is used for table SBOOK?

4. Inspect the secondary indexes defined for transparent table SBOOK.

   How many secondary indexes are defined for transparent table SBOOK?

   Which of the indexes are generated when the ABAP system is deployed on an SAP HANA database, and which are not created on SAP HANA?

Analyze SAP HANA-specific settings in ABAP Dictionary

Business Example

Your company has recognized SAP HANA as an important strategic topic and has asked you to refresh your ABAP skills, paying particular attention to those required to develop or review code that leverages the strengths of SAP HANA.

Analyze Transparent Table SBOOK
On application server T8N, which uses SAP HANA as a primary database, analyze the DB-specific settings and secondary indexes of transparent table SBOOK.

1. If you are not already logged on, log on to system T8N, which uses SAP HANA as its primary database.
   a) Perform this step as in previous exercises.
2. In the *Object Browser* (transaction `SE80`), open the definition of transparent table SBOOK.
   a) Choose *Edit Object* or select *Shift* + *F5*, enter the name of the table definition and double-click *Database Table* in the following dialog box.
3. Open the DB-specific properties of this transparent table.
   a) Open *Technical Settings* and go to the *DB-Specific Properties* tab.

   Which of SAP HANA's table storage types is used for table SBOOK?

   The storage type is *Column Store*.
4. Inspect the secondary indexes defined for transparent table SBOOK.
   a) Return to *Display Table*, then open *Indexes*.

   How many secondary indexes are defined for transparent table SBOOK?

   Two secondary indexes are defined for transparent table SBOOK.

   Which of the indexes are generated when the ABAP system is deployed on an SAP HANA database, and which are not created on SAP HANA?

   None of the secondary indexes are created if the ABAP system runs on SAP HANA, but they are created on all other supported databases.

Unit 1
Exercise 4

Create a Project in ABAP Development Tools

Business Example

Your company has identified SAP HANA as an important strategic topic and has asked you to refresh your ABAP skills, paying particular attention to those required to develop or review code that leverages the strengths of SAP HANA.

Create a Project in ABAP Development Tools

Open ABAP Development Tools and create a project for the application server T8N, which uses SAP HANA as its primary database. Add your development package ZHA400_## and development package HA400_PRIMDB_EXERCISE to the favorites of this project.

1. Open ABAP Development Tools (that is, the *ABAP* perspective in Eclipse).
2. Create a project connected to system T8N, which uses SAP HANA as its primary database. Use the user ID and password provided by your instructor.
3. Add your package ZHA400_## as a favorite under your project.
4. Add package HA400_PRIMDB_EXERCISE to the favorites of this project.
5. Optionally, if you also want to use ABAP Development Tools for the exercises performed in system ZME, which uses Sybase as its primary database, create a second ABAP project that is connected to this system. Add your own package ZHA400_## and the package HA400_SECDB_EXERCISE as favorite packages to this project.

Unit 1
Solution 4

Create a Project in ABAP Development Tools

Business Example

Your company has identified SAP HANA as an important strategic topic and has asked you to refresh your ABAP skills, paying particular attention to those required to develop or review code that leverages the strengths of SAP HANA.

Create a Project in ABAP Development Tools
Open ABAP Development Tools and create a project for the application server T8N, which uses SAP HANA as its primary database. Add your development package ZHA400_## and development package HA400_PRIMDB_EXERCISE to the favorites of this project.

1. Open ABAP Development Tools (that is, the *ABAP* perspective in Eclipse).
    a) If you have not opened *eclipse*, yet, do so now. Choose *eclipse* from the task bar of the Windows Terminal Server.

    b) If the *Welcome* page is open, close it or choose the *Workbench* link.

    c) Change to the *ABAP* perspective using the button in the upper right-hand corner.

2. Create a project connected to system T8N, which uses SAP HANA as its primary database. Use the user ID and password provided by your instructor.
    a) Choose *Create an ABAP project* on the *Project Explorer* window on the left.

    b) Choose system ID *T8N*.

    c) Choose *Next >*.

    d) Enter client `001`, your user, and your password.

    e) Choose *Finish*.

3. Add your package ZHA400_## as a favorite under your project.
    a) In the *Project Explorer*, right-click *Favorite Packages* under your project and choose *Add Package...*.

    b) Enter the name of your own package and choose *OK*.

4. Add package HA400_PRIMDB_EXERCISE to the favorites of this project.
    a) In the *Project Explorer*, right-click *Favorite Packages* under your project and choose *Add Package...*.

    b) Enter package name HA400_PRIMDB_EXERCISE and choose *OK*.

5. Optionally, if you also want to use ABAP Development Tools for the exercises performed in system ZME, which uses Sybase as its primary database, create a second ABAP project that

is connected to this system. Add your own package ZHA400_## and the package HA400_SECDB_EXERCISE as favorite packages to this project.

a) Repeat the actions of previous steps with the appropriate adjustments to the system ID, client, and package names.

Unit 2
Exercise 5

Search for Potential Functional Issues with Code Inspector

Business Example

You have an ABAP program that you want migrate to a system with an SAP HANA database. You want to perform a static code analysis to find potential functional issues that might occur when executing the program on SAP HANA.

Template:

Report HA400_CODE_CHECK_T1

Solution:

Report HA400_CODE_CHECK_S1

Task 1: Copy and Understand the Template

Log on to the application server ZME that uses Sybase ASE as primary database. Create a copy of report HA400_CODE_CHECK_T1 in your package ZHA400_## (suggested name: ZHA400_##_CHECK_1, where ## is your group number). Activate and execute the program, first on the primary database (Sybase ASE), then on the secondary database (SAP HANA).

1. Create a copy of the report. Place it in your package ZHA400_## and assign it to your workbench task.
2. Activate and execute the program with the default settings.

   What data does the report display?

3. Execute the program again, but this time let it read its data from the secondary database.

   What does the report display this time?

Task 2: Analyze the Program with ABAP Code Inspector

Analyze the program with the Code Inspector (transaction `SCI`) to identify the source of this functional issue. Use globally shipped check variant *FUNCTIONAL_DB* for this check.

1. In Code Inspector (transaction `SCI`), create a local inspection for your program ZHA400_##_CHECK_1 based on check variant *FUNCTIONAL_DB* (suggested name for the inspection: HA400_##_FUNCTIONAL).

2. Execute the inspection and analyze the inspection result. Navigate to the ABAP source code. Read the check documentation for hints on how to correct the program.

   Which check reports an error?

   What is the reason for this error?

   What can you do to correct the coding?

Task 3: Improve the Program

1. Edit subroutine *get_data* of your program. Apply one of the possible corrections for the Code Inspector error.

2. Activate and test your program. Make sure both databases deliver exactly the same data.

Unit 2
Solution 5

Search for Potential Functional Issues with Code Inspector

Business Example

You have an ABAP program that you want migrate to a system with an SAP HANA database. You want to perform a static code analysis to find potential functional issues that might occur when executing the program on SAP HANA.

Template:

Report HA400_CODE_CHECK_T1

Solution:

Report HA400_CODE_CHECK_S1

Task 1: Copy and Understand the Template

Log on to the application server ZME that uses Sybase ASE as primary database. Create a copy of report HA400_CODE_CHECK_T1 in your package ZHA400_## (suggested name: ZHA400_##_CHECK_1, where ## is your group number). Activate and execute the program, first on the primary database (Sybase ASE), then on the secondary database (SAP HANA).

1. Create a copy of the report. Place it in your package ZHA400_## 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 with the default settings.
   a) Complete this step as you learned to do in previous classes.

   What data does the report display?

   The report displays a list of carriers – each with revenues separated by economy, business, and first class.
3. Execute the program again, but this time let it read its data from the secondary database.
   a) Complete this step as before.

   What does the report display this time?

   The report displays a much longer list on which the same carrier appears several times.

Task 2: Analyze the Program with ABAP Code Inspector

Analyze the program with the Code Inspector (transaction `SCI`) to identify the source of this functional issue. Use globally shipped check variant *FUNCTIONAL_DB* for this check.

1. In Code Inspector (transaction `SCI`), create a local inspection for your program ZHA400_##_CHECK_1 based on check variant *FUNCTIONAL_DB* (suggested name for the inspection: HA400_##_FUNCTIONAL).

   a) Enter transaction `SCI`.

   b) In the *Inspection* frame, enter the name for the inspection and choose *Create*.

   c) In the *Object Selection* frame, enter *Single*, *Program*, and the name of your program.

   d) In the *Check Variant* frame, enter the name of the check variant.

   Hint:
   You might have to choose the button to the left of the input field before you can choose global check variants.

2. Execute the inspection and analyze the inspection result. Navigate to the ABAP source code. Read the check documentation for hints on how to correct the program.

   a) On the toolbar, choose *Execute*.

   b) To see the inspection result, choose *Results* on the toolbar.

   c) Check the reported error(s). To navigate to the ABAP source, double-click a message code.

   d) To display the check documentation, choose the “i”-icon next to the error message.

   Which check reports an error?

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

   What is the reason for this error?

   Internal table *lt_sbook* is filled with a SELECT statement from database table SBOOK. A subsequent LOOP over this internal table makes use of control structures AT ... ENDAT. This only works correctly if the content of *lt_sbook* is sorted by column CARRID.

What can you do to correct the coding?

Make sure the content of *lt_sbook* is sorted by column CARRID. There are several ways to achieve this: 1. Add `ORDER BY CARRID` to the SELECT statement. 2. Add statement `SORT lt_sbook BY carrid.` before the loop. 3. Declare lt_sbook as Sorted Table.

Task 3: Improve the Program

1. Edit subroutine *get_data* of your program. Apply one of the possible corrections for the Code Inspector error.
   a) Alternative 1: Add addition `ORDER BY CARRID` at the end of the SELECT statement .
   b) Alternative 2: Add statement `SORT lt_sbook BY carrid.` before the loop.
   c) Alternative 3: Replace the declaration of *lt_sbook* with statement `DATA lt_sbook TYPE SORTED TABLE OF sbook with NON-UNIQUE KEY carrid.`
   d) See the source code extract from the model solution.
2. Activate and test your program. Make sure both databases deliver exactly the same data.
   a) Complete this step as you learned to do in previous classes.
   b) Verify that your code matches the following solution:

Source code extract from model solution (Program HA400_OPT_OSQL_S1)

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

FORM get_data    USING pv_dbcon     TYPE dbcon-con_name
              CHANGING ct_carriers TYPE ty_t_carriers.

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

  DATA ls_carrier LIKE LINE OF ct_carriers.

* Alternative 3: Sorted table
*   DATA:
*     lt_sbook TYPE SORTED TABLE OF sbook
*              WITH NON-UNIQUE KEY carrid.

  DATA:
    lt_sbook TYPE TABLE OF sbook,
    ls_sbook TYPE sbook,
    ls_scarr TYPE scarr.

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

*Alternative 1: ORDER BY
*   SELECT * FROM sbook
*      CONNECTION (pv_dbcon)
*      INTO TABLE lt_sbook
```

```
  SELECT * FROM sbook
     CONNECTION (pv_dbcon)
     INTO TABLE lt_sbook
      ORDER BY carrid.

* Alternative 2: SORT ...
*  SORT lt_sbook BY carcid.

  LOOP AT lt_sbook INTO ls_sbook.

    AT NEW carrid.

      IF ls_carrier-carrid IS NOT INITIAL.
        APPEND ls_carrier TO ct_carriers.
        CLEAR ls_carrier.
      ENDIF.

      SELECT SINGLE * FROM scarr
                CONNECTION (pv_dbcon)
                      INTO ls_scarr
                     WHERE carrid = ls_sbook-carrid.

      MOVE-CORRESPONDING ls_scarr TO ls_carrier.

    ENDAT.

    CASE ls_sbook-class.
      WHEN 'Y'.
        ls_carrier-revenue_economy  = ls_carrier-revenue_economy
                                    + ls_sbook-loccuram.
      WHEN 'C'.
        ls_carrier-revenue_business = ls_carrier-revenue_economy
                                    + ls_sbook-loccuram.
      WHEN 'F'.
        ls_carrier-revenue_first    = ls_carrier-revenue_economy
                                    + ls_sbook-loccuram.
    ENDCASE.

    AT LAST.
      APPEND ls_carrier TO ct_carriers.
      CLEAR ls_carrier.
    ENDAT.
  ENDLOOP.

  SORT ct_carriers BY carrid.

