TABC42 ABAP Programmiertechniken 1/2

TABC42 1/2

R/3 System

Release 46B

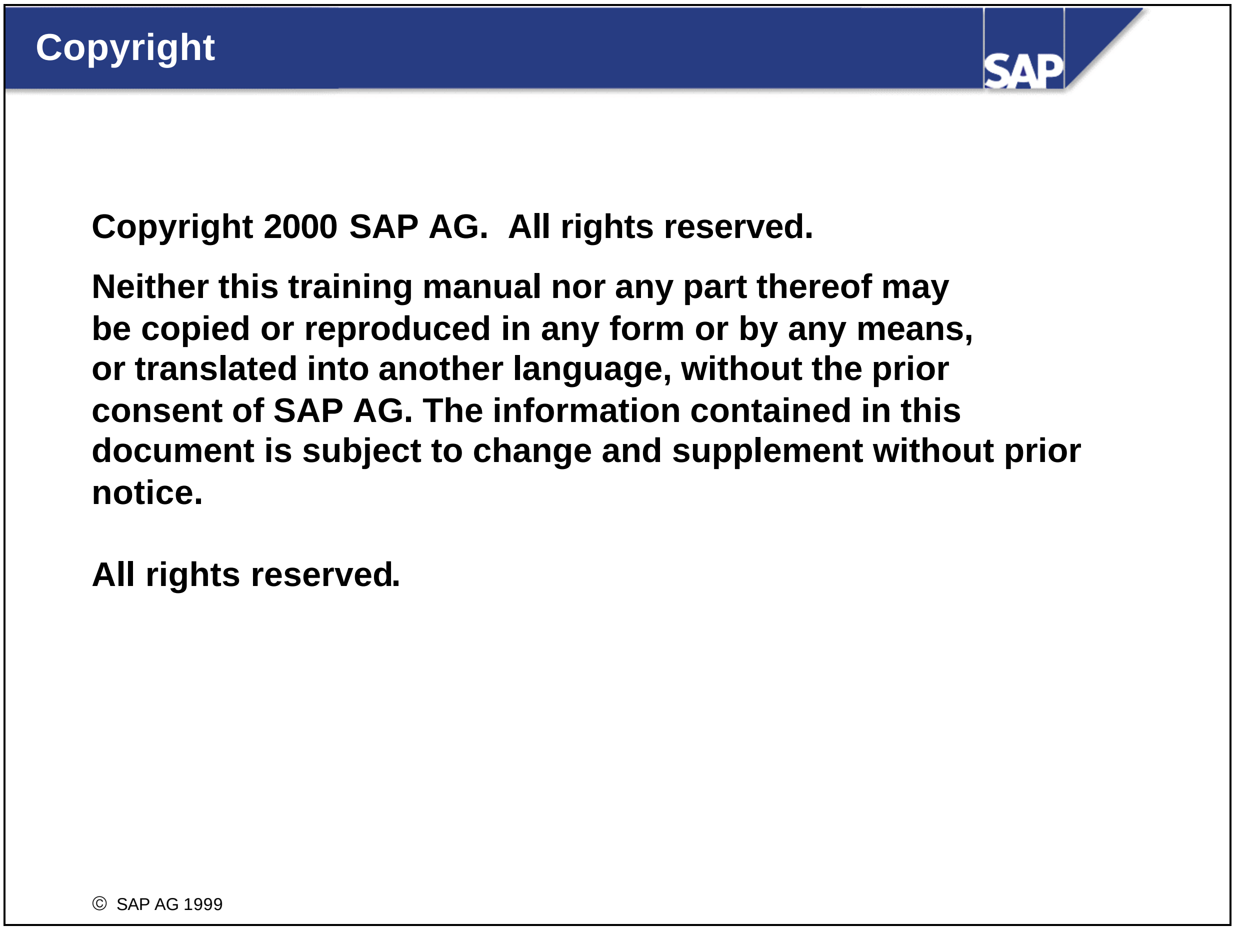
30.05.2000

TABC42 1/2

ABAP Programming Techniques

Part 1 of 2

- R/3 System
- Release 4.6B
- May 2000
- Material number 50039584



Copyright

Section Overview

Section **Basis Technology Overview**

Section **ABAP Workbench Concepts and Tools**

Section **Managing ABAP Development Projects**

Section **ABAP Dictionary**

Section **ABAP Programming Techniques**

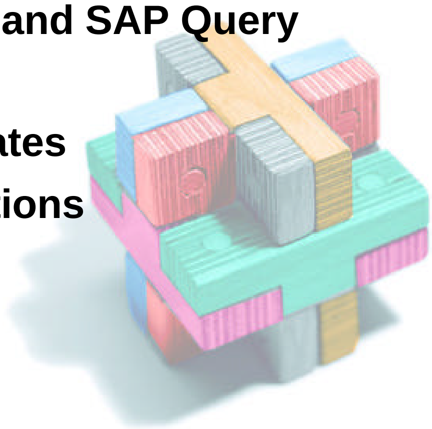
Section **Techniques for List Creation and SAP Query**

Section **Transaction Programming**

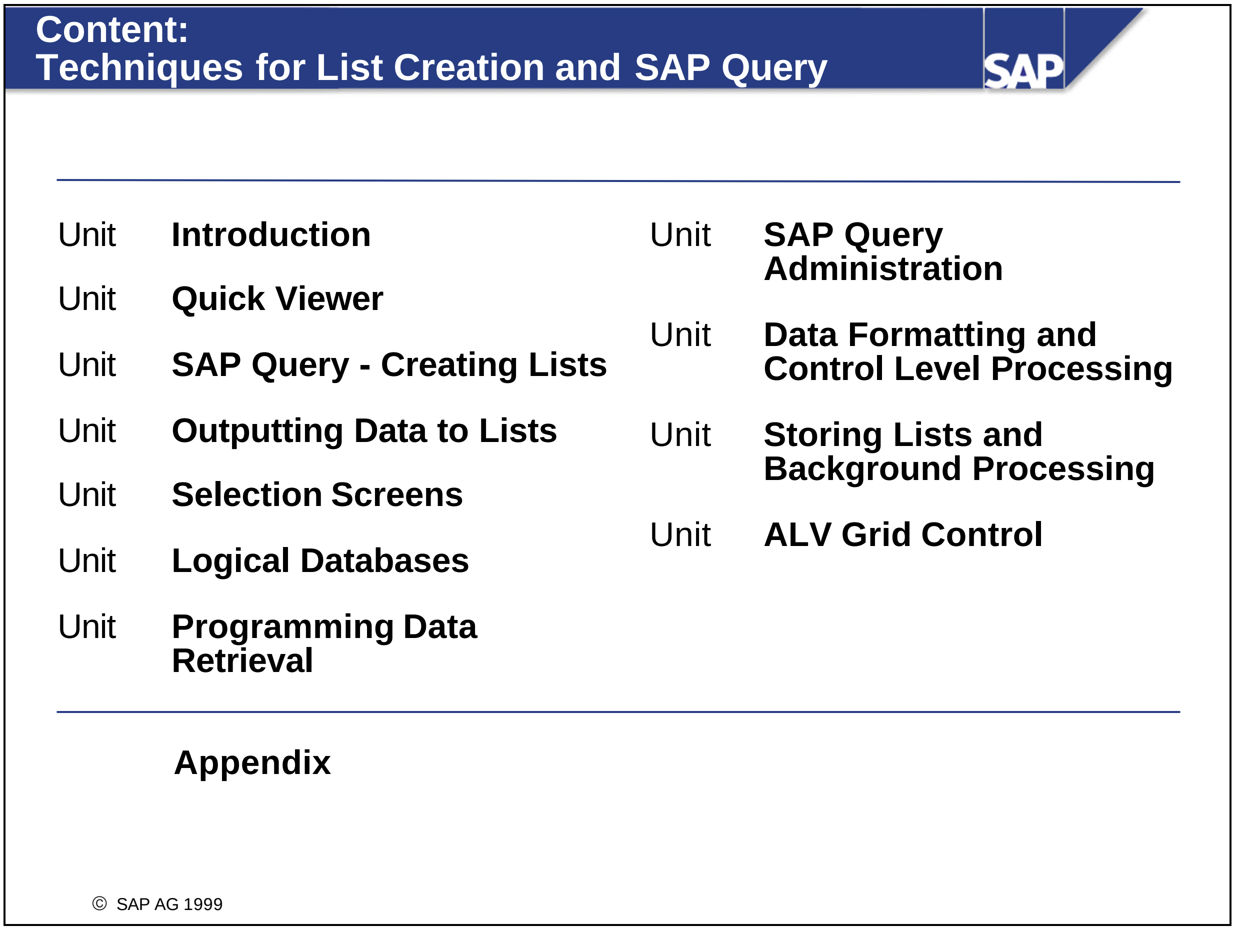
Section **Programming Database Updates**

Section **Enhancements and Modifications**

Section **Data Transfer**



Section: Techniques for List Creation and SAP Query



Content: Techniques for List Creation and SAP Query

Unit **Introduction**

Unit **Quick Viewer**

Unit **SAP Query - Creating Lists**

Unit **Outputting Data to Lists**

Unit **Selection Screens**

Unit **Logical Databases**

Unit **Programming Data Retrieval**

Unit **SAP Query Administration**

Unit **Data Formatting and Control Level Processing**

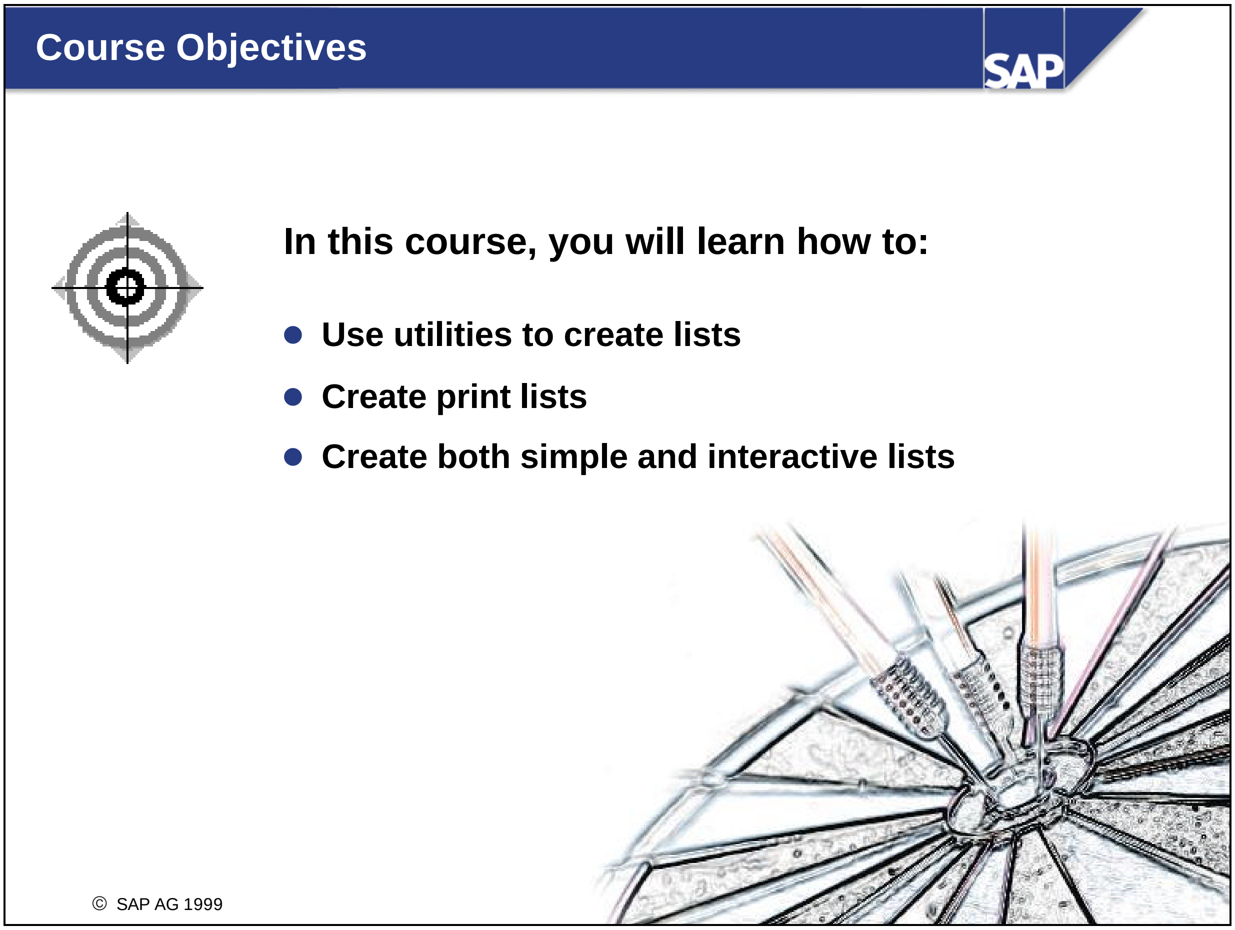
Unit **Storing Lists and Background Processing**

Unit **ALV Grid Control**

Appendix

Introduction

- Course Goals
- Course Objectives
- Course Content
- Course Overview Diagram
- Main Business Scenario
- Getting Started

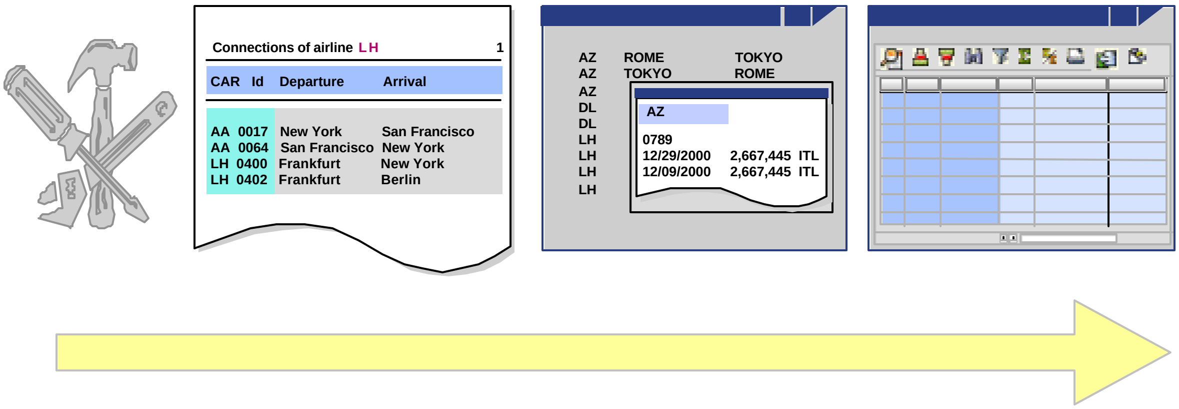


Course Objectives

In this course, you will learn how to:

- Use utilities to create lists
- Create print lists
- Create both simple and interactive lists

Course Overview Diagram



```
Connections of airline LH                                1
CAR  Id    Departure        Arrival

AA   0017  New York         San Francisco
AA   0064  San Francisco    New York
LH   0400  Frankfurt        New York
LH   0402  Frankfurt        Berlin
```

```
AZ   ROME          TOKYO
AZ   TOKYO         ROME
AZ
DL
DL
LH
LH
LH
LH
```

```
AZ

0789
12/29/2000     2,667,445  ITL
12/09/2000     2,667,445  ITL
```

Tools

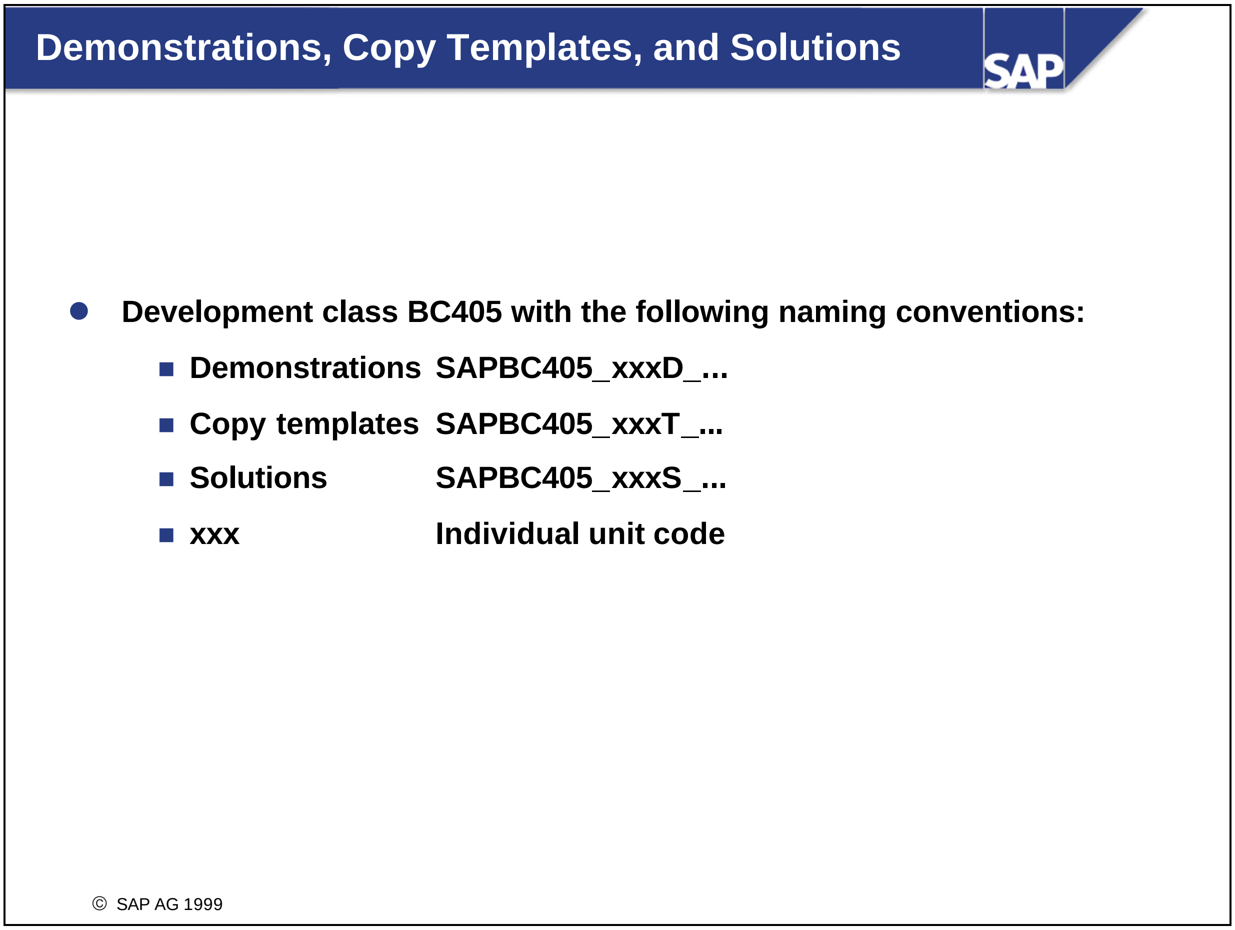
Simple lists

Interactive Lists

ALV Grid Control

Business Scenario

- You are an employee of a very large tour company
- The tour company wants to increase its offerings
- To allow for the increased number of tours, the company needs a list of the most current flight data
- You are assigned the task of writing a program that outputs the required flight data to a list



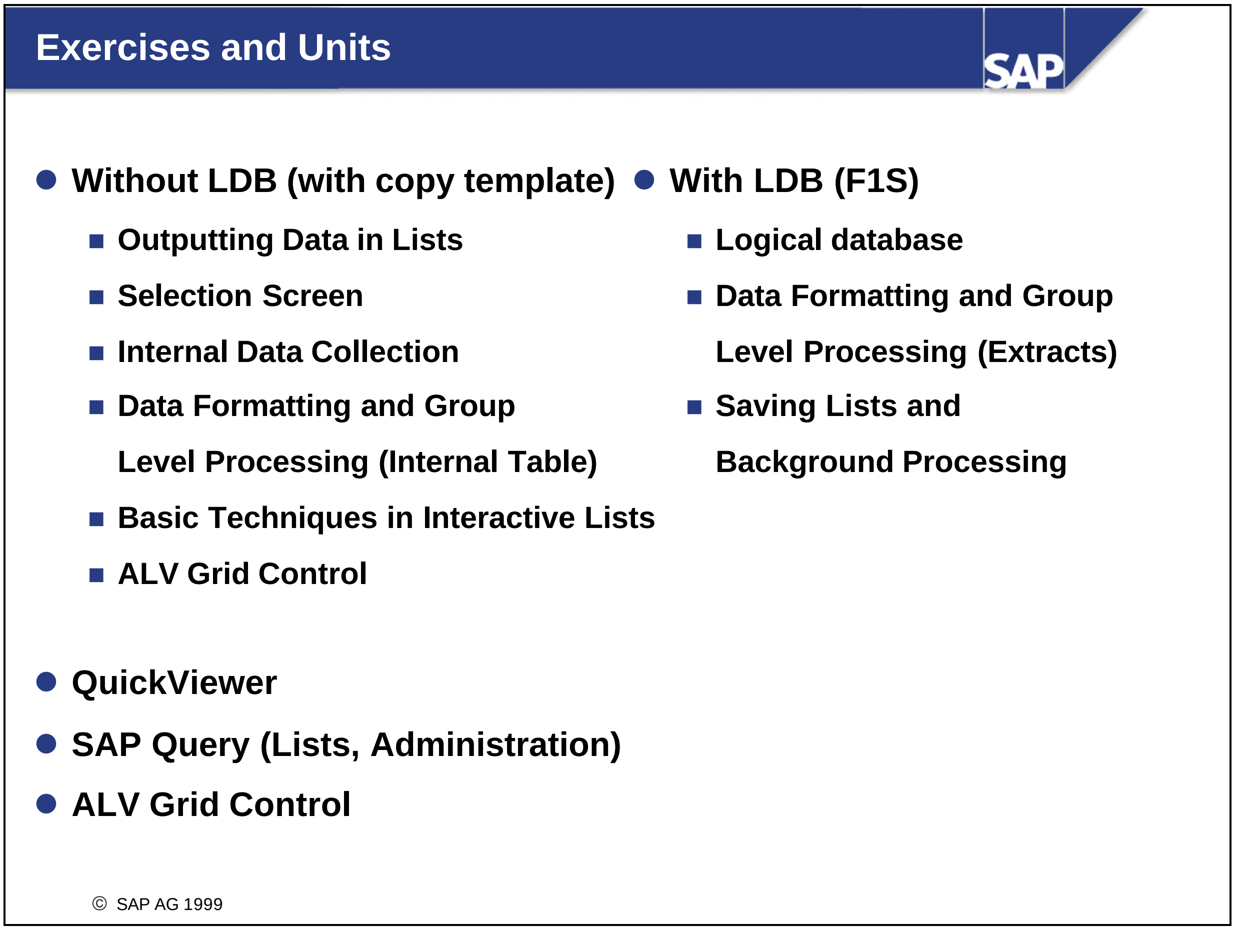
Demonstrations, Copy Templates, and Solutions

- Development class BC405 with the following naming conventions:

Demonstrations	SAPBC405_xxxD_...
Copy templates	SAPBC405_xxxT_...
Solutions	SAPBC405_xxxS_...
xxx	Individual unit code

- Abbreviations for individual units:

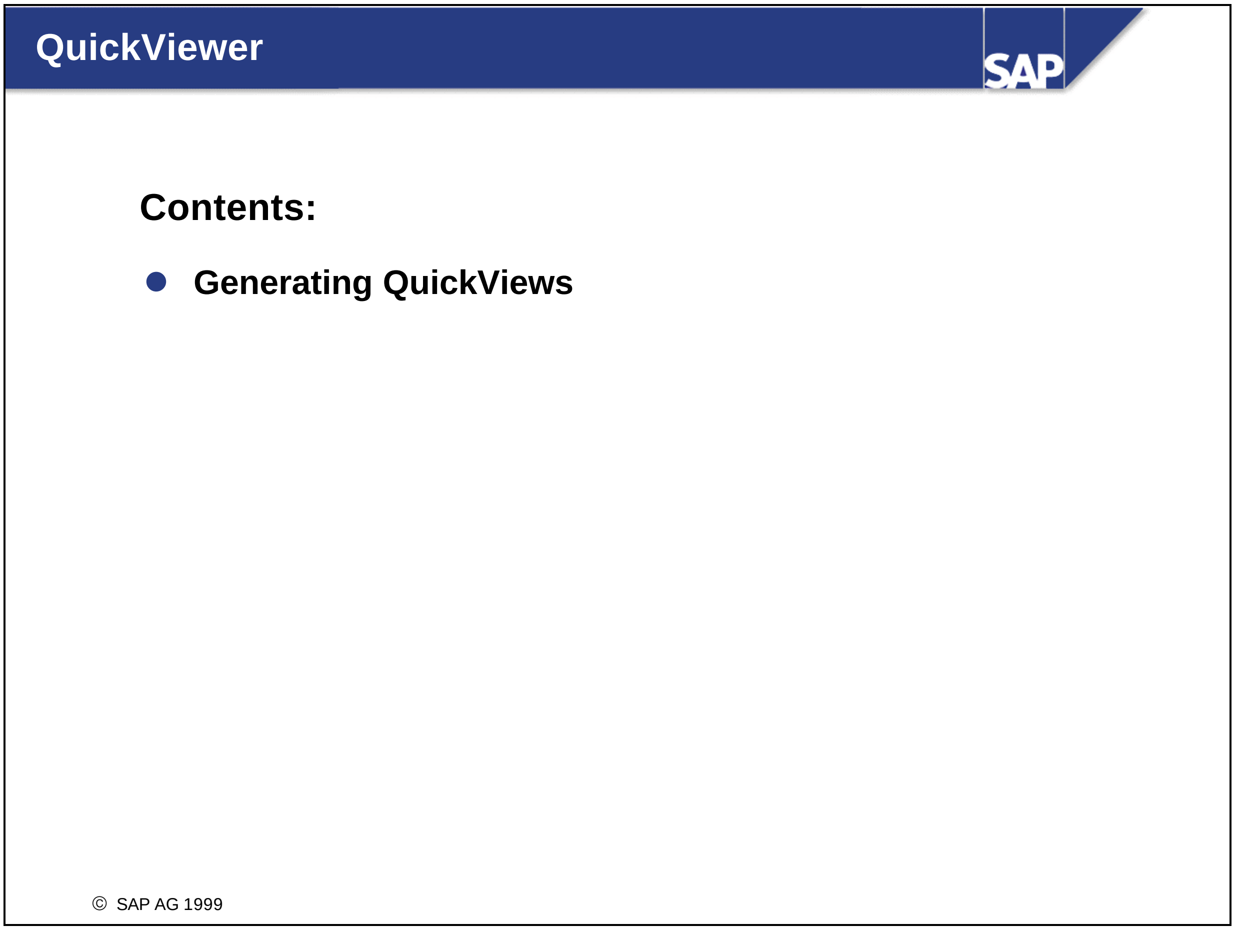
- QUV	Unit 2:	QuickViewer
- AQL	Unit 3:	SAP Query - Creating Lists
- FOL	Unit 4:	Outputting Data in Lists
- SSC	Unit 5:	Selection Screen
- LDB	Unit 6:	Logical database
- GDA	Unit 7:	Internal Data Collection
- AQA	Unit 8:	SAP Query - Administration
- DAP	Unit 9:	Data Formatting and Control Level Processing
- STL	Unit 10:	Saving Lists and Background Processing
- ILB	Unit 11:	Basic Techniques in Interactive Lists
- ALV	Unit 12:	ALV Grid Control



Exercises and Units

- Without LDB (with copy template)
  - Outputting Data in Lists
  - Selection Screen
  - Internal Data Collection
  - Data Formatting and Group Level Processing (Internal Table)
  - Basic Techniques in Interactive Lists
  - ALV Grid Control
- With LDB (F1S)
  - Logical database
  - Data Formatting and Group Level Processing (Extracts)
  - Saving Lists and Background Processing
- QuickViewer
- SAP Query (Lists, Administration)
- ALV Grid Control

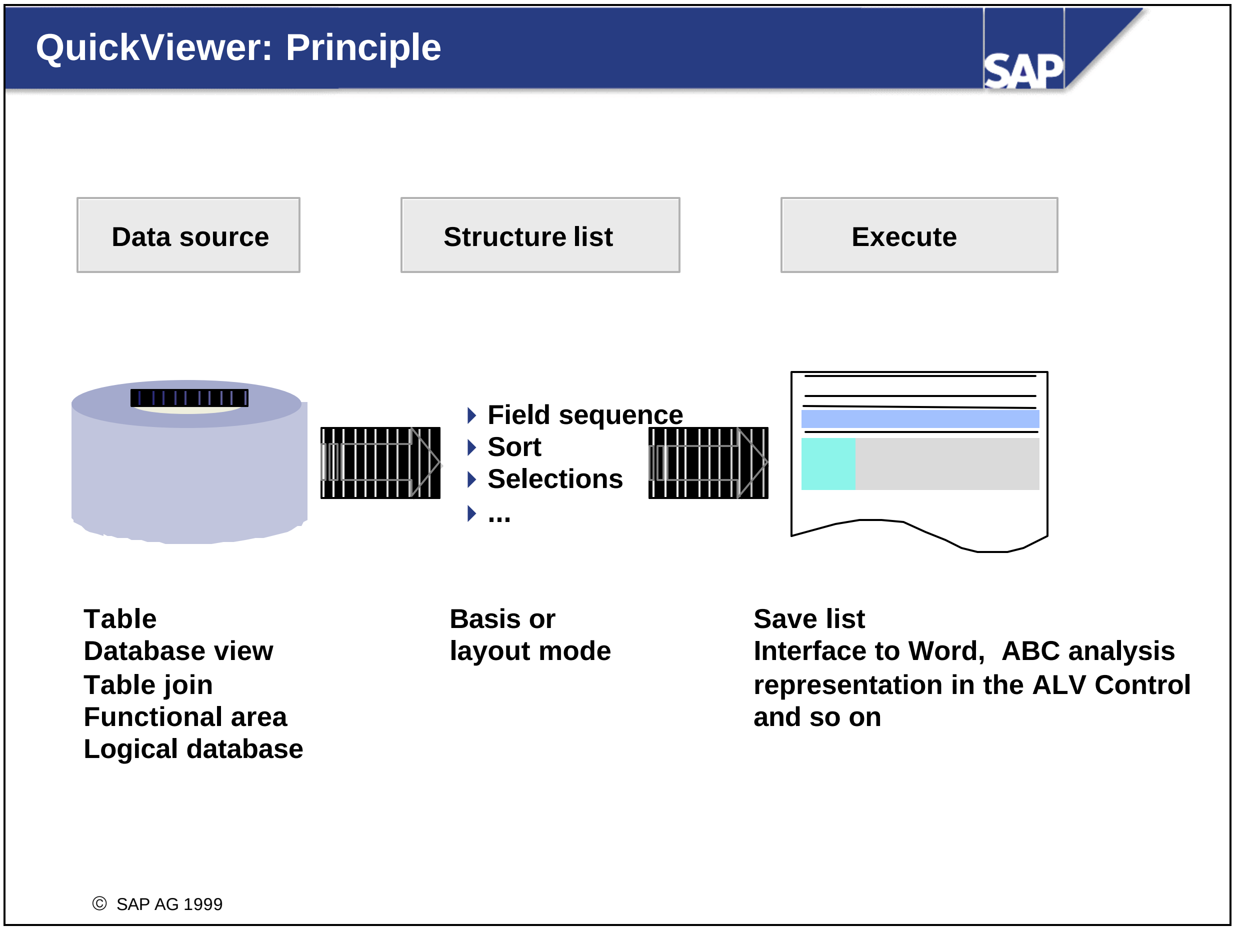
- The exercises stretch across several units. Each intermediate step has a sample solution that can be used for the subsequent exercise.