ENDFORM.                    "
```

Unit 2
Exercise 6

Analyze Potential Performance Issues

Business Example

You have an ABAP program that does not perform well. You want to perform a static code analysis to get clues about what can be the cause, especially if the program is supposed to perform well when executed on SAP HANA.

Template:

Report HA400_OPT_OSQL_T1

Solution:

Report HA400_OPT_OSQL_S1

Task 1: Copy and Understand the Template

Log on to application server ZME that uses Sybase ASE as primary database. Create a copy of report HA400_OPT_OSQL_T1 in your package ZHA400_## (suggested name: ZHA400_##_OSQL_1, where ## is your group number). Activate and execute the program.

1. Create a copy of the report. Place it in your package ZHA400_## and assign it to your workbench task.

2. Activate and execute the program with default settings.

   What does the report display?

   ____
   ____
   ____

Task 2: Analyze the Program with ABAP Code Inspector

Analyze the program with the Code Inspector (SCI) to identify potential performance problems. Use globally shipped check variant *PERFORMANCE_DB* for this check.

1. Perform an inspection of your program based on your new check variant (suggested name for the inspection: *HA400_PERF##*).

2. Execute the inspection and analyze the inspection result. Navigate to the ABAP source code. Read the check documentation for hints on how to correct the program.

In the search for problematic SELECT *-statements you should find an error related to database table SBOOK. Why is there no error related to database table SCUSTOM?

Task 3: Analyze the Suggested Improvements

1. Uncomment the source code of subroutine *get_data_solution1*.

   Analyze the coding in the *get_data_solution1* subroutine and compare it to the coding in the *get_data_template* subroutine. What are the two main improvements?

   Why is it important that internal table *lt_sbook* is declared as a sorted table and not as a standard table?

2. Uncomment the source code of subroutine *get_data_solution2*.

   Analyze the coding in the *get_data_solution2* subroutine and compare it to the coding in the *get_data_template* subroutine. What is the main difference?

   Why is it important that the content of internal table *lt_cust_sbook* is sorted?

3. Activate and test your program. Make sure all three subroutines deliver exactly the same data.

Task 4: Quantify the Improvements

Perform a runtime measurement with the ABAP Trace (SAT). Compare the runtime consumption of *get_data_template*, *get_data_solution1* and *get_data_solution2*. Do this analysis on both, primary and secondary database.

1. Perform a runtime measurement with the ABAP Trace (transaction `SAT`) on the primary database and compare the gross runtime of the three subroutines.

On the primary database, the runtime of *get_data_template* is __________(1) microseconds, the runtime of *get_data_solution1* is __________(2) microseconds and the runtime of *get_data_solution2* is __________(3) microseconds.

On the secondary database (SAP HANA), the runtime of *get_data_template* is __________(1) microseconds, the runtime of *get_data_solution1* is __________(2) microseconds and the runtime of *get_data_solution2* is __________(3) microseconds.

2. Repeat the runtime measurement with data retrieval from the secondary database (SAP HANA).

   On SAP HANA, the runtime of *get_data_template* is __________(1) microseconds, the runtime of *get_data_solution1* is __________(2) microseconds and the runtime of *get_data_solution2* is __________(3) microseconds.

Analyze Potential Performance Issues

Business Example

You have an ABAP program that does not perform well. You want to perform a static code analysis to get clues about what can be the cause, especially if the program is supposed to perform well when executed on SAP HANA.

Template:

Report HA400_OPT_OSQL_T1

Solution:

Report HA400_OPT_OSQL_S1

Task 1: Copy and Understand the Template

Log on to application server ZME that uses Sybase ASE as primary database. Create a copy of report HA400_OPT_OSQL_T1 in your package ZHA400_## (suggested name: ZHA400_##_OSQL_1, where ## is your group number). Activate and execute the program.

1. Create a copy of the report. Place it in your package ZHA400_## 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 with default settings.
   a) Complete this step as you learned to do in previous classes.

   What does the report display?

   The report displays a list of customers – each with the average number of days between a booking and the actual flight.

Task 2: Analyze the Program with ABAP Code Inspector

Analyze the program with the Code Inspector (SCI) to identify potential performance problems. Use globally shipped check variant *PERFORMANCE_DB* for this check.

1. Perform an inspection of your program based on your new check variant (suggested name for the inspection: *HA400_PERF##*).
   a) Start transaction `SCI`.
   b) In the *Inspection* frame, enter the name for the inspection and choose *Create*.
   c) In the *Object Selection* frame, enter *Single*, *Program* and type in the name of your program.

d) In the *Check Variant* frame, enter the name of the check variant.

> Hint:
> You might have to select the button left of the input field before you can choose global check variants.

2. Execute the inspection and analyze the inspection result. Navigate to the ABAP source code. Read the check documentation for hints on how to correct the program.

   a) On the toolbar, choose *Execute*.

   b) To see the inspection result, choose *Results* on the toolbar.

   c) Check the reported error(s). To navigate to the ABAP source, double-click a message code.

   d) To display the check documentation, choose the "i"-icon next to the error message.

   In the search for problematic SELECT *-statements you should find an error related to database table SBOOK. Why is there no error related to database table SCUSTOM?

   The check has an input parameter that only reports SELECT * statements where less than a certain percentage of all fields is actually needed. This parameter is 20% by default.

Task 3: Analyze the Suggested Improvements

1. Uncomment the source code of subroutine *get_data_solution1*.

   a) Select the entire source code between FORM and ENDFORM.

   b) Right-click on the selection and choose *Format* → *Uncomment Lines* or select Ctrl + Shift + < on your keyboard.

   Analyze the coding in the *get_data_solution1* subroutine and compare it to the coding in the *get_data_template* subroutine. What are the two main improvements?

   NO SELECT * STATEMENTS: From both tables, only those fields are selected that are actually needed by the program. NO NESTED SELECTS: In subroutine get_data_solution1, the two nested SELECT-loops have been replaced by two array fetches on SCUSTOM and SBOOK and two nested loops over the internal tables.

   Why is it important that internal table *lt_sbook* is declared as a sorted table and not as a standard table?

   If *lt_sbook* was a standard table, the runtime environment would not be able to optimize the loop over this table and we would lose a lot of runtime by searching for the required entries.

2. Uncomment the source code of subroutine *get_data_solution2*.

a) Select the entire source code between FORM and ENDFORM.

b) Right-click on the selection and choose *Format* → *Uncomment Lines* or select Ctrl + Shift + < on your keyboard.

Analyze the coding in the *get_data_solution2* subroutine and compare it to the coding in the *get_data_template* subroutine. What is the main difference?

Instead of two separate SELECTs there is one SELECT statement with a join.

Why is it important that the content of internal table *lt_cust_sbook* is sorted?

If *lt_cust_sbook* was not sorted, the AT NEW and AT LAST logic inside LOOP would not work.

3. Activate and test your program. Make sure all three subroutines deliver exactly the same data.

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

Task 4: Quantify the Improvements

Perform a runtime measurement with the ABAP Trace (SAT). Compare the runtime consumption of *get_data_template*, *get_data_solution1* and *get_data_solution2*. Do this analysis on both, primary and secondary database.

1. Perform a runtime measurement with the ABAP Trace (transaction `SAT`) on the primary database and compare the gross runtime of the three subroutines.

   a) Start transaction `SAT`.

   b) Enter the name of your program and, if necessary, the name of the variant (**`DEFAULT`**).

   c) Choose *Execute*.

   d) On the selection screen of the report, choose *All, one after the other* and *Primary database*, and choose *Execute* again.

   e) After the report has finished, choose *Back* from the system toolbar or select F3 on your keyboard to return to the selection screen.

   f) Choose *Back* again or select F3 on your keyboard again to display the result of the runtime measurement.

   g) On the *Profile Trace Results* screen, double-click the node *Runtime Measurement* → *Internal Processing Blocks* → *Subroutines* → *PERFORM* to filter the *Hit List* on the right to display only subroutine calls.

   h) Sort the *Hit List* by choosing the header of the *Gross [microsec]* column.

On the primary database, the runtime of *get_data_template* is __________(1) microseconds, the runtime of *get_data_solution1* is __________(2) microseconds and the runtime of *get_data_solution2* is __________(3) microseconds.

Your instructor will provide their measurement for comparison.

On the secondary database (SAP HANA), the runtime of *get_data_template* is __________(1) microseconds, the runtime of *get_data_solution1* is __________(2) microseconds and the runtime of *get_data_solution2* is __________(3) microseconds.

Your instructor will provide their measurement for comparison.

2. Repeat the runtime measurement with data retrieval from the secondary database (SAP HANA).

   On SAP HANA, the runtime of *get_data_template* is __________(1) microseconds, the runtime of *get_data_solution1* is __________(2) microseconds and the runtime of *get_data_solution2* is __________(3) microseconds.

   Your instructor will provide his measurement for comparison.

   a) Execute the runtime analysis as before but this time, choose the *Secondary Database* option on the selection screen of the report.

      Code Extract from program HA400_OPT_OSQL_S1

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

FORM get_data_template    USING pv_dbcon     TYPE dbcon-con_name
                       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 (pv_dbcon)
     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
       CONNECTION (pv_dbcon)
             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_solution1: Nested Loops
*&---------------------------------------------------------------------*
FORM get_data_solution1  USING pv_dbcon     TYPE dbcon-con_name
                      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.

* help variables
  DATA lv_count TYPE i.

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

  CLEAR ct_customers.

  SELECT id name postcode city country
    FROM scustom
    CONNECTION (pv_dbcon)
    INTO TABLE lt_scustom.
*    ORDER BY id no improvement, sorting on Appl. server more efficient
  SELECT customid fldate order_date
    FROM sbook
    CONNECTION (pv_dbcon)
    INTO TABLE lt_sbook
   WHERE cancelled = space.                            "#EC CI_NOFIELD
*    ORDER BY customid  no impr., sorting on AS 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.                    "

*&---------------------------------------------------------------------*
*&      Form  get_data_solution_2: Join
*&---------------------------------------------------------------------*
FORM get_data_solution2  USING pv_dbcon     TYPE dbcon-con_name
                      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
*****************

  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 (pv_dbcon)
    INTO TABLE lt_cust_book
         WHERE b~cancelled = space
    ORDER BY c~id.

  LOOP AT lt_cust_book INTO ls_cust_book.
```

```
    AT NEW country.
      IF ls_customer-id IS NOT INITIAL.
        ls_customer-days_ahead = ls_customer-days_ahead / lv_count.
        APPEND ls_customer TO ct_customers.
      ENDIF.

      CLEAR lv_count.
      CLEAR ls_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.
    ENDAT.

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

    AT LAST.
      IF ls_customer-id IS NOT INITIAL.
        ls_customer-days_ahead = ls_customer-days_ahead / lv_count.
        APPEND ls_customer TO ct_customers.
      ENDIF.
    ENDAT.
  ENDLOOP.

*   SORT ct_customers BY id. "already sorted
ENDFORM.
```

Perform a Structured SQL Performance Analysis

Business Example

You performed a static code analysis of a poorly performing ABAP program using the SAP Code Inspector and collected run-time data for the same code. You want to combine the results of both using the SAP Performance Tuning Worklist to analyze which pieces of code should be improved with highest priority. You improve the corresponding pieces of code and want to repeat the analysis to quantify the performance improvement achieved.

Sample classes:

CL_HA400_CI_PERFORMANCE_ISSUES
CL_HA400_CI_OPTIMIZED

Task 1: Analyze the Source Code of Class CL_HA400_CI_PERFORMANCE_ISSUES
Log on to the application server *ZME on Sybase* with system ID ZME. Open class CL_HA400_CI_PERFORMANCE_ISSUES and analyze the coding.

1. Open class CL_HA400_CI_PERFORMANCE_ISSUES.

   What are the public methods of the class?

   ____
   ____
   ____

   What are the private methods and where are they called?

   ____
   ____
   ____

   Are there any conditions for these calls?

   ____
   ____
   ____

Task 2: Analyze Existing Code Inspector Result for Class CL_HA400_CI_PERFORMANCE_ISSUES
Use the Code Inspector tool, and display the settings and result of existing global inspection HA400_SWLT.

1. Open the code inspector and analyze the properties of global inspection *HA400_SWLT*.

Which object(s) have been analyzed with this inspection?

Which check variant has been used for this inspection?

2. Analyze the inspection result.

   How many SAP HANA-relevant performance errors did the check find and in which methods?

   What information is missing for prioritizing the performance errors?

Task 3: Analyze Existing SQL Monitor Data for Class CL_HA400_CI_PERFORMANCE_ISSUES
With the SQL Monitor tool, analyze a snapshot of SQL Monitor data. Restrict the display to measurements related to class CL_HA400_CI_PERFORMANCE_ISSUES.

Note:
The description of the snapshot begins with "SQLM Snapshot for HA400".

1. Start the SQL Monitor tool and display the data of the snapshot.
2. Find the statement with the maximum consumption of database time.

   Which SQL statement has consumed the maximum database time?

Which processing block contains this statement?

Task 4: Combine Static and Dynamic Analysis for Class CL_HA400_CI_PERFORMANCE_ISSUES

With the SQL Performance Tuning Worklist tool, combine the Code Inspector findings and the SQL Monitor data to build a prioritized work list for SQL Performance tuning.

1. Start the SQL Performance Tuning Worklist tool and restrict the analysis to your class.
2. Activate the analysis of SQL Monitor data and select the same SQLM snapshot as before.
3. Activate the analysis of static check data and specify the same code inspector inspection as before. Make sure the program aggregates findings by code position.
4. Execute the analysis in the SQL Performance Tuning Worklist tool.
5. Analyze the static and SQL Monitor findings for the various SQL statements of class CL_HA400_CI_PERFORMANCE_ISSUES. Start with the finding with the maximum DB time.

   How often has the statement been executed?

   What can you do to reduce the DB time consumption?

Task 5: Optional: Repeat the Analysis for Class CL_HA400_CI_OPTIMIZED

Assume you have developed an improved version of the class. Understand the coding in the class and repeat your analysis to prove that there are no SAP HANA relevant findings in Code Inspector and that you reduced the overall DB time consumption drastically.

1. Analyze the source code of class CL_HA400_CI_OPTIMIZED.
2. Analyze the CL_HA400_CI_OPTIMIZED class with the code inspector.
3. Combine your Code Inspector check result with SQL Monitor data from the same snapshot as above.

Perform a Structured SQL Performance Analysis

Business Example

You performed a static code analysis of a poorly performing ABAP program using the SAP Code Inspector and collected run-time data for the same code. You want to combine the results of both using the SAP Performance Tuning Worklist to analyze which pieces of code should be improved with highest priority. You improve the corresponding pieces of code and want to repeat the analysis to quantify the performance improvement achieved.

Sample classes:

CL_HA400_CI_PERFORMANCE_ISSUES
CL_HA400_CI_OPTIMIZED

Task 1: Analyze the Source Code of Class CL_HA400_CI_PERFORMANCE_ISSUES
Log on to the application server *ZME on Sybase* with system ID ZME. Open class CL_HA400_CI_PERFORMANCE_ISSUES and analyze the coding.

1. Open class CL_HA400_CI_PERFORMANCE_ISSUES.
   a) Start the ABAP Workbench (for example using transaction code `SE80`).
   b) Choose *Repository Browser* if it is not chosen yet.
   c) Choose the *Class/Interface* object category.
   d) Enter the class name as the object name and choose Enter.
   e) Double-click the class.

   What are the public methods of the class?

   The public methods are *get_customers* and *get_agencies*.

   What are the private methods and where are they called?

   The private method *get_days_ahead_for_customer* is called from method *get_customers* and method *get_days_ahead_for_agencies* is called from method *get_agencies*.

Are there any conditions for these calls?

Both private methods are only executed if import parameter *iv_with_days* of the corresponding public method is set to *abap_true*.

Task 2: Analyze Existing Code Inspector Result for Class CL_HA400_CI_PERFORMANCE_ISSUES

Use the Code Inspector tool, and display the settings and result of existing global inspection HA400_SWLT.

1. Open the code inspector and analyze the properties of global inspection *HA400_SWLT*.

   Which object(s) have been analyzed with this inspection?

   Global class CL_HA400_CI_PERFORMANCE_ISSUES.

   Which check variant has been used for this inspection?

   Global check variant PERFORMANCE_DB.

   a) Start transaction `SCI`.

   b) In the *Inspection* frame, enter **`HA400_SWLT`** and choose *Display*.

   Hint:
   You might have to choose the button on the left of the input field before you can choose global inspections.

2. Analyze the inspection result.

   a) To see the inspection result, on the toolbar, choose *Results (Shift F6)*.

   b) Check the reported errors. To navigate to the ABAP source code, double-click the message.

   How many SAP HANA-relevant performance errors did the check find and in which methods?

   There are three performance errors in methods *get_days_ahead_for_customer* and *get_days_ahead_for_agencies*.

What information is missing for prioritizing the performance errors?

How often the different methods are actually executed in the productive system, in particular, how often import parameter *iv_with_days* is set to *abap_true*.

Task 3: Analyze Existing SQL Monitor Data for Class CL_HA400_CI_PERFORMANCE_ISSUES
With the SQL Monitor tool, analyze a snapshot of SQL Monitor data. Restrict the display to measurements related to class CL_HA400_CI_PERFORMANCE_ISSUES.

Note:
The description of the snapshot begins with “SQLM Snapshot for HA400”.

1. Start the SQL Monitor tool and display the data of the snapshot.
   a) Start transaction `SQLM`.
   b) In the *Snapshots* frame, choose *Display Snapshots*.
   c) In the *Results* frame, choose *Select Snapshot*.
   d) Select the line where the content of column *Description* starts with SQLM Snapshot for HA400 and choose *Enter*.
   e) In the *Object name* field, enter the name of the class and choose *Execute (F8)*.
2. Find the statement with the maximum consumption of database time.
   a) Sort the list descending by column *Total DB time*.

   Which SQL statement has consumed the maximum database time?

   A SELECT from table SBOOK.

   Which processing block contains this statement?

   Method *get_days_ahead_for_customer*.

Task 4: Combine Static and Dynamic Analysis for Class CL_HA400_CI_PERFORMANCE_ISSUES
With the SQL Performance Tuning Worklist tool, combine the Code Inspector findings and the SQL Monitor data to build a prioritized work list for SQL Performance tuning.

1. Start the SQL Performance Tuning Worklist tool and restrict the analysis to your class.
   a) Start transaction `SWLT`.
   b) In the *Object name* field, enter the name of the class.
2. Activate the analysis of SQL Monitor data and select the same SQLM snapshot as before.

a) Choose the *SQL Monitor* tab.

b) Choose *Use SQL Monitor Data* if it is not yet chosen.

c) Choose *Select Snapshot*.

d) Choose the SQLM snapshot with the description starting with “SQLM Snapshot for HA400 ....”

3. Activate the analysis of static check data and specify the same code inspector inspection as before. Make sure the program aggregates findings by code position.

   a) Choose the *Static Checks* tab.

   b) Choose *Use Static Check Data* if it is not yet chosen.

   c) Choose *Code Inspector* if it is not yet chosen.

   d) Choose *Select Inspection Result*.

   e) Select the inspection (HA400_SWLT).

   f) In the *Further Options* frame, in the *Aggregate of Findings in Result Overview* field, choose the *Aggregate By: Code Position* entry.

4. Execute the analysis in the SQL Performance Tuning Worklist tool.

   a) Choose *Execute*.

5. Analyze the static and SQL Monitor findings for the various SQL statements of class CL_HA400_CI_PERFORMANCE_ISSUES. Start with the finding with the maximum DB time.

   a) In the *Result Overview*, make sure the results are ordered by column *Total DB time*.

   b) Choose the SQL statement with the largest value in the *Total DB time* column.

   c) Analyze the entries in *SQL Monitor Results* and *Static Check Findings*.

   How often has the statement been executed?

   Several hundred thousand times.

   What can you do to reduce the DB time consumption?

   Avoid the repeated single record accesses and get rid of the SELECT * in this statement.

Task 5: Optional: Repeat the Analysis for Class CL_HA400_CI_OPTIMIZED

Assume you have developed an improved version of the class. Understand the coding in the class and repeat your analysis to prove that there are no SAP HANA relevant findings in Code Inspector and that you reduced the overall DB time consumption drastically.

1. Analyze the source code of class CL_HA400_CI_OPTIMIZED.

   a) Perform this step as before.

2. Analyze the CL_HA400_CI_OPTIMIZED class with the code inspector.

   a) Perform this step as before.

3. Combine your Code Inspector check result with SQL Monitor data from the same snapshot as above.

   a) Perform this step as before.

Unit 3
Exercise 8

Use Features of Enhanced Open SQL

Business Example

You are asked to improve an ABAP program which reads a few columns from the database, computes a few additional columns, and aggregates and groups the result. You want to use the new Open SQL to perform these computations in the database.

Template:

Report HA400_NEW_OSQL_T1

Solution:

Report HA400_NEW_OSQL_S1

Task 1: Copy and Understand Templates

In system ZME, which uses Sybase as its primary database, create a copy of report HA400_NEW_OSQL_T1 in your package (suggested name: ZHA400_##_NEW_OSQL, where ## is your group number). Analyze the source code, activate and execute the program.

1. If you are not still logged on, log on to system *ZME on Sybase* and start the ABAP workbench.
2. Create a copy of the report. Place it in your ZHA400_## package and assign it to your workbench task.
3. Analyze the source code of the *get_data_template* subroutine.

   What data is read and from which database tables?

   How is this data aggregated?

4. Activate and execute the program.

Task 2: Use New SQL Features to Do the Aggregation in One SELECT Statement

In the *get_data_solution* subroutine, implement a left outer join on SCARR and SBOOK. Use SQL CASE and aggregation function SUM( ) to calculate all three sums in one SELECT Statement.

1. Edit your program. In the *get_data_solution* subroutine, you find a SELECT statement that is commented out. Remove the comments and perform a syntax check.

   The syntax check asks you to use new Open SQL Syntax. Which part of the statement is the reason for that?

   Which changes are necessary to switch from classic Open SQL to new Open SQL syntax?

2. Make these changes to switch from classic Open SQL to new Open SQL syntax.
3. Optionally, you can change the sequence of clauses. Place the field list after the from clause and add the introductory key word FIELDS. Move the INTO and CONNECTION clauses to the very end of the statement.

   Note:
   The field list after the from clause allows you to use code completion when editing the field list. The INTO clause at the end makes the target data object easier to spot.

4. Edit the field list. Use SQL CASE within each of the three SUM( ) functions to make sure that LOCCURAM is only added if the booking belongs to the respective flight class, that is, if field CLASS has the right value.
5. Activate and test your program to verify that the data has been read correctly.

Unit 3
Solution 8

Use Features of Enhanced Open SQL

Business Example

You are asked to improve an ABAP program which reads a few columns from the database, computes a few additional columns, and aggregates and groups the result. You want to use the new Open SQL to perform these computations in the database.

Template:

Report HA400_NEW_OSQL_T1

Solution:

Report HA400_NEW_OSQL_S1

Task 1: Copy and Understand Templates

In system ZME, which uses Sybase as its primary database, create a copy of report HA400_NEW_OSQL_T1 in your package (suggested name: ZHA400_##_NEW_OSQL, where ## is your group number). Analyze the source code, activate and execute the program.

1. If you are not still logged on, log on to system *ZME on Sybase* and start the ABAP workbench.
    a) Perform this step as before.
2. Create a copy of the report. Place it in your ZHA400_## package and assign it to your workbench task.
    a) Complete this step as you learned to do in previous classes.
3. Analyze the source code of the *get_data_template* subroutine.

    What data is read and from which database tables?

    Carrier ID (field CARRID), carrier name (field CARRNAME), and currency code (Field CURRCODE) from table SCARR, plus the flight class (field CLASS) and the ticket price (field LOCCURAM) in the local currency of the carrier.

    How is this data aggregated?

    In a nested loop, three sums are calculated per carrier. The total revenue in first class, business class, and economy class.
4. Activate and execute the program.
    a) Complete this step as you learned to do in previous classes.

Task 2: Use New SQL Features to Do the Aggregation in One SELECT Statement

In the *get_data_solution* subroutine, implement a left outer join on SCARR and SBOOK. Use SQL CASE and aggregation function SUM( ) to calculate all three sums in one SELECT Statement.

1. Edit your program. In the *get_data_solution* subroutine, you find a SELECT statement that is commented out. Remove the comments and perform a syntax check.

   The syntax check asks you to use new Open SQL Syntax. Which part of the statement is the reason for that?

   The field CANCELLED in the WHERE clause. In classic Open SQL fields of the right table of a left outer join are not allowed in the WHERE clause.

   Which changes are necessary to switch from classic Open SQL to new Open SQL syntax?

   Column-separated field list after SELECT, column-separated fields after GROUP BY, and prefix @ before data object *ct_carriers*.

2. Make these changes to switch from classic Open SQL to new Open SQL syntax.
   a) See source code extract from the model solution.

3. Optionally, you can change the sequence of clauses. Place the field list after the from clause and add the introductory key word FIELDS. Move the INTO and CONNECTION clauses to the very end of the statement.

   Note:
   The field list after the from clause allows you to use code completion when editing the field list. The INTO clause at the end makes the target data object easier to spot.

   a) See source code extract from the model solution.

4. Edit the field list. Use SQL CASE within each of the three SUM( ) functions to make sure that LOCCURAM is only added if the booking belongs to the respective flight class, that is, if field CLASS has the right value.
   a) See source code extract from the model solution.

5. Activate and test your program to verify that the data has been read correctly.
   a) Complete this step as before.

   b) Verify that your code matches the following solution:

   Source code extract from the model solution (Program HA400_NEW_OSQL_S1)

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

```
FORM get_data_template    USING pv_dbcon     TYPE dbcon-con_name
                       CHANGING ct_carriers TYPE ty_t_carriers.

  TYPES: BEGIN OF ty_s_scarr,
           carrid   TYPE scarr-carrid,
           carrname TYPE scarr-carrname,
           currcode TYPE scarr-currcode,
         END OF ty_s_scarr.
  TYPES: BEGIN OF ty_s_sbook,
           carrid    TYPE sbook-carrid,
           class     TYPE sbook-class,
           loccurkey TYPE sbook-loccurkey,
           loccuram  TYPE sbook-loccuram,
         END OF ty_s_sbook.

  DATA ls_carrier TYPE ty_s_carrier.

  DATA: lt_scarr TYPE SORTED TABLE OF ty_s_scarr
                      WITH UNIQUE KEY carrid,
        ls_scarr TYPE ty_s_scarr.

  DATA: lt_sbook TYPE SORTED TABLE OF ty_s_sbook
                      WITH NON-UNIQUE KEY carrid,
        ls_sbook TYPE ty_s_sbook.

* Classic Open SQL
  SELECT carrid carrname currcode
    FROM scarr
    CONNECTION (pv_dbcon)
    INTO TABLE lt_scarr.

  SELECT carrid class loccurkey loccuram
    FROM sbook
    CONNECTION (pv_dbcon)
    INTO TABLE lt_sbook
     WHERE cancelled <> 'X'.

  LOOP AT lt_scarr INTO ls_scarr.
    CLEAR ls_carrier.

    LOOP AT lt_sbook INTO ls_sbook
                     WHERE carrid = ls_scarr-carrid.
      IF ls_sbook-loccurkey <> ls_scarr-currcode.
        MESSAGE 'Inconsistent data' TYPE 'A'.
      ENDIF.
      CASE ls_sbook-class.
        WHEN 'F'.  "First Class Booking
          ls_carrier-revenue_first =     ls_carrier-revenue_first
                                       + ls_sbook-loccuram.
        WHEN 'C'. "Business Class Booking
          ls_carrier-revenue_business =  ls_carrier-revenue_business
                                       + ls_sbook-loccuram.
        WHEN 'Y'.  "Economy Class Booking
          ls_carrier-revenue_economy =   ls_carrier-revenue_economy
                                       + ls_sbook-loccuram.
      ENDCASE.
    ENDLOOP.
    MOVE-CORRESPONDING ls_scarr TO ls_carrier.
    APPEND ls_carrier TO ct_carriers.
  ENDLOOP.

ENDFORM.                    "
*&---------------------------------------------------------------------*
*&      Form  get_data_solution
*&---------------------------------------------------------------------*
FORM get_data_solution    USING pv_dbcon     TYPE dbcon-con_name
                       CHANGING ct_carriers TYPE ty_t_carriers.

* Use new SQL features like CASE to read all data in one SELECT statement
*
* Note the possibility to change the sequence of clauses * in Enhanced
Open SQL:
*    - FROM clause before the field list enables code completion
*        (key word FIELD required for this)
*    - INTO clause at the end makes target data object easier to spot

  SELECT FROM scarr AS a LEFT OUTER JOIN sbook AS b
                           ON a~carrid = b~carrid
       FIELDS  a~carrid,
               a~carrname,
               a~currcode,
               SUM( CASE b~class
                      WHEN 'F' THEN b~loccuram
                      ELSE          0
                    END
                   ) AS revenue_first,
               SUM( CASE b~class
                      WHEN 'C' THEN b~loccuram
                      ELSE          0
                    END
                   ) AS revenue_business,
               SUM( CASE b~class
                      WHEN 'Y' THEN b~loccuram
                      ELSE          0
                    END
                   ) AS revenue_economy
      WHERE b~cancelled <> 'X'
         OR b~cancelled IS NULL
      GROUP BY a~carrid, a~carrname, a~currcode
      ORDER BY a~carrid
      INTO TABLE @ct_carriers
      CONNECTION (pv_dbcon).

ENDFORM.                    "
```

Unit 3
Exercise 9

Create a Core Data Services View and Use It in ABAP

Business Example

Your ABAP program contains a considerably complicated SELECT statement. You want to reuse this select statement in other programs. To achieve this, you define a CDS View and use it in your ABAP program.

Template:

Report HA400_CDS_VIEW_T1

Solution:

Report HA400_CDS_VIEW_S1
Data Definition HA400_CDS_S1 (defining CDS View Ha400_Cds_S1 and SQL View HA400_CDS_SQL_S1)

Task 1: Copy and Understand the Template

In the system that has SAP HANA as a primary database, create a copy of report HA400_CDS_VIEW_T1 in your package (suggested name: ZHA400_##_CDS_1, where ## is your group number). Analyze the source code, activate and execute the program.

Hint:
In this exercise, you have to create and implement the definition of a CDS view. This new type of repository object can only be edited in ABAP Development Tools (ADT). The ABAP program you can edit in the classical ABAP workbench. But you might want to take the opportunity and do the complete exercise in ADT.

1. If you are still logged on to system ZME, log off and log on to the system labelled *T8N on SAP HANA*.

2. Create a copy of the report. Place it in your package ZHA400_## and assign it to your workbench task.

   Hint:
   In ABAP Development Tools, you copy the report via context menu item *Duplicate*.

3. Analyze the SELECT statement in subroutine *get_data_template*.

From which database tables does the SELECT statement read its data?

Which fields are read from these database tables?

How is field COUNT calculated?

How is field DAYS_SUM calculated?

4. Analyze the LOOP ... ENDLOOP in subroutine *get_data_template*.

   What's the purpose of this LOOP?

   Why can't this average be calculated via aggregate function AVG( )?

5. Activate and execute the program.

Task 2: Create and Implement a Data Definition (DDL Source)

Create a new data definition (suggested name: ZHA400_##_CDS1, where ## stands for your group number) that defines a CDS view (suggested name: Zha400_##_Cds1) and an SQL view (suggested name: ZHA400_##_SQL1). Make sure it selects the same data as subroutine *get_data_template* of your program.

Note:
In CDS ABAP, you can not use an SQL function (*dats_days_between()* in our example) as argument of an aggregate function (AVG( ) in our example). The final solution will be a CDS view, that reads from another CDS view (view-on-view approach). In a first step, we implement a CDS view that only calculates the number of days between booking date and flight date. The aggregation can then be done by the Open SQL statement.

1. Using ABAP Development Tools for Eclipse, create a new data definition (= DDL Source) based on template *Define View With Join*.
2. Within the DDL source, set the name of the SQL view within the *sqlViewName* annotation.
3. **Edit the FROM clause of the generated SELECT statement. Replace** `data_source_name` **with `SBOOK as B` and** `joined_data_source_name` **with `SCUSTOM as C` to read data from these two tables**.
4. In the FROM clause, replace the generated join condition with **`ON B.customid = C.ID`**
5. Edit the field list of the generated SELECT statement. Read the same fields, in the same sequence as in the SELECT statement of subroutine *get_data_template*. But instead of the two aggregate functions SUM( ) and COUNT(*), only calculate the number of days between booking date and flight date.

   Read the following fields:

   a. From table SCUSTOM: fields ID, NAME, POSTCODE, CITY, COUNTRY.
   b. From table SBOOK: expression dats_days_between(ORDER_DATE, FLDATE). Don't forget to provide an alias for the calculated field (suggested name: *days_ahead*).

Hint:
You can reduce the typing effort by making use of the editor's code completion capabilities.

6. After the field list, add a WHERE clause. Exclude all bookings that are cancelled.
7. Why can't you add the ORDER BY addition?

   ____

   ____

   ____

8. Save and activate the data definition.

Task 3: Test the CDS View

Test your CDS View by displaying the data preview.

1. Open the data preview for the CDS View.

Task 4: Use the ABAP CDS View in an Open SQL Statement

In subroutine *get_data_solution* of your program, implement a select statement from your CDS View.

1. Edit your program (either in `SE80` or ADT). In subroutine *get_data_solution*, implement a select statement that reads the relevant data from your CDS view .

   Hint:
   Remember that the CDS view contains neither an ORDER BY clause nor an aggregation of the calculated number of days.

   Note:
   If you use the CDS view name (recommended), you have to use the new Open SQL syntax. If you use the SQL view name (not recommended) you may still use classical Open SQL Syntax.

2. Activate and test your program to verify that the data has been read correctly.

Unit 3
Solution 9

Create a Core Data Services View and Use It in ABAP

Business Example

Your ABAP program contains a considerably complicated SELECT statement. You want to reuse this select statement in other programs. To achieve this, you define a CDS View and use it in your ABAP program.

Template:

Report HA400_CDS_VIEW_T1

Solution:

Report HA400_CDS_VIEW_S1
Data Definition HA400_CDS_S1 (defining CDS View Ha400_Cds_S1 and SQL View HA400_CDS_SQL_S1)

Task 1: Copy and Understand the Template

In the system that has SAP HANA as a primary database, create a copy of report HA400_CDS_VIEW_T1 in your package (suggested name: ZHA400_##_CDS_1, where ## is your group number). Analyze the source code, activate and execute the program.

Hint:
In this exercise, you have to create and implement the definition of a CDS view. This new type of repository object can only be edited in ABAP Development Tools (ADT). The ABAP program you can edit in the classical ABAP workbench. But you might want to take the opportunity and do the complete exercise in ADT.

1. If you are still logged on to system ZME, log off and log on to the system labelled *T8N on SAP HANA*.

   a) Perform this step as before.

2. Create a copy of the report. Place it in your package ZHA400_## and assign it to your workbench task.

   Hint:
   In ABAP Development Tools, you copy the report via context menu item *Duplicate*.

   a) Complete this step as before.

3. Analyze the SELECT statement in subroutine *get_data_template*.
    a) Complete this step as before.

    From which database tables does the SELECT statement read its data?

    From tables SBOOK and SCUSTOM (inner join).

    Which fields are read from these database tables?

    Customer number (field ID), customer name (field NAME) and the address data (fields POSTCODE, CITY, COUNTRY) are read from table SCUSTOM. From table SBOOK come the booking date (field ORDER_DATE) and flight date (field FLDATE).

    How is field COUNT calculated?

    Aggregate function COUNT(*) is used to calculate the total number of bookings per customer.

    How is field DAYS_SUM calculated?

    Aggregate function SUM( ) is used to calculate the total number of days between booking date and flight date per customer. The number of days itself is calculated via SQL function *dats_days_between( )*.

4. Analyze the LOOP ... ENDLOOP in subroutine *get_data_template*.
    a) Complete this step as before.

    What's the purpose of this LOOP?

    Calculate the average number of days between booking date and flight date per customer.

    Why can't this average be calculated via aggregate function AVG( )?

    In Open SQL, aggregate function AVG( ) does not support expressions like SQL function *dats_days_between( )* as argument.

5. Activate and execute the program.
    a) Complete this step as you learned to do in previous classes.

Task 2: Create and Implement a Data Definition (DDL Source)

Create a new data definition (suggested name: ZHA400_##_CDS1, where ## stands for your group number) that defines a CDS view (suggested name: Zha400_##_Cds1) and an SQL view (suggested name: ZHA400_##_SQL1). Make sure it selects the same data as subroutine *get_data_template* of your program.

Note:
In CDS ABAP, you can not use an SQL function (*dats_days_between()* in our example) as argument of an aggregate function (AVG( ) in our example). The final solution will be a CDS view, that reads from another CDS view (view-on-view approach). In a first step, we implement a CDS view that only calculates the number of days between booking date and flight date. The aggregation can then be done by the Open SQL statement.