QuickViewer

Contents:

- Generating QuickViews



QuickViewer: Principle

Data source

Structure list

Execute

- Field sequence
- Sort
- Selections
- ...

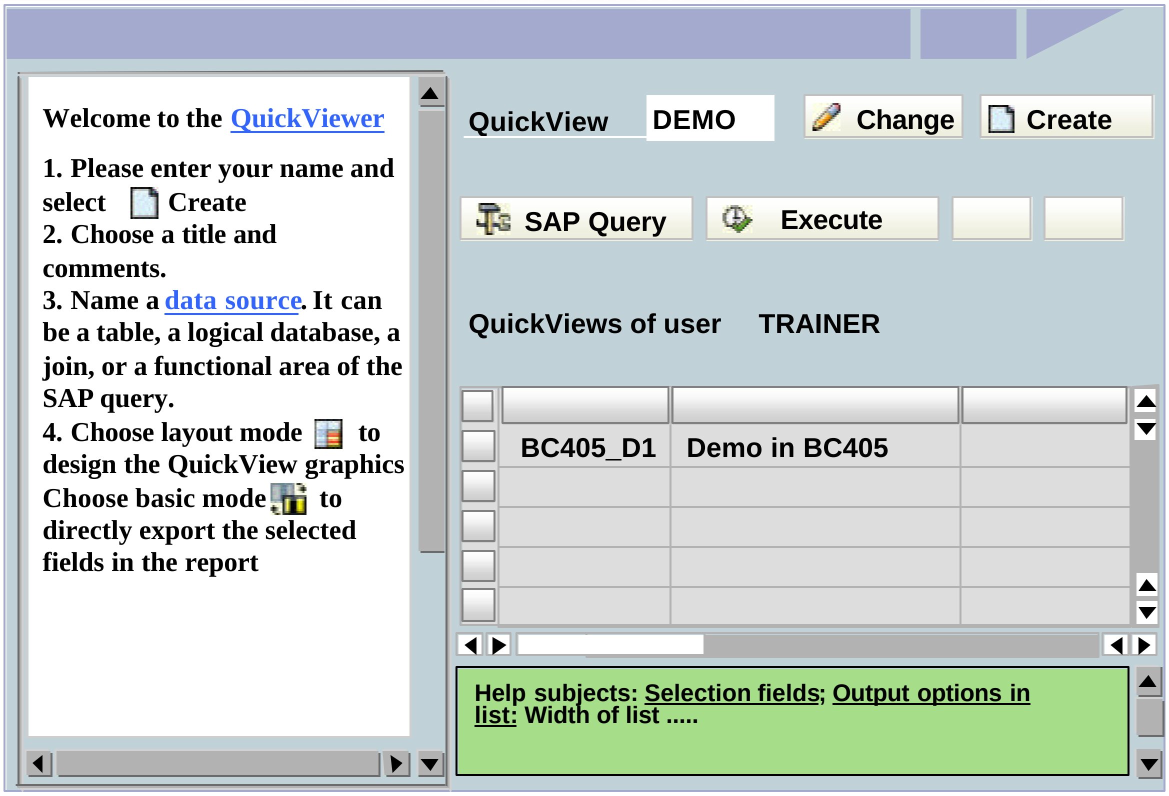
Table
Database view
Table join
Functional area
Logical database

Basis or
layout mode

Save list
Interface to Word, ABC analysis
representation in the ALV Control
and so on

- The QuickViewer is a tool for developing ad hoc reports that is new in Release 4.6A. You can start the QuickViewer using the menu path QUV-1.
- The QuickViewer can use a database table or a database view as a data source. Lists can be generated using the fields in the data source specified. Two modes are available for this: basis mode and layout mode
- The QuickViewer provides interfaces, for example, to the EIS, ABC analysis or the ALV Grid Control. The list can also be processed further in external programs, such as Word.
- The generated list can be saved and then displayed again in the QuickViewer. Selection criteria are also saved along with the list, and can be queried again at any time.

QuickViewer: Initial Access



Welcome to the QuickViewer

1. Please enter your name and select Create
2. Choose a title and comments.
3. Name a data source. It can be a table, a logical database, a join, or a functional area of the SAP query.
4. Choose layout mode to design the QuickView graphics Choose basic mode to directly export the selected fields in the report

QuickView DEMO

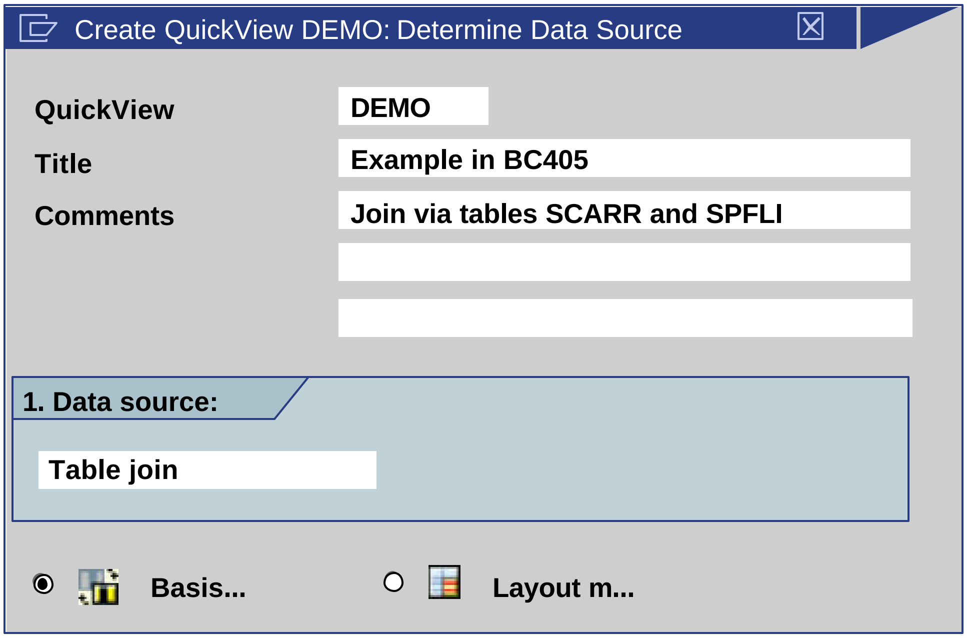
QuickViews of user TRAINER

BC405_D1	Demo in BC405	

Help subjects: Selection fields; Output options in list: Width of list .....

- Each user defines their own user-specific QuickViews which only they can display. This means that you cannot copy other users' QuickViews. You can, however, compile an SAP Query from a QuickView, if the QuickView uses a functional area from the standard system as a data source (see unit 'SAP Query - Creating Lists'). The query is then visible to the user group.
- QuickViews are not connected to the correction and transport system.

Creating a QuickView



Create QuickView DEMO: Determine Data Source

QuickView	DEMO
Title	Example in BC405
Comments	Join via tables SCARR and SPFLI

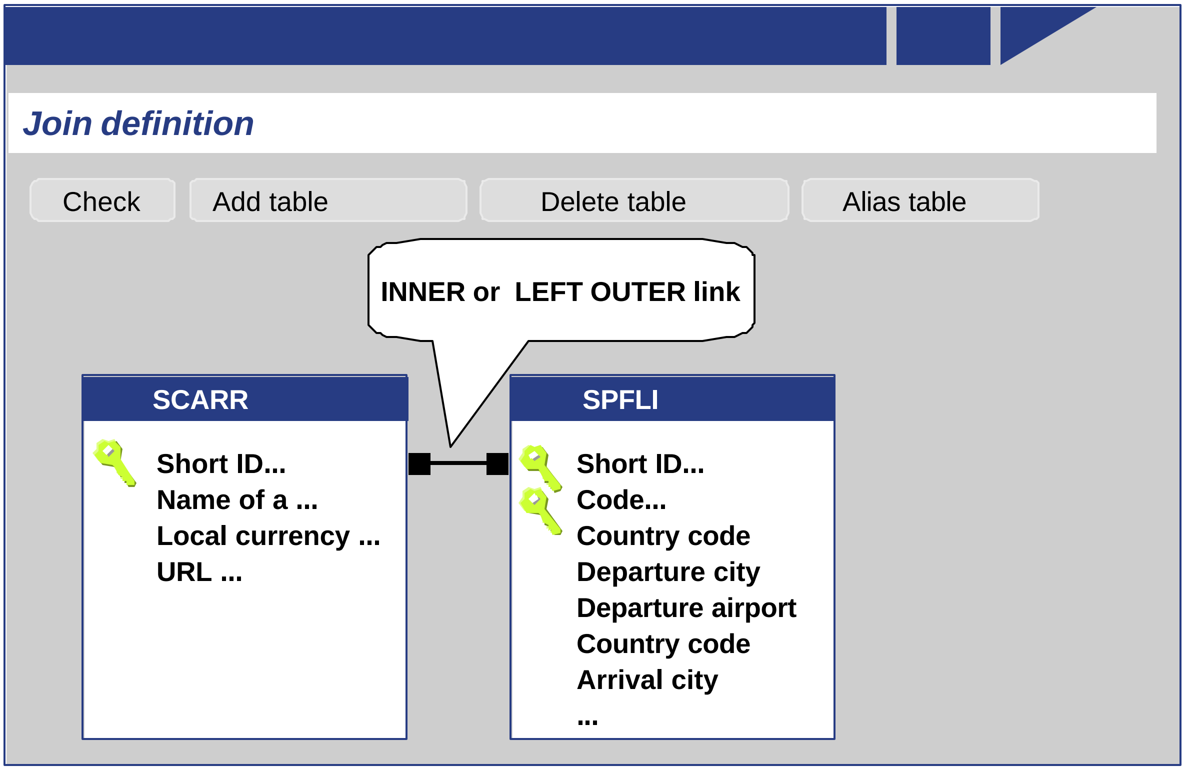
1. Data source:

Table join

● Basis... ○ Layout m...

- You must name a data source in order to generate a QuickView. The data source can be a database table, a database view, a logical database, a table join, or even a functional area of SAP query. The functional area must lie in the (client-specific) standard area.
- You can access the specified data, but you cannot extend it with additional fields (also see *Local fields* under SAP Query).

Join Definition: Graphical



Join definition

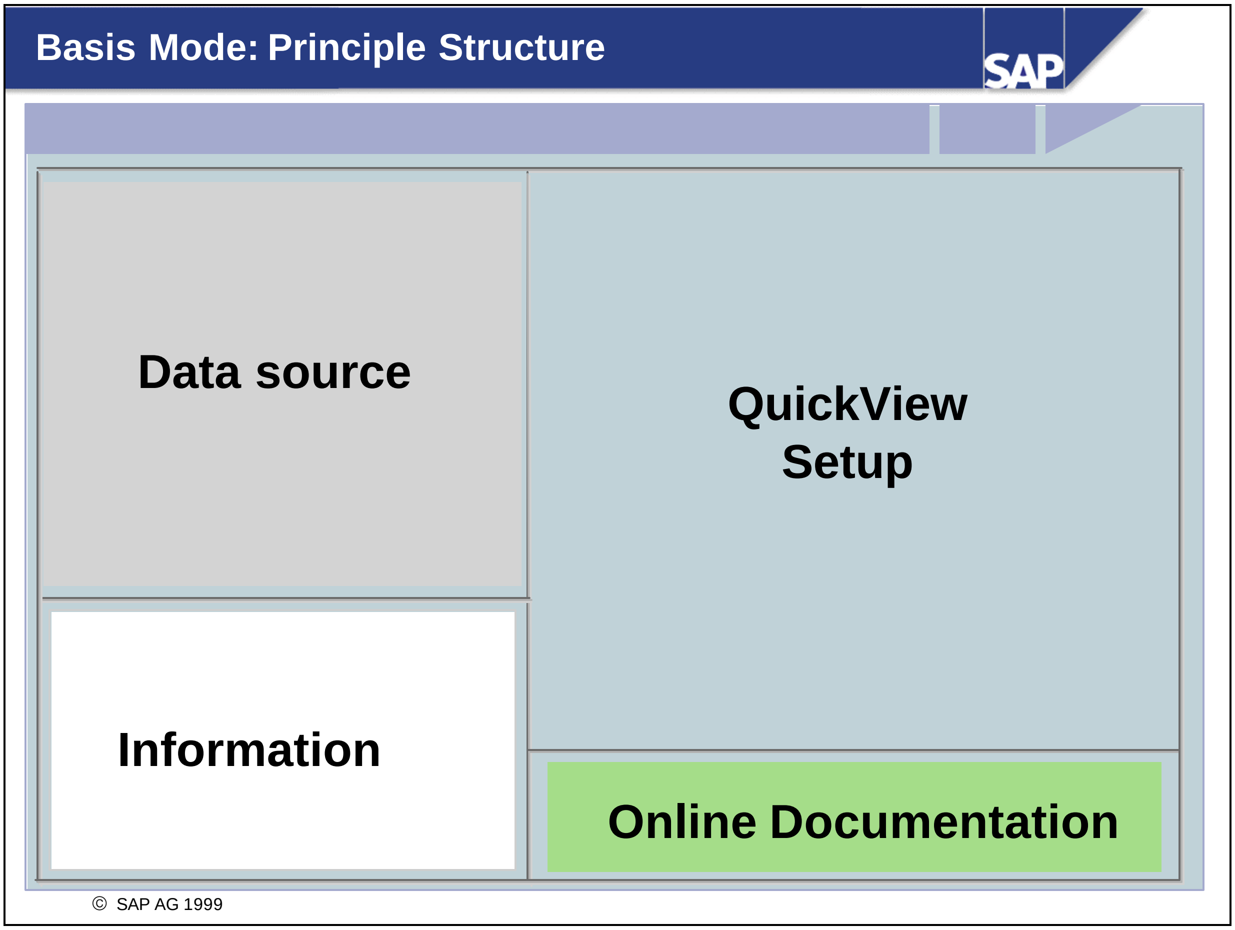
SCARR

- Short ID...
- Name of a ...
- Local currency ...
- URL ...

SPFLI

- Short ID...
- Code...
- Country code
- Departure city
- Departure airport
- Country code
- Arrival city
- ...

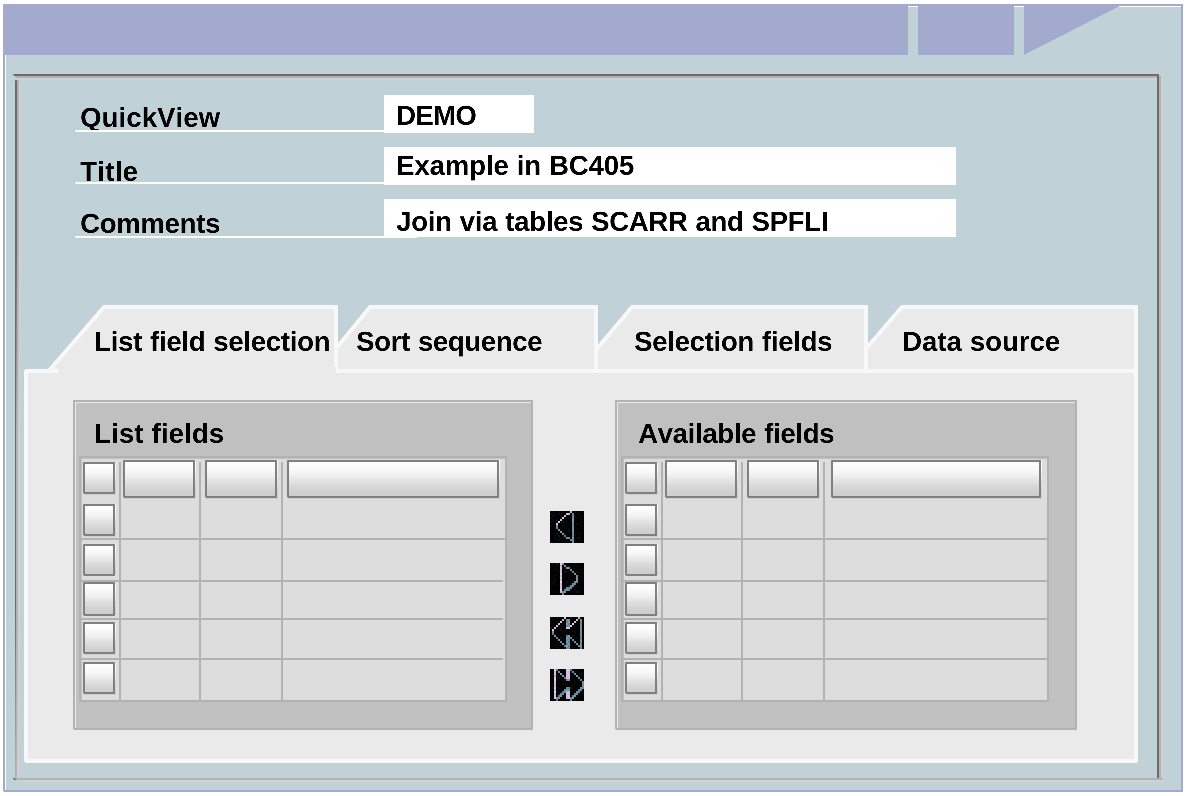
- When you specify a table join as the data source, you have to define the join before you can structure the list in Query Painter.
- You define the table join graphically. You have to specify the links between the tables, and you can have the system propose a value. It does this using information from the Dictionary .
- You determine the resulting quantity by deciding on either Inner or Left Outer Join logic. For example, if you only want to output airlines from table SCARR in a list when these airlines have flights in table SPFLI, this corresponds to the Inner Join logic. In contrast, if you want to output all the airlines regardless of whether flights exist in table SPFLI, then you would link both tables using Left Outer Join logic. In this case, the left table is SCARR.
- Alias tables enable you to use the same (database) table several times when defining the join



Basis Mode: Principle Structure

- In basic mode, the screen is divided into four areas. The available fields (data source) are displayed to the left in tree form. Further information on how to work in the basic mode is displayed in the lower left window. You can maintain the title and comments and control the output (list or Excel) in the upper right area. This is also where you control the list structure, set the sort sequence and define the selection criteria. You can branch to the online documentation from the lower right window.

Structuring a QuickView in the Basis Mode

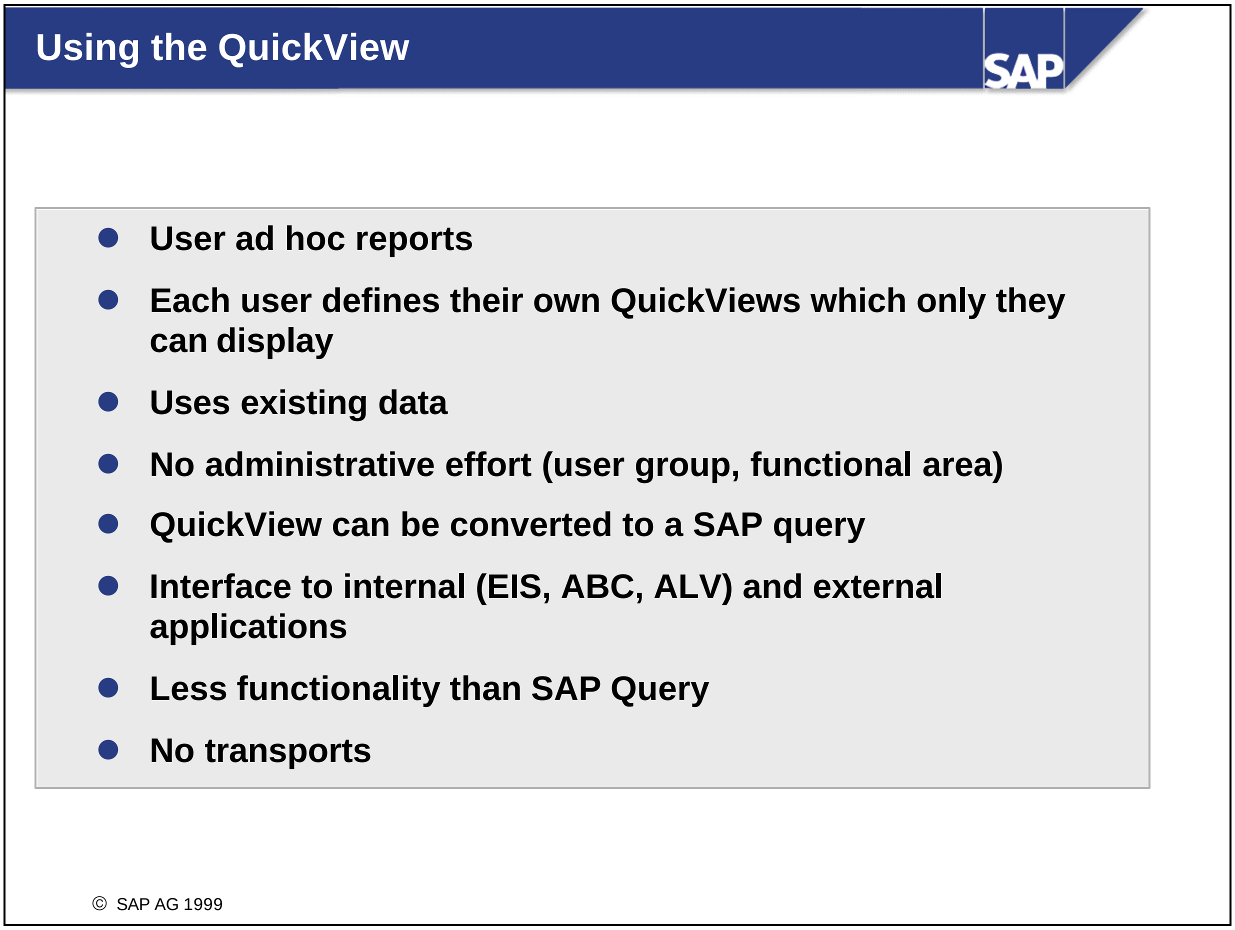


QuickView: DEMO

Title: Example in BC405

Comments: Join via tables SCARR and SPFLI

- You can structure your QuickView using two table controls. Select the fields you want in your list in the right table control and use the transfer functions to move them to the left table control ('List fields'). You can also control how many lines the list should have (using the 'Add line' function) in the left table control ('List fields').
- Follow the same procedure for the sort and selection fields: select the fields you require in the right table control and copy them to the left control.



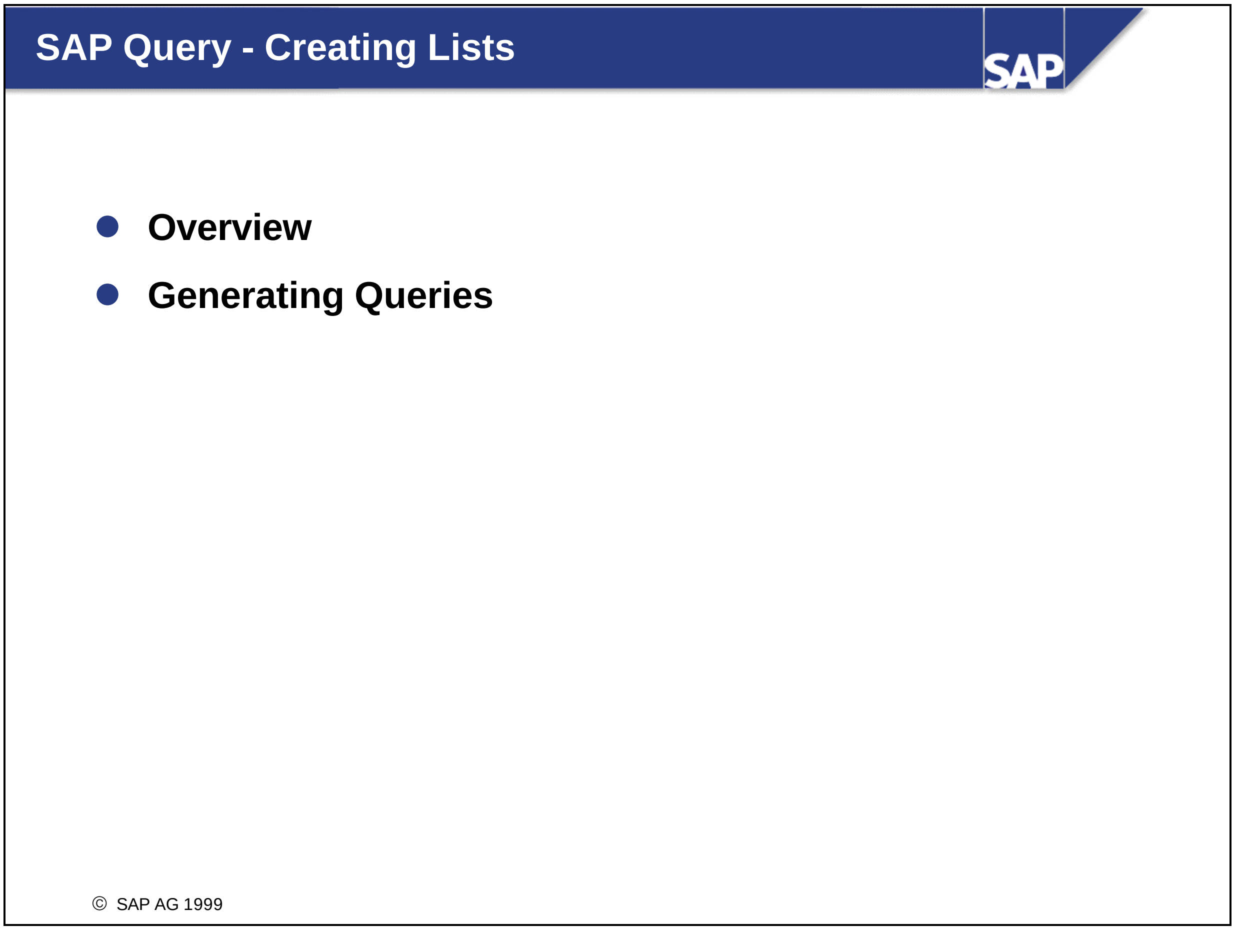
Using the QuickView

- User ad hoc reports
- Each user defines their own QuickViews which only they can display
- Uses existing data
- No administrative effort (user group, functional area)
- QuickView can be converted to a SAP query
- Interface to internal (EIS, ABC, ALV) and external applications
- Less functionality than SAP Query
- No transports

Summary

You are now able to:

- **Use the QuickViewer to generate ad hoc reports**



SAP Query - Creating Lists

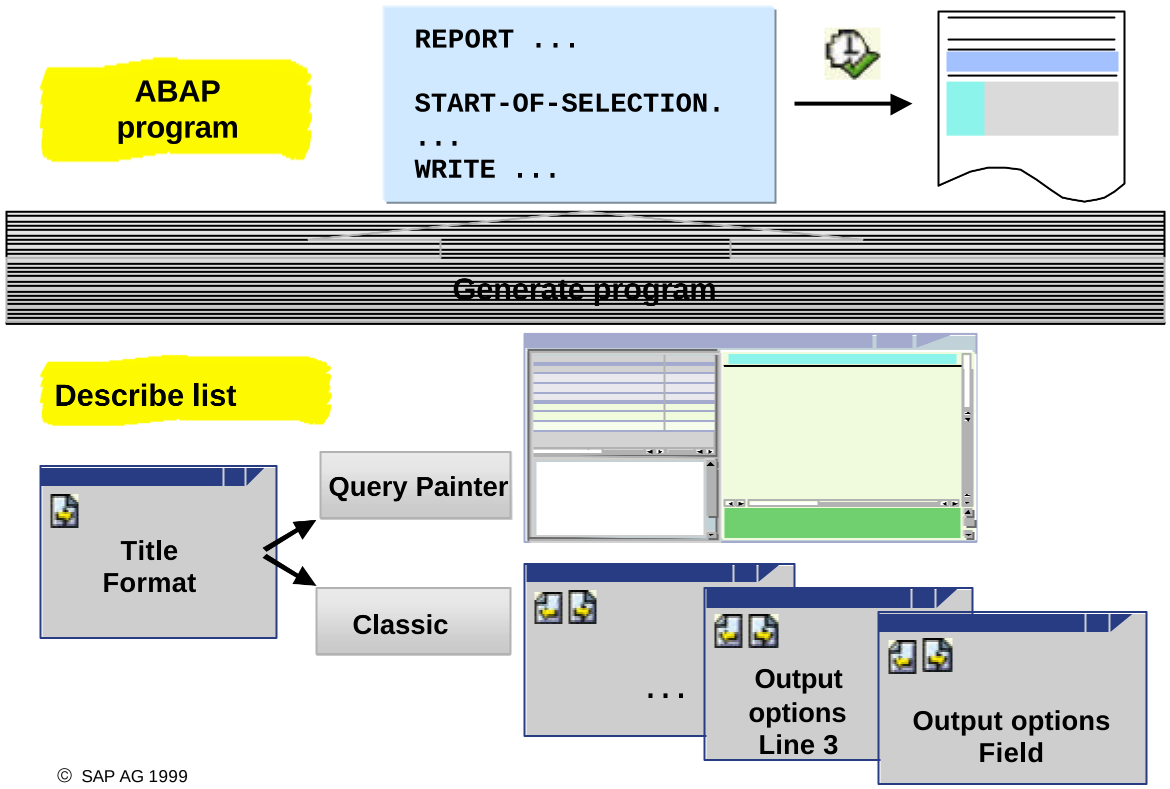
- Overview
- Generating Queries

SAP Query - Creating Lists

➤ Overview

Queries

Overview: Programs and Query



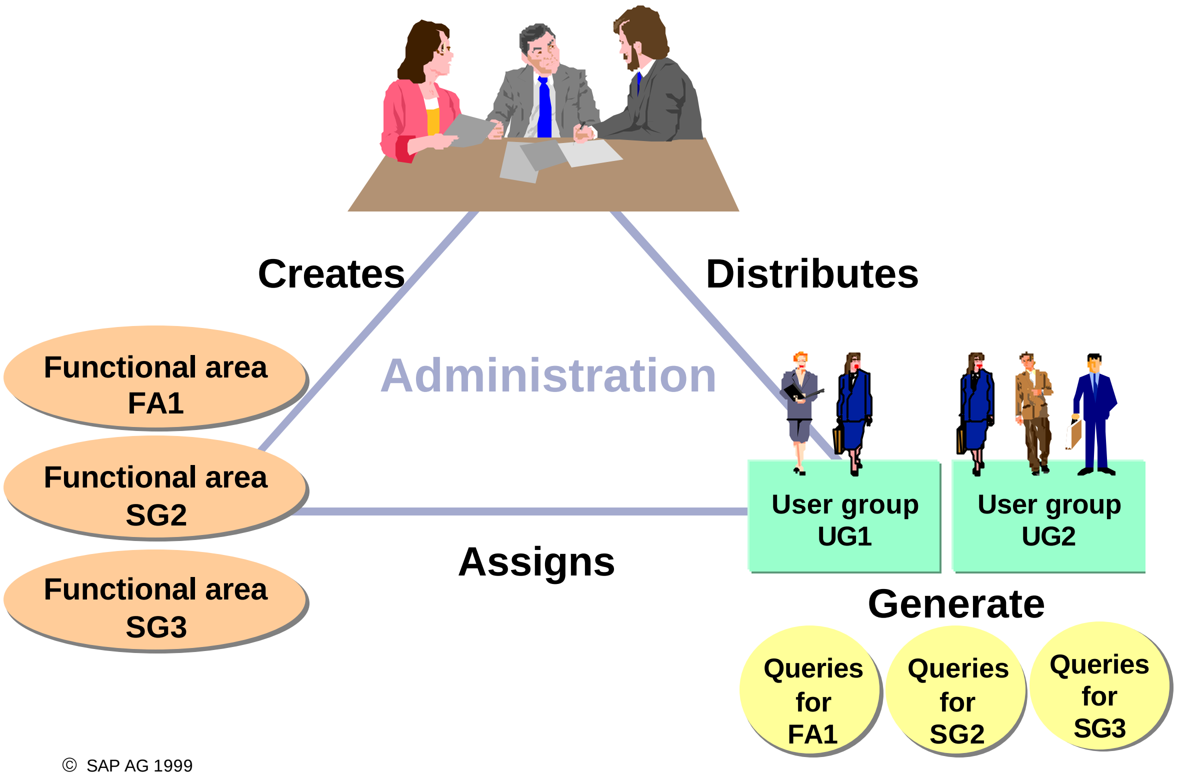
```
REPORT ...

START-OF-SELECTION.
...
WRITE ...
```

- When you create a list with a report, the data is usually retrieved via a logical database, processed by the report and then output as a list.
- Queries evaluate data and can be created without any prior programming knowledge using the SAP Query tool.
- The query results in a sequence of screen fields which you use to describe the line structure and list layout. Starting in Release 4.6A, you can use the Query Painter to add graphics to query lists.
- When the query is started, an internal report generator creates a program that corresponds to the list definition. That program then reads the data, processes it, and outputs the data as a list. The program is named AQmmbbbbbbbbbbbbqqqqqqqqqqqqqq. You can display the report names with the menu path displayed in appendix documentation AQL-1.

  mm - encoded client (standard area) or ZZ (global area)
  bbbbbbbbbbbb - Name of user group (12 places)
  qqqqqqqqqqqqqq - Name of query (14 places)

  Spaces in query program names are replaced with '='.

Organization of Query



Creates

Distributes

Administration

Functional area FA1

Functional area SG2

Functional area SG3

Assigns

User group UG1

User group UG2

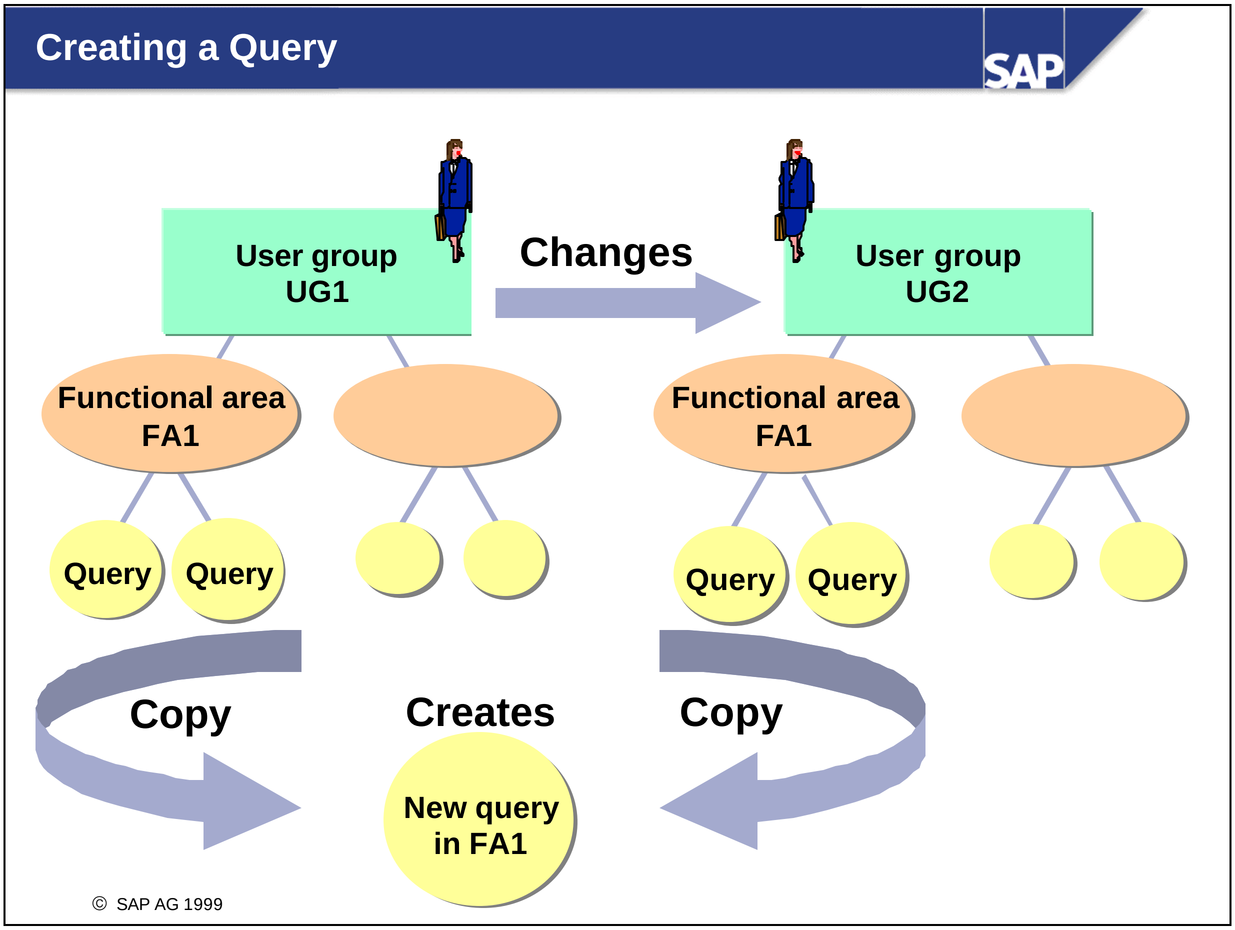
Generate

Queries for FA1

Queries for SG2

Queries for SG3

- The administrative tasks in the query environment include creating functional areas and user groups, as well as assigning the functional areas to the user groups.
- The functional area determines the tables (and the fields of those tables) to which a query can refer. Functional areas are frequently based on logical databases.
- Users may create and start queries only when they belong to at least one user group. A given user can belong to several user groups. Users in a user group all have the same privileges.
- Functional areas are allocated to a user group; the members of a group can access the functional area to which the group is allocated.
- A functional area can be allocated to several user groups.
- Several functional areas can be allocated to a user group.
- Queries are always created for a specific user group and a specific functional area. Users in a user group have access to all the queries allocated to that group.



Creating a Query

User group UG1

Changes

User group UG2

Functional area FA1

Query

Query

Functional area FA1

Query

Query

Copy

Creates

Copy

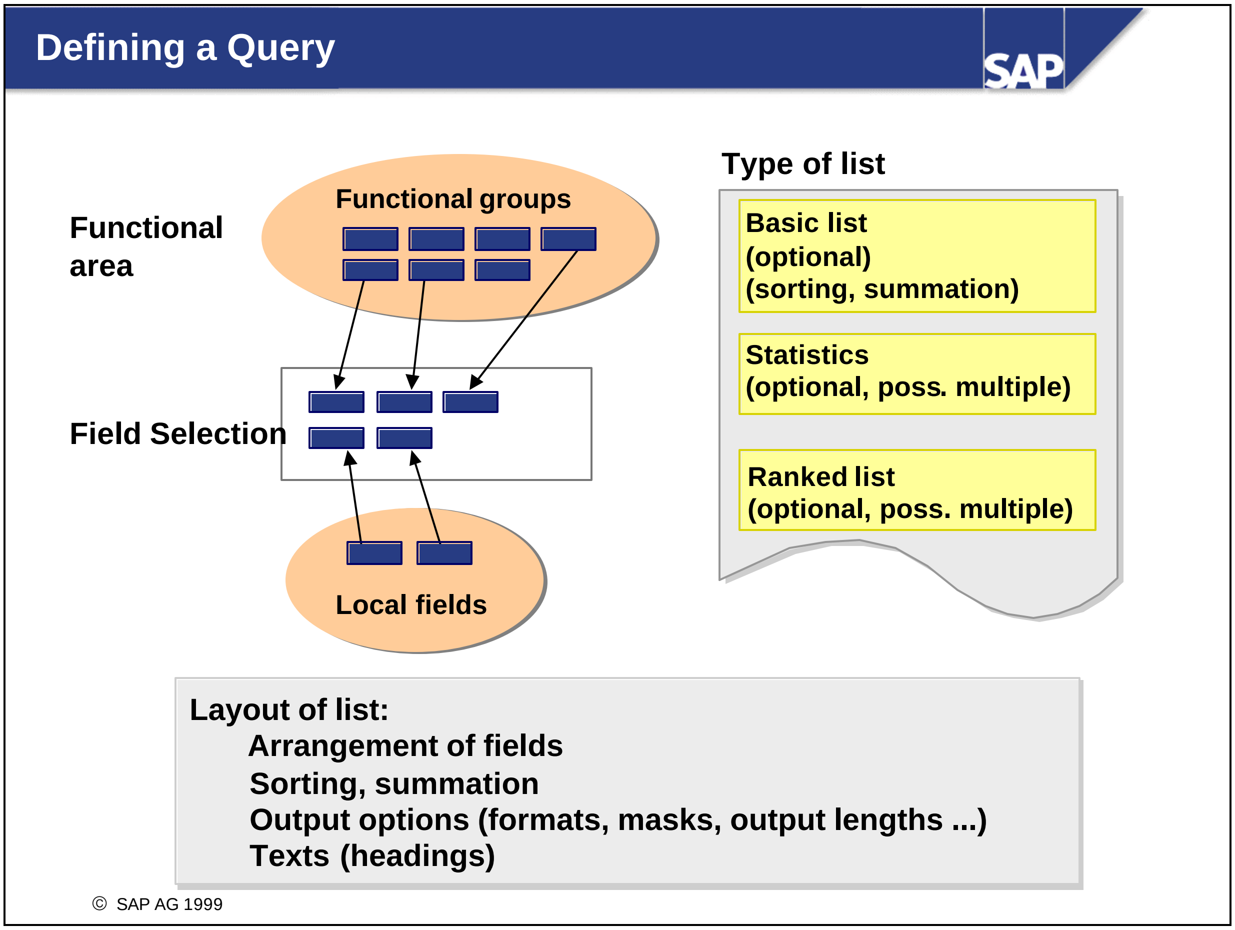
New query in FA1

- If you have been allocated to several user groups, you can switch within these groups.
- A query is always created from a specific functional area. The functional area must be allocated to the user group in which the query was created.
- You can access all queries that have been allocated to your user group.
- If you are authorized to define a query with a functional area, you can list all the queries for that functional area.
- You can only copy a query from a different user group to your user group when the functional area of the query to be copied has also been allocated to your user group.

SAP Query - Creating Lists

Overview

Queries



Defining a Query

Functional area

Functional groups

Field Selection

Local fields

Type of list

Basic list
(optional)
(sorting, summation)

Statistics
(optional, poss. multiple)

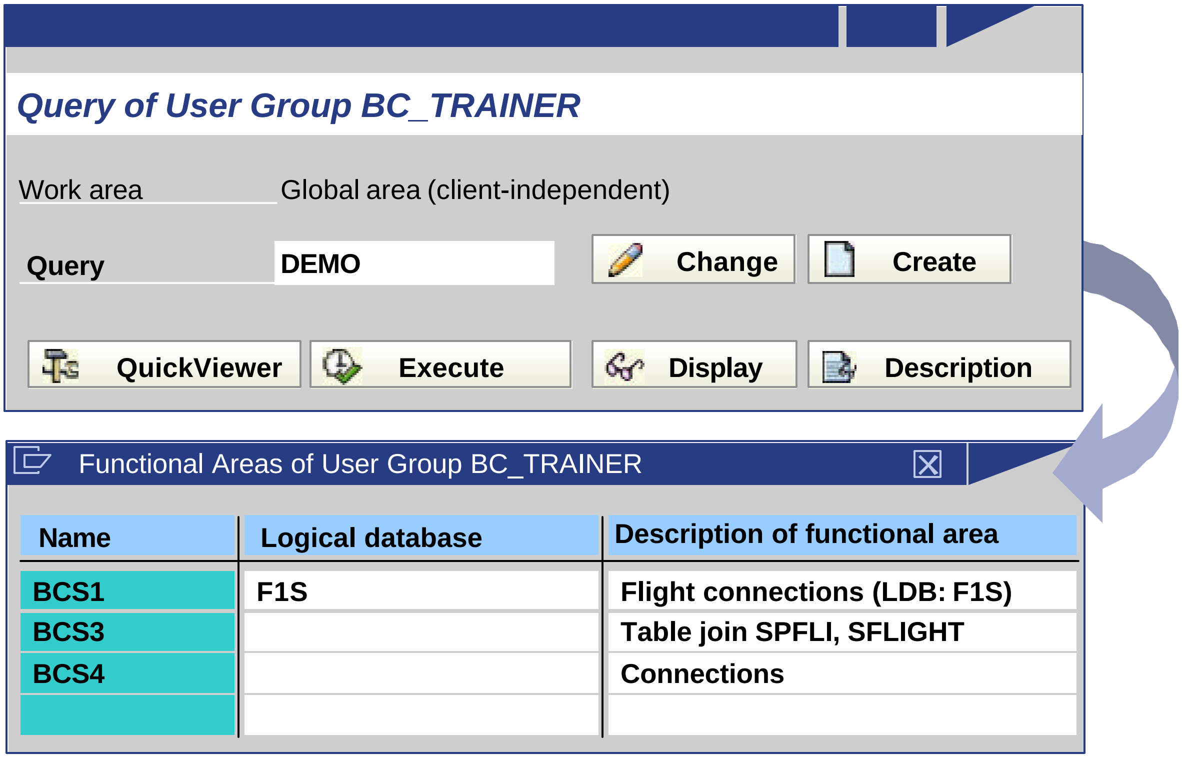
Ranked list
(optional, poss. multiple)

Layout of list:
Arrangement of fields
Sorting, summation
Output options (formats, masks, output lengths ...)
Texts (headings)

- The query results in a sequence of screen fields in which you use
  - Selection (checkboxes)
  - Number assignment (sequence, sort, ...)
  - Texts (headers, group level texts)

  to determine the line structure and the list layout.
- Starting in Release 4.6A, you can use the Query Painter to add graphics to basic lists.
- You can use SAP Query to generate different types of lists (partial lists):
  - Basic List: Single line or multiline. Multiline basic lists can be compressed.
  - Statistics, ranked lists: Require a numeric field. Data can be compressed.
  - You can combine different partial lists in a single query. Starting in 4.6A, you can also print the individual partial lists.
- You can also define local fields within a query, which means you can calculate new values from the collected data.
- While you cannot generate interactive lists you have defined yourself, some standard interaction functions are available. For example, you can pass on the generated lists for further processing (Excel, EIS, ABC analysis), display them in graphical form (SAP Graphics), save them, or edit them in table form (table control and ALV grid control).

Selecting the Work Area and Functional Area



Query of User Group BC_TRAINER

Work area Global area (client-independent)

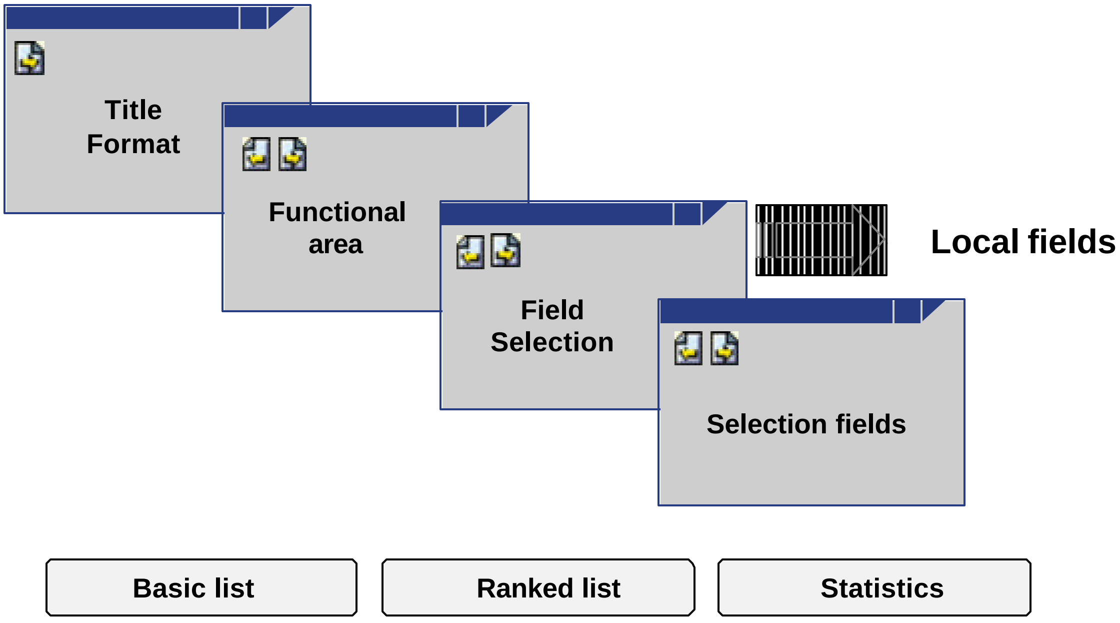
Query DEMO

Functional Areas of User Group BC_TRAINER

Name	Logical database	Description of functional area
BCS1	F1S	Flight connections (LDB: F1S)
BCS3		Table join SPFLI, SFLIGHT
BCS4		Connections

- You can use the menu paths displayed in appendix documentation AQL-2 to create, change, and execute queries with the ABAP Workbench.
- Queries are created either in the standard area or the global area. A query area covers a set of query objects that are internally complete and consistent - this means objects with the same name but with a different meaning can exist in the various query areas. The global and standard areas have separate namespaces.
- The standard area is client-specific and is not linked to the Workbench Organizer (WBO). The query objects in the global area are available in all clients and linked to the WBO. If you create a query in the global area, you have to assign it to a development class.
- When creating a query, you must first choose a functional area. The system displays all the functional areas that have been assigned to your user group. Once you have chosen a functional area, you cannot modify your choice: the functional area is the basis for data retrieval.
- SET/GET parameters AQW and AQB are available and can be used in your user parameters to define default settings for the query area (global area: AQW = G) and your user group.

Creating Local Fields



- When selecting fields, the system leads you through the following screens:
  - Title, format:

    Used to assign the query title You can set the page layout by making entries for the format. You can also set additional characteristics for the query with special attributes.
  - Functional area

    Functional areas are divided into functional groups. These form logical groups of data. You choose the required functional groups here.
  - Field Selection

    Here you choose the required data fields of the previously selected functional groups. If you require local fields, you can also define them here.
  - Selection fields:

    You can define fields to add to the selection screen and further limit the selection criteria.
- Depending on which type of list you want to generate, edit the screen fields or use layout mode (Query Painter) for the basic list. You always have to use the *Field selection* screen field to create local fields.

- By defining local fields, you can generate additional information from the fields that are available in a functional area.
- If pre-existing fields are required for the definition of a local field, short descriptions must be provided (see the menu path displayed in appendix documentation AQL-3).
- A short description can be assigned for each field.
- Short descriptions are also used to retrieve values of the corresponding fields in the list headers.

- You can define local fields for a query (menu path AQL-4)
- Local fields are defined with calculation rules. In the simplest case, calculation rules consist of a single formula formed with normal mathematical rules and consisting of operands and operators.
- The calculation of a field's value can be made condition-dependent. In this case, values are calculated according to certain rules only when a particular condition is met. If the condition remains unmet, the field receives a default value. Multiple conditions are allowed.

- You can sort the values of key columns of statistics in ascending or descending order.
- Numerical fields in statistics are accumulated. Statistics only make sense with numerical fields.
- Statistics allow you to display the average value, the percentage breakdown, and the number of records read for each numerical field.
- You can define up to 9 statistics individually or as a supplement to a basic list.
- If you work with different currency or quantity fie lds within statistics, you must enter a reference currency or a reference unit for each field, so that the system can convert it into that currency or unit.
- The list displays the conversions processed by the system. In the event of an error, the system logs any conversions that did not take place. In addition, the system highlights the affected currency or quantity fields within the statistics.
- With the appropriate definition, subtotal lines can also appear within statistics. If you compress the statistics, the system displays only the subtotal lines and the grand total.

- Ranked lists are special forms of statistics. However, they are always sorted based on **one** numerical value. This value is referred to as the ranked list criterion. In addition, the system only outputs a certain number of records. As a result, ranked lists are appropriate for tasks like: "Which 10 flight connections have the highest sales"?
- Ranked lists are sorted according to only one field, and the number of output lines is limited.
- You can define up to 9 ranked lists individually or as supplements to a basic list.
- You can also define each ranked list as statistics.
- The rules for conversions of currency and quantity fields also apply to ranked lists.

- To create basic lists, use the Query Painter. In the Query Painter, the screen is divided into four areas. The available fields (data source) are displayed to the left in tree form. The list structure is displayed with sample data in the upper right area. Information for the currently active element is displayed in the lower left portion of the window. Links to documentation and any warnings that are output while formatting the list are displayed in the lower right section of the window.
- You can edit list characteristics (frame, width) by selecting a field, right-clicking with the mouse and choosing 'List options' from the menu. While editing, you are working in the lower left window. If you have created new characteristics, then you need to confirm the values you have changed using the *APPLY* function.
- You can edit list line characteristics (color, separators, and so on) by selecting a field, right-clicking with the mouse and choosing 'Line options' from the menu. While editing, you are working in the lower left window. If you have created new characteristics, then you need to confirm the values you have changed using the *APPLY* function.
- You can edit field characteristics in the lower left window by selecting the appropriate field. Further field characteristics are available in the menu displayed with the right mouse button.
- You can move column and list headers to a mode that is ready for input by double-clicking.

- Selecting a field in the upper left window automatically adds that field to the list (is appended at the end of the current line). The individual fields are represented by field values. Sample data records are read from the source. If this is not possible, field values are simulated. The structure of the layout determines the structure of the subsequent list - that is, it contains the order of the fields, the headers, the colors, totals lines, and so on. To display the list structure for multiline hierarchy lists, several sample records are read and displayed.
- In addition, tools are available in the Query Painter to design the list. You can change the arrangement of the tools with *drag and drop*. Select the tool, such as the trash (a frame is displayed), with the left mouse button. You can now drag the selected area to the new position as long as you keep pressing the left mouse button.
- You can also use *drag and drop* to edit the list. Example:You want to change the field sequence. To do this, point the mouse at the field you want to move, click and hold the left mouse button (the cursor changes), drag the field to the desired location, and release the mouse button. To delete a field, just drag it to the trash.
- You can also change the output position and output length with entries in the lower left window. Press *Apply* to apply your values to the list structure.

- You can set up control level lists. To do this, you have to determine the sort fields. The sort sequence can be defined in either ascending or descending order separately for each field. To create a sort field, drag a field from the list to the *Sort* tool.
- You can define control levels with or without a total at the end of the control level (subtotal). You can change the text accompanying the subtotals.
- If you total a field, the total is output to the same column as the field, with the same output length. Accordingly, the output length may be too short and result in an overflow (an asterisk appears in the first position of the value). To prevent overflows of totals, you can simply increase the output length of the field you wish to total.
- You can output blank lines and/or force a page break before outputting control levels.
- You can hide and change introductory and concluding texts for control levels.
- The system automatically creates a currency distribution for currency totals.

- **List overview:** If your list consists of several partial lists, for example a basic list, two statistical lists and a ranked list, the system offers you the ability to display the partial lists individually. The partial lists can also be printed separately.
- **Report/report interface (RRI):** You can use this interface to call query programs (receiver) and other reports (sender). Additional information is available in the online documentation.
- **Table display:** The list is displayed as a table control or using the ALV grid control. Starting in Release 4.6A, you can also display multiline lists. The different lines are summarized in one line.
- **Graphics:** The information contained in a list can be displayed with SAP Presentation Graphics.
- **File storage, private storage:** Saves the data as a file on the presentation server or in the private folders. For more information, please refer to online documentation QD02.
- **Word processing** and spreadsheets: Transfer data to MS Word or Excel (for example)
- **ABC analysis, EIS:** Additional information is available in appendix documentation QD02.
- **Selection:** Indicates which selections were input in the selection screen.
- **Drilldown functions:** For expanding and collapsing the list.
- **Totaling:** Totals for numeric fields.

- You can save a list generated by a query using the menu path AQL-5 and re-display it later.
- Subsequent display of a saved list does not require database access to retrieve data. Such a display is therefore much quicker than restructuring the data running the query again.
- Saving a list stores the list itself and supplemental information. Storage of additional information is a special function of saving lists that is supported only by query. This makes it possible to perform interactive functions in the saved list.
- When a query is integrated in an area menu (not the AQ... query program), then all the saved lists are automatically passed on to the area menu, and can be displayed there. All interactive functions remain available.
- If you save the list 'normally' (using menu path AQL-6), then no interactive functions are available in the saved list.

Unit: SAP Query - Creating Lists

Topic: Creating a Query List

When you have completed these exercises, you will be able to:

- Create a multi-line query list with local fields

1-1 Create a **query QE1-##** in user group **BC_STUDENTS** using **functional area BCS1** in the **global work area**. Note: ## stands for your group number. The sample solution, EXERS_01, is available in the global work area under user group BC_TRAINER.

1-2 Maintain the short texts for the query and set the column width to 90 columns.

1-3 Create a local field.

1-3-1 Assign short names for the fields ***Occupied seats*** (OCC) and ***Maximum occupancy*** (MAX).

1-3-2 Create a local field, ***Empty seats*** with short name ***FREE*** and header ***Free*** in functional group ***Flights***. The field should have the same attributes as the ***Maximum occupancy*** field.

1-3-3 Use this field to calculate the number of available seats as the difference between the maximum occupancy and the number of occupied seats.

2-1 Create a multi-line basic list in the Query Painter (layout mode).

Line 1 Airline and flight code

Line 2 Departure city, Arrival city, Departure time, Distance, Distance unit of measure

Line 3 Flight date, Occupied seats, Available seats, Maximum occupancy, Percentage occupancy of the flight, Price, Current total revenue for the flight

3-1 List format

3-1-1 Output the list with frames.

3-1-2 Line 1: Color *Header(intensify),* one blank line before the line

Line 2: Color *Header*

Line 3: Color *Normal*

Field Available seats: Color *Positive*

3-1-3 Modify the standard length for the following fields:

Occupied seats to 8 places

Available seats to 8 places

Maximum occupancy to 8 places

Percentage occupancy to 6 places

Current total revenue for the flight to 15 places

Flight price to 10 places

3-1-4 Change the header of field “Percentage occupancy” to “%”.

Optional (sample solution EXERS_01_OPT, see above)

4-1 Copy query QE1-## to QE1-##_OPT.