1. Using ABAP Development Tools for Eclipse, create a new data definition (= DDL Source) based on template *Define View With Join*.
   a) In the *ABAP* perspective of *Eclipse*, right-click on your package and choose *New* → *Other ABAP Repository Object*.
   b) On the next window, choose *Core Data Services* →*Data Definition*.
   c) Enter the name of your package, a name, and a short description for the new repository object and choose *Next*.
   d) Assign the object to your transport request and choose *Next*.
   e) Choose template *Define View With Join* and choose *Finish*.
2. Within the DDL source, set the name of the SQL view within the *sqlViewName* annotation.
   a) In the first line of the editor, replace `sql_view_name` with ZHA400_##_SQL1, where ## stands for your group number.
   See source code extract from the model solution.
3. **Edit the FROM clause of the generated SELECT statement. Replace** `data_source_name` **with** **`SBOOK as B`** **and** `joined_data_source_name` **with** **`SCUSTOM as C`** **to read data from these two tables**.
   a) See source code extract from the model solution.
4. In the FROM clause, replace the generated join condition with **`ON B.customid = C.ID`**
   a) See source code extract from the model solution.
5. Edit the field list of the generated SELECT statement. Read the same fields, in the same sequence as in the SELECT statement of subroutine *get_data_template*. But instead of the two aggregate functions SUM( ) and COUNT(*), only calculate the number of days between booking date and flight date.
   Read the following fields:
   a. From table SCUSTOM: fields ID, NAME, POSTCODE, CITY, COUNTRY.
   b. From table SBOOK: expression dats_days_between(ORDER_DATE, FLDATE). Don't forget to provide an alias for the calculated field (suggested name: *days_ahead*).

Hint:
You can reduce the typing effort by making use of the editor's code completion capabilities.

a) Place the cursor between the curly brackets "{ }" . Then invoke the code-completion by selecting Ctrl + Space on your keyboard.

b) See source code extract from the model solution.

6. After the field list, add a WHERE clause. Exclude all bookings that are cancelled.

   a) See source code extract from the model solution.

7. Why can't you add the ORDER BY addition?

   ORDER BY is not supported in CDS View definitions.

8. Save and activate the data definition.

   a) Choose *Save*.

   b) Choose *Activate*.

   c) Verify that your code matches the following model solution:

Source Code extract from the Model Solution (Data Definition HA400_CDS_S1)

```
@AbapCatalog.sqlViewName: 'HA400_CDS_SQL_S1'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'Solution: CDS view Booking with Customer'

define view HA400_CDS_S1 as select
      from sbook as b inner join scustom as c
         on b.customid = c.id
          {
            c.id,
            c.name,
            c.postcode,
            c.city,
            c.country,
            dats_days_between( b.order_date, b.fldate ) as days_ahead
          }
      where b.cancelled <> 'X'
```

Task 3: Test the CDS View

Test your CDS View by displaying the data preview.

1. Open the data preview for the CDS View.

   a) Right-click anywhere in the data definition (DDL Source) and choose the contextual menu item *Open With → Data Preview*.

Task 4: Use the ABAP CDS View in an Open SQL Statement

In subroutine *get_data_solution* of your program, implement a select statement from your CDS View.

1. Edit your program (either in `SE80` or ADT). In subroutine *get_data_solution*, implement a select statement that reads the relevant data from your CDS view .

> Hint:
> Remember that the CDS view contains neither an ORDER BY clause nor an aggregation of the calculated number of days.

> Note:
> If you use the CDS view name (recommended), you have to use the new Open SQL syntax. If you use the SQL view name (not recommended) you may still use classical Open SQL Syntax.

   a) See source code extract from the model solution.

2. Activate and test your program to verify that the data has been read correctly.

   a) Complete this step as before.

   b) Verify that your code matches the following solution:

Source Code extract from the Model Solution (Program HA400_CDS_VIEW_S1)

```
*&---------------------------------------------------------------------*
*&      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,
           days_sum TYPE i,
           count    TYPE i,
         END OF lty_s_cust.


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

* Targets for Select
  DATA: lt_cust TYPE SORTED TABLE OF lty_s_cust
                     WITH NON-UNIQUE KEY id,
        ls_cust TYPE lty_s_cust.
```

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

  CLEAR ct_customers.

* Note 1: SQL Function dats_days_between( )
*        available in Open SQL as off Re. 7.51

* Note 2: In Open SQL, no expressions are allowed
*        as input for aggregate function AVG( )

  SELECT FROM sbook AS b INNER JOIN scustom AS c
           ON b~customid = c~id
       FIELDS c~id,
              c~name,
              c~postcode,
              c~city,
              c~country,
              SUM( dats_days_between( b~order_date, b~fldate) )
                                                 AS days_sum,
              COUNT( * ) AS count
        WHERE b~cancelled <> 'X'
     GROUP BY c~id, c~name, c~postcode, c~city, c~country
     ORDER BY c~id
         INTO TABLE @lt_cust.



  LOOP AT lt_cust INTO ls_cust.

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

* Note 3: Aggregate function AVG( ) always rounds down
*         ABAP rounds differently by default
*         to achieve identical results we enforce rounding down in ABAP

    ls_customer-days_ahead =
                      round( val = ls_cust-days_sum / ls_cust-count
                             dec = 4
                            mode = cl_abap_math=>round_down ).

    INSERT ls_customer INTO TABLE ct_customers.
  ENDLOOP.


ENDFORM.                    "

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

  SELECT FROM ha400_cds_s1
       FIELDS id,
              name,
              postcode,
              city,
              country,
              AVG( days_ahead )  AS days_ahead
     GROUP BY id, name, postcode, city, country
     ORDER BY id
   INTO TABLE @ct_customers.

ENDFORM.                    "
```

Unit 3
Exercise 10

Use Associations in Core Data Services in ABAP

Business Example

You want to read from CDS views that contain associations to other data sources.

Template:

Report HA400_CDS_VIEW_S1

Solution:

Report HA400_CDS_VIEW_S2

Task 1: Copy and Understand Template Program
In system T8N, which has SAP HANA as a primary database, create a copy of your report ZHA400_##_CDS_1 and place the new program in your package (suggested name: ZHA400_##_CDS_2, where ## is your group number). If you did not complete the previous exercise, you can copy from program HA400_CDS_View_S1 instead. Analyze the source code, activate, and execute the program.

1. Create a copy of the report. Place it in your package ZHA400_## and assign it to your workbench task.
2. Analyze the source code of subroutine *get_data_solution*.
3. Activate and execute the program.

Task 2: Analyze the Definition of a CDS View that Contains an Association
In ABAP Development Tools (ADT), open the definition of CDS View *HA400_Booking_With_Association*. Analyze the DDL Source and display a data preview.

1. In ABAP Development Tools, open the definition of CDS View *HA400_Booking_With_Association*.
2. In the DDL Source, analyze the FROM clause of the SELECT statement and compare it to the FROM clause in the definition of CDS View *HA400_CDS_S1*.

   What difference is there in the FROM clause of the SELECT statement?

   ____

   ____

   ____

3. In the DDL Source, analyze the field list of the SELECT statement and compare it to the field list in the definition of CDS View *HA400_CDS_S1*.

What is the meaning of element _*customer* of the field list?

Why are there no fields NAME, POSTCODE, CITY, COUNTRY?

4. Open the SQL create statement for the SQL view.

   Why is there no mention of table SCUSTOM?

5. Open the data preview for CDS View *HA400_Booking_With_Association*, choose one of the bookings and display details on the associated customer.

Task 3: Use the CDS View with Association in an Open SQL Statement
In subroutine *get_data_solution* of your program, read from CDS view *HA400_Booking_With_Association*.

1. Edit your program. In subroutine *get_data_solution*, replace the CDS View in the FROM clause with CDS view *HA400_Booking_With_Association*.

   Why do you get syntax errors in the field list and the GROUP BY clause?

2. Adjust the field list and the GROUP BY clause to use components of the exposed association.

   Hint:
   in Open SQL, the name of an exposed association has to be prefixed with "\" and the component selector for associations is "-".

3. Activate and test your program to verify that the data has been read correctly.

Use Associations in Core Data Services in ABAP

Business Example

You want to read from CDS views that contain associations to other data sources.

Template:

Report HA400_CDS_VIEW_S1

Solution:

Report HA400_CDS_VIEW_S2

Task 1: Copy and Understand Template Program

In system T8N, which has SAP HANA as a primary database, create a copy of your report ZHA400_##_CDS_1 and place the new program in your package (suggested name: ZHA400_##_CDS_2, where ## is your group number). If you did not complete the previous exercise, you can copy from program HA400_CDS_View_S1 instead. Analyze the source code, activate, and execute the program.

1. Create a copy of the report. Place it in your package ZHA400_## and assign it to your workbench task.
   a) Complete this step as described in the previous exercise.
2. Analyze the source code of subroutine *get_data_solution*.
   a) Complete this step as described in the previous exercise.
3. Activate and execute the program.
   a) Complete this step as described in the previous exercise.

Task 2: Analyze the Definition of a CDS View that Contains an Association

In ABAP Development Tools (ADT), open the definition of CDS View *HA400_Booking_With_Association*. Analyze the DDL Source and display a data preview.

1. In ABAP Development Tools, open the definition of CDS View *HA400_Booking_With_Association*.
   a) Complete this step as described in the previous exercise.
2. In the DDL Source, analyze the FROM clause of the SELECT statement and compare it to the FROM clause in the definition of CDS View *HA400_CDS_S1*.
   a) Complete this step as described in the previous exercise.

What difference is there in the FROM clause of the SELECT statement?

Instead of a join of database tables SBOOK and SCUSTOM, the view selects from database table SBOOK only. In addition, it defines an association to database table SCUSTOM.

3. In the DDL Source, analyze the field list of the SELECT statement and compare it to the field list in the definition of CDS View *HA400_CDS_S1*.
   a) Complete this step as described in the previous exercise.

   What is the meaning of element *_customer* of the field list?

   This is the name of the association, defined in the FROM clause. By listing the association name in the element list, the view exposes the association, which means it makes the association visible to a consumer of the view.

   Why are there no fields NAME, POSTCODE, CITY, COUNTRY?

   This information is accessible through the exposed association _Customer. It would be redundant as well as ineffective to list them.

4. Open the SQL create statement for the SQL view.
   a) In the editor window of the data definition (DDL Source), right-click and choose the contextual menu item *Show SQL CREATE statement*.

   Why is there no mention of table SCUSTOM?

   Associations are not evaluated immediately. Table SCUSTOM will be joined to table SBOOK if – and only if – the consumer of the view actually accesses fields from the associated table.

5. Open the data preview for CDS View *HA400_Booking_With_Association*, choose one of the bookings and display details on the associated customer.
   a) Right-click anywhere in the data definition (DDL Source) and choose the contextual menu item *Open With → Data Preview*.
   b) Right-click one of the data records and choose *Follow Association*.
   c) In the dialog box, choose *_customer → scustom[0..1]*.

Task 3: Use the CDS View with Association in an Open SQL Statement

In subroutine *get_data_solution* of your program, read from CDS view *HA400_Booking_With_Association*.

1. Edit your program. In subroutine *get_data_solution*, replace the CDS View in the FROM clause with CDS view *HA400_Booking_With_Association*.
   a) See source code extract from the model solution.

Why do you get syntax errors in the field list and the GROUP BY clause?

The new CDS view does not define fields NAME, POSTCODE, CITY, COUNTRY. It exposes association _*Customer*, instead.

2. Adjust the field list and the GROUP BY clause to use components of the exposed association.

Hint:
in Open SQL, the name of an exposed association has to be prefixed with “\” and the component selector for associations is “-”.

   a) See source code extract from the model solution.

3. Activate and test your program to verify that the data has been read correctly.

   a) Complete this step as before.

   b) Verify that your code matches the following solution:

Source Code extract from the Model Solution (Program HA400_CDS_VIEW_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,
           days_sum TYPE i,
           count    TYPE i,
         END OF lty_s_cust.


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

* Targets for Select
  DATA: lt_cust TYPE SORTED TABLE OF lty_s_cust
                     WITH NON-UNIQUE KEY id,
        ls_cust TYPE lty_s_cust.


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

```
  CLEAR ct_customers.

* Note 1: SQL Function dats_days_between( )
*        available in Open SQL as off Re. 7.51

* Note 2: In Open SQL, no expressions are allowed
*        as input for aggregate function AVG( )

  SELECT FROM sbook AS b INNER JOIN scustom AS c
            ON b~customid = c~id
        FIELDS c~id,
               c~name,
               c~postcode,
               c~city,
               c~country,
               SUM( dats_days_between( b~order_date, b~fldate) )
                                                   AS days_sum,
               COUNT( * ) AS count
         WHERE b~cancelled <> 'X'
      GROUP BY c~id, c~name, c~postcode, c~city, c~country
      ORDER BY c~id
          INTO TABLE @lt_cust.



  LOOP AT lt_cust INTO ls_cust.

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

* Note 3: Aggregate function AVG( ) always rounds down
*         ABAP rounds differently by default
*         to achieve identical results we enforce rounding down in ABAP

    ls_customer-days_ahead =
                      round( val = ls_cust-days_sum / ls_cust-count
                             dec = 4
                            mode = cl_abap_math=>round_down ).

    INSERT ls_customer INTO TABLE ct_customers.
  ENDLOOP.

ENDFORM.                    "

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

* SELECT FROM ha400_booking_with_days
*       FIELDS customid,
*              name,
*              postcode,
*              city,
*              country,
*               AVG( days_ahead ) AS days_ahead
*      GROUP BY customid, name, postcode, city, country
*      ORDER BY customid
*    INTO TABLE @ct_customers.
*
* With use of associations
*
  SELECT FROM Ha400_Booking_With_Association
       FIELDS customid as id,
              \_customer-name,
              \_customer-postcode,
              \_customer-city,
              \_customer-country,
              AVG( days_ahead ) AS days_ahead
     GROUP BY customid,
              \_customer-name,
              \_customer-postcode,
              \_customer-city,
              \_customer-country
    ORDER BY  id
  INTO TABLE @ct_customers.

ENDFORM.                    "
```

Optional: Define and Use a CDS View with Input Parameter

Business Example

You have a program that reads from an existing CDS View. You now want to define your own CDS view to move the complete aggregation expression into the CDS view.

Template:

ABAP program: HA400_CDS_VIEW_T3

Solution:

ABAP program: HA400_CDS_VIEW_S3
Data definition: HA400_CDS_S3 (Defining CDS View HA400_CDS_S3 and SQL View HA400_CDS_SQL_S3)

Task 1: Copy and Understand the Template Program
In system T8N, which has SAP HANA as a primary database, create a copy of report HA400_CDS_VIEW_T3 in your package (suggested name: ZHA400_##_CDS_3, 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_## and assign it to your workbench task.
2. Analyze the source code of subroutine *get_data_template*.

   Which data is read and from which database tables?

   ____
   ____
   ____

   What is the purpose of the selection criterion in the WHERE clause?

   ____
   ____
   ____

3. Activate and execute the program.

Task 2: Create a New Data Definition (DDL Source)
You want to move the entire data selection of subroutine *get_data_template* into a CDS view. To do so, create a new ABAP DDL source (suggested name: ZHA400_##_DDL3) which defines an SQL view in the ABAP Dictionary (suggested name: ZHA400_##_SQL3). Define the view in a way

that it reads the same data as the SELECT statement of subroutine *get_data_template* of your program.

Define an input parameter for the CDS view to support exclusion of bookings on flights after a given date.

1. Using ABAP Development Tools for Eclipse, create a new ABAP DDL source (= Data Definition) based on template *Define View With Parameters*.

2. Within the DDL source, set the name of the SQL view within the *sqlViewName* annotation.

3. Edit the WITH PARAMETERS addition of the generated DEFINE VIEW statement. Define a parameter for the flight date up to which bookings should be considered in the aggregation (suggested name: P_UPTO_DATE). To do so, replace `parameter_name` with **`P_UPTO_DATE`**. To define the parameter with dictionary type DATS, replace `parameter_type` with **`abap.dats`**.

4. **Edit the FROM clause of the generated SELECT statement. Replace `data_source_name` with `HA400_Booking_With_Association` to read from CDS view *HA400_Booking_With_Association*.**

5. Edit the field list of the generated SELECT statement. Read the same fields, in the same sequence as in the SELECT statement of subroutine *get_data_template*.

   Read the following fields:

   a. FROM the CDS view itself: CUSTOMID and AVG(days_ahead).

   b. From association *_Customer*: NAME, POSTCODE, CITY, COUNTRY.

   Hint:
   You can reduce the typing effort by making use of the editor's code completion capabilities.

6. After the field list, add a WHERE clause. Exclude all bookings with a flight date after the value of your input parameter.

   Hint:
   In CDS syntax, you have to use prefix *$parameters.* before the parameter name.

7. After the WHERE clause, add a GROUP BY clause with all fields of the field list except the aggregate expression.

   Hint:
   You can have the editor generate the entire GROUP BY clause by using the editor's "Quick Fix" functionality for the respective error message.

8. Why can't you add the ORDER BY addition?

9. Save and activate the data definition.

Task 3: Use Your SQL View in an Open SQL Statement
In subroutine *get_data_solution* of your program, implement a select statement from your newly defined view.

1. Edit your program. In subroutine *get_data_solution*, implement a select statement that reads the relevant data from your new CDS view. Pass the system date (field *sy-datum)* as actual value for the input parameter of the CDS view.

2. Activate and test your program to verify that the data has been read correctly.

Task 4: Optional: Default Value for Input Parameter
Use annotation *@Environment.systemField* to use the system of AS ABAP as the default value for your parameter. Then remove the parameter passing from your Open SQL statement.

1. Edit your data definition (DDL source). Between the key word WITH PARAMETERS and the name of the parameter add the annotation *@Environment.systemField* followed by the value *#SYSTEM_DATE*.

2. Save and activate the data definition.

3. Open the data preview for this view.

   Why do you still have to enter a value for the input parameter?

4. Edit your program. In subroutine *get_data_solution*, remove the parameter passing and brackets after the name of your CDS view.

5. Activate and test your program to verify that the data has been read correctly.

Unit 3
Solution 11

Optional: Define and Use a CDS View with Input Parameter

Business Example

You have a program that reads from an existing CDS View. You now want to define your own CDS view to move the complete aggregation expression into the CDS view.

Template:

ABAP program: HA400_CDS_VIEW_T3

Solution:

ABAP program: HA400_CDS_VIEW_S3
Data definition: HA400_CDS_S3 (Defining CDS View HA400_CDS_S3 and SQL View HA400_CDS_SQL_S3)

Task 1: Copy and Understand the Template Program
In system T8N, which has SAP HANA as a primary database, create a copy of report HA400_CDS_VIEW_T3 in your package (suggested name: ZHA400_##_CDS_3, 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_## and assign it to your workbench task.
   a) Complete this step as before.

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

   Which data is read and from which database tables?

   All data is read from CDS view *HA400_Booking_With_Association*. Besides some customer information (fields CUSTOMID, NAME, POSTCODE, CITY, COUNTRY) there is an aggregation (aggregate function *AVG( )*) of the number of days between booking date and flight date (field DAYS_AHEAD).

   What is the purpose of the selection criterion in the WHERE clause?

   Bookings on future flights should not be considered.

3. Activate and execute the program.
   a) Complete this step as in the previous exercise.

Task 2: Create a New Data Definition (DDL Source)

You want to move the entire data selection of subroutine *get_data_template* into a CDS view. To do so, create a new ABAP DDL source (suggested name: ZHA400_##_DDL3) which defines an SQL view in the ABAP Dictionary (suggested name: ZHA400_##_SQL3). Define the view in a way that it reads the same data as the SELECT statement of subroutine *get_data_template* of your program.

Define an input parameter for the CDS view to support exclusion of bookings on flights after a given date.

1. Using ABAP Development Tools for Eclipse, create a new ABAP DDL source (= Data Definition) based on template *Define View With Parameters*.
   a) In the *ABAP* perspective of *Eclipse*, right-click your package and choose *New* → *Other ABAP Repository Object*.

   b) In the next window, choose *Core Data Services* → *Data Definition*.

   c) Enter the name of your package, a name, and a short description for the new object, and choose *Next*.

   d) Assign the object to your transport request and choose *Next*.

   e) Choose template *Define View With Parameters* and choose *Finish*.

2. Within the DDL source, set the name of the SQL view within the *sqlViewName* annotation.
   a) In the first line of the editor, replace `sql_view_name` with **`ZHA400_##_SQL3`**, where **`##`** stands for your group number.

      See the source code extract from the model solution.

3. Edit the WITH PARAMETERS addition of the generated DEFINE VIEW statement. Define a parameter for the flight date up to which bookings should be considered in the aggregation (suggested name: P_UPTO_DATE). To do so, replace `parameter_name` with **`P_UPTO_DATE`**. To define the parameter with dictionary type DATS, replace `parameter_type` with **`abap.dats`**.
   a) See the source code extract from the model solution.

4. **Edit the FROM clause of the generated SELECT statement. Replace** `data_source_name` **with `HA400_Booking_With_Association` to read from CDS view *HA400_Booking_With_Association*.**
   a) See the source code extract from the model solution.

5. Edit the field list of the generated SELECT statement. Read the same fields, in the same sequence as in the SELECT statement of subroutine *get_data_template*.

   Read the following fields:

   a. FROM the CDS view itself: CUSTOMID and AVG(days_ahead).

   b. From association *_Customer*: NAME, POSTCODE, CITY, COUNTRY.

Hint:
You can reduce the typing effort by making use of the editor's code completion capabilities.

a) Place the cursor between the curly brackets "{ }". Then invoke the code-completion by selecting Ctrl + Space on your keyboard.

b) See the source code extract from the model solution.

6. After the field list, add a WHERE clause. Exclude all bookings with a flight date after the value of your input parameter.

Hint:
In CDS syntax, you have to use prefix *$parameters.* before the parameter name.

a) See the source code extract from the model solution.

7. After the WHERE clause, add a GROUP BY clause with all fields of the field list except the aggregate expression.

Hint:
You can have the editor generate the entire GROUP BY clause by using the editor's "Quick Fix" functionality for the respective error message.

a) Place the cursor on the source code line that contains key word SELECT.

b) Select Ctrl + 1 on your keyboard. Alternatively, you can right-click that code line and choose *Quick Fix.*

c) Choose *Add GROUP BY clause*.

d) See the source code extract from the model solution.

8. Why can't you add the ORDER BY addition?

ORDER BY is not supported in CDS view definitions.

9. Save and activate the data definition.

a) Choose *Save*.

b) Choose *Activate*.

Task 3: Use Your SQL View in an Open SQL Statement

In subroutine *get_data_solution* of your program, implement a select statement from your newly defined view.

1. Edit your program. In subroutine *get_data_solution*, implement a select statement that reads the relevant data from your new CDS view. Pass the system date (field *sy-datum)* as actual value for the input parameter of the CDS view.
   a) Immediately after the CDS view name (without a space), add `( p_upto_date = @sy-datum ).`

   b) See the source code extract from the model solution.

2. Activate and test your program to verify that the data has been read correctly.
   a) Complete this step as before.

Task 4: Optional: Default Value for Input Parameter
Use annotation *@Environment.systemField* to use the system of AS ABAP as the default value for your parameter. Then remove the parameter passing from your Open SQL statement.

1. Edit your data definition (DDL source). Between the key word WITH PARAMETERS and the name of the parameter add the annotation *@Environment.systemField* followed by the value *#SYSTEM_DATE*.
   a) See the source code extract from the model solution.

2. Save and activate the data definition.
   a) Choose *Save*.

   b) Choose *Activate*.

3. Open the data preview for this view.
   a) Right-click anywhere in the data definition (DDL source) and choose the contextual menu item *Open With → Data Preview.*

   Why do you still have to enter a value for the input parameter?

   Because annotation *@Environment.systemField* is only relevant when accessing the view from ABAP via Open SQL.

4. Edit your program. In subroutine *get_data_solution*, remove the parameter passing and brackets after the name of your CDS view.
   a) See commented coding in the source code extract from the model solution.

5. Activate and test your program to verify that the data has been read correctly.
   a) Complete this step as before.

   b) Verify that your code matches the following solution:

   Data Definition HA400_CDS_S3

```
@AbapCatalog.sqlViewName: 'HA400_CDS_SQL_S3'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'CDS view with input parameter'

define view HA400_CDS_S3
```

```
       with parameters
            // optional: use system date of AS ABAP as
            //            default value of this parameter
            @Environment.systemField: #SYSTEM_DATE
            p_upto_date : abap.dats
  as select
       from HA400_Booking_With_Association
          {
           customid as id,

           _customer.name,
           _customer.postcode,
           _customer.city,
           _customer.country,

           avg( days_ahead ) as days_ahead
         }

     where fldate <= $parameters.p_upto_date

  group by  customid,
           _customer.name,
           _customer.postcode,
           _customer.city,
           _customer.country
```

ABAP program HA400_CDS_VIEW_S3

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

FORM get_data_template CHANGING ct_customers TYPE ty_t_customers.

  SELECT FROM HA400_Booking_With_Association
       FIELDS customid AS id,
              \_customer-name,
              \_customer-postcode,
              \_customer-city,
              \_customer-country,

              AVG( days_ahead ) AS days_ahead

        WHERE fldate <= @sy-datum
     GROUP BY customid,
              \_customer-name,
              \_customer-postcode,
              \_customer-city,
              \_customer-country
    ORDER BY customid
  INTO TABLE @ct_customers.

ENDFORM.                        "

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

```
  SELECT FROM ha400_cds_s3( p_upto_date = @sy-datum )
       FIELDS id,
              name,
              postcode,
              city,
              country,
              days_ahead
     ORDER BY id
   INTO TABLE @ct_customers.

* Optional: if parameter is implicitly set to system field sy-datum

*   SELECT FROM ha400_cds_s3
*        FIELDS id,
*               name,
*               postcode,
*               city,
*               country,
*               days_ahead
*      ORDER BY id
*    INTO TABLE @ct_customers.

ENDFORM.                    "
```

Unit 4
Exercise 12

Create and Call an ABAP-Managed Database Procedure

Business Example

You want to implement an ABAP-Managed Database Procedure to push down application logic onto the HANA database.

Template:

Report HA400_AMDP_T1
Class CL_HA400_AMDP_T1

Solution:

Report HA400_AMDP_S1
Class CL_HA400_AMDP_S1 (including AMDP method *early_bird_and_last_minute*)

Task 1: Copy and Understand Template Program
In system T8N, which uses SAP HANA as its primary database, create a copy of report HA400_AMDP_T1 in your package (suggested name: ZHA400_##_AMDP, where ## is your group number). Analyze the source code, activate and execute the program.

Note:
Be sure to copy the program with **all subobjects**, such as screens, GUI status and GUI title.

1. Create a copy of the report. Place it in your package ZHA400_## 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.

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

3. Activate the program and all its subobjects, and execute the program.

Task 2: Copy and Understand Template Class
Create a copy of class CL_HA400_AMDP_T1 in your package (suggested name: ZCL_HA400_##_AMDP, where ## is your group number) and analyze it.

1. Create a copy of the class. Place it in your package ZHA400_## and assign it to your workbench task.
2. Analyze the definition of the class.

   Which components does the class contain?

   ____

   ____

   ____

Task 3: Define and Implement an ABAP-Managed Database Procedure

In your copy of the template class, make the existing static method *early_bird_and_last_minute* an ABAP-Managed Database Procedure (AMDP). To do so, adjust the definition of the class and the method where necessary. Then implement the method with two SELECT statements similar to those you find in subroutine *get_data_template*. Finally, activate and test your class.

> Hint:
> As soon as a global class contains an AMDP, you cannot edit it in a classic ABAP Workbench anymore. You have to use ABAP Development Tools for Eclipse to edit your class.

1. Go to the *ABAP* perspective in *Eclipse* and open your copy of the template class there. Go to the implementation of method *early_bird_and_last_minute* and add the `BY DATABASE PROCEDURE...` addition to indicate that the method is an ABAP-managed database procedure.
2. Which changes are necessary to the class definition and the method definition?

   ____

   ____

   ____

3. Add the marker interface IF_AMDP_MARKER_HDB to your class and add value(...) to all parameters of the method.
4. Edit the method implementation. Write a value assignment to fill output parameter *et_early* (assignment operator “=”). On the right-hand side of the assignment, implement a SELECT statement that is similar to the first SELECT statement in subroutine *get_data_template* of your program.

Hint:
Note the following important syntax differences between SAP HANA SQL and (enhanced) Open SQL:

- No FIELDS addition in SAP HANA SQL. Move the field list to before the FROM clause.
- In SQLScript, you cannot access CDS views directly (only the SQL views are created on the database). Read from the generated SQL view instead.
- No implicit client handling in SAP HANA SQL. Add a WHERE clause to restrict the client.
- No system fields sy-nnnnn in SAP HANA SQLScript. Use SAP HANA built in function *session_context( )* with value 'CLIENT' to access the ABAP session client.
- In SAP HANA SQLScript you need to escape host variables with character ":" instead of character "@".
- In an ORDER BY clause, you specify the sort order with DESC and ASC instead of DESCENDING and ASCENDING.
- No addition UP TO ... ROWS. Use addition TOP ... right after the key word SELECT.

5. Add another value assignment with output parameter *et_late* on the left and a second SELECT statement on the right. The only difference between the two SELECT statements should be the sort order (ascending instead of descending).
6. Use addition `USING ...` to make the ABAP Dictionary object HA400_CDS_SQL_S1 (SQL View) known inside the AMDP source code.
7. Optional: Use addition OPTIONS READ-ONLY to document that the procedure does not contain any write access to the database.
8. Activate and test the class.

Task 4: Call the ABAP-Managed Database Procedure
In subroutine *get_data_solution* of your program ZHA400_##_AMDP, use your newly defined ABAP-Managed Database Procedure to read the data from the database.

1. Edit your program. In subroutine *get_data_solution*, call the static method that defines the AMDP.
2. Activate and test your program.

Task 5: Debug the AMDP
In ABAP Development Tools in Eclipse (ADT), set break-points in the AMDP method and execute the program that calls the method.

1. In the *ABAP* perspective in Eclipse, navigate to the implementation of method *early_bird_and_last_minute* in your class ZCL_HA400_##_AMDP.
2. In the SQLScript code of the AMDP, set two breakpoints, one at each select statement.
3. Remain in the *ABAP* perspective in Eclipse, navigate back to program, and execute the ABAP report that calls the database procedure.
4. In the *Debugger* perspective of Eclipse, analyze the content of the AMDP procedure’s parameters.
5. Resume execution to the next breakpoint in the AMDP method and display the content of parameter *et_late*.
6. Resume execution until the application displays the result.
7. Close the application and the *Debug* perspective.

Unit 4
Solution 12

Create and Call an ABAP-Managed Database Procedure

Business Example

You want to implement an ABAP-Managed Database Procedure to push down application logic onto the HANA database.

Template:

Report HA400_AMDP_T1
Class CL_HA400_AMDP_T1

Solution:

Report HA400_AMDP_S1
Class CL_HA400_AMDP_S1 (including AMDP method *early_bird_and_last_minute*)

Task 1: Copy and Understand Template Program

In system T8N, which uses SAP HANA as its primary database, create a copy of report HA400_AMDP_T1 in your package (suggested name: ZHA400_##_AMDP, where ## is your group number). Analyze the source code, activate and execute the program.

Note:
Be sure to copy the program with **all subobjects**, such as screens, GUI status and GUI title.

1. Create a copy of the report. Place it in your package ZHA400_## 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.

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

   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 a list of customers with the shortest average time between order date and flight date.
3. Activate the program and all its subobjects, and execute the program.

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

Task 2: Copy and Understand Template Class

Create a copy of class CL_HA400_AMDP_T1 in your package (suggested name: ZCL_HA400_##_AMDP, where ## is your group number) and analyze it.

1. Create a copy of the class. Place it in your package ZHA400_## and assign it to your workbench task.

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

2. Analyze the definition of the class.

   Which components does the class contain?

   Structure type *ty_s_customer*, table type *ty_t_customers*, and static method *early_bird_and_last_minute* with 4 parameters (*iv_number*, *iv_mandt*, *et_early* and *et_late*)

Task 3: Define and Implement an ABAP-Managed Database Procedure

In your copy of the template class, make the existing static method *early_bird_and_last_minute* an ABAP-Managed Database Procedure (AMDP). To do so, adjust the definition of the class and the method where necessary. Then implement the method with two SELECT statements similar to those you find in subroutine *get_data_template*. Finally, activate and test your class.

Hint:
As soon as a global class contains an AMDP, you cannot edit it in a classic ABAP Workbench anymore. You have to use ABAP Development Tools for Eclipse to edit your class.

1. Go to the *ABAP* perspective in *Eclipse* and open your copy of the template class there. Go to the implementation of method *early_bird_and_last_minute* and add the `BY DATABASE PROCEDURE...` addition to indicate that the method is an ABAP-managed database procedure.

   a) Open the *ABAP* perspective in *Eclipse*.

   b) In the *Project* navigator on the left, expand your project and your package.

   c) Expand the *Source Library* and double-click your class to open it in the *ABAP editor*.

   d) Edit the source code as usual. For the result, see source code extract from the model solution.

2. Which changes are necessary to the class definition and the method definition?

   The class has to implement interface IF_AMDP_MARKER_HDB and all parameters of the method have to be passed *by value*.

3. Add the marker interface IF_AMDP_MARKER_HDB to your class and add value(...) to all parameters of the method.

   a) See source code extract from the model solution.

4. Edit the method implementation. Write a value assignment to fill output parameter *et_early* (assignment operator “=”). On the right-hand side of the assignment, implement a SELECT statement that is similar to the first SELECT statement in subroutine *get_data_template* of your program.

   Hint:
   Note the following important syntax differences between SAP HANA SQL and (enhanced) Open SQL:

   - No FIELDS addition in SAP HANA SQL. Move the field list to before the FROM clause.
   - In SQLScript, you cannot access CDS views directly (only the SQL views are created on the database). Read from the generated SQL view instead.
   - No implicit client handling in SAP HANA SQL. Add a WHERE clause to restrict the client.
   - No system fields sy-nnnnn in SAP HANA SQLScript. Use SAP HANA built in function *session_context( )* with value 'CLIENT' to access the ABAP session client.
   - In SAP HANA SQLScript you need to escape host variables with character “:” instead of character “@”.
   - In an ORDER BY clause, you specify the sort order with DESC and ASC instead of DESCENDING and ASCENDING.
   - No addition UP TO ... ROWS. Use addition TOP ... right after the key word SELECT.

   a) See source code extract from the model solution.

5. Add another value assignment with output parameter *et_late* on the left and a second SELECT statement on the right. The only difference between the two SELECT statements should be the sort order (ascending instead of descending).

   a) See source code extract from the model solution.

6. Use addition `USING ...` to make the ABAP Dictionary object HA400_CDS_SQL_S1 (SQL View) known inside the AMDP source code.

   a) See source code extract from the model solution.

7. Optional: Use addition OPTIONS READ-ONLY to document that the procedure does not contain any write access to the database.

   a) See source code extract from the model solution.

8. Activate and test the class.

   a) Choose *Activate* from the toolbar or select Ctrl + F3 on your keyboard.

   b) Choose *Run As ...* → *ABAP Application* from the toolbar or select F8 on your keyboard.

c) Verify that your code matches the following solution:

Source code extract from model solution (Class CL_HA400_AMDP_S1)