4-1-1 Instead of outputting the number of available seats, you can use a traffic signal icon to display the information.

4-1-2 Now assign property *Icon* to local field *Available seats*, and use the complex calculations to determine the logical conditions under which ICON_RED_LIGHT (red light) or ICON_GREEN_LIGHT (green light) will be displayed. You can reduce the output length of the field to 6 places, and make the field color identical to the line color.

4-1-3 Specify the *Flight connection code* as a sort field. Sort in descending order.

4-1-4 Display the group level header in a frame. Do not total or count at the end of the group level.

Unit: SAP Query - Creating Lists

Topic: Creating a Query List

1-1 Start SAP Query from the Workbench. Use menu *Environment* -> *Query areas* to switch to the global query area. Press the *Change user group* button and change to user group BC_STUDENTS. Create a query named QE1-## (## stands for the group number). Create a query using functional area BCS1.

1-1-1 Maintain the short text and list width in the initial screen.

1-1-2 Press the *Next screen* pushbutton and scroll forward to screen *FIELD SELECTION*.

1-1-3 Activate the short names under menu *Edit* -> *Short names* and enter short names for all specified fields.

1-1-4 Use menu item *Edit* -> *Local field* to create the required field. Enter short name MAX in field *Same attributes as field*. Model the difference between MAX and OCC in the calculation formula. Save the local field.

2-1 Now press the *Basic list* pushbutton. Start the Query Painter.

2-1-1 Select the required fields in the upper left corner of the Query Painter (data fields). Make sure that you set up the list in the order in which you selected the fields. Otherwise you will have to re-sort the fields accordingly.

2-1-2 To change the attributes of a field, select the field. This displays the field in the lower left window, and you can now change its attributes. Press *APPLY* to activate your changes.

2-1-3 Choose item *Line options* from the context menu (right mouse button). Choose *Line options* from the context menu.

2-1-4 Double-click on a header text to change it. The text field is displayed ready for input.

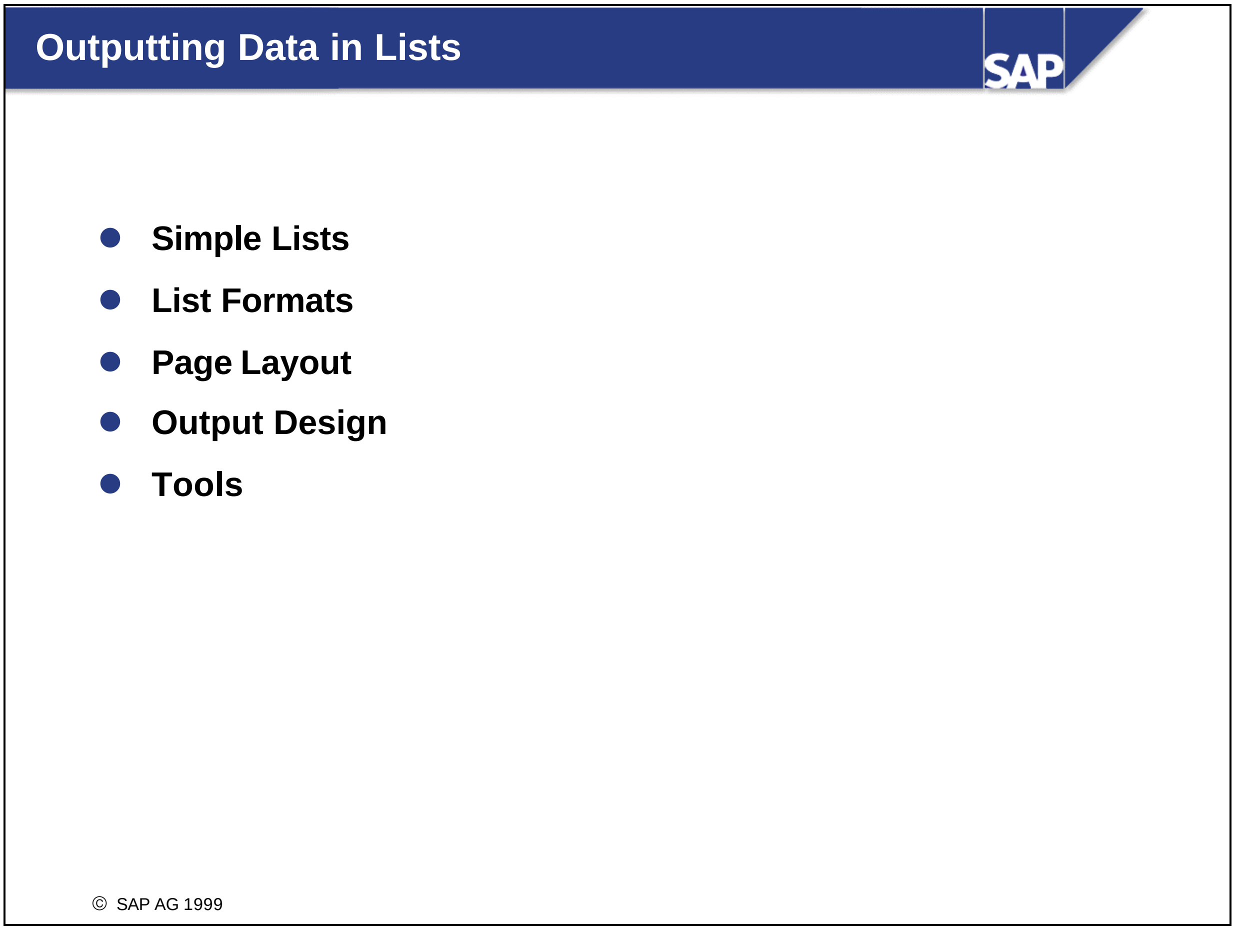
2-1-5 Save the Query.

3-1 Optional

3-1-1 Branch to the maintenance of the local fields (see above). Change the attribute of the local field to *Icon*. Press the *Complex calculations* pushbutton. Enter MAX = OCC under the first condition and enter ICON_RED_LIGHT under the corresponding formula. Enter OCC < MAX under the second condition and enter ICON_GREEN_LIGHT under the corresponding formula.

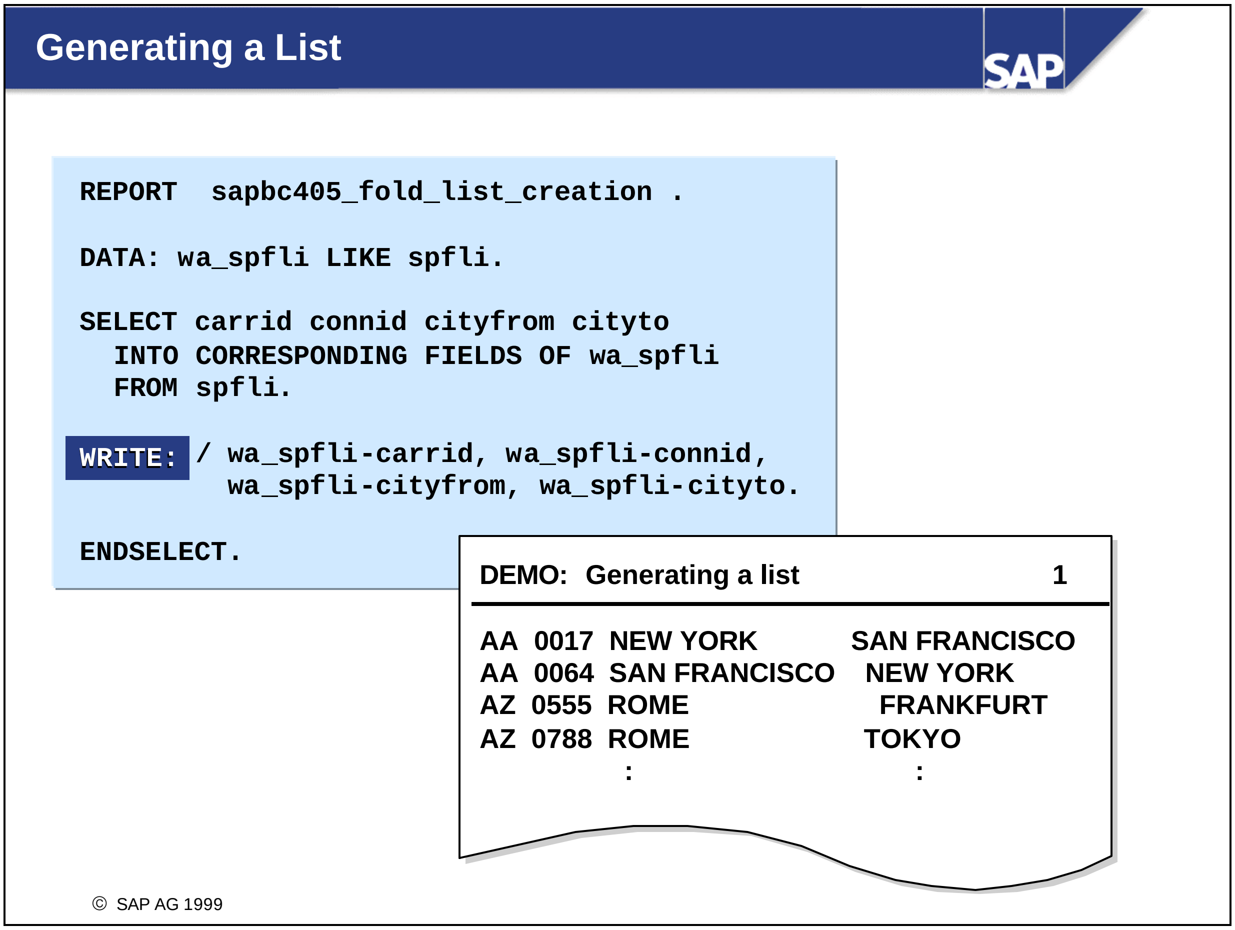
3-1-2 Start the Query Painter. Select the required sort field and drag it to the *Sort fields* box. When you select a field in the *Sort fields* box, the attributes of the selected control level appear in the lower left window. Press *APPLY* to activate your changes.

3-1-3 Save the Query.



Outputting Data in Lists

- Simple Lists
- List Formats
- Page Layout
- Output Design
- Tools

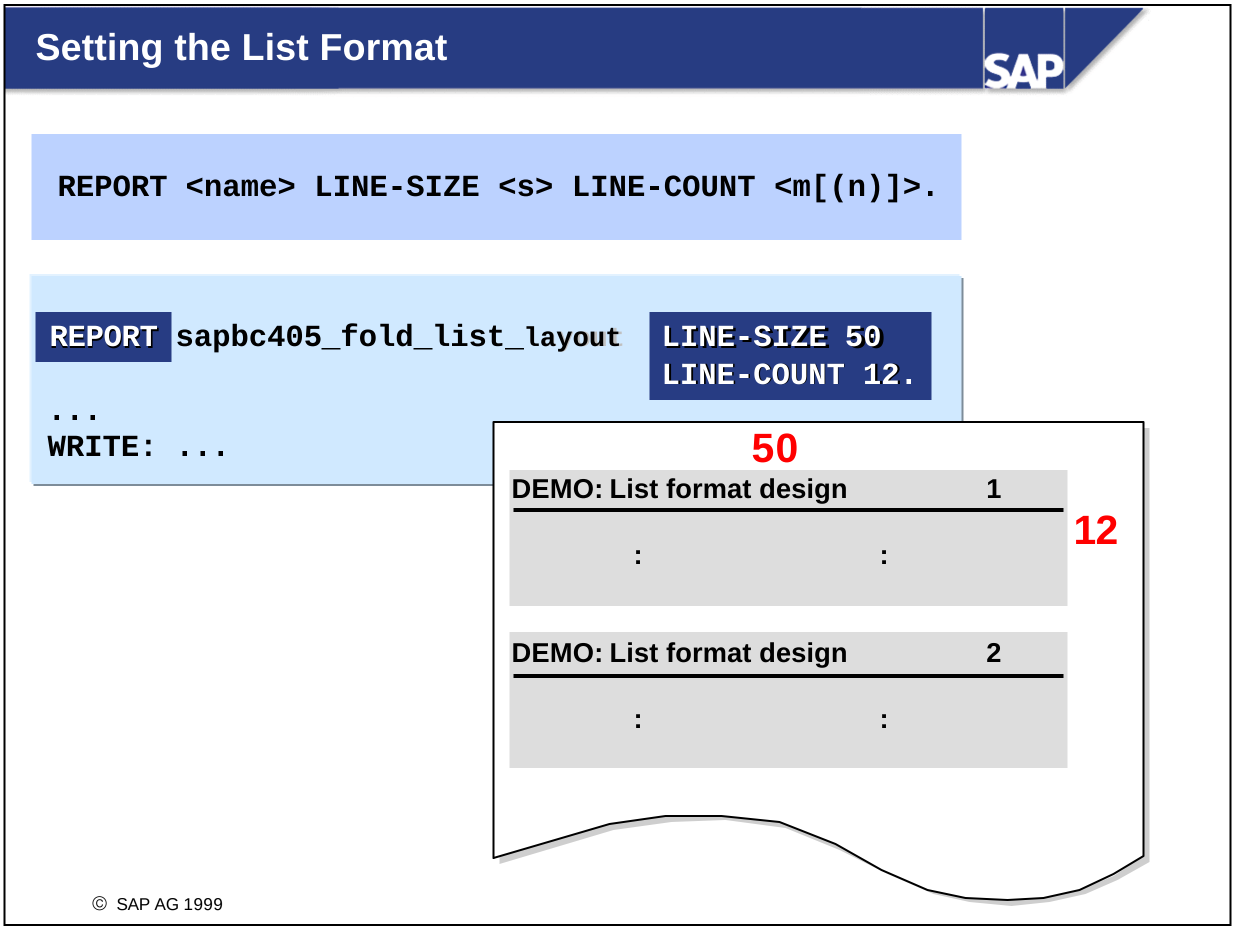


Generating a List

```
REPORT  sapbc405_fold_list_creation .

DATA: wa_spfli LIKE spfli.

SELECT carrid connid cityfrom cityto
  INTO CORRESPONDING FIELDS OF wa_spfli
  FROM spfli.

WRITE: / wa_spfli-carrid, wa_spfli-connid,
         wa_spfli-cityfrom, wa_spfli-cityto.

ENDSELECT.
```

```
DEMO:  Generating a list                       1
------------------------------------------------
AA  0017  NEW YORK           SAN FRANCISCO
AA  0064  SAN FRANCISCO       NEW YORK
AZ  0555  ROME                 FRANKFURT
AZ  0788  ROME                NOTKYO
                :                   :
```

- The first WRITE statement in an ABAP program triggers list generatio n. The system first writes the data intended for output to a list buffer. Once all the data has accumulated in the list buffer and the system has processed all events, the system generates the screen image from the list buffer.
- By default, the list consists of a "continuous" page (maximum 60,000 lines).
- The maximum length of a line is 1,023 characters. To support maintenance and improve performance, lists should be only as long as necessary.
- As a standard function, the system generates two header lines (standard header). The first header line contains the program title in the upper left corner and the page number in the upper right corner. The second header line consists of an unbroken line. Both header lines remain in the window when you scroll.
- When you print a list, the first line of the header appears as follows: Upper left: System date Center: Program title Upper right: Page number

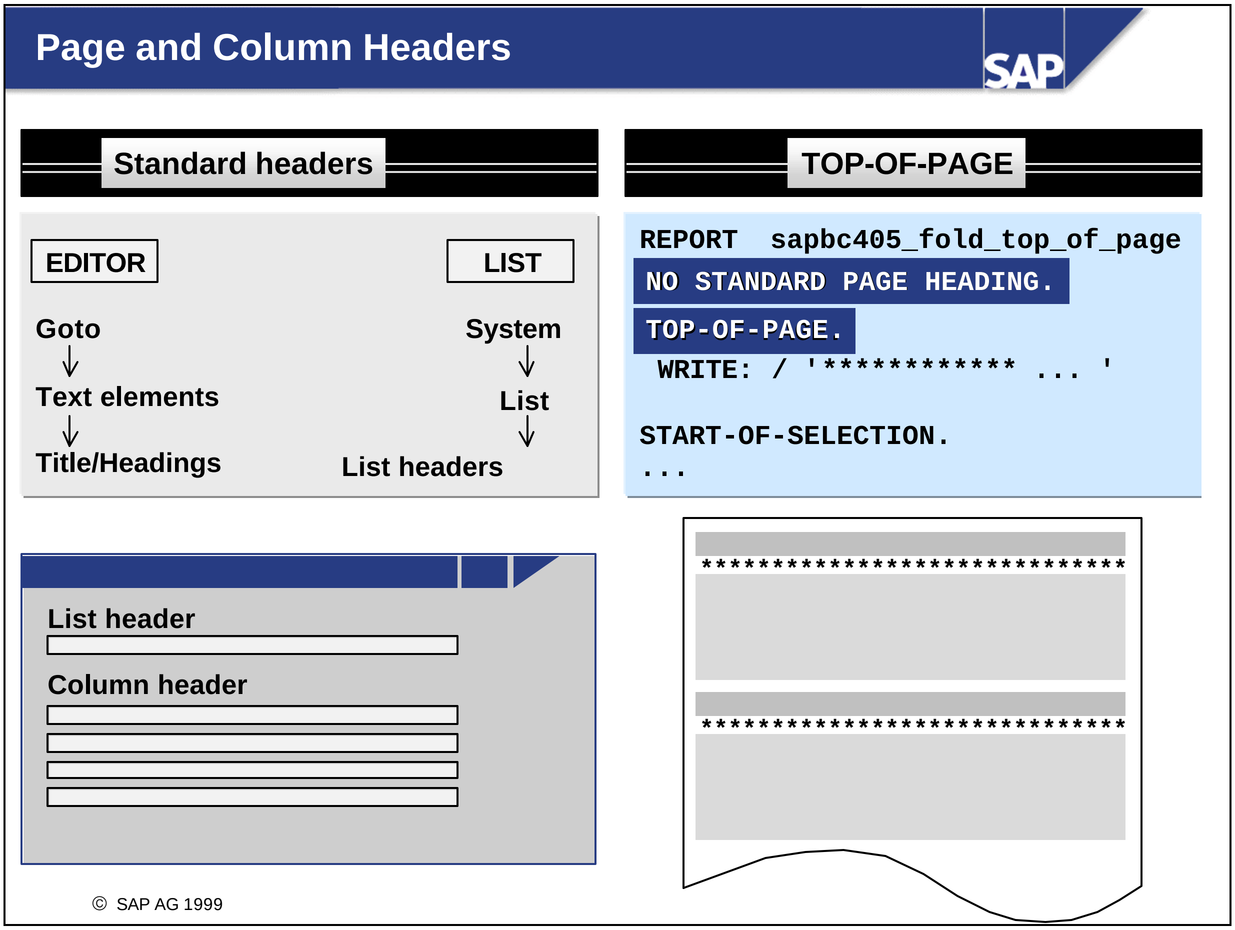


Setting the List Format

```
REPORT <name> LINE-SIZE <s> LINE-COUNT <m[(n)]>.
```

```
REPORT sapbc405_fold_list_layout    LINE-SIZE 50
                                    LINE-COUNT 12.
...
WRITE: ...
```

- Use the additions **LINE-SIZE <s>** and **LINE-COUNT <m>** with the **REPORT** statement to create global definitions for column and line length for all list levels. The different list levels are created during interactive reporting.
- Within a list level, you can use **NEW-PAGE LINE-COUNT <s>** to change the number of lines on a page - this value overrides the definition in the **REPORT** statement. The width of a list can only be changed by creating a new list level. If you want to use the default values, then set <s> and/or <m> to zero.
- You cannot use variables for <s> or <m>.
- An optional addition in the **REPORT** statement, n, reserves a line for the footer. To create a footer, you have to program the **END-OF-PAGE** event.



Page and Column Headers

Standard headers

EDITOR

Goto
↓
Text elements
↓
Title/Headings

LIST

System
↓
List
↓
List headers

List header

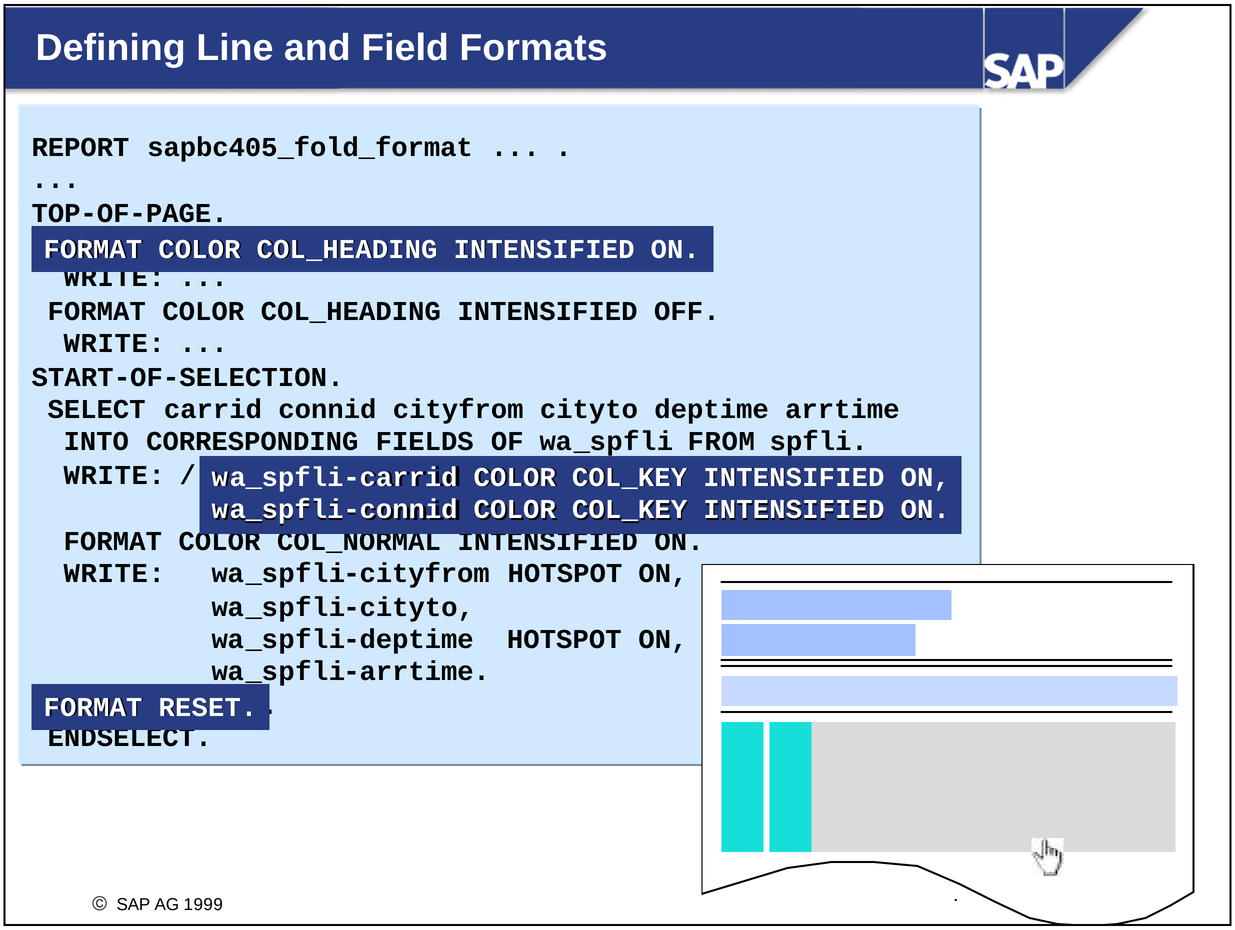
Column header

TOP-OF-PAGE

```
REPORT  sapbc405_fold_top_of_page
NO STANDARD PAGE HEADING.
TOP-OF-PAGE.
 WRITE: / '************  ... '

START-OF-SELECTION.
...
```

```
*******************************
*******************************
```

- You can maintain a list header (page header) and up to four column headers for a list. You can maintain the headers with the Editor or from the list itself. Maintenance from the list itself offers an advantage: since it is displayed on the screen, positioning of elements, especially column headers, is simpler. Headers appear automatically the next time the program is started in that list. If you have not maintained a list title, the system uses the program name as a default (system field SY-TITLE).
- The addition **NO STANDARD PAGE HEADING** in the **REPORT/PROGRAM** statement suppresses the output of list/column headers. You can override this global setting with **NEW-PAGE NO-TITLE/WITH-TITLE** and **NO-HEADING/WITH-HEADING**. All texts that you enter in the standard list headers are saved language-specifically, and can be translated later.
- The **TOP-OF-PAGE** event can be used to generate any page headers. **TOP-OF-PAGE** is especially useful when you want to output variables in the headers. All texts should be written as text elements, to allow them to be translated later.
- **TOP-OF-PAGE** is triggered whenever a new page is created (**WRITE**, **ULINE**, and so on). If you do not suppress the standard list headers, they appear above the lines generated by **TOP-OF-PAGE**. Lines generated by **TOP-OF-PAGE** remain in the window during vertical scrolling.



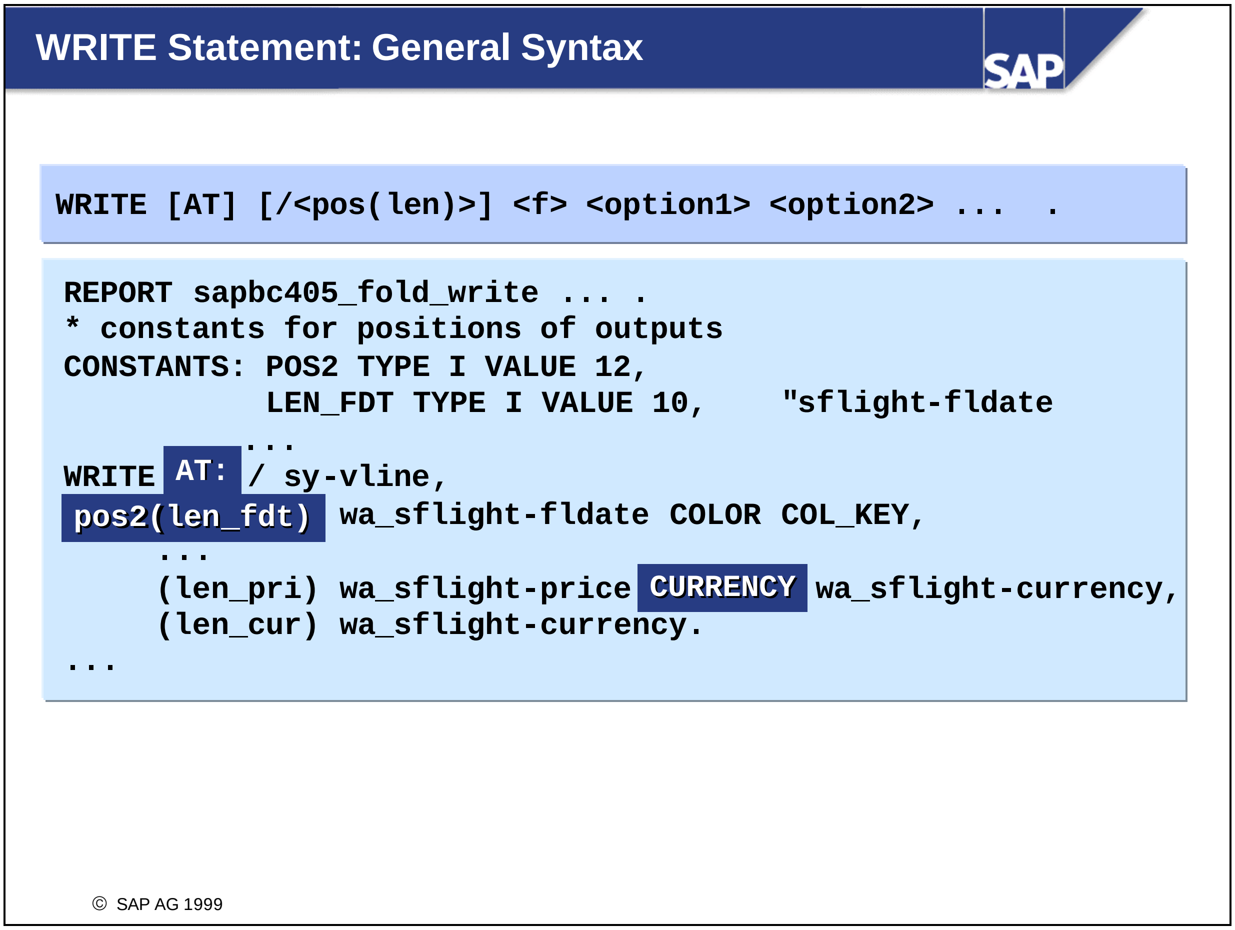
Defining Line and Field Formats

```
REPORT sapbc405_fold_format ... .
...
TOP-OF-PAGE.
FORMAT COLOR COL_HEADING INTENSIFIED ON.
  WRITE: ...
 FORMAT COLOR COL_HEADING INTENSIFIED OFF.
  WRITE: ...
START-OF-SELECTION.
 SELECT carrid connid cityfrom cityto deptime arrtime
  INTO CORRESPONDING FIELDS OF wa_spfli FROM spfli.
  WRITE: / wa_spfli-carrid COLOR COL_KEY INTENSIFIED ON,
           wa_spfli-connid COLOR COL_KEY INTENSIFIED ON.
  FORMAT COLOR COL_NORMAL INTENSIFIED ON.
  WRITE:   wa_spfli-cityfrom HOTSPOT ON,
           wa_spfli-cityto,
           wa_spfli-deptime  HOTSPOT ON,
           wa_spfli-arrtime.
FORMAT RESET.
 ENDSELECT.
```

- You can use any of the following **FORMAT** options:

COLOR <n>	[ON|OFF]	Colors the line background
INTENSIFIED	[ON|OFF]	Intensify colors YES|NO
INVERSE	[ON|OFF]	Inverse: Background/text color
HOTSPOT	[ON|OFF]	Display mouse pointer as hand and single click with mouse button (see **AT LINE-SELECTION**)
INPUT	[ON|OFF]	Input field
RESET		Resets all formats to their default values

- The formats set with **FORMAT** take effect with the next **WRITE** statement.
- You can use all **FORMAT** options with the **WRITE** statement, but the options will affect only the one field in which they appear.
- **FORMAT** options in a **WRITE** statement change the global formatting instructions (set with a **FORMAT** statement) for the field.
- At each new event, the system resets all **FORMAT** options to their default values.