```
CLASS cl_ha400_amdp_s1 DEFINITION
  PUBLIC
  FINAL
  CREATE PUBLIC .

  PUBLIC SECTION.

    INTERFACES if_amdp_marker_hdb.

    TYPES: BEGIN OF ty_s_customer,
             id         TYPE scustom-id,
             name       TYPE scustom-name,
             postcode   TYPE scustom-postcode,
             city       TYPE scustom-city,
             country    TYPE scustom-country,
             days_ahead TYPE ha400_avgnd,
           END OF ty_s_customer.

    TYPES: ty_t_customers TYPE STANDARD TABLE OF ty_s_customer
                             WITH NON-UNIQUE KEY id name.

    CLASS-METHODS early_bird_and_last_minute
      IMPORTING VALUE(iv_number) TYPE i
      EXPORTING VALUE(et_early)  TYPE ty_t_customers
                VALUE(et_late)   TYPE ty_t_customers.

ENDCLASS.

CLASS CL_HA400_AMDP_S1 IMPLEMENTATION.


  METHOD early_bird_and_last_minute
                          BY DATABASE PROCEDURE
                          FOR HDB LANGUAGE SQLSCRIPT
                          OPTIONS READ-ONLY
                          USING HA400_CDS_SQL_S1.


 et_early = select top :iv_number
                   id,
                   name,
                   postcode,
                   city,
                   country,
                   avg( days_ahead ) as days_ahead
              from HA400_CDS_SQL_S1

             where mandt = session_context('CLIENT')

          group by id,
                   name,
                   postcode,
                   city,
                   country
          order by days_ahead DESC;

    et_late = select top :iv_number
                   id,
                   name,
                   postcode,
                   city,
                   country,
                   avg( days_ahead ) as days_ahead
              from HA400_CDS_SQL_S1

             where mandt = session_context('CLIENT')

          group by id,
                   name,
                   postcode,
                   city,
                   country

             order by days_ahead ASC;

  ENDMETHOD.
ENDCLASS.
```

Task 4: Call the ABAP-Managed Database Procedure

In subroutine *get_data_solution* of your program ZHA400_##_AMDP, use your newly defined ABAP-Managed Database Procedure to read the data from the database.

1. Edit your program. In subroutine *get_data_solution*, call the static method that defines the AMDP.

   a) See source code extract from the model solution.

2. Activate and test your program.

   a) Activate and execute the program as you have learned in other classes.

   b) Verify that your code matches the following solution:

   Source code extract from model solution (Program HA400_AMDP_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.
* early bookers


  SELECT from ha400_booking_with_days
       fields customid as id,
              name,
              postcode,
              city,
              country,
              avg( days_ahead ) as days_ahead
```

```
     GROUP BY customid,
              name,
              postcode,
              city,
              country

     ORDER BY days_ahead DESCENDING

   INTO TABLE @ct_cust_early
        UP TO @pv_number ROWS .

* late bookers

  SELECT FROM ha400_booking_with_days
       FIELDS customid as id,
              name,
              postcode,
              city,
              country,
              avg( days_ahead ) as days_ahead

     GROUP BY customid,
              name,
              postcode,
              city,
              country

     ORDER BY days_ahead ASCENDING

   INTO TABLE @ct_cust_late
        UP TO @pv_number ROWS .

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.

* Call an AMDP here

  cl_ha400_amdp_s1=>early_bird_and_last_minute(
    EXPORTING
      iv_number = pv_number
    IMPORTING
      et_early  = ct_cust_early
      et_late   = ct_cust_late ).


ENDFORM.                     "
```

Task 5: Debug the AMDP

In ABAP Development Tools in Eclipse (ADT), set break-points in the AMDP method and execute the program that calls the method.

1. In the *ABAP* perspective in Eclipse, navigate to the implementation of method *early_bird_and_last_minute* in your class ZCL_HA400_##_AMDP.

   a) Perform this step as before.

2. In the SQLScript code of the AMDP, set two breakpoints, one at each select statement.
   a) In each of the two lines with the select statements, double-click the grey bar to the left of the code.

3. Remain in the *ABAP* perspective in Eclipse, navigate back to program, and execute the ABAP report that calls the database procedure.
   a) Return to the source code of your program ZHA400_##_AMDP and execute the program by selecting **`F8`** on your keyboard.

   b) On the selection screen, select *Solution only* to make sure the AMDP is called.

   c) When prompted, confirm the switch to the *Debugger* perspective.

4. In the *Debugger* perspective of Eclipse, analyze the content of the AMDP procedure's parameters.
   a) On the right, you find a window labelled *Variables* with a list of the data objects.

      What is the content of the import parameters?

      Import parameter *iv_number* contains the user input from the selection screen. Import parameter *iv_mandt* contains the current logon client.

      What is the content of the export parameters?

      The export parameters of the AMDP, *et_late* and *et_early* contain no data yet.

5. Resume execution to the next breakpoint in the AMDP method and display the content of parameter *et_late*.
   a) Select F8 on your keyboard or choose *Resume*.

   b) In the variable list, right-click parameter *et_late* and choose *Show in Data Preview*.

6. Resume execution until the application displays the result.
   a) Select F8 on your keybaord or choose *Resume*.

7. Close the application and the *Debug* perspective.
   a) Close the tab with the embedded SAP GUI window showing the results.

   b) In the list of open perspectives, right-click *Debug* and choose *Close*.

Unit 5
Exercise 13

Implement a Value Help with Fuzzy Search

Business Example

You have a report which takes a customer ID as input and lists how many flights the given customer booked for each carrier. You want to improve the usability of the report by enabling a type-ahead value help for the customer ID input field.

Template Objects:

Search Help: HA400_CUST_T1
Structure: HA400_S_CUST_T1
Report: HA400_FUZZY_T2

Solution Objects:

Search Help: HA400_CUST_S1
Structure: HA400_S_CUST_S1
Report: HA400_FUZZY_S2

Task 1: Copy and Analyze the Template Program

In system T8N, which uses SAP HANA as its primary database, create a copy of template report HA400_FUZZY_T2 (suggested name: ZHA400_##_FUZZY_2). Activate and test your copy. Analyze the coding and find the object that defines the value help on the selection screen.

1. Create a copy of report HA400_FUZZY_T2. Place it in your package ZHA400_## and assign it to your workbench task.
2. Activate and test your copy.

   Is there a value help (often referred to as F4 help) for the input field?

3. Analyze your copy. Where does the value help come from?

   The input help comes from search help __________(1) that is assigned to component __________(2) of structure __________(3).

Task 2: Copy Search Help and Structure and Use the Copies in Your Program

Copy the search help and the structure. Use these copies in your program so that the value help for the input field is defined by your own search help.

1. Create a copy of search help HA400_CUST_T1 (suggested name: ZHA400_##_CUST), place it in your package ZHA400_## and assign it to your workbench task.
2. Activate the search help.
3. Create a copy of structure HA400_S_CUST_T1 (suggested name: ZHA400_S_##_CUST), place it in your package ZHA400_## and assign it to your workbench task.
4. Open your copy of structure HA400_S_CUST_T1. Instead of search help HA400_CUST_S1, assign your own search help to component ID.
5. Activate the structure.
6. Edit your program. In the definition of the input field, replace structure HA400_S_CUST_T1 with your own structure. As a result the value help for the input field is now defined by your own search help.

Task 3: Adjust the Search Help to Make it Support Proposal Search and Fuzzy Search

Adjust the definition of your search help to make it support proposal search and fuzzy search. Activate it and check the result by executing your program again.

1. Adjust the definition of your search help ZHA400_##_CUST to make it support proposal search.
2. Execute your program and check the impact on the value help.
3. Adjust the definition of your search help ZHA400_##_CUST again, this time to make it support fuzzy search.
4. Execute your program once more and check the impact on the value help.

Unit 5
Solution 13

Implement a Value Help with Fuzzy Search

Business Example

You have a report which takes a customer ID as input and lists how many flights the given customer booked for each carrier. You want to improve the usability of the report by enabling a type-ahead value help for the customer ID input field.

Template Objects:

Search Help: HA400_CUST_T1
Structure: HA400_S_CUST_T1
Report: HA400_FUZZY_T2

Solution Objects:

Search Help: HA400_CUST_S1
Structure: HA400_S_CUST_S1
Report: HA400_FUZZY_S2

Task 1: Copy and Analyze the Template Program
In system T8N, which uses SAP HANA as its primary database, create a copy of template report HA400_FUZZY_T2 (suggested name: ZHA400_##_FUZZY_2). Activate and test your copy. Analyze the coding and find the object that defines the value help on the selection screen.

1. Create a copy of report HA400_FUZZY_T2. Place it in your package ZHA400_## and assign it to your workbench task.
   a) Complete this step as before.

2. Activate and test your copy.
   a) Perform this step as you learned to do in previous classes.

   Is there a value help (often referred to as F4 help) for the input field?

   Yes

3. Analyze your copy. Where does the value help come from?
   a) Look for the definition of the parameter (statement `PARAMETERS`).
   b) Navigate to the data-type (`HA400_S_CUST_T1-ID`).
   c) In the definition of structure HA400_S_CUST_T1, look for the search help assigned to component ID.

The input help comes from search help __________(1) that is assigned to component __________(2) of structure __________(3).

(1) HA400_CUST_T1, (2) ID, (3) HA400_S_CUST_T1.

Task 2: Copy Search Help and Structure and Use the Copies in Your Program
Copy the search help and the structure. Use these copies in your program so that the value help for the input field is defined by your own search help.

1. Create a copy of search help HA400_CUST_T1 (suggested name: ZHA400_##_CUST), place it in your package ZHA400_## and assign it to your workbench task.
   a) Perform this step as you learned to do in previous classes.
2. Activate the search help.
   a) Perform this step as you learned to do in previous classes.
3. Create a copy of structure HA400_S_CUST_T1 (suggested name: ZHA400_S_##_CUST), place it in your package ZHA400_## and assign it to your workbench task.
   a) Perform this step as you learned to do in previous classes.
4. Open your copy of structure HA400_S_CUST_T1. Instead of search help HA400_CUST_S1, assign your own search help to component ID.
   a) Perform this step as you learned to do in previous classes.
5. Activate the structure.
   a) Perform this step as you learned to do in previous classes.
6. Edit your program. In the definition of the input field, replace structure HA400_S_CUST_T1 with your own structure. As a result the value help for the input field is now defined by your own search help.
   a) `PARAMETERS pa_cust TYPE ZHA400_S_##_CUST-ID.`

Task 3: Adjust the Search Help to Make it Support Proposal Search and Fuzzy Search
Adjust the definition of your search help to make it support proposal search and fuzzy search. Activate it and check the result by executing your program again.

1. Adjust the definition of your search help ZHA400_##_CUST to make it support proposal search.
   a) Open the search help and switch to change mode.
   b) In the *Enhanced Options* frame, select the *Autosuggest in input fields* indicator.
   c) Activate the search help.
2. Execute your program and check the impact on the value help.
   a) Execute your program.
   b) Place the cursor inside the input field and start typing a customer name.
3. Adjust the definition of your search help ZHA400_##_CUST again, this time to make it support fuzzy search.

   a) Open the search help and switch to change mode.
   b) In the *Enhanced Options* frame, select the *Multi-Column Fulltext Seach (Database-Specific)* indicator.
   c) Activate the search help.

4. Execute your program once more and check the impact on the value help.
   a) Execute your program.
   b) Place the cursor inside the input field and start typing a customer name.

Unit 5
Exercise 14

Use ALV with Integrated Data Access for a CDS View

Business Example

You are asked to develop a report listing, for a range of customers to be specified on the selection screen, how many days on average each customer books flights in advance. You have a program that retrieves this information based on a CDS view. Now you want to improve the user experience by making use of the SAP List Viewer with Integrated Data Access (ALV with IDA).

Template:

Report HA400_ALV_IDA_T1

Solution:

Report HA400_ALV_IDA_S1

Task 1: Copy and Understand Templates

In system T8N, which uses SAP HANA as its primary database, create a copy of program HA400_ALV_IDA_T1 in your package ZHA400_## (suggested name: ZHA400_##_ALV, where ## is your group number). Activate the program.

1. Create a copy of the report. Place it in your package ZHA400_## and assign it to your workbench task.
2. Analyze the source code of subroutine *display_alv_classic*.

   Where is the data retrieved and where does it come from?

   ____
   ____
   ____

   Which ABAP data objects are used in the SELECT statements and for what purpose?

   ____
   ____
   ____

3. Activate the program and test it with default settings.

Task 2: Use the ALV with IDA to Display the Data

Edit your program. Implement the *display_alv_ida* subroutine so that the data from CDS view HA400_CDS_S3 is displayed in the new ALV with integrated data access. Make sure you provide an actual value for the import parameter and to restrict the data to be displayed based on the user input.

1. In the *display_alv_ida* subroutine, declare a reference variable of type ref to IF_SALV_GUI_TABLE_IDA and use it to receive the object from the factory method CREATE_FOR_CDS_VIEW of class CL_SALV_GUI_TABLE_IDA, which creates and returns an instance of the ALV with IDA class. Supply the name of the CDS view (HA400_CDS_S3) for parameter IV_TABLE_NAME.
2. Call method SET_VIEW_PARAMETERS for the ALV instance, to provide the current date for parameter P_UPTO_DATE of the view. To do this, populate an internal table of type if_salv_gui_types_ida=>yt_parameter with an appropriate entry and then pass this internal table to the method.
3. Make the select options in data objects *pt_range_id* and *pt_rang_name* known to the ALV with IDA instance. In a first step, declare a reference variable of type ref to CL_SALV_RANGE_TAB_COLLECTOR, and then create an instance of this class.
4. Call the method ADD_RANGES_FOR_NAME for the above object, once for each select-option on the selection screen. Assign the respective range table to parameter IT_RANGES, and the field name to parameter IV_NAME.
5. Call method GET_COLLECTED_RANGES which exports an internal table of type if_salv_service_types=>yt_named_ranges that contains a combination of the various select-options.
6. Call method SET_SELECT_OPTIONS for the ALV with IDA instance to pass the internal table from the previous step to parameter IT_RANGES.
7. Call functional method FULLSCREEN for the ALV with IDA instance, which returns a reference of type IF_SALV_GUI_FULLSCREEN_IDA. For this new object, call method DISPLAY.
8. Activate and test your program.

Unit 5
Solution 14

Use ALV with Integrated Data Access for a CDS View

Business Example

You are asked to develop a report listing, for a range of customers to be specified on the selection screen, how many days on average each customer books flights in advance. You have a program that retrieves this information based on a CDS view. Now you want to improve the user experience by making use of the SAP List Viewer with Integrated Data Access (ALV with IDA).

Template:

Report HA400_ALV_IDA_T1

Solution:

Report HA400_ALV_IDA_S1

Task 1: Copy and Understand Templates

In system T8N, which uses SAP HANA as its primary database, create a copy of program HA400_ALV_IDA_T1 in your package ZHA400_## (suggested name: ZHA400_##_ALV, where ## is your group number). Activate the program.

1. Create a copy of the report. Place it in your package ZHA400_## and assign it to your workbench task.
   a) Complete this step as you learned before.
2. Analyze the source code of subroutine *display_alv_classic*.
   a) Open the source code in the editor.

   Where is the data retrieved and where does it come from?

   The data is retrieved in a SELECT statement from CDS view HA400_CDS_S3.

   Which ABAP data objects are used in the SELECT statements and for what purpose?

   Internal table *lt_customers* is used as target for the read data. Range tables *pt_range_id* and *pt_range_name* are used as selection criteria in the WHERE clause. System field *sy-datum* is used as actual parameter for the CDS view's input parameter *p_upto_date*.
3. Activate the program and test it with default settings.
   a) Complete this step as before.

Task 2: Use the ALV with IDA to Display the Data

Edit your program. Implement the *display_alv_ida* subroutine so that the data from CDS view HA400_CDS_S3 is displayed in the new ALV with integrated data access. Make sure you provide an actual value for the import parameter and to restrict the data to be displayed based on the user input.

1. In the *display_alv_ida* subroutine, declare a reference variable of type ref to IF_SALV_GUI_TABLE_IDA and use it to receive the object from the factory method CREATE_FOR_CDS_VIEW of class CL_SALV_GUI_TABLE_IDA, which creates and returns an instance of the ALV with IDA class. Supply the name of the CDS view (HA400_CDS_S3) for parameter IV_TABLE_NAME.
   a) See source code extract from model solution.
2. Call method SET_VIEW_PARAMETERS for the ALV instance, to provide the current date for parameter P_UPTO_DATE of the view. To do this, populate an internal table of type if_salv_gui_types_ida=>yt_parameter with an appropriate entry and then pass this internal table to the method.
   a) See source code extract from model solution.
3. Make the select options in data objects *pt_range_id* and *pt_rang_name* known to the ALV with IDA instance. In a first step, declare a reference variable of type ref to CL_SALV_RANGE_TAB_COLLECTOR, and then create an instance of this class.
   a) See source code extract from model solution.
4. Call the method ADD_RANGES_FOR_NAME for the above object, once for each select-option on the selection screen. Assign the respective range table to parameter IT_RANGES, and the field name to parameter IV_NAME.
   a) See source code extract from model solution.
5. Call method GET_COLLECTED_RANGES which exports an internal table of type if_salv_service_types=>yt_named_ranges that contains a combination of the various select-options.
   a) See source code extract from model solution.
6. Call method SET_SELECT_OPTIONS for the ALV with IDA instance to pass the internal table from the previous step to parameter IT_RANGES.
   a) See source code extract from model solution.
7. Call functional method FULLSCREEN for the ALV with IDA instance, which returns a reference of type IF_SALV_GUI_FULLSCREEN_IDA. For this new object, call method DISPLAY.
   a) See source code extract from model solution.
8. Activate and test your program.
   a) Perform this step as before.
   b) Verify that your code matches the following solution:

Source code extract from model solution (Program HA400_ALV_IDA_S1)

```
*&---------------------------------------------------------------------*
*&      Form  display_alv_ida
*&---------------------------------------------------------------------*
FORM display_alv_ida USING pt_range_id   LIKE so_id[]
                            pt_range_name LIKE so_name[].

* control specific:
  DATA: lo_salv TYPE REF TO if_salv_gui_table_ida.

  DATA:
    lo_range_collector TYPE REF TO cl_salv_range_tab_collector,
    lt_named_ranges    TYPE if_salv_service_types=>yt_named_ranges.


  DATA: lo_fcat TYPE REF TO if_salv_gui_field_catalog_ida,
        lo_exc  TYPE REF TO cx_salv_ida_contract_violation.


  DATA: lt_params TYPE if_salv_gui_types_ida=>yt_parameter,
        ls_param  LIKE LINE OF lt_params.

* Create ALV Instance
*---------------------*
  lo_salv = cl_salv_gui_table_ida=>create_for_cds_view(
              iv_cds_view_name = 'HA400_CDS_S3'
            ).

* Set Input Parameter
*---------------------*
  ls_param-name = 'P_UPTO_DATE'.
  ls_param-value = sy-datum.

  APPEND ls_param TO lt_params.
  lo_salv->set_view_parameters( lt_params ).

* Alternative: Based on expression VALUE (> AS ABAP 7.40)
*lo_salv->set_view_parameters( value #(
*                                        ( name  = 'P_UPTO_DATE'
*                                          value = sy-datum
*                                         )
*                                       )
*                             ).

* Transform Ranges to one Range Table
*----------------------------------------*
  CREATE OBJECT lo_range_collector.

  lo_range_collector->add_ranges_for_name(
      iv_name   = 'ID'
      it_ranges = pt_range_id ).

  lo_range_collector->add_ranges_for_name(
      iv_name   = 'NAME'
      it_ranges = pt_range_name ).

  lo_range_collector->get_collected_ranges(
    IMPORTING
```

```
      et_named_ranges = lt_named_ranges ).

* Set Select Options
*---------------------*
  lo_salv->set_select_options( it_ranges = lt_named_ranges ).

* Invoke fullscreen display
*----------------------------*
  lo_salv->fullscreen( )->display( ).

ENDFORM.                    "display_solution
```

Unit 6
Exercise 15

Case Study Part 1: Code Push Down with ABAP Core Data Services

Business Example

You are asked to improve an ABAP program that determines, for a set of customers to be entered, the total amount (in your local currency) each customer spends for flights. As a first step, you want to push the currency conversion into the database, making use of an ABAP CDS view yet to be developed.

Template:

Report HA400_CASE_STUDY_T1

Solution:

DDL Source HA400_BOOKING_W_CONV_AMOUNT
Report HA400_CASE_STUDY_S1

Task 1: Copy and Analyze Template

In system T8N, which uses SAP HANA as its primary database, create a copy of report HA400_CASE_STUDY_T1 in your package ZHA400_## (suggested name: **`ZHA400_##_CSTUDY_1`**, where ## is your group number). Activate, analyze, and execute the program.

Hint:
Remember that there is no SAP GUI editor for repository object DDL source. You have to develop at least your CDS view in ABAP Development Tools (ADT). The rest of the exercise you can still do in the classical ABAP Workbench. But you might want to take the opportunity and perform this complete exercise in Eclipse.

1. Create a copy of the report and all its subobjects. Place it in your package ZHA400_## and assign it to your workbench task.

   Hint:
   To copy the report using the ABAP Development Tools, use the contextual menu item *Duplicate*.

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

3. Which database tables occur in the FROM clause of the select statement? Which data are read from them?

   ____
   ____
   ____

4. What is calculated in the loop over the customer's flight bookings? How is this done?

   ____
   ____
   ____

5. Activate the program with all its subobjects and execute the program.

Task 2: Create an ABAP CDS View

Using the ABAP Development Tools in Eclipse, create a data definition (= DDL source) that defines an ABAP CDS view which contains the complete calculation logic found in subroutine *get_data*. Make use of the fact that ABAP CDS supports string expressions and currency conversions. Define the view with an import parameter for the target currency of the conversion.

1. Go to the *ABAP* perspective in Eclipse. In the *Project Explorer*, go to your package ZHA400–## on System T8N.

2. In your package ZHA400–##, create a new data definition (suggested name: **`ZHA400_##_BOOKING_W_CONVAM`**).

3. Within the data definition, adjust annotation @*ABAPCatalog.sqlViewName* to specify the name of the SQL view.

4. After the *with parameters* addition, define an input parameter for the target currency of the currency conversion (suggested name: *p_target_curr*). Use code completion to choose the appropriate type for currency codes.

5. Edit the from clause of the SELECT statement. Add all tables, joins and join conditions you find in the SELECT statement in subroutine *get_data*.

   Hint:
   Remember that, inside a DDL source, you use session variable *$session.system_language* instead of system field *sy-langu*.

6. Within the curly brackets { }, edit the field list. Provide exactly the fields defined in local structure type *ty_s_booking* of the ABAP program. Use SQL function *concat_with_space( )* to concatenate ID and NAME into view field CUSTOMER. Use SQL function *currency_conversion( )* to convert the content of field FORCURAM into the target currency and return it via view field CONVAM. Return the input parameter value as view field CURRENCY.

7. Use annotations @*EndUserText.label* to add descriptions for the calculated fields. Use annotations @*Semantics.currencyCode* and @*Semantics.amount.currencyCode* to establish the connection between view fields CONVAM and CURRENCY.

8. Save and activate your DDL source.

Note:
You can ignore the syntax warning that tells you that search help SCUSTOM is not inherited.

Task 3: Use the CDS View to Optimize the Data Retrieval of the Program

Edit your program ZHA400_##_CSTUDY_1. Replace subroutine *get_data* with a new subroutine *get_data_cds*. Optimize the data retrieval in this new subroutine by performing just one `SELECT` from your CDS view.

Hint:
Remember that you need to use the new syntax of Open SQL if you want to select from the CDS entity.

1. Copy subroutine *get_data* to a new subroutine *get_data_cds*.
2. Comment out the call of subroutines *get_data* and replace it with a call of the new subroutine.
3. In subroutine *get_data_cds*, implement an Open SQL SELECT that reads data from the CDS view you just created. Let the SELECT read into data object *ct_bookings* directly. Make sure you provide a value for the input parameter of the CDS view.
4. Activate and test your program.

Unit 6
Solution 15

Case Study Part 1: Code Push Down with ABAP Core Data Services

Business Example

You are asked to improve an ABAP program that determines, for a set of customers to be entered, the total amount (in your local currency) each customer spends for flights. As a first step, you want to push the currency conversion into the database, making use of an ABAP CDS view yet to be developed.

Template:

Report HA400_CASE_STUDY_T1

Solution:

DDL Source HA400_BOOKING_W_CONV_AMOUNT
Report HA400_CASE_STUDY_S1

Task 1: Copy and Analyze Template

In system T8N, which uses SAP HANA as its primary database, create a copy of report HA400_CASE_STUDY_T1 in your package ZHA400_## (suggested name: `ZHA400_##_CSTUDY_1`, where ## is your group number). Activate, analyze, and execute the program.

Hint:
Remember that there is no SAP GUI editor for repository object DDL source. You have to develop at least your CDS view in ABAP Development Tools (ADT). The rest of the exercise you can still do in the classical ABAP Workbench. But you might want to take the opportunity and perform this complete exercise in Eclipse.

1. Create a copy of the report and all its subobjects. Place it in your package ZHA400_## and assign it to your workbench task.

   Hint:
   To copy the report using the ABAP Development Tools, use the contextual menu item *Duplicate*.

   a) Complete this step as you learned before.

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

   a) Complete this step as you learned before.

3. Which database tables occur in the FROM clause of the select statement? Which data are read from them?

   Table SCUSTOM (Customer name and address data), table T005T (country name), table SBOOK (Booking details), and table SPFLI (Departure City and Arrival City).

4. What is calculated in the loop over the customer's flight bookings? How is this done?

   The customer's ID and NAME are concatenated into a single field CUSTOMER. This is done in a string expression using operator &&. The payment in local currency is converted to the currency provided in USING-parameter *pv_taget_currency*. This is done via function module CONVERT_TO_LOCAL_CURRENCY.

5. Activate the program with all its subobjects and execute the program.
   a) Complete this step as you learned to do in previous classes.

Task 2: Create an ABAP CDS View

Using the ABAP Development Tools in Eclipse, create a data definition (= DDL source) that defines an ABAP CDS view which contains the complete calculation logic found in subroutine *get_data*. Make use of the fact that ABAP CDS supports string expressions and currency conversions. Define the view with an import parameter for the target currency of the conversion.

1. Go to the *ABAP* perspective in Eclipse. In the *Project Explorer*, go to your package ZHA400–## on System T8N.
   a) Log on to the ABAP project on system T8N, if you are not already logged on.
   b) Expand the project and open the *Favorite Packages* folder.
   c) Expand your package ZHA400–##, which you added as a favorite package in a previous exercise.
2. In your package ZHA400–##, create a new data definition (suggested name: **`ZHA400_##_BOOKING_W_CONVAM`**).
   a) Right-click the package and choose *New → Other ABAP Repository Object*.
   b) In the tree below, choose *Core Data Services → Data Definition* and choose *Next*.
   c) Enter the name of your package, a name, and a short description for the new object, and choose *Next*.
   d) Assign the object to your transport request and choose *Next*.
   e) Choose *Use the selected template*, choose *Define View with Parameters* and choose *Finished*.
3. Within the data definition, adjust annotation @*ABAPCatalog.sqlViewName* to specify the name of the SQL view.
   a) In the first line of the editor, look for statement @*AbapCatalog.sqlViewName:*.
   b) After the colon, enter a name for the SQL view in single quotes (suggested name: ZHA400_##_BOOK).

See the source code extract from the model solution.

4. After the *with parameters* addition, define an input parameter for the target currency of the currency conversion (suggested name: *p_target_curr*). Use code completion to choose the appropriate type for currency codes.
   a) See the source code extract from the model solution.

5. Edit the from clause of the SELECT statement. Add all tables, joins and join conditions you find in the SELECT statement in subroutine *get_data*.

   Hint:
   Remember that, inside a DDL source, you use session variable *$session.system_language* instead of system field *sy-langu*.

   a) See the source code extract from the model solution.

6. Within the curly brackets { }, edit the field list. Provide exactly the fields defined in local structure type *ty_s_booking* of the ABAP program. Use SQL function *concat_with_space( )* to concatenate ID and NAME into view field CUSTOMER. Use SQL function *currency_conversion( )* to convert the content of field FORCURAM into the target currency and return it via view field CONVAM. Return the input parameter value as view field CURRENCY.
   a) See the source code extract from the model solution.

7. Use annotations *@EndUserText.label* to add descriptions for the calculated fields. Use annotations *@Semantics.currencyCode* and *@Semantics.amount.currencyCode* to establish the connection between view fields CONVAM and CURRENCY.
   a) See the source code extract from the model solution.

8. Save and activate your DDL source.

   Note:
   You can ignore the syntax warning that tells you that search help SCUSTOM is not inherited.

   a) Choose *Save*.

   b) Choose *Activate*.

Task 3: Use the CDS View to Optimize the Data Retrieval of the Program
Edit your program ZHA400_##_CSTUDY_1. Replace subroutine *get_data* with a new subroutine *get_data_cds*. Optimize the data retrieval in this new subroutine by performing just one `SELECT` from your CDS view.

Hint:
Remember that you need to use the new syntax of Open SQL if you want to select from the CDS entity.