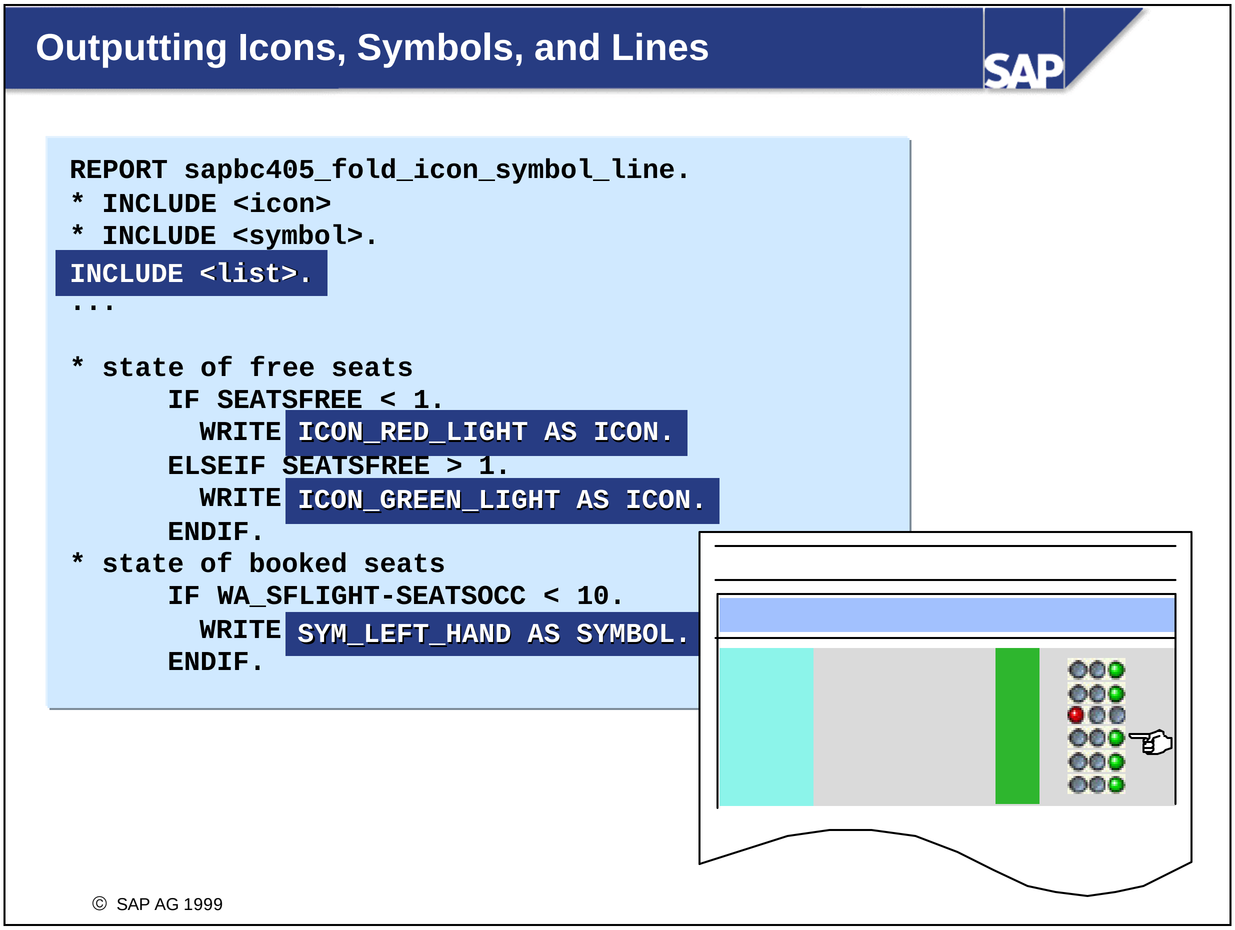
WRITE Statement: General Syntax

```
WRITE [AT] [/<pos(len)>] <f> <option1> <option2> ...  .
```

```
REPORT sapbc405_fold_write ... .
* constants for positions of outputs
CONSTANTS: POS2 TYPE I VALUE 12,
           LEN_FDT TYPE I VALUE 10,    "sflight-fldate
          ...
WRITE AT: / sy-vline,
pos2(len_fdt) wa_sflight-fldate COLOR COL_KEY,
     ...
     (len_pri) wa_sflight-price CURRENCY wa_sflight-currency,
     (len_cur) wa_sflight-currency.
...
```

- **NO-GAP** Suppresses output of spaces after the <f> field.
  Fields output directly after each other appear without gaps.
- **NO-ZERO** If the contents of field <f> are equal to zero, only spaces are output.
  If f is of type C or N, spaces replace leading zeros.
- **DD/MM/YY** If <f> is a date field (type D), its contents are not processed according to user parameters (and according to the option).
- **CURRENCY <key>** determines the number of decimal places for currency amounts in the list output. The specified key is used to read the number of decimal places in table TCURX.
- **UNIT <key>** determines the number of decimal places for quantities in the list output. The specified key is used to read the number of decimal places in table T006.
- **USING EDIT MASK <mask>** Outputs according to the formatting template<mask>.
- **UNDER** <g> The output begins at the column in which field <g> was output.
- **LEFT-JUSTIFIED** Left-justified output (default for types C, N, D, T, X).
- **CENTERED** Centered output within the output length.
- **RIGHT-JUSTIFIED** Right-justified output (standard for all number fields: I, P and F)

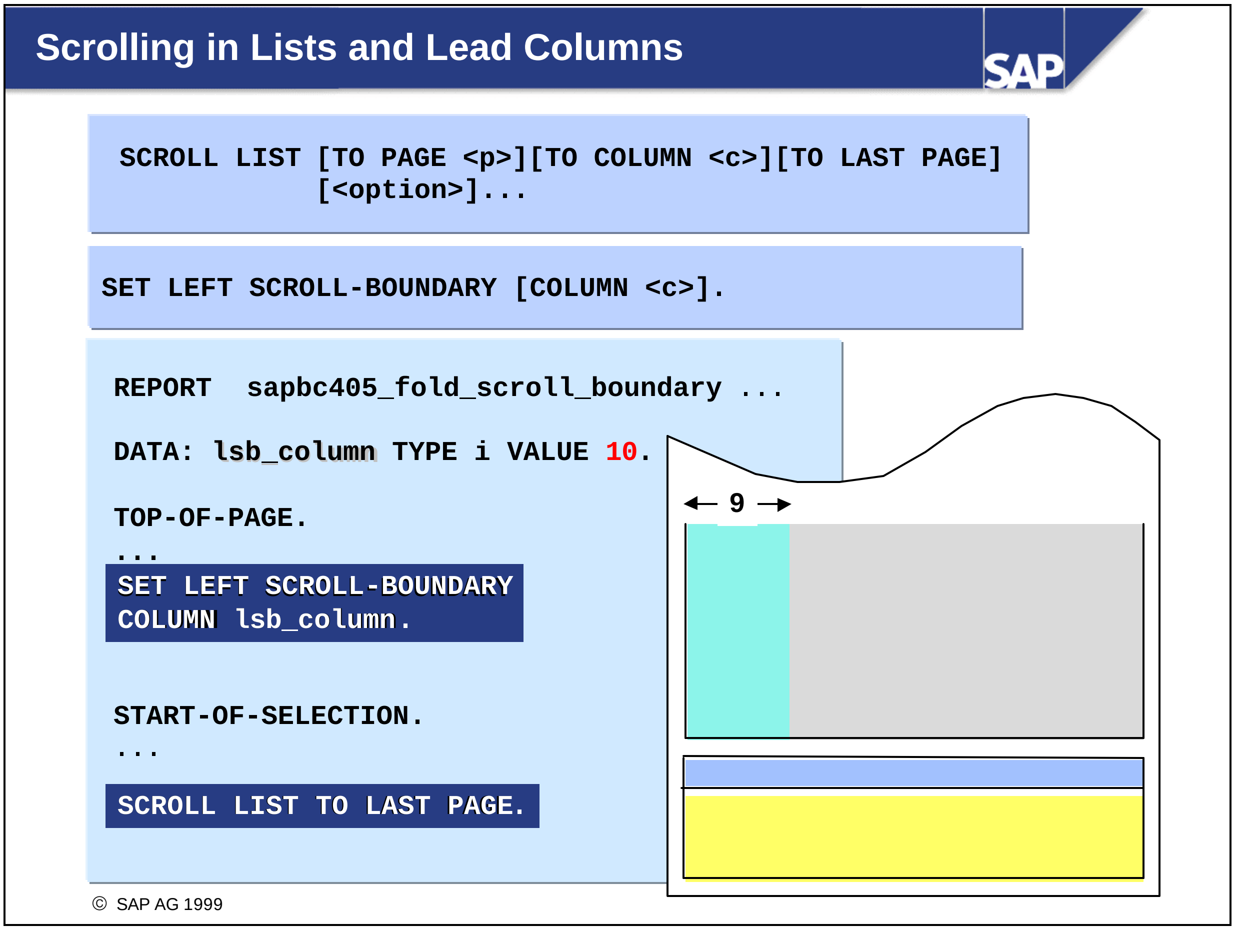
■ You can find a complete list of all WRITE options in the online documentation.



Outputting Icons, Symbols, and Lines

```
REPORT sapbc405_fold_icon_symbol_line.
* INCLUDE <icon>
* INCLUDE <symbol>.
INCLUDE <list>.
...

* state of free seats
      IF SEATSFREE < 1.
        WRITE ICON_RED_LIGHT AS ICON.
      ELSEIF SEATSFREE > 1.
        WRITE ICON_GREEN_LIGHT AS ICON.
      ENDIF.
* state of booked seats
      IF WA_SFLIGHT-SEATSOCC < 10.
        WRITE SYM_LEFT_HAND AS SYMBOL.
      ENDIF.
```

- You can use the **AS SYMBOL** option of the **WRITE** statement to include symbols in lists. The symbolic names of these characters are defined in include program <symbol>.
- You can also insert icons into the list with **WRITE <f> AS ICON**. To do this, you have to link the include program <icon> in your program.
- You can link the include program <list> to use both symbols and icons in the list.
- You can find an overview of available symbols and icons in the online documentation or in the statement examples for WRITE.
- To generate a horizontal line, use the **ULINE** statement, system field sy-uline in a **WRITE** statement, or several minus signs in a WRITE statement
- To generate a vertical line, use system field sy-vline in a **WRITE** statement
- To generate special lines, like the upper-right corner, use **line_top_right_corner AS LINE** .
- You can use these elements to frame a list, to separate titles from a list with horizontal lines, to separate columns with vertical lines, and to create table and trees.



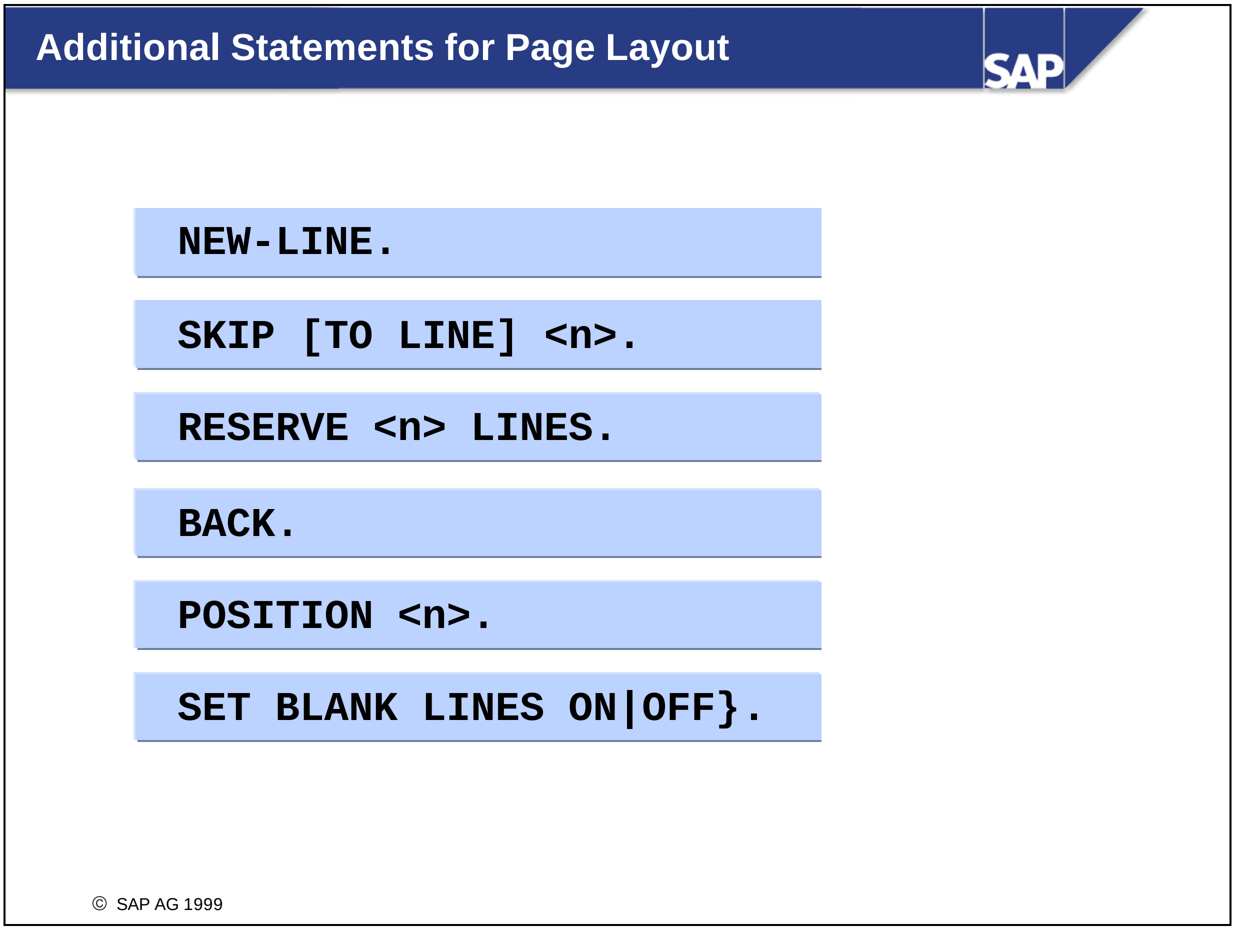
Scrolling in Lists and Lead Columns

```
SCROLL LIST [TO PAGE <p>][TO COLUMN <c>][TO LAST PAGE]
            [<option>]...
```

```
SET LEFT SCROLL-BOUNDARY [COLUMN <c>].
```

```
REPORT  sapbc405_fold_scroll_boundary ...

DATA: lsb_column TYPE i VALUE 10.

TOP-OF-PAGE.
...
SET LEFT SCROLL-BOUNDARY
COLUMN lsb_column.


START-OF-SELECTION.
...

SCROLL LIST TO LAST PAGE.
```

- You can use **SET LEFT SCROLL-BOUNDARY** to set hard lead columns for a list: the lead columns remain in place during horizontal scrolling. Without an additional parameter, the system uses the current write position (SY-COLNO) as the left margin. The margin (limit) must be reset for every new page (at **TOP-OF-PAGE** for example).
- You can use **NEW-LINE NO-SCROLLING** to prevent shifting the next list line during horizontal scrolling. For example, you can use this function to ensure that the comment lines are always visible.
- You can use the **SCROLL** statement to scroll to any place in the list at runtime; for example, the system could automatically display the last page of the list.



Additional Statements for Page Layout

```
NEW-LINE.

SKIP [TO LINE] <n>.

RESERVE <n> LINES.

BACK.

POSITION <n>.

SET BLANK LINES ON|OFF}.
```

- NEW-LINE new line, corresponds to "/" in the WRITE statement.
- RESERVE <n> LINES If the current page does not have space for at least <n> more lines in the list structure, a page feed is generated.
- SKIP <n> <n> blank lines are output
  SKIP TO LINE <n> Next output in line n (you can also skip backwards in the list)
- BACK without RESERVE Return to the first line in the current page after TOP-OF-PAGE
  With RESERVE: Return to the first line output after RESERVE
- POSITION <n> Next output position in column <n> of the current line
- SET BLANK LINES ON Any blank lines created through the output of blank fields are output
- SET BLANK LINES OFF No blank lines are output (default setting)

- Text elements include the standard headers, text symbols, and selection texts. The text elements are saved language-specifically, separate from the source text. This allows subsequent translation. The logon language determines the language in which the system displays text elements.
- You should define the output length of the text symbols as large as possible, as this determines how much space is available for the translations.
- Text symbols can be addressed in programs in the following ways:
  - TEXT-xxx (xxx is a three-character string)
  - 'string' (xxx)
- You can reconcile the text symbols and the program when you use the second method of creating the text symbols. If a text symbol has been maintained, it is always output in the list. The extended syntax check returns an error if you forgot to maintain the text symbols.

- During generation of a list, the ABAP runtime system fills the following system fields:

SY-LINCT	Number of lines from REPORT statement (LINE-COUNT)
SY-LINSZ	Line width from REPORT statement (LINE-SIZE)
SY-SROWS	Number of lines in the display window
SY-SCOLS	Number of columns in the display window
SY-PAGNO	Page number of the current page
SY-LINNO	Line number of the current line of the current page (SY-PAGNO)
SY-COLNO	Column number of the current column

- During generation of the list, the system fills the last three system fields continuously.

- A number of standard list functions are available in the standard list interface.

Unit: Outputting Data in Lists
Topic: Formatting

When you have completed these exercises, you will be able to:

- Define list formats
- Set headers
- Set Hard Lead Columns
- Create a page break

1-1 Copy program template SAPBC405_FOLT_1 to Z##FOL1_... Sample solution for exercise: SAPBC405_FOLS_1.

The following functionality has been implemented in the template: A selection screen is displayed with a selection for the airline codes. The data is read from the database to an internal table, ***it_flights***, using a database view at START-OF-SELECTION, and should be output in subroutine DATA_OUTPUT at END-OF-SELECTION.

The objective of this exercise is to insert the data output in subroutine DATA_OUTPUT in the LOOP … ENDLOOP loop.

1-1-1 In the TOP include, define the list width as 100 columns and suppress the standard list header.

1-1-2 Output the following data in the list:

Airline code	wa_flights-carrid
Flight number	wa_flights-connid
Flight date	wa_flights-fldate
Departure city	wa_flights-cityfrom
Arrival location	wa_flights-cityto
Flight price	wa_flights-price
Local currency of airline	wa_flights-currency

Display the data according to the template.

1-1-3 Output the list header in color COL_HEADING with intensive display, and the column headers in color COL_HEADING with a less intensive display. Output the key information (CARRID, CONNID) in color COL_KEY with intensive display and the list body in COL_NORMAL with intensive display. Use horizontal lines to separate the headers from the list body. Output the price as a currency amount.

Optional: Add a frame (sy-vline) as shown in the template.

1-1-4 Flag the international flights (wa_flights-countryto <> wa_flights-countryfr) with an icon (ICON_BW_GIS).

1-1-5 Set the page break so that the data for **one** flight connection fits on exactly **one** page. To do this, use the control structure: ON CHANGE OF … ENDON.

1-2 Implement event TOP-OF-PAGE.

1-2-1 Output the list and column headers there. Refer to the template for details. Use text elements to allow your texts to be translated. To position the elements, use constants that you declare in the TOP include.

1-2-1 Make sure that the airline code and connection number are fixed during horizontal scrolling. Ensure that the statement SET LEFT SCROLL-BOUNDARY is only valid for a single page.

Template (Displaying Data in Lists):

```
Flight data
-----------------------------------------------------------------------
     Flight     Date          Departure city  Arrival location     Price
-----------------------------------------------------------------------
 @   AZ    0555               30.09.1999      ROME                 FRANKFURT
                360.202  ITL
 @   AZ    0555               19.11.1999      ROME                 FRANKFURT
                360.202  ITL
 @   AZ    0555               22.11.1999      ROME                 FRANKFURT
                360.202  ITL
 @   AZ    0555               29.11.1999      ROME                 FRANKFURT
                360.202  ITL
 @   AZ    0555               19.12.1999      ROME                 FRANKFURT
                360.202  ITL
```

```
Flight data
-----------------------------------------------------------------------
     Flight     Date          Departure city  Arrival location     Price
-----------------------------------------------------------------------
 @   LH     0400              30.09.1999      FRANKFURT            NEW YORK
                672.50 DEM
 @   LH     0400              19.11.1999      FRANKFURT            NEW YORK
                672.50 DEM
 @   LH     0400              22.11.1999      FRANKFURT            NEW YORK
                672.50 DEM
 @   LH     0400              29.11.1999      FRANKFURT            NEW YORK
                672.50 DEM
 @   LH     0400              19.12.1999      FRANKFURT            NEW YORK
                672.50 DEM
 @   LH     0400              21.12.1999      FRANKFURT            NEW YORK
                672.50 DEM
```

@ = ICON_ BW_GIS

Unit: Outputting Data in Lists

Topic: Formatting

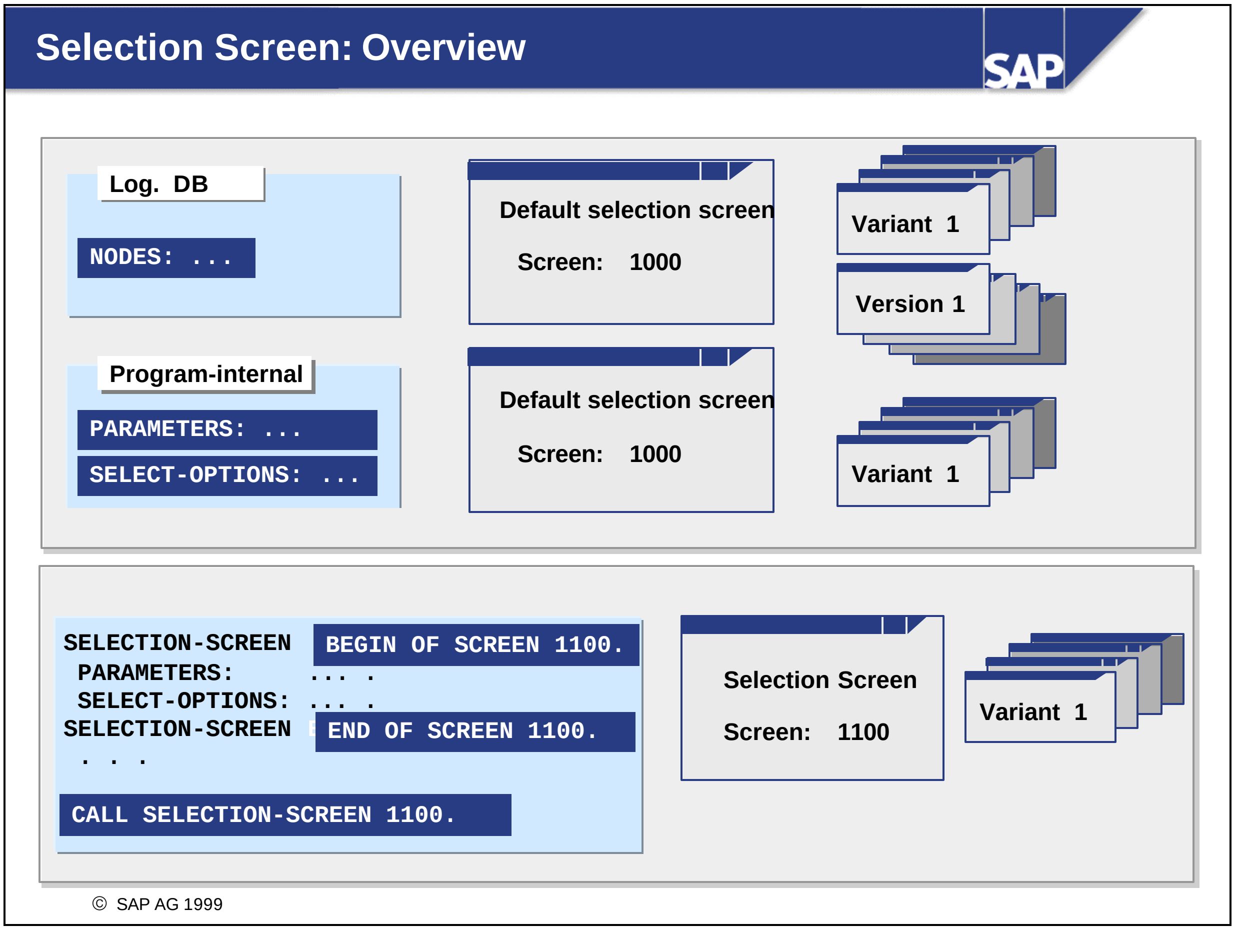
```
*&---------------------------------------------------------------------*
*& Report  SAPBC405_FOLS_1                                              *
*&                                                              *
*&---------------------------------------------------------------------*
*& Exercise 1; Outputting Data in Lists                           *
*&                                                              *
*&---------------------------------------------------------------------*

INCLUDE bc405_fols_1top.

*&---------------------------------------------------------------------*
*&   Event TOP-OF-PAGE
*&---------------------------------------------------------------------*
TOP-OF-PAGE.

* Title
  FORMAT COLOR COL_HEADING INTENSIFIED ON.
  ULINE.
  WRITE:      / sy-vline,
              'Flight data'(001),
              AT line_size sy-vline.
  ULINE.

* Column header
  FORMAT COLOR COL_HEADING INTENSIFIED OFF.
  WRITE:  sy-vline, AT pos_c1 'Flight'(002).

* Fix left scroll boundary
  SET LEFT SCROLL-BOUNDARY.
  WRITE:      'Date'(003) ,
              'Departure location'(004),
                      'Price'(006),
                      AT line_size sy-vline.
  ULINE.

*&---------------------------------------------------------------------*
*&   Event START-OF-SELECTION
*&---------------------------------------------------------------------*
START-OF-SELECTION.

* Filling internal table with flight data using a DDIC view.
  SELECT * FROM dv_flights INTO TABLE it_flights
  WHERE carrid IN so_car.

*&---------------------------------------------------------------------*
*&   Event END-OF-SELECTION
*&---------------------------------------------------------------------*
END-OF-SELECTION.

SORT it_flights BY carrid connid fldate.

* Data output
  PERFORM data_output.

*&---------------------------------------------------------------------*
*&      Form  DATA_OUTPUT
*&---------------------------------------------------------------------*
*       List output of flight data
*----------------------------------------------------------------------*
FORM data_output.

* Loop at the internal table for writing data
  LOOP AT it_flights INTO wa_flights.

* Get a new page if CONNID has changed.
    ON CHANGE OF wa_flights-connid.
      NEW-PAGE.
    ENDON
```

```
* Mark international flights
   FORMAT COLOR COL_KEY INTENSIFIED ON.
   IF wa_flights-countryfr EQ wa_flights-countryto.
     WRITE: / sy-vline, icon_space AS ICON CENTERED.
   ELSE.
     WRITE: / sy-vline, icon_bw_gis AS ICON CENTERED.
   ENDIF.

* Data output
   WRITE:    wa_flights-carrid,
             wa_flights-connid.
   FORMAT COLOR COL_NORMAL INTENSIFIED OFF.
   WRITE:    wa_flights-fldate,
                     wa_flights-cityfrom,
                     wa_flights-cityto,
                     wa_flights-price CURRENCY wa_flights-currency,
                     wa_flights-currency,
                     AT line_size sy-vline.

  ENDLOOP.

ENDFORM.                              " DATA_OUTPUT



*&---------------------------------------------------------------------*
*& Include BC405_FOLS_1TOP                                             *
*&                                                                     *
*&---------------------------------------------------------------------*

REPORT bc405_fols_1top LINE-SIZE 100 NO STANDARD PAGE HEADING.

* Include for using icons
INCLUDE <icon>.

* Constants for writing position
CONSTANTS: pos_c1 TYPE i VALUE 6
```

```
            line_size TYPE i VALUE 100.

* Internal table like DDIC view DV_FLIGHTS
DATA:       it_flights LIKE TABLE OF dv_flights,
            wa_flights LIKE dv_flights.

* Selection screen
SELECT-OPTIONS so_car FOR wa_flights-carrid.
```

Selection Screen

- Generate
- Design
- Input checks
- Variants



Selection Screen: Overview

Log. DB

```
NODES: ...
```

Program-internal

```
PARAMETERS: ...
SELECT-OPTIONS: ...
```

Default selection screen

Screen: 1000

Variant 1

Version 1

Default selection screen

Screen: 1000

Variant 1

```
SELECTION-SCREEN BEGIN OF SCREEN 1100.
 PARAMETERS:     ... .
 SELECT-OPTIONS: ... .
SELECTION-SCREEN END OF SCREEN 1100.
 . . .

CALL SELECTION-SCREEN 1100.
```

Selection Screen

Screen: 1100

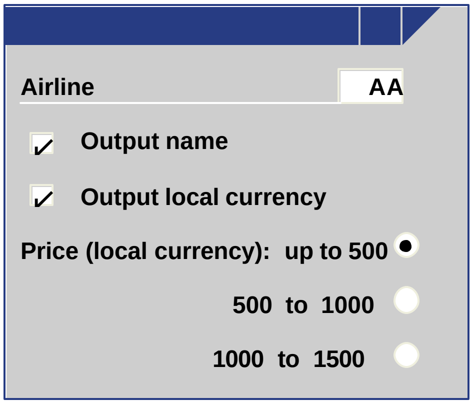
Variant 1

- Selection screens serve as the interface between the program and the user, and allow, for example, limitation of the amount of data to be read from the database.
- Logical databases supply selection screens whose concrete appearance is dependent on the specified node name (**NODES<name>**). Selection screen versions (if supplied by the logical database) offer a subset of default selection screens.
- You can use the declarative language elements **PARAMETERS** and **SELECT-OPTIONS** to generate a default selection screen (screen 1000) with input-ready fields.
- In addition to the default selection screen, you can generate additional selection screens with **SELECTION-SCREEN BEGIN ....** and call them with **CALL SELECTION-SCREEN**.
- You can create variants of a selection screen. A variant is a user-specific selection variant. You would create a screen variant if you frequently start a program with the same selection variants or start in background processing.

Declaring Fields with PARAMETERS

```
PARAMETERS: <f>[TYPE <type>][DECIMALS <n>][LIKE <f1>][MEMORY ID <pid>]
               [OBLIGATORY][DEFAULT <wert>]
               [AS CHECKBOX]
               [RADIOBUTTON GROUP <grp>]
```

```
REPORT  sapbc405_sscd_checkbox_radiobutton.
... .
PARAMETERS: pa_carr  LIKE sflight-carrid,
            pa_name  AS CHECKBOX DEFAULT 'X',
            pa_curr  AS CHECKBOX DEFAULT 'X',
            pa_lim_1  RADIOBUTTON GROUP lim,
            pa_lim_2  RADIOBUTTON GROUP lim,
            pa_lim_3  RADIOBUTTON GROUP lim.
CONSTANTS mark VALUE 'X'.

* Check, if any checkbox has been selected
IF pa_name EQ mark. ... . ENDDIF.
IF pa_curr EQ mark. ... . ENDDIF.

* Check, which radiobutton has been selected
CASE mark.
  WHEN pa_lim_1. ... .
  WHEN pa_lim_2. ... .
  WHEN pa_lim_3. ... .
ENDCASE.
```



Airline AA

☑ Output name

☑ Output local currency

Price (local currency): up to 500 ●

500 to 1000 ○

1000 to 1500 ○

- The **PARAMETERS** statement is a declarative language element. As in the case of the **DATA** statement, you can declare the fields with **TYPE** or **LIKE**. The system generates input-ready fields in the selection screen. The names of **PARAMETERS** fields can be up to 8 characters long. You can maintain selection texts (parameter names) with the function Text elements/Selection texts.
- You can set a default value with the **DEFAULT <value>** addition. If you assign a **MEMORY ID <pid>**, the system uses SAP Memory and the SET/GET parameter to set the default value. If you declare mandatory fields with the **OBLIGATORY** addition, users cannot leave the selection screen until values have been entered in these fields.
- You can also define parameters as checkboxes (**AS CHECKBOX**). Doing so creates a one-character field that can contain a " "(SPACE) or an "X". You can evaluate the contents of checkboxes using **IF/ENDIF** control structures.
- You can also define a series of radio buttons for the selection screen with the addition **RADIOBUTTON GROUP <grp>**. The maximum length name for a RADIOBUTTON GROUP <grp> is 4 characters. Only **one** radio button in a group can be active and can be evaluated during program processing. You can evaluate the contents of radio buttons using **CASE/ENDCASE** control structures.

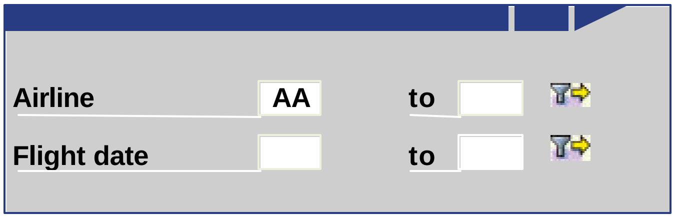
Selections with SELECT-OPTIONS

```
SELECT-OPTIONS: <seltab> FOR <f>.
```

```
REPORT  sapbc405_sscd_select_options .
... .

SELECT-OPTIONS: so_carr FOR sflight-carrid DEFAULT 'AA',
                so_fldt FOR sflight-fldate.
```

Internal Table
so_carr

Sign	Option	Low	High
I	EQ	AA	



Airline AA to

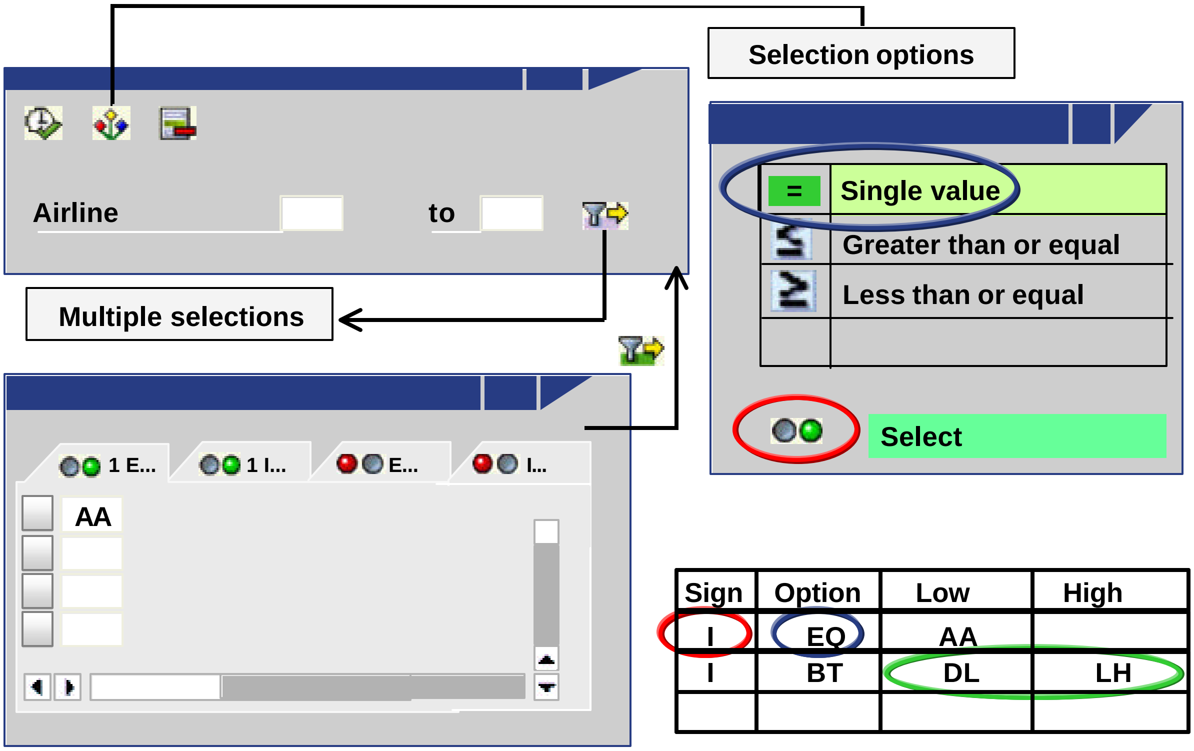
Flight date to

- The **SELECT-OPTIONS** statement is a declarative language element. In contrast to the **PARAMETERS** statement, it allows complex selections instead of just ***one*** input-ready field.
- **SELECT-OPTIONS** generates an internal table <seltab> with a standard structure. This consists of 4 fields: seltab-sign, seltab-option, seltab-low, and seltab-high. The name of selection table <seltab> can contain up to 8 characters. You can maintain selection texts (name of the selections) with the function **Text elements/Selection texts**.
- Use the addition FOR to specify the field against which the system should check the selection entries. This field must be declared in a **DATA** or **TABLES** statement. The fields seltab-low and seltab-high possess the same field characteristics as the check field.
- Each line of the selection table <seltab> formulates a condition using one of the following relational operators. The following values are possible:

  SIGN: I (Include), E (Exclude)
  OPTION: EQ, NE, LE, LT, GE, GT, BT(Between), NB (Not Between),
  CP (Contains Pattern), NP (Contains Pattern not)

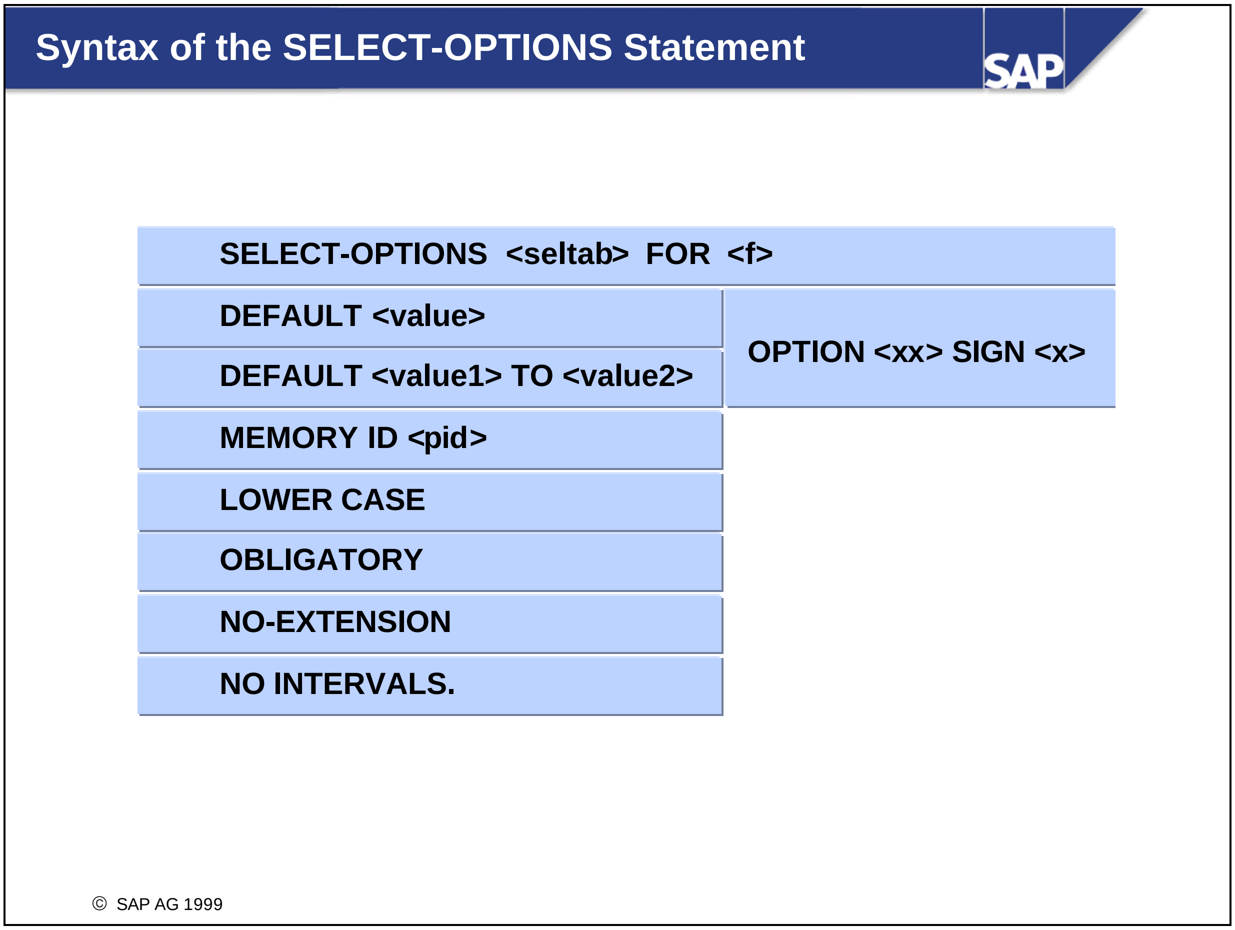
- The selection set is the union of all includes (I1,..., In) minus the union of all excludes (E1, ..., Em). If the table remains empty, selection is performed using the total selection set, if you are working in the **SELECT** statement with **WHERE IN <seltab>**.

Selection Options and Multiple Selections



Sign	Option	Low	High
I	EQ	AA	
I	BT	DL	LH

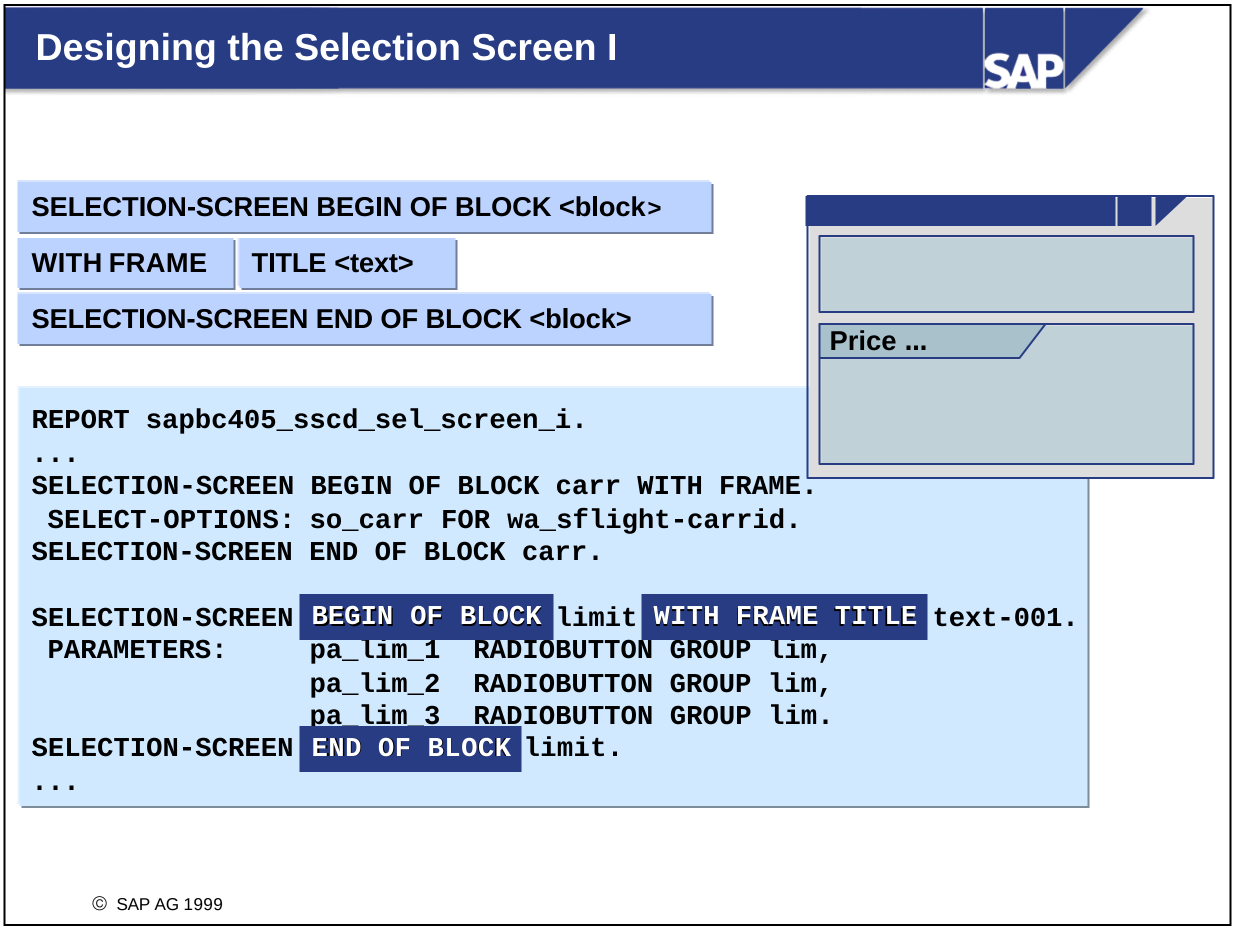
- When you make entries on a selection screen, the system populates the internal table <seltab>. Standard entries for the fields seltab-sign and seltab-option are **I** and **EQ** for individual selections, and **I** and **BT** for ranges.
- To change the default entries for seltab-sign and seltab-option, choose ***Selection options*** (double click on the appropriate entry field or activate the pushbutton). The system offers all the alternatives for fields seltab-sign and seltab-option that are appropriate for the selection. If the traffic signal icon is green during ***Select***, there is an **I** in seltab-sign; a red light indicates **E**.
- To delete a table entry, use the appropriate pushbutton (***Delete selection***).
- Every selection criterion can be used to make multiple selections unless defined otherwise. If multiple selections are present, the color of the arrow changes from white to green.



Syntax of the SELECT-OPTIONS Statement

```
SELECT-OPTIONS  <seltab>  FOR  <f>
DEFAULT <value>                          OPTION <xx> SIGN <x>
DEFAULT <value1> TO <value2>
MEMORY ID <pid>
LOWER CASE
OBLIGATORY
NO-EXTENSION
NO INTERVALS.
```

- Additions to the **SELECT-OPTIONS** statement:
  - **DEFAULT** enables you to set default values for seltab-low (single value) or seltab-low and seltab-high (interval). You can use **OPTION** and **SIGN** to set default values for seltab-option and seltab-sign that differ from the normal defaults.
  - **MEMORY ID <pid>** allocates a SPA/GPA parameter. The value stored in SAP Memory with the ID <pid> is placed in seltab-low (lower interval limit) when you call the selection screen.
  - **LOWER CASE** suppresses conversion of the entry into upper-case. This addition is not permitted for Dictionary fields, since the attribute set in the Dictionary takes precedence.
  - **OBLIGATORY** generates a mandatory field. A question mark appears in the entry field in the selection screen, and the user must enter a value.
  - **NO-EXTENSION** suppresses multiple single or multiple range selections.
  - **NO INTERVALS** suppresses the seltab-high (upper interval limit) entry on the selection screen. You can use the additional screen, **Multiple selection,** to enter ranges.
- If you entered a logical database in the attributes of the type 1 program, the selection screen of the logical database is processed. If you have programmed additional **SELECTION-OPTIONS** or **PARAMETERS** statements, the system displays them after the selections of the logical database.



Designing the Selection Screen I

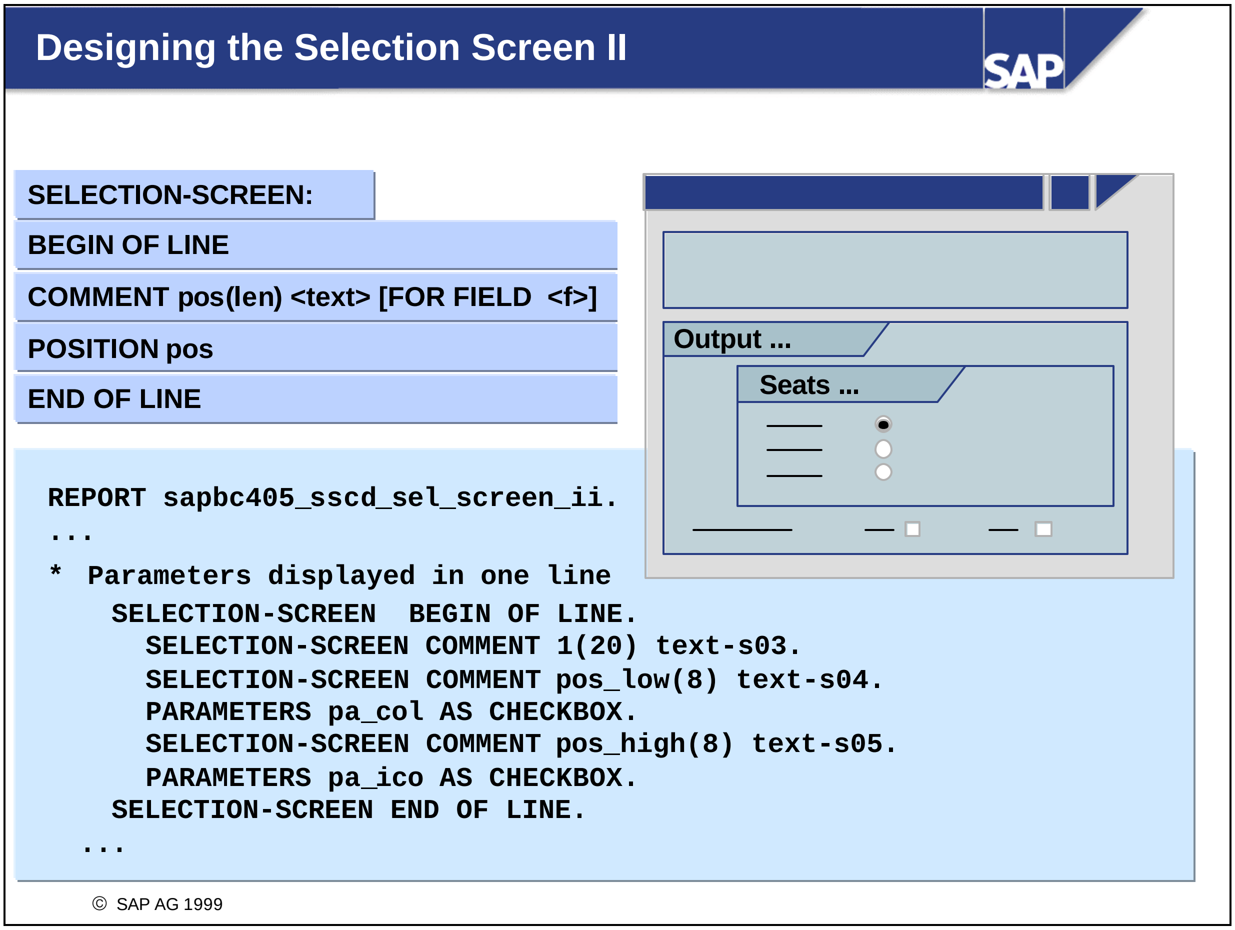
SELECTION-SCREEN BEGIN OF BLOCK <block>

WITH FRAME TITLE <text>

SELECTION-SCREEN END OF BLOCK <block>

```
REPORT sapbc405_sscd_sel_screen_i.
...
SELECTION-SCREEN BEGIN OF BLOCK carr WITH FRAME.
 SELECT-OPTIONS: so_carr FOR wa_sflight-carrid.
SELECTION-SCREEN END OF BLOCK carr.

SELECTION-SCREEN BEGIN OF BLOCK limit WITH FRAME TITLE text-001.
 PARAMETERS:      pa_lim_1  RADIOBUTTON GROUP lim,
                  pa_lim_2  RADIOBUTTON GROUP lim,
                  pa_lim_3  RADIOBUTTON GROUP lim.
SELECTION-SCREEN END OF BLOCK limit.
...
```

- You can use the **SELECTION-SCREEN** statement to design the layout of the selection screen. You can group selections that belong together logically with the supplemental **BEGIN OF BLOCK <block>** and place a frame around them using WITH FRAME. You can assign a title to the block, but you can only use the addition **TITLE <text>** together with a frame.
- You can nest framed blocks to a maximum of 5 frames.
- Before designing a selection screen, you should orient yourself to the screen design guidelines found in the sample transaction BIBS.



Designing the Selection Screen II

SELECTION-SCREEN:

BEGIN OF LINE

COMMENT pos(len) <text> [FOR FIELD <f>]

POSITION pos

END OF LINE

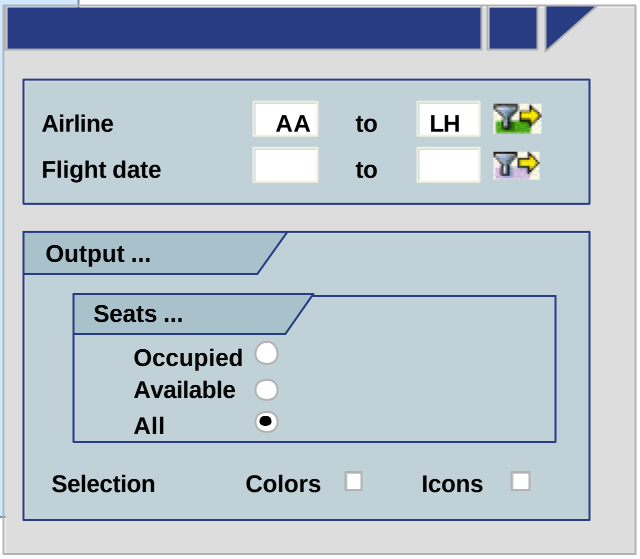
```
REPORT sapbc405_sscd_sel_screen_ii.
...
*  Parameters displayed in one line
     SELECTION-SCREEN  BEGIN OF LINE.
       SELECTION-SCREEN COMMENT 1(20) text-s03.
       SELECTION-SCREEN COMMENT pos_low(8) text-s04.
       PARAMETERS pa_col AS CHECKBOX.
       SELECTION-SCREEN COMMENT pos_high(8) text-s05.
       PARAMETERS pa_ico AS CHECKBOX.
     SELECTION-SCREEN END OF LINE.
  ...
```

- You can display multiple parameters and comments in one output line. To do so, you must enclose them between the **SELECTION-SCREEN BEGIN OF LINE** and **SELECTION-SCREEN END OF LINE** statements. The **COMMENT** parameter enables you to include text in the line.
- Comment texts must always have a format (position and output length). The position can be set with a data field, pos_low or pos_high. These are the positions for fields seltab-low and seltab-high on the selection screen.
- Adding **COMMENT ... FOR FIELD <f>** ensures that the F1 Help for field <f> is displayed for the comment text and for the parameter itself. If you hide the parameter (selection variant: attribute invisible) the comment text is also hidden.
- You can use **POSITION <pos**> to set the cursor for the next output position (only within **... BEGIN OF LINE ... END OF LINE**).

Initializing the Selection Screen

`INITIALIZATION.`

```
REPORT  sapbc405_sscd_initialization.
...
INITIALIZATION.
  MOVE:  mark TO  pa_all.

   MOVE: 'I'  TO so_carr-sign,
         'BT' TO so_carr-option,
         'AA' TO so_carr-low,
         'LH' TO so_carr-high.
    APPEND so_carr.
  CLEAR so_carr.
    MOVE:  'E'  TO so_carr-sign,
           'EQ' TO so_carr-option,
           'DL' TO so_carr-low.
     APPEND so_carr.
...
```



Airline AA to LH

Flight date to

Output ...

Seats ...

Occupied ○

Available ○

All ●

Selection Colors □ Icons □

- The **INITIALIZATION** event is processed exactly once in an executable program. You can supply default values to the selection screen fields of the logical database during this event. You can use F1 Help (Technical help) to determine the names of the selection fields.
- You can use the addition **DEFAULT <value>** to supply additional report-specific default values to selection screen fields in a selection-option statement. The value sets are entered in the internal table <seltab> during this event.
- The selection screen can generally be initialized during event **AT SELECTION-SCREEN OUTPUT**. This event corresponds to event Process Before Output (PBO) of the selection screen, and therefore may be passed several times. A typical task for the selection screen's PBO event is dynamic screen modification (**LOOP AT SCREEN**), that is, showing or hiding fields, enabling or preventing input, and so on.

- You can perform an error dialog check of the selection screen fields within the **AT SELECTION-SCREEN** processing block. The event belongs to the PAI (Process After Input) processing of the selection screen. In case of errors (**MESSAGE Exxx** or **MESSAGE Wxxx**), all fields are made ready for input again.
- You can refer to individual selections with the parameters **ON <f>** or **ON <seltab>**. In case of errors, only these selections are made input-ready again.
- To check the entry combinations of a logical group, you can use the event **AT SELECTION-SCREEN ON BLOCK <block>**. Fields in this block are made ready for input when an error message is issued.
- The event **AT SELECTION-SCREEN ON END OF <field>** belongs to the PAI processing of the selection screen for **Multiple selections.**
- You can perform entry checks for selection criteria of the logical database and for your own program-specific selections.

- You can work with several selection screens in one program. The default selection screen always has the screen number 1000.
- You can also define a selection screen with **SELECTION-SCREEN BEGIN OF SCREEN <nnnn> ... END OF SCREEN <nnnn**>. Between the **BEGIN ... END ...** statements, you declare the required selections with **SELECT-OPTIONS** and **PARAMETERS**. The selection screen is assigned the screen number <nnnn> and is called with **CALL SELECTION-SCREEN <nnnn>**.
- The system takes care of the return from the selection screen, which means you do not have to program it yourself with **LEAVE SCREEN** (as is the case with **CALL SCREEN**). The program is continued immediately after the call. However, you must use system field sy-subrc to query whether the user chose ***Execute*** (F8) or ***Cancel*** (green and yellow arrows, red X). *Execute* (F8) returns sy-subrc = 0; *Cancel* returns sy-subrc = 4.
- You can supply the selection screen with default values at **INITIALIZATION.**
- You can determine which selection screen is currently processing with the `AT SELECTION-SCREEN` event. You can do so with a **CASE** control structure and evaluate the system field sy-dynnr.

- You can create any number of selection sets (variants) for a program. The variants are allocated to the program uniquely.
- Creating variants makes sense when you frequently start a program with the same selection default values.
- You can mark ***Start with variants*** in the program attributes. Users (system, services, reporting) can then start the program only with a variant.
- If the program uses several selection screens, you can choose to create a variant for all the selection screens or individually for each selection screen.
- Naming conventions and transporting variants
  - "SAP&xxx" are supplied by SAP
  - "CUS&xxx" are created by customers (in client 000)

    Variants that follow these naming conventions are client-independent and will automatically be transported along with the report. If these naming conventions are not followed, an entry for a request (task) must be added to the object list: LIMU VARI <variant_name>.

- You have to assign a name and a description to each variant. By default, variants are available for both online and background processing. You can also define a variant exclusively for use with background processing.

- You can ***protect*** the ***variant*** itself and the individual selection criteria and parameters against unauthorized changes. If you select ***Display only in catalog***, this variant will not be displayed in the general value help (F4).

- The ***type*** of a selection is determined in its declaration: Type s for **SELECT-OPTIONS**, type p for **PARAMETERS**. If you select ***Selections protected***, then the field(s) will not be ready for input. You can use the ***hide*** attribute to suppress selection criteria and parameters on the screen, if required, resulting in a less cluttered selection screen.

- When you use ***selection variables***, there are three basic ways of supplying your selections with values at runtime:

  - From table TVARV (type T)

  - Date fields using dynamic date calculation (type D), such as today's date

  - User-specific variables (type B); Prerequisite: The selection must be declared with the **MEMORY ID <pid>** addition.