1. Copy subroutine *get_data* to a new subroutine *get_data_cds*.
   a) Perform this step as you learned to do in previous classes.
2. Comment out the call of subroutines *get_data* and replace it with a call of the new subroutine.
   a) Perform this step as you learned to do in previous classes.
3. In subroutine *get_data_cds*, implement an Open SQL SELECT that reads data from the CDS view you just created. Let the SELECT read into data object *ct_bookings* directly. Make sure you provide a value for the input parameter of the CDS view.
   a) See the source code extract from the model solution.
4. Activate and test your program.
   a) Complete this step as you learned to do in previous classes.
   b) Verify that your code matches the following solution:

Source Code Extract: DDL Source

```
@AbapCatalog.sqlViewName: 'HA400_BOOK_CONV'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #NOT_REQUIRED
@EndUserText.label: 'HA400 CDS View for Case Study'
define view Ha400_Booking_W_Conv_Amount
       with parameters p_target_curr:abap.cuky
         as select from scustom as c
            inner join t005t as t
                    on c.country = t.land1
                   and t.spras = $session.system_language
               inner join sbook as b
                       on c.id = b.customid
                      and b.cancelled <> 'X'
                 inner join spfli as p
                         on b.carrid = p.carrid
                        and b.connid = p.connid
{
    c.id,
    c.name,
   @EndUserText.label: 'Flight Customer'
    concat_with_space(c.id, c.name, 1) as customer,
    c.city,
    t.landx as country,
    b.carrid,
    b.connid,
    b.fldate,
    p.cityfrom,
    p.cityto,
   @EndUserText.label: 'Booking Price (Converted)'
   @Semantics.amount.currencyCode: 'CONVAM_CURRENCY'
    currency_conversion(          amount => b.forcuram,
                         source_currency => b.forcurkey,
                         target_currency => $parameters.p_target_curr,
                      exchange_rate_date => b.order_date,
                          error_handling => 'FAIL_ON_ERROR' ) as convam,
   @EndUserText.label: 'Currency'
   @Semantics.currencyCode: true
    $parameters.p_target_curr as convam_currency
}
```

Source Code Extract: Subroutine *get_data_cds*

```
FORM get_data  USING pt_range_id         LIKE so_id[]
                     pt_range_name       LIKE so_name[]
                     pv_target_currency  TYPE s_curr
            CHANGING ct_bookings          TYPE ty_t_bookings.

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

  CLEAR ct_bookings.

  SELECT country, city, customer, carrid, connid, fldate,
         cityfrom, cityto, convam, convam_currency
    FROM ha400_booking_w_conv_amount( p_target_curr =
@pv_target_currency )
    INTO TABLE @ct_bookings
    WHERE   id IN @pt_range_id
      AND name IN @pt_range_name.

ENDFORM.
```

Unit 6
Exercise 16

Case Study Part 2: Use ALV Optimized for SAP HANA

Business Example

You are asked to improve an ABAP program that determines, for a set of customers to be entered, the total amount (in your local currency) each customer spends for flights. In the second step, you want to use the SAP List Viewer with Integrated Data Access (ALV with IDA) to display the result.

Template:

DDL Source HA400_BOOKING_W_CONV_AMOUNT
Report HA400_CASE_STUDY_S1

Solution:

Report HA400_CASE_STUDY_S2

Task 1: Copy and Analyze Template
In system T8N, which uses SAP HANA as its primary database, create a copy of report HA400_CASE_STUDY_S1 in your package ZHA400_## (suggested name: **`ZHA400_##_CSTUDY_2`**, where ## is your group number). Activate, analyze, and execute the program.

1. Create a copy of the report. Place it in your package ZHA400_## and assign it to your workbench task.

2. Analyze the source code of subroutine *get_data_cds*. Where does the data come from?

3. Activate and execute the program.

Task 2: Replace the Classical ALV with ALV with IDA
Edit your program. Replace the instantiation of the classical ALV with an instantiation of the ALV with Integrated Data Access (ALV with IDA). Comment out the call of subroutines *get_data_cds* and replace it with a call of a new subroutine (suggested name: *set_selection_alv_ida*) in which you set the data selection for the new ALV instance. Comment out the call of subroutine *configure_alv* and replace it with a call of a new subroutine (suggested name: *configure_alv_ida* in which you do the same settings as before but using the API of the new ALV.

1. Comment out the call of method *factory* of class *cl_salv_table* and implement a call of method *create_for_cds_view* of class *cl_salv_gui_table_ida* instead. Supply input parameter

iv_cds_view_name with the name of the CDS view you developed in the previous exercise. (Alternatively, you can use CDS view HA400_BOOKING_W_CONV_AMOUNT). Adjust the definition of data object *go_salv* to be compatible with the returning parameter of the method.

2. Define a new subroutine (suggested name: *set_selection_alv*) in which you pass the select options and a value for the CDS view's input parameter to the ALV instance. Replace the call of subroutine *get_data_cds* with a call to this new subroutine.

   Hint:
   Call appropriate methods of the ALV instance (interface *if_salv_gui_table_ida*) and make use of service class *cl_salv_range_tab_collector*.

3. Define a new subroutine (suggested name: *configure_alv_ida*) in which you hide unwanted fields, define an aggregation in field CONVAM, a sorting in column FLDATE and a grouping in columns COUNTRY, CITY and CUSTOMER. Replace the call of subroutine *configure_alv* with a call to this new subroutine.

   Hint:
   Call appropriate methods of the layout object (interface *if_salv_gui_layout_ida*), which you can retrieve from the ALV instance via method *default_layout*. Note that grouping of values is considered to be a part of the sorting.

4. Replace the call of method *display* of the classical ALV instance with a call of the corresponding method of the new ALV instance.

   Hint:
   There is no method *display* in interface *if_salv_gui_table_ida*. You have to call method *fullscreen* first and then method *display* for the resulting object reference.

5. Implement appropriate exception handling.
6. Activate and test the report.

Case Study Part 2: Use ALV Optimized for SAP HANA

Business Example

You are asked to improve an ABAP program that determines, for a set of customers to be entered, the total amount (in your local currency) each customer spends for flights. In the second step, you want to use the SAP List Viewer with Integrated Data Access (ALV with IDA) to display the result.

Template:

DDL Source HA400_BOOKING_W_CONV_AMOUNT
Report HA400_CASE_STUDY_S1

Solution:

Report HA400_CASE_STUDY_S2

Task 1: Copy and Analyze Template

In system T8N, which uses SAP HANA as its primary database, create a copy of report HA400_CASE_STUDY_S1 in your package ZHA400_## (suggested name: **`ZHA400_##_CSTUDY_2`**, where ## is your group number). Activate, analyze, and execute the program.

1. Create a copy of the report. Place it in your package ZHA400_## and assign it to your workbench task.
   a) Complete this step as you learned in previous exercises.
2. Analyze the source code of subroutine *get_data_cds*. Where does the data come from?

   The subroutine selects from CDS view HA400_BOOKING_W_CONV_AMOUNT, which is defined in the DDL source with the same name.
3. Activate and execute the program.
   a) Complete this step as you learned to do in previous classes.

Task 2: Replace the Classical ALV with ALV with IDA

Edit your program. Replace the instantiation of the classical ALV with an instantiation of the ALV with Integrated Data Access (ALV with IDA). Comment out the call of subroutines *get_data_cds* and replace it with a call of a new subroutine (suggested name: *set_selection_alv_ida*) in which you set the data selection for the new ALV instance. Comment out the call of subroutine *configure_alv* and replace it with a call of a new subroutine (suggested name: *configure_alv_ida* in which you do the same settings as before but using the API of the new ALV.

1. Comment out the call of method *factory* of class *cl_salv_table* and implement a call of method *create_for_cds_view* of class *cl_salv_gui_table_ida* instead. Supply input parameter *iv_cds_view_name* with the name of the CDS view you developed in the previous exercise. (Alternatively, you can use CDS view HA400_BOOKING_W_CONV_AMOUNT). Adjust the definition of data object *go_salv* to be compatible with the returning parameter of the method.
   a) See the source code extract from the model solution.

2. Define a new subroutine (suggested name: *set_selection_alv*) in which you pass the select options and a value for the CDS view's input parameter to the ALV instance. Replace the call of subroutine *get_data_cds* with a call to this new subroutine.

   Hint:
   Call appropriate methods of the ALV instance (interface *if_salv_gui_table_ida*) and make use of service class *cl_salv_range_tab_collector*.

   a) See the source code extract from the model solution.

3. Define a new subroutine (suggested name: *configure_alv_ida*) in which you hide unwanted fields, define an aggregation in field CONVAM, a sorting in column FLDATE and a grouping in columns COUNTRY, CITY and CUSTOMER. Replace the call of subroutine *configure_alv* with a call to this new subroutine.

   Hint:
   Call appropriate methods of the layout object (interface *if_salv_gui_layout_ida*), which you can retrieve from the ALV instance via method *default_layout*. Note that grouping of values is considered to be a part of the sorting.

   a) See the source code extract from the model solution.

4. Replace the call of method *display* of the classical ALV instance with a call of the corresponding method of the new ALV instance.

   Hint:
   There is no method *display* in interface *if_salv_gui_table_ida*. You have to call method *fullscreen* first and then method *display* for the resulting object reference.

   a) See the source code extract from the model solution.

5. Implement appropriate exception handling.
   a) See the source code extract from the model solution.

6. Activate and test the report.
   a) Perform this step as before.
   b) Verify that your code matches the following solution:

Code Extract: Report HA400_CASE_STUDY_S2

```
*&---------------------------------------------------------------------*
*& Report HA400_CASE_STDY_S2
*&---------------------------------------------------------------------*
*&
*&---------------------------------------------------------------------*
REPORT ha400_case_study_s2 MESSAGE-ID ha400.

* Structure for Result
TYPES: BEGIN OF ty_s_booking,
         country         TYPE t005t-landx,
         city            TYPE scustom-city,
         customer        TYPE ha400_customer,
         carrid          TYPE sbook-carrid,
         connid          TYPE sbook-connid,
         fldate          TYPE sbook-fldate,
         cityfrom        TYPE spfli-cityfrom,
         cityto          TYPE spfli-cityto,
         convam          TYPE s_sum,
         convam_currency TYPE s_curr,
       END OF ty_s_booking.

TYPES: ty_t_bookings TYPE STANDARD TABLE OF ty_s_booking
                          WITH NON-UNIQUE KEY customer.

DATA: gt_bookings TYPE        ty_t_bookings,
      gs_booking  TYPE        ty_s_booking.

* ALV Processing

*DATA go_salv TYPE REF TO cl_salv_table.
*DATA gx_excp TYPE REF TO cx_salv_error.

DATA go_salv TYPE REF TO if_salv_gui_table_ida.
DATA gx_excp TYPE REF TO cx_salv_ida_dynamic.

* selection screen

DATA gs_scustom TYPE scustom.
SELECTION-SCREEN BEGIN OF BLOCK cus WITH FRAME TITLE TEXT-cus.
SELECT-OPTIONS:
     so_id FOR gs_scustom-id,
   so_name FOR gs_scustom-name.
SELECTION-SCREEN END OF BLOCK cus.


AT SELECTION-SCREEN.

* message not needed any more: Application is much faster now
*   IF so_id IS INITIAL AND so_name IS INITIAL.
*     MESSAGE w050.
*   ENDIF.

START-OF-SELECTION.

* Provide Data
*----------------------*
```

```
*  PERFORM get_data_cds   USING
*                            so_id[]
*                            so_name[]
*                            'USD'
*                          CHANGING
*                            gt_bookings.
*
* Create ALV Instance
*----------------------*

  TRY.

*      cl_salv_table=>factory(
*        IMPORTING
*          r_salv_table   = go_salv
*        CHANGING
*          t_table        = gt_bookings
*      ).

      go_salv = cl_salv_gui_table_ida=>create_for_cds_view(
                          iv_cds_view_name = 'HA400_BOOKING_W_CONV_AMOUNT'
                                                          ).
* New: Set Data Selection for ALV Instance
*---------------------------------------------*

      PERFORM set_selection_alv_ida USING
                                     go_salv
                                     so_id[]
                                     so_name[]
                                    'USD'.


* Configure ALV Instance
*-------------------------*

*      PERFORM configure_alv
*                   USING
*                     go_salv.

      PERFORM configure_alv_ida
                   USING
                     go_salv.

* Display
*-------------------------*


*      go_salv->display( ).
      go_salv->fullscreen( )->display( ).

* Error Handling
*--------------------------*
*    CATCH cx_salv_error INTO gx_excp.    "
    CATCH cx_salv_ida_dynamic INTO gx_excp.    "
      MESSAGE gx_excp TYPE 'I'.
  ENDTRY.

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

```
*&  Form set_selection_alv_ida
*&---------------------------------------------------------------------*
FORM set_selection_alv_ida
            USING po_salv TYPE REF TO if_salv_gui_table_ida
                  pt_range_id         LIKE so_id[]
                  pt_range_name       LIKE so_name[]
                  pv_target_currency  TYPE s_curr.

  DATA:
    lo_range_collector TYPE REF TO cl_salv_range_tab_collector,
    lt_named_ranges    TYPE        if_salv_service_types=>yt_named_ranges.

  DATA: lt_parameter TYPE if_salv_gui_types_ida=>yt_parameter,
        ls_parameter LIKE LINE OF lt_parameter.

* Set parameter
*---------------------------------------*

  ls_parameter-name = 'P_TARGET_CURR'.
  ls_parameter-value = 'USD'.

  APPEND ls_parameter TO lt_parameter.

  po_salv->set_view_parameters( lt_parameter ).

* Set ranges for preselection
*---------------------------------------*
  CREATE OBJECT lo_range_collector.

  lo_range_collector->add_ranges_for_name(
      iv_name   = 'ID'
      it_ranges = pt_range_id
  ).

  lo_range_collector->add_ranges_for_name(
      iv_name   = 'NAME'
      it_ranges = pt_range_name
  ).

  lo_range_collector->get_collected_ranges(
    IMPORTING
      et_named_ranges = lt_named_ranges
  ).

  po_salv->set_select_options(
    EXPORTING
      it_ranges    = lt_named_ranges
  ).


ENDFORM.

*&---------------------------------------------------------------------*
*&  Form configure_alv_ida
*&---------------------------------------------------------------------*
FORM configure_alv_ida
            USING po_salv TYPE REF TO if_salv_gui_table_ida.
```

```
* ALV processing

  DATA: lo_layout TYPE REF TO if_salv_gui_layout_ida.

  DATA:  lt_visible_fields TYPE if_salv_gui_types_ida=>yt_field_name.

  DATA: lt_sort_order TYPE         if_salv_gui_types_ida=>yt_sort_rule,
        ls_sort_order LIKE LINE OF lt_sort_order.

  DATA: lt_aggregation TYPE if_salv_gui_types_ida=>yt_aggregation_rule,
        ls_aggregation LIKE LINE OF lt_aggregation.


  DATA: lx_excp  TYPE REF TO cx_salv_ida_dynamic.

  TRY.

      lo_layout = po_salv->default_layout( ).

* Hide unwanted fields
*-----------------------*
      APPEND 'COUNTRY'         TO lt_visible_fields.
      APPEND 'CITY'            TO lt_visible_fields.
      APPEND 'CUSTOMER'        TO lt_visible_fields.
      APPEND 'CARRID'          TO lt_visible_fields.
      APPEND 'CONNID'          TO lt_visible_fields.
      APPEND 'FLDATE'          TO lt_visible_fields.
      APPEND 'CITYFROM'        TO lt_visible_fields.
      APPEND 'CITYTO'          TO lt_visible_fields.
      APPEND 'CONVAM'         TO lt_visible_fields.
      APPEND 'CONVAM_CURRENCY' TO lt_visible_fields.

      lo_layout->set_visible_fields( lt_visible_fields ).

* Preset sort order, grouping and aggregations
*-----------------------------------------------*

* Calculate Sum in Column CONVAM

      ls_aggregation-field_name = 'CONVAM'.
      ls_aggregation-function   = if_salv_service_types=>cs_function_code-
sum.
      APPEND  ls_aggregation TO lt_aggregation.

      lo_layout->set_aggregations( lt_aggregation  ).

* Sorting and subtotals for columns COUNTRY, CITY and CUSTOMER

      ls_sort_order-field_name = 'COUNTRY'.
      ls_sort_order-descending = abap_false.
      ls_sort_order-is_grouped = abap_true.
      APPEND ls_sort_order TO lt_sort_order.

      ls_sort_order-field_name = 'CITY'.
      ls_sort_order-descending = abap_false.
      ls_sort_order-is_grouped = abap_true.
      APPEND ls_sort_order TO lt_sort_order.
      ls_sort_order-field_name = 'CUSTOMER'.
      ls_sort_order-descending = abap_false.
      ls_sort_order-is_grouped = abap_true.
      APPEND ls_sort_order TO lt_sort_order.

* Sorting in column FLDATE

      ls_sort_order-field_name = 'FLDATE'.
      ls_sort_order-descending = abap_false.
      ls_sort_order-is_grouped = abap_false.
      APPEND ls_sort_order TO lt_sort_order.

      lo_layout->set_sort_order( lt_sort_order ).

* Exception Handling
*--------------------*
    CATCH cx_salv_ida_dynamic INTO lx_excp.     "
      MESSAGE lx_excp TYPE 'I'.
  ENDTRY.


ENDFORM.
```