Unit: Selection Screen

Topic: Designing, initializing (optional) and checking (optional) a selection screen

When you have completed these exercises, you will be able to:

- Use the SELECT-OPTIONS statement
- Use the PARAMETERS statement
- Design a selection screen
- Initialize and check (optional)

1-1 Copy or enhance your program Z##FOL1_..., or copy the sample solution, SAPBC405_FOLS_1, to program Z##SSC1_... . Sample solution for exercise: SAPBC405_SSCS_1.

Extend the selection screen with selections (SELECT-OPTIONS) for the connection number and the flight date, as well as parameters for output control.

1-1-1 Extend the selection screen with one selection each for the connection number and the flight date.

1-1-2 Suppress the multiple selection option for the flight date.

1-1-3 Group the selections for the codes of the airline and the connection number to a block. Create a frame with a title around the block.

Group the flight date into a block. Create a frame with a title around the block.

1-1-4 Maintain the selection texts.

1-2 Implement a group of parameters for output control

1-2-1 Generate a set of radio buttons with three possible settings

- **all** flights are read
- **domestic** flights only are read
- **international** flights only are read

The standard setting is that international flights are read.

1-2-2 Allow the user to enter a country code for domestic flights in an additional parameter. To do this, use field wa_flights-countryfr.

1-2-3 Create a frame without a title around the radio button set. Create a frame around the complete display parameters and assign a title and selection texts. Arrange the frames and texts as shown in the template.

1-3 Make sure that only the data records requested are read from the database.

1-3-1 To do this, supplement the WHERE clause of the SELECT statement with the conditions that result from the selections

1-3-2 Implement the logic for the radio button group.

Note: You need three separate SELECT statements with different WHERE conditions. You can map the national/international condition directly on the database:countryto = dv_flights~countryfr or countryto <> dv_flights~countryfr. You use the tilde (~) to address the database field.

2-1 OPTIONAL

2-1-1 Initialize the selection table for the airline name such that the flights of airlines AA through QF will be displayed, but not AZ.

2-1-2 Output error message 003 in message class BC405 if the user has selected "domestic flights " and the input parameter for the country is initial. In case of error, only the radio button group and the country parameter should be ready for input.

Unit: Selection Screen

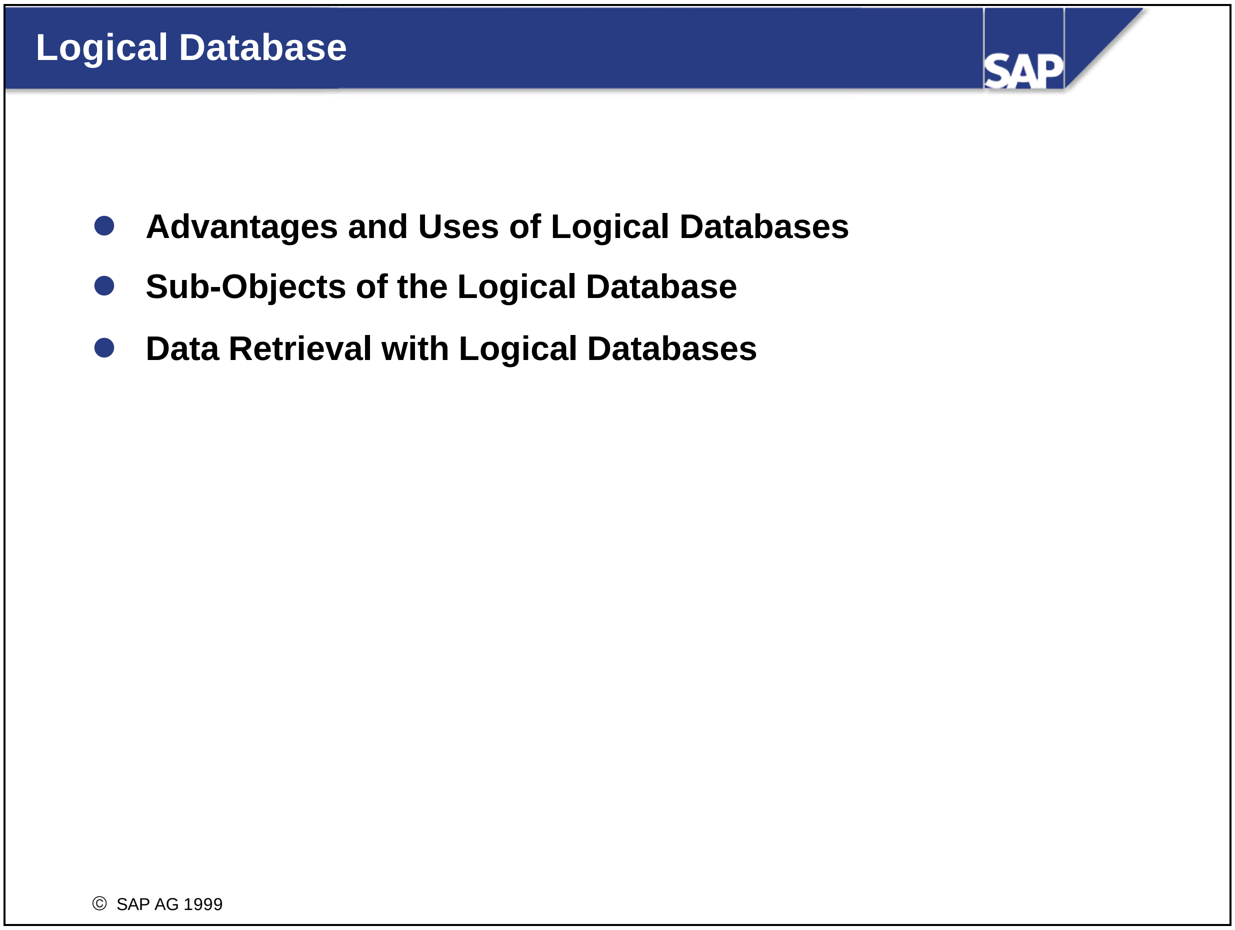
Topic: Designing, initializing (optional) and checking (optional) a selection screen

```
*&---------------------------------------------------------------------*
*& Report  SAPBC405_SSCS_1                                             *
*&                                                                     *
*&---------------------------------------------------------------------*
*&    Solution: Exercise 1; Selection Screen                           *
*&                                                                     *
*&---------------------------------------------------------------------*

INCLUDE bc405_sscs_1top.

*&---------------------------------------------------------------------*
*&   Event TOP-OF-PAGE
*&---------------------------------------------------------------------*
TOP-OF-PAGE.

* Title
  FORMAT COLOR COL_HEADING INTENSIFIED ON.
  ULINE.
  WRITE: / sy-vline,
         'Flight data'(001),
          AT line_size sy-vline.
  ULINE.

* Column header
  FORMAT COLOR COL_HEADING INTENSIFIED OFF.
  WRITE:  sy-vline, AT pos_c1 'Flight'(002).

* Fix left scroll boundary
  SET LEFT SCROLL-BOUNDARY.
  WRITE: 'Date'(003) ,
         'Departure location'(004),
         'Arrival location'(005),
         'Price'(006),
         AT line_size sy-vline.
  ULINE.

*********************************************************************
Optional parts: Initializing and checking a selection screen
*********************************************************************

*&-------------------------------------------------------------------*
*&   Event   INITIALIZATION
*&-------------------------------------------------------------------*
INITIALIZATION.                    " OPTIONAL

* Initialize select-options for CARRID
  MOVE:       'AA' TO so_car-low,
          'QF' TO so_car-high,
          'BT' TO so_car-option,
          'I' TO so_car-sign.
          APPEND so_car.
  CLEAR so_car.
  MOVE:       'AZ' TO so_car-low
          'EQ' TO so_car-option,
          'E' TO so_car-sign.
          APPEND so_car.
  CLEAR so_car.

*&-------------------------------------------------------------------*
*&   Event AT SELECTION-SCREEN ON BLOCK PARAM
*&-------------------------------------------------------------------*
AT SELECTION-SCREEN ON BLOCK param.    " OPTIONAL

* check country for national flights is not empty
  CHECK national = 'X' AND country = space.
  MESSAGE e003(bc405).

*&-------------------------------------------------------------------*
*&   Event START-OF-SELECTION
```

```
*&-----------------------------------------------------------------*
START-OF-SELECTION.

* Checking the output parameters
  CASE mark.
    WHEN all.
* Radiobutton ALL is marked
      SELECT * FROM dv_flights INTO TABLE it_flights
        WHERE carrid IN so_car
          AND connid IN so_con
          AND fldate IN so_fdt.

    WHEN national.
* Radiobutton NATIONAL is marked
      SELECT * FROM dv_flights INTO TABLE it_flights
        WHERE carrid IN so_car
          AND connid IN so_con
          AND fldate IN so_fdt
          AND countryto = dv_flights~countryfr
          AND countryto = country.

WHEN internat.
* Radiobutton INTERNAT is marked
      SELECT * FROM dv_flights INTO TABLE it_flights
        WHERE carrid IN so_car
          AND connid IN so_con
          AND fldate IN so_fdt
          AND countryto <> dv_flights~countryfr.

ENDCASE.

*&-----------------------------------------------------------------*
*&   Event END-OF-SELECTION
*&-----------------------------------------------------------------*
END-OF-SELECTION.

SORT it_flights BY carrid connid fldate
```

```
* Data output
  PERFORM data_output.

*&---------------------------------------------------------------------*
*&      Form  DATA_OUTPUT
*&---------------------------------------------------------------------*
*       List output of flight data
*----------------------------------------------------------------------*
FORM data_output.

* Loop at the internal table for writing data
  LOOP AT it_flights INTO wa_flights.

* Get a new page if CONNID has changed.
    ON CHANGE OF wa_flights-connid.
      NEW-PAGE.
    ENDON.

* Mark international flights
    FORMAT COLOR COL_KEY INTENSIFIED ON.
    IF wa_flights-countryfr EQ wa_flights-countryto.
      WRITE: / sy-vline, icon_space AS ICON CENTERED.
    ELSE.
      WRITE: / sy-vline, icon_bw_gis AS ICON CENTERED.
    ENDIF.

* Data output
    WRITE: wa_flights-carrid,
           wa_flights-connid.
    FORMAT COLOR COL_NORMAL INTENSIFIED OFF.
    WRITE: wa_flights-fldate,
           wa_flights-cityfrom,
           wa_flights-cityto,
           wa_flights-price CURRENCY wa_flights-currency,
           wa_flights-currency,
           AT line_size sy-vline.
  ENDLOOP.

ENDFORM.                    " DATA_OUTPUT




*&---------------------------------------------------------------------*
*& Include BC405_SSCS_1TOP                                             *
*&                                                                     *
*&---------------------------------------------------------------------*

REPORT bc405_sscs_1top LINE-SIZE 100 NO STANDARD PAGE HEADING.

* Include for using icons
INCLUDE <icon>.

* Constants for writing position
CONSTANTS       : pos_c1 TYPE i VALUE  6,
                  line_size TYPE i VALUE 100.

* Constant for CASE statement
CONSTANTS mark VALUE 'X'.

* Internal table like DDIC view DV_FLIGHTS
DATA:       it_flights LIKE TABLE OF dv_flights,
            wa_flights LIKE dv_flights.

* Selections for connections
SELECTION-SCREEN BEGIN OF BLOCK conn
                        WITH FRAME TITLE text-tl1.
SELECT-OPTIONS:         so_car FOR wa_flights-carrid,
                        so_con FOR wa_flights-connid.
SELECTION-SCREEN END OF BLOCK conn.

* Selections for flights
SELECTION-SCREEN BEGIN OF BLOCK flight
```

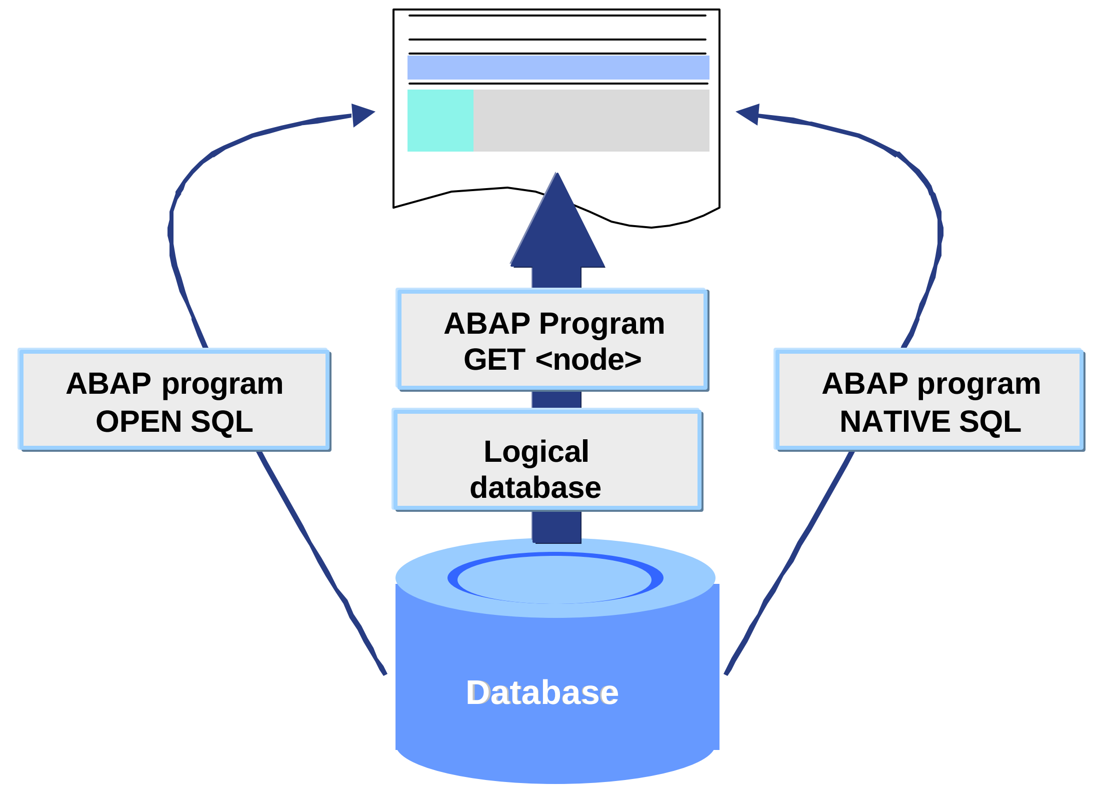
```
                              WITH FRAME TITLE text-tl2.
SELECT-OPTIONS                so_fdt FOR wa_flights-fldate NO-EXTENSION.
SELECTION-SCREEN END OF BLOCK flight.

* Output parameter
SELECTION-SCREEN BEGIN OF BLOCK param
                              WITH FRAME TITLE text-tl3.
SELECTION-SCREEN BEGIN OF BLOCK radio WITH FRAME.
PARAMETERS:      all RADIOBUTTON GROUP rbg1,
                                      national RADIOBUTTON GROUP rbg1,
                                      internat RADIOBUTTON GROUP rbg1 DEFAULT
'X'.
SELECTION-SCREEN END OF BLOCK radio.

PARAMETERS country LIKE wa_flights-countryfr.
SELECTION-SCREEN END OF BLOCK param.
```



Logical Database

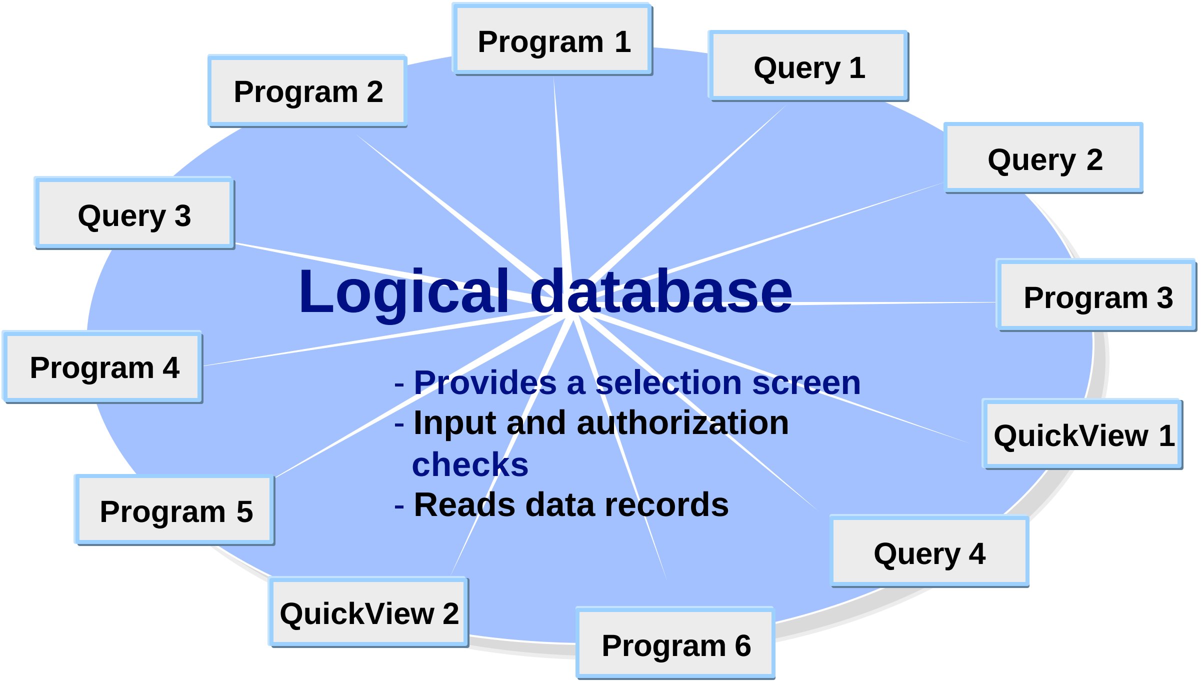
- Advantages and Uses of Logical Databases
- Sub-Objects of the Logical Database
- Data Retrieval with Logical Databases

Generating Lists



- In general, the system reads data that will appear in a list from the database.
- You can use OPEN SQL or NATIVE SQL statements to read data from the database.
- The use of a logical database provides you with an alternative to having to program database accesses individually. Logical databases retrieve data records and make them available to ABAP programs.

Advantages of a Logical Database



Program 1

Query 1

Program 2

Query 2

Query 3

Logical database

Program 3

Program 4

- Provides a selection screen
- Input and authorization checks
- Reads data records

QuickView 1

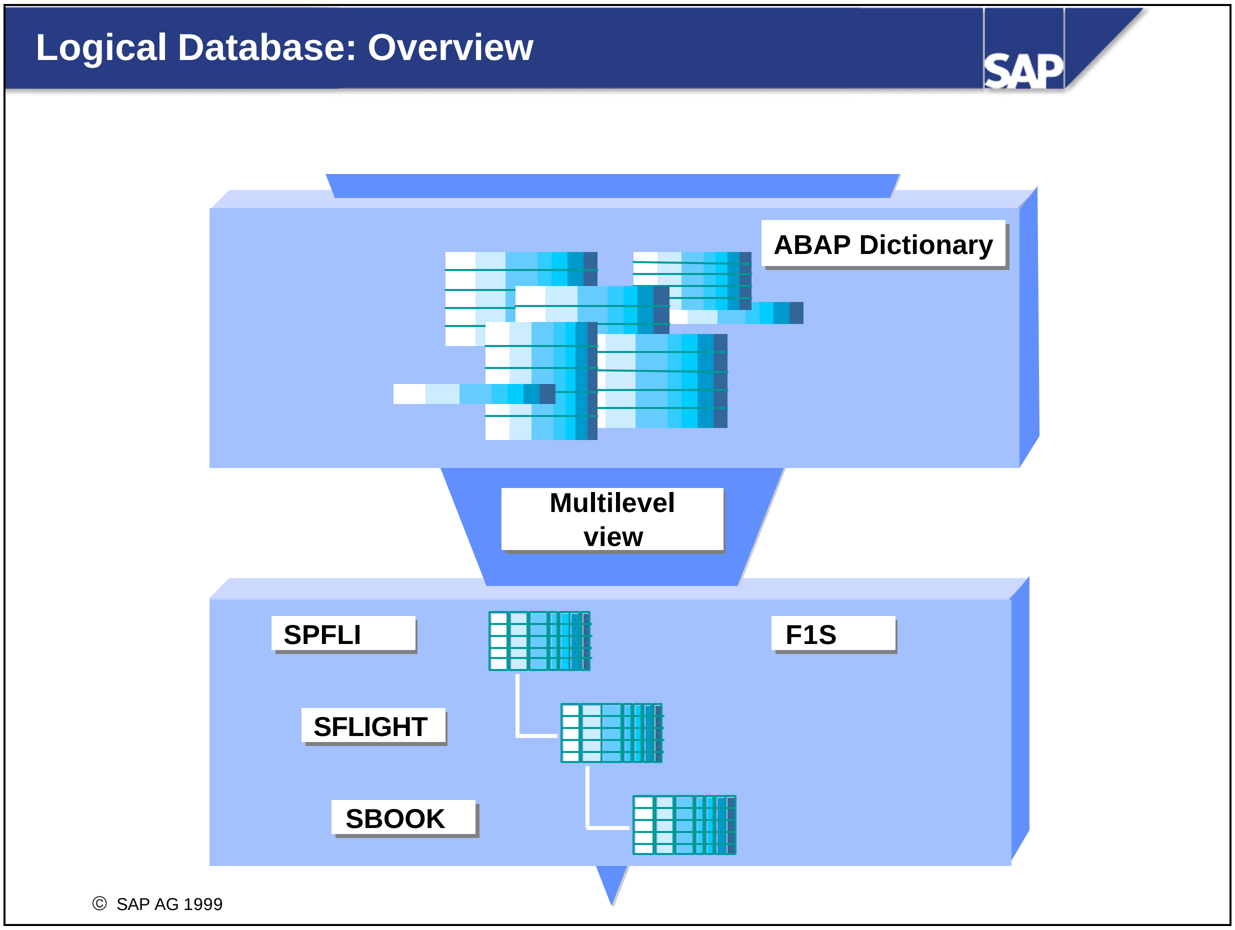
Program 5

Query 4

QuickView 2

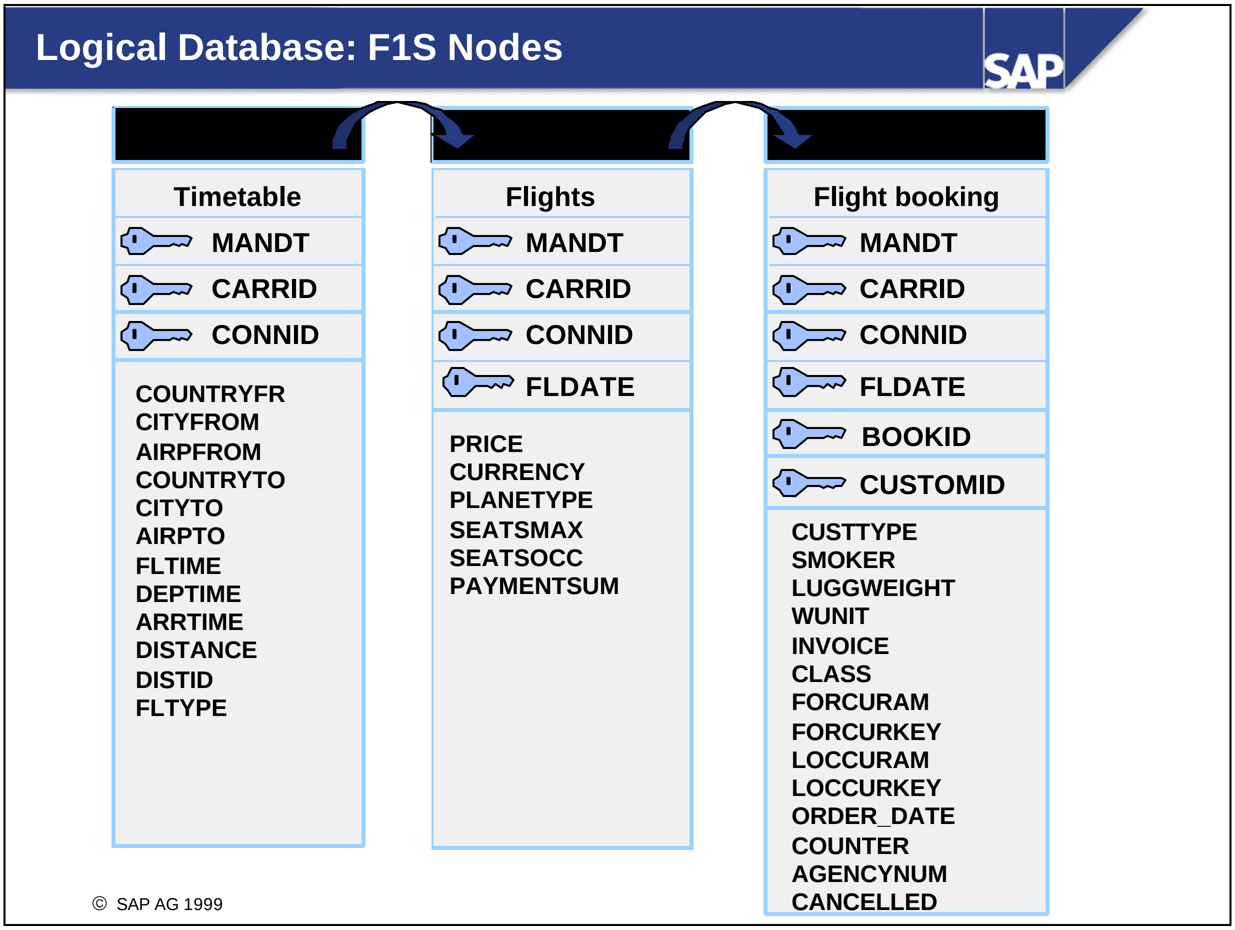
Program 6

- The same logical database can be the data source for several QuickViews, queries, and programs. In the QuickView, the LDB can be specified directly as a data source. A query works with the logical database when the functional area that generated the query is defined with a logical database. In the case of type 1 programs, the LDB is entered in the attributes or called using function module LDB_PROCESS. See appendix for information on how to use the function module.
- Logical databases offer several advantages:
  - The system generates a selection screen. The use of selection screen versions or variants provides the required flexibility.
  - The user does not have to know the exact structure of the tables involved (especially the foreign key dependencies); the data is made available in the correct order at GET events.
  - Performance improvements within logical databases directly affect all programs linked to the logical database, without having to change the programs themselves.
  - Maintenance can be performed at a central location.
  - Authorization checks can also be performed centrally.



Logical Database: Overview

- A logical database is an ABAP program that reads predefined data from the database and makes it available to other programs.
- A hierarchical structure determines the order in which the data is supplied to the programs. A logical database also provides a selection screen that checks user entries and conducts error dialogs. These can be extended in programs.
- SAP provides some 200 logical databases in Release 4.6. The names of logical databases have been extended to 20 places in Release 4.0 (namespace prefix max. 10 characters).



Logical Database: F1S Nodes

Timetable
MANDT
CARRID
CONNID
COUNTRYFR
CITYFROM
AIRPFROM
COUNTRYTO
CITYTO
AIRPTO
FLTIME
DEPTIME
ARRTIME
DISTANCE
DISTID
FLTYPE

Flights
MANDT
CARRID
CONNID
FLDATE
PRICE
CURRENCY
PLANETYPE
SEATSMAX
SEATSOCC
PAYMENTSUM

Flight booking
MANDT
CARRID
CONNID
FLDATE
BOOKID
CUSTOMID
CUSTTYPE
SMOKER
LUGGWEIGHT
WUNIT
INVOICE
CLASS
FORCURAM
FORCURKEY
LOCCURAM
LOCCURKEY
ORDER_DATE
COUNTER
AGENCYNUM
CANCELLED

- The demo programs and exercises for SAP courses and ABAP documentation refer to SAP's BC_TRAVEL flight data model, which is found in development class **BC_DATAMODEL**.
- The tables

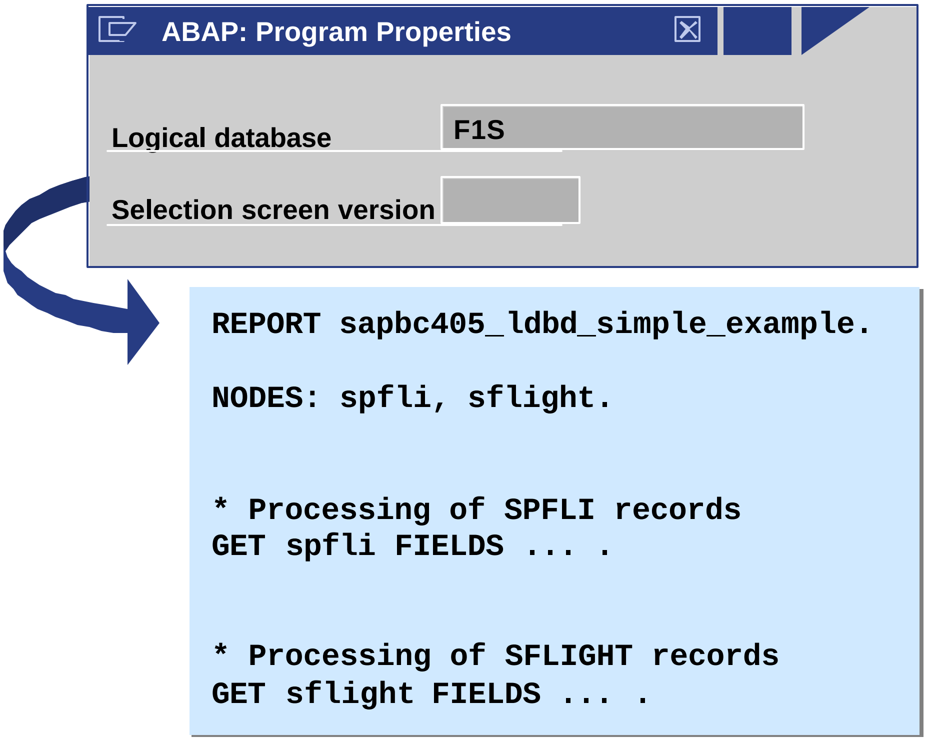
  SPFLI: Flight connections

  SFLIGHT: Flights

  SBOOK: Bookings

  form the nodes of logical database F1S.

Sample Program for a Logical Database



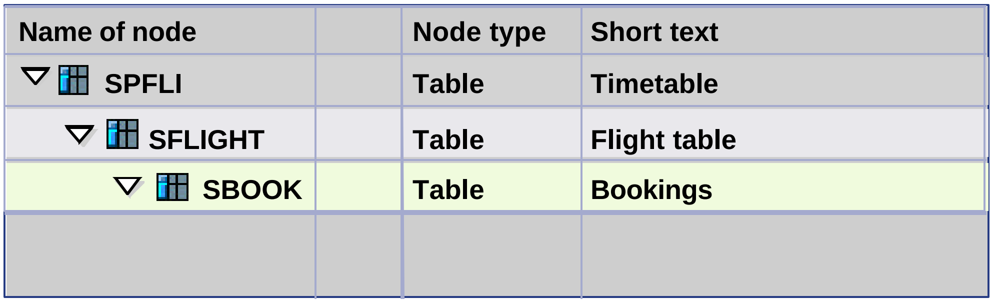
Logical database: F1S

Selection screen version:

```
REPORT sapbc405_ldbd_simple_example.

NODES: spfli, sflight.


* Processing of SPFLI records
GET spfli FIELDS ... .


* Processing of SFLIGHT records
GET sflight FIELDS ... .
```

- In the case of executable programs, you can enter a logical database in the attributes.
- Use the **NODES <node>** statement to specify the nodes of the logical database that you want to use in the program. **NODES** allocates the appropriate storage space for the node - that is, a work area or a table area depending on the node type.
- The logical database makes the data records available for the corresponding **GET** events. The sequence in which these events are processed is determined by the structure of the logical database.

LDB Sub-Objects: Structure



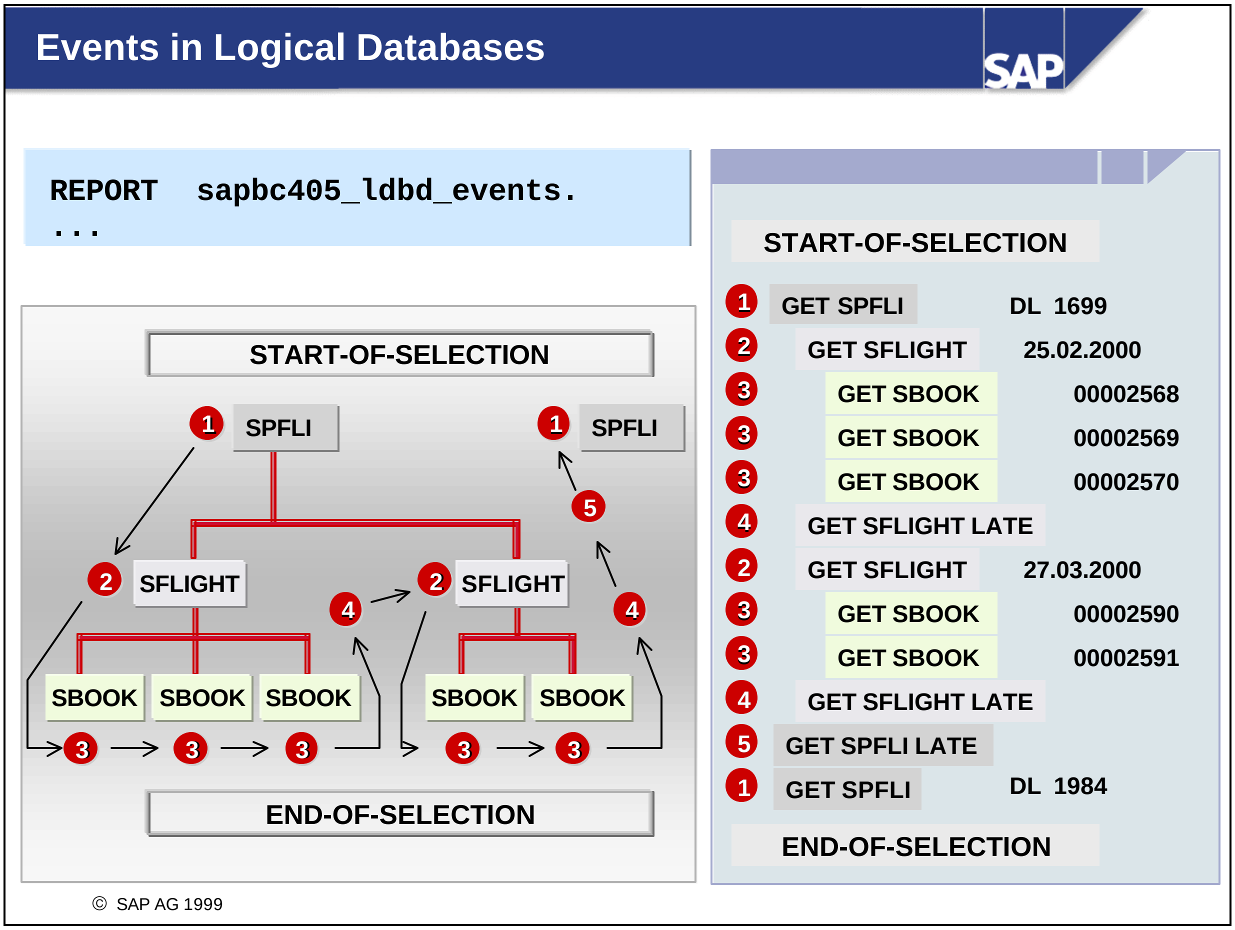
Name of node		Node type	Short text
▽ ▦ SPFLI		Table	Timetable
▽ ▦ SFLIGHT		Table	Flight table
▽ ▦ SBOOK		Table	Bookings

Establishes a data hierarchy (read sequence)

Possible node types:

Database table	Table or structure from the DDIC Name must be identical to the node name
DDIC type	DDIC type: Table or structure. Name can differ from node name; deep structures are possible.
Data type	Type that was defined in a type group
Dynamic type	Type is specified in program

- Logical databases are made up of several sub-objects. The structure determines the hierarchy, and thus the read sequence of the data records.
- Node names can contain up to 14 characters. There are four different node types.
  - Table (type T): The node name is the name of a transparent table (this type corresponds to the concept prior to Release 4.0A). The table name must be identical to the node name. Deep types (complex) are not allowed.
  - DDIC type (type S): Any node name is possible. It is assigned a structure or a table type from the Dictionary. The node name can differ from the type name. Deep structures are possible.
  - Type groups (type C): The node type is defined in a type group. The name of the type group must be maintained in the "Type group" field. You should generally prefer DDIC types, as the other applications that use the logical database (such as SAP Query) can access them (short texts, and so on).
  - Dynamic nodes (type A): These nodes do not have a fixed type; they are not classified until the program runtime. Which types are generally allowed is determined when the structure is created.
- Nodes are declared using language element NODES.



Events in Logical Databases

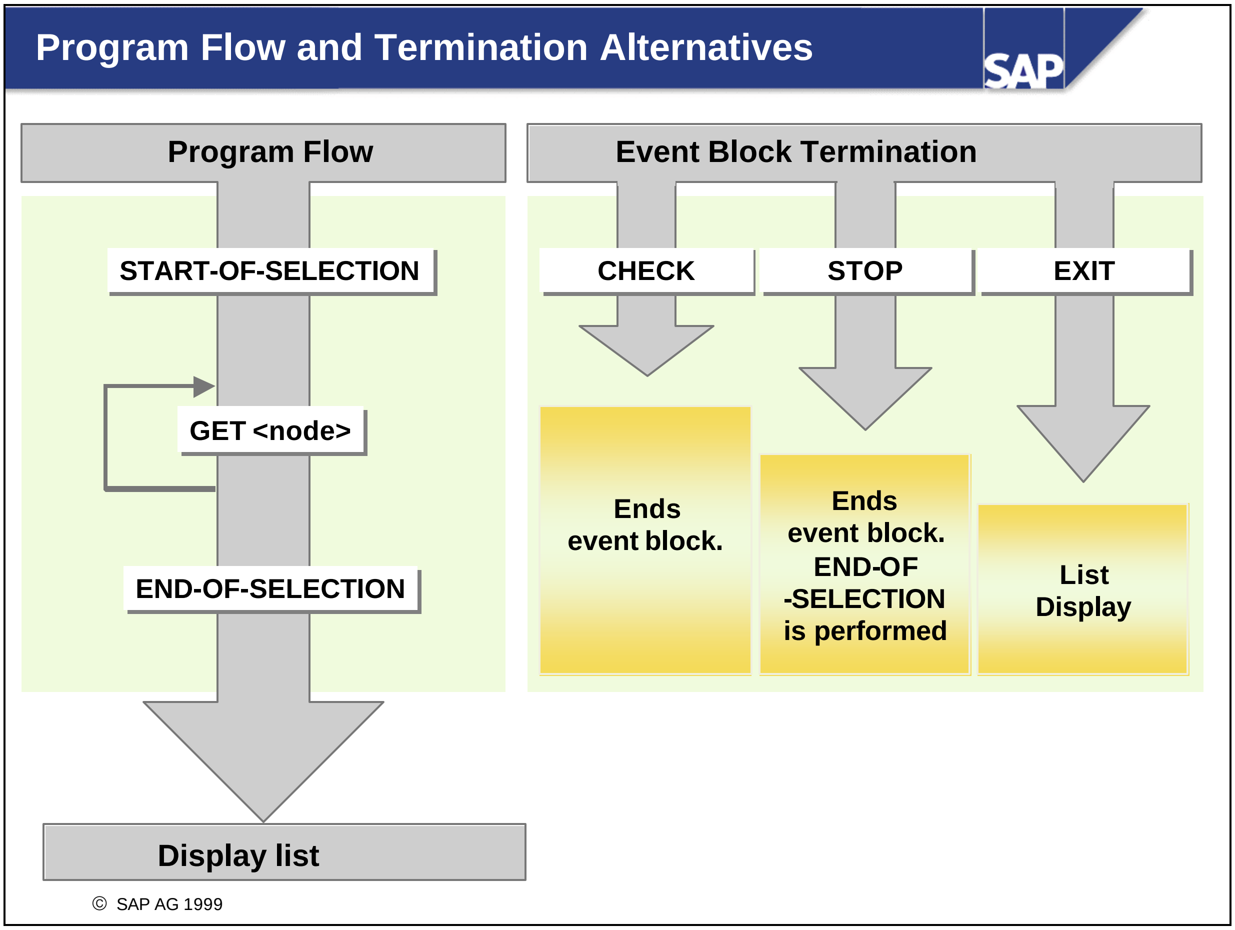
```
REPORT  sapbc405_ldbd_events.
...
```

START-OF-SELECTION

1	GET SPFLI	DL 1699
2	GET SFLIGHT	25.02.2000
3	GET SBOOK	00002568
3	GET SBOOK	00002569
3	GET SBOOK	00002570
4	GET SFLIGHT LATE	
2	GET SFLIGHT	27.03.2000
3	GET SBOOK	00002590
3	GET SBOOK	00002591
4	GET SFLIGHT LATE	
5	GET SPFLI LATE	
1	GET SPFLI	DL 1984

END-OF-SELECTION

- Processing blocks are always allocated to an event. A processing block is closed by the next event key word, the start of form routines, or by the end of the program.
- The **START-OF-SELECTION** event is triggered before control is given to the read routine of the logical database. The **END-OF-SELECTION** event is triggered after all **GET** events have been processed - that is, all data records have been read and processed.
- The **GET <node>** event is triggered whenever the logical database supplies data for this node. This means that **GET** events are processed several times, and that data has already been read from the database for these events. The sequence in which the **GET** events are processed is determined by the structure of the logical database.
- The **GET <node> LATE** event is triggered when all subordinate nodes of node <node> have been processed, before the data is read for the next <node>; that is, whenever a hierarchy level has been completed.
- At the start of the event, the system automatically adds a line feed and configures the default formats (for example, **INTENSIFIED ON**).



Program Flow and Termination Alternatives

- **CHECK** statements end the current processing block.
- **STOP** statements end program processing. However, in contrast to the **EXIT** statement, the processing block **END-OF-SELECTION** is processed first (if it exists).
- If there is a **STOP** statement within the **END-OF-SELECTION** processing block, program processing ends immediately and a list is displayed.
- The **EXIT** statement exits the program and displays the list.
- You can also use the **REJECT** statement. The data record is not processed further. Processing continues on the same hierarchy level when the next data record is read. **REJECT**, unlike the **CHECK** statement, can also be used within a subroutine.

- Use the selection include db<name>sel to define selection screens for logical databases. The addition **FOR NODE** assigns selections to individual logical nodes. The appearance of a selection screen thus directly depends on the **NODES** statement contained within your program.
- A **field selection** can be defined for the individual nodes. To do this, you have to specify the addition **FIELD SELECTION FOR NODE** in the **SELECTION-SCREEN** statement. You can then use **GET <node> FIELDS <field list**> to restrict the amount of data returned.
- You can designate individual nodes for **dynamic selection** using the addition **DYNAMIC SELECTIONS FOR NODE**. The *Dynamic selection* pushbutton then appears on your selection screen. You can determine which selection fields can be set by choosing a particular selection view yourself (type: CUS) or by using the selection view delivered by SAP (type: SAP).
- With large logical databases you can define several selection screen versions. Each selection screen version contains a subset of your selection criteria (language element: **EXCLUDE**). Specify the name of a selection screen version in the program attributes.

- When you enter a logical database in the attributes of your type 1 program, the system processes the selection screen of the logical database. The concrete characteristics of the selection screen depend upon the node specified in the **NODES** statement. If you specify a node of type T (table), you can also declare the table work area with the TABLES statement.
- If you address only subordinate nodes (in the hierarchy) of the logical database in the program (for example sflight), the selection screen criteria for the superior node in the hierarchy (spfli) also appear. You can thus restrict the dataset to be read so that it meets your specific requirements.
- Note: A logical database **always reads in accordance with its structure**. This means that if you only need data from a node deep in the hierarchy, you will achieve better performance by programming the access yourself. This avoids unnecessary reading of the database.

- If the logical database supports dynamic selections, the pushbutton for ***Dynamic selections*** appears on the selection screen. When the user presses this button, a second selection screen is displayed. This screen allows the user to select additional database fields. The system transfers the selections directly to the logical database program and therefore to the database (dynamic selections).
- The selection view determines which fields are displayed on the selection screen. Create your own view with type CUS, and have it override the view with type SAP.

- Database program sapdb<ldbname> for logical database <ldbname> is a collection of subroutines, each of which is performed for specific events. For example, subroutine <init> is processed once at the start of the database program. This program can be used to define default values for the selection screen of the LDB.
- Other subroutines also exist that are processed during events PBO (**P**rocess **B**efore **O**utput) and PAI (**P**rocess **A**fter **I**nput) of the selection screen. Checks, such as authorization checks (**AUTHORITY-CHECK**), are usually performed during event PAI.
- The database accesses (**SELECT** statements) are programmed in the put_<node> subroutines. These subroutines may be processed several times, depending on which selection criteria the user specifies. The sequence in which these subroutines are processed is determined by the structure of the logical database.
- Database access (**SELECT** statements) should be programmed with optimal performance in mind. When creating a logical database you generate the corresponding database program after first having determined its structure and selection attributes. You can find performance tips in the comment lines.

- When a program that has been assigned a logical database is started, control is initially passed to the database program of the logical database. Each event has a corresponding subroutine in the database program - for example, subroutine **init** for event **INITIALIZATION**. During the interaction between the LDB and the associated program, the subroutine is always processed first, followed by the event (if there is one in the report).
- Logical database programs read data from a database according to the structure declared for the logical database. They begin with the root node and then process the individual "branches" consecutively from top to bottom.
- The logical database reads the data in the put_<node> subroutines. During event **PUT**, control is passed from the database program to the **GET** event of the associated report. The data is made available in the corresponding work areas in the report. The processing block defined for the **GET** event is performed. Control then returns to the logical database. **PUT** activates the next form subroutine found in the structure. This flow is continued until the report has collected all the available data.
- The **depth** of data read in the structure depends upon a program's **GET** events. A logical database reads to the lowest **GET** event contained within the structure attributes. Only those **GET** events for which processing is supposed to take place are written into the report program. Logical databases read all data records found on the direct access path.

- If you specify a logical database and declare additional selections in the program attributes that refer to the fields of a node **not** designated for dynamic selection, you must use the **CHECK <seltab>** statement to see if the current data record fulfills the selection criteria.
- If the data record does not fulfill these selection criteria, current event block processing ends.

Unit: Logical database

Topic: GET Events

When you have completed these exercises, you will be able to:

- Create a list whose data is read from a logical database

1-1 Create program Z##LDB1_... with TOP include (Z##LDB1_...TOP) and enter logical database F1S in the program attributes. Make sure you specify ***Executable program*** as the program type. Sample solution for exercise: SAPBC405_LDBS_1.

1-1-1 The logical database should supply the program with data for nodes SPFLI, SFLIGHT, and SBOOK.

1-2-2 Create a list that displays the following data:

Table SPFLI: CARRID, CONNID, CITYFROM,
AIRPFROM, CITYTO, AIRPTO.

Table SFLIGHT: FLDATE, SPRICE, CURRENCY,
PLANETYPE, SEATSMAX,
SEATSOCC,
FREE_SEATS.

Table SBOOK: BOOKID, CUSTOMID, SMOKER,
LUGGWEIGHT, WUNIT.

Field FREE_SEATS is not a table field – it has to be calculated in the program. The price and luggage weight should be output with the appropriate units.

1-2-3 Formatting the list (optional)

Create a three-line list in which each line outputs the information for one node (see above).

Output the first line in color COL_HEADING not intensified, the second line in COL_NORMAL intensified, and the third line in COL_NORMAL not intensified. The list should have 83 columns and have a frame. Maintain the column headers (standard list header).

Note: You will have to program the events GET spfli, GET sflight, GET sbook, and END-OF-SELECTION. To output the fields, use the pattern functions available in the ABAP Editor.

Exercises

Unit: Logical database

Topic: GET LATE Events and Checks from Internal Program Selections

When you have completed these exercises, you will be able to:

- Create a list whose data is read from a logical database
- Check internal selections for their validity

1-1 Copy program Z##LDB1_... or the sample solution, SAPBC405_LDBS_1, from exercise 1 to program Z##LDB2_... . Sample solution for exercise: SAPBC405_LDBS_2.

1-1-1 Add a SELECT-OPTIONS statement for the posting date (table SBOOK) to the selection screen. Frame the selection and maintain the selection text.

1-2-2 Make sure that only bookings that meet the specified selection criteria are output in the list. Include the booking date in the list output. Maintain the column header (standard list header).

1-2-3 In the list, output a solid line when all the bookings for a date have been output, and when a flight has been completely output. Output each flight on a new page.

Optional: Sample solution for exercise: SAPBC405_LDBS_2_OPT.

2-1 Enhance your program.

2-1-1 Add a radio button group with two radio buttons to the selection screen. Draw a frame around the group and maintain the selection texts.

The group has to model the following functionality: The user can select between charter flights only and regular flights only in the list. Whether a flight is a charter or not is determined in field SPFLI-FLTYPE: Charter flight: SPFLI –FLTYPE = 'X'.

2-2-2 Make sure that only dates and bookings that meet the specified selection criteria are output in the list. You will need an auxiliary variable to evaluate the radio button group.

Unit: Logical database

Topic: GET Events

```
*&---------------------------------------------------------------------*
*& Report  SAPBC405_LDBS_2                                             *
*&                                                                *
*&---------------------------------------------------------------------*
*&                                                       *
*&                                                       *
*&---------------------------------------------------------------------*

INCLUDE bc405_ldbs_2top.

*&---------------------------------------------------------------------*
*&   Event GET SPFLI
*&---------------------------------------------------------------------*
GET spfli.
* Data output SPFLI
  FORMAT COLOR COL_HEADING INTENSIFIED OFF.
  WRITE:  / sy-vline, spfli-carrid,
            spfli-connid,
            spfli-cityfrom,
            spfli-airpfrom,
            spfli-cityto,
            spfli-airpto,
            AT line_size sy-vline.


*&---------------------------------------------------------------------*
*&   Event GET SFLIGHT
*&---------------------------------------------------------------------*
GET sflight.
* Calculate free seats
* Data output SFLIGHT
  FORMAT COLOR COL_NORMAL INTENSIFIED ON.
  WRITE: / sy-vline, sflight-fldate,
           sflight-price CURRENCY sflight-currency,
           sflight-currency,
           sflight-planetype,
           sflight-seatsmax,
           sflight-seatsocc,
           free_seats,
           AT line_size sy-vline.

*&---------------------------------------------------------------------*
*&   Event GET SBOOK
*&---------------------------------------------------------------------*
GET sbook.

* Check select-option
  CHECK so_odat.

  FORMAT COLOR COL_NORMAL INTENSIFIED OFF.
  WRITE: / sy-vline, sbook-bookid,
           sbook-customid,
           sbook-smoker,
           sbook-luggweight UNIT sbook-wunit,
           sbook-wunit,
           sbook-order_date,
           AT line_size sy-vline.

*&---------------------------------------------------------------------*
*&   Event GET SPFLI LATE
*&---------------------------------------------------------------------*
GET spfli LATE.
  ULINE.
  NEW-PAGE.
*&---------------------------------------------------------------------*
*&   Event GET SFLIGHT LATE
```

```
*&---------------------------------------------------------------------*
GET sflight LATE.
  ULINE.



*&---------------------------------------------------------------------*
*& Include BC405_LDBS_2TOP                                              *
*&                                                                     *
*&---------------------------------------------------------------------*

REPORT   sapbc405_ldbs_2 LINE-SIZE 83.

* Used nodes of the structure of the logical database F1S
NODES: spfli, sflight, sbook.

* Additional selections
SELECTION-SCREEN BEGIN OF BLOCK order WITH FRAME.
SELECT-OPTIONS: so_odat FOR sbook-order_date.
SELECTION-SCREEN END OF BLOCK order.

* Variables
DATA: free_seats LIKE sflight-seatsocc.

* Constants
CONSTANTS: line_size LIKE sy-linsz VALUE 83.
```

Solutions

Unit: Logical database

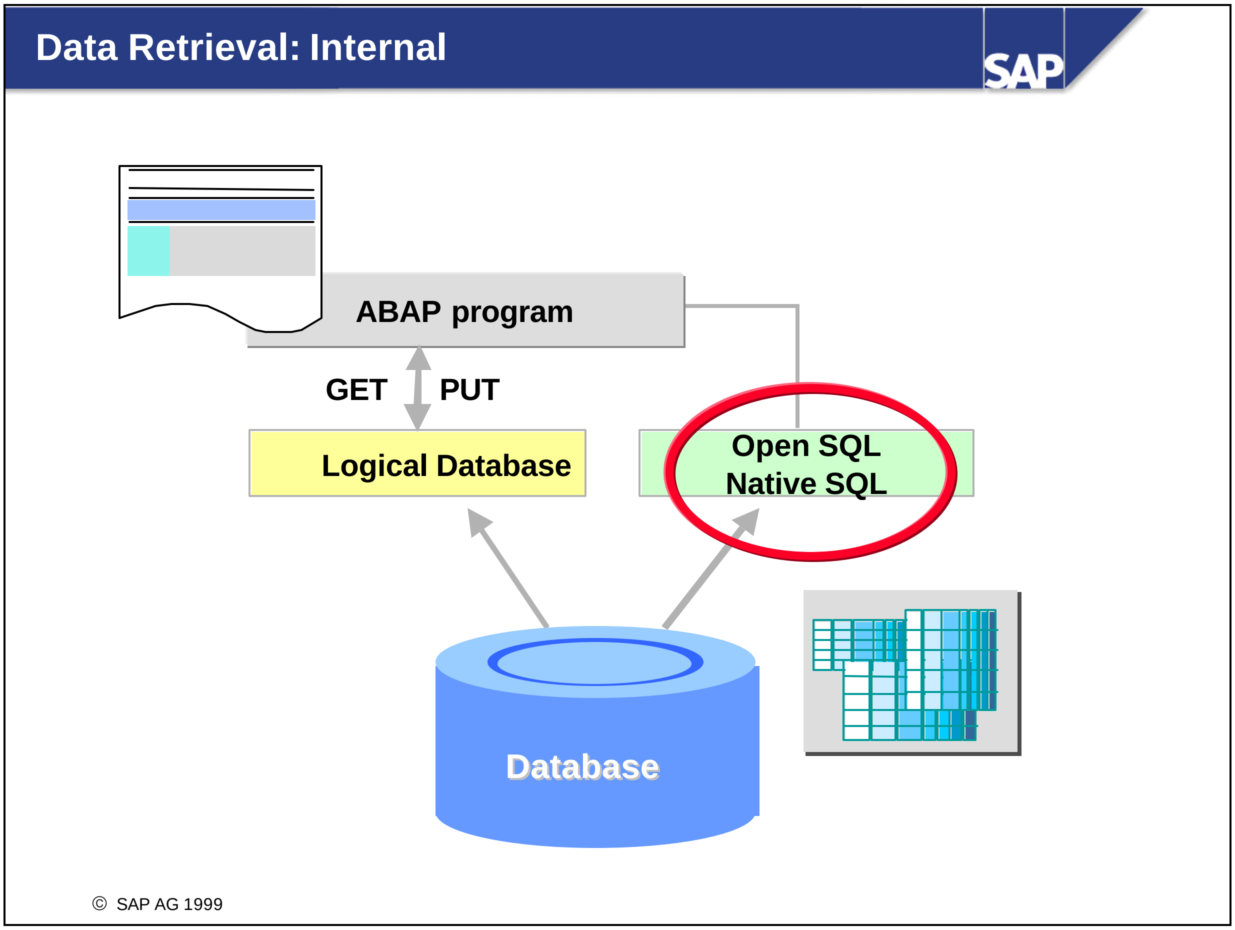
Topic: GET LATE Events and Checks from Internal Program Selections

```
*&-------------------------------------------------------------------*
*& Report  SAPBC405_LDBS_2_OPT                                *
*&                                                          *
*&-------------------------------------------------------------------*
*&                                                          *
*&                                                          *
*&-------------------------------------------------------------------*

INCLUDE BC405_LDBS_2_OPTTOP.

*&-------------------------------------------------------------------*
*&   Event GET SPFLI
*&-------------------------------------------------------------------*
GET spfli.

*+++++++++++++++++++++++++++++++++++++> optional
* Check radio button group using a help variable
* Flight type charter or scheduled)

  CLEAR check_negative.
  IF pa_fty1 = 'X'.
    IF NOT spfli-fltype = pa_fty1.
      check_negative = 'X'.
    ENDIF.
  ELSEIF pa_fty2 = 'X'.
    IF NOT spfli-fltype = space.
      check_negative = 'X'.
    ENDIF.
  ENDIF.

  CHECK check_negative = space.
*<---------------------------------------------------------- optional

* Data output SPFLI
  FORMAT COLOR COL_HEADING INTENSIFIED OFF.
  WRITE:  / sy-vline, spfli-carrid,
            spfli-connid,
            spfli-cityfrom,
            spfli-airpfrom,
            spfli-cityto,
            spfli-airpto,
            AT line_size sy-vline.



*&---------------------------------------------------------------------*
*&   Event GET SFLIGHT
*&---------------------------------------------------------------------*
GET sflight.
* Calculate free seats
  free_seats = sflight-seatsmax - sflight-seatsocc.

* Data output SFLIGHT
  FORMAT COLOR COL_NORMAL INTENSIFIED ON.
  WRITE: / sy-vline, sflight-fldate,
           sflight-price CURRENCY sflight-currency,
           sflight-currency,
           sflight-planetype,
           sflight-seatsmax,
           sflight-seatsocc,
           free_seats,
           AT line_size sy-vline.

*&---------------------------------------------------------------------*
*&   Event GET SBOOK
*&---------------------------------------------------------------------*
```

```
GET sbook.

* Check select-option
  CHECK so_odat.

  FORMAT COLOR COL_NORMAL INTENSIFIED OFF.
  WRITE: / sy-vline, sbook-bookid,
           sbook-customid,
           sbook-smoker,
           sbook-luggweight UNIT sbook-wunit,
           sbook-wunit,
           sbook-order_date,
           AT line_size sy-vline.

*&---------------------------------------------------------------------*
*&   Event GET SPFLI LATE
*&---------------------------------------------------------------------*
GET spfli LATE.
  ULINE.
  NEW-PAGE.
*&---------------------------------------------------------------------*
*&   Event GET SFLIGHT LATE
*&---------------------------------------------------------------------*
GET sflight LATE.
  ULINE.



*&---------------------------------------------------------------------*
*& Include BC405_LDBS_2OPTTOP                                           *
*&                                                                     *
*&---------------------------------------------------------------------*

REPORT   sapbc405_ldbs_2_opt LINE-SIZE 83.

* Used nodes of the structure of the logical database F1S
NODES: spfli, sflight, sbook.
```

```
* Additional selections
SELECTION-SCREEN BEGIN OF BLOCK order WITH FRAME.
SELECT-OPTIONS: so_odat FOR sbook-order_date.
SELECTION-SCREEN END OF BLOCK order.

* Additional selections (optional part)
SELECTION-SCREEN BEGIN OF BLOCK type WITH FRAME.
PARAMETERS: pa_fty1 RADIOBUTTON GROUP ftyp,
            pa_fty2 RADIOBUTTON GROUP ftyp.
SELECTION-SCREEN: END OF BLOCK type.

* Variables
DATA: free_seats LIKE sflight-seatsocc.
DATA  check_negative.

* Constants
CONSTANTS: line_size LIKE sy-linsz VALUE 83.
```

Programming Data Retrieval



Data Retrieval: Internal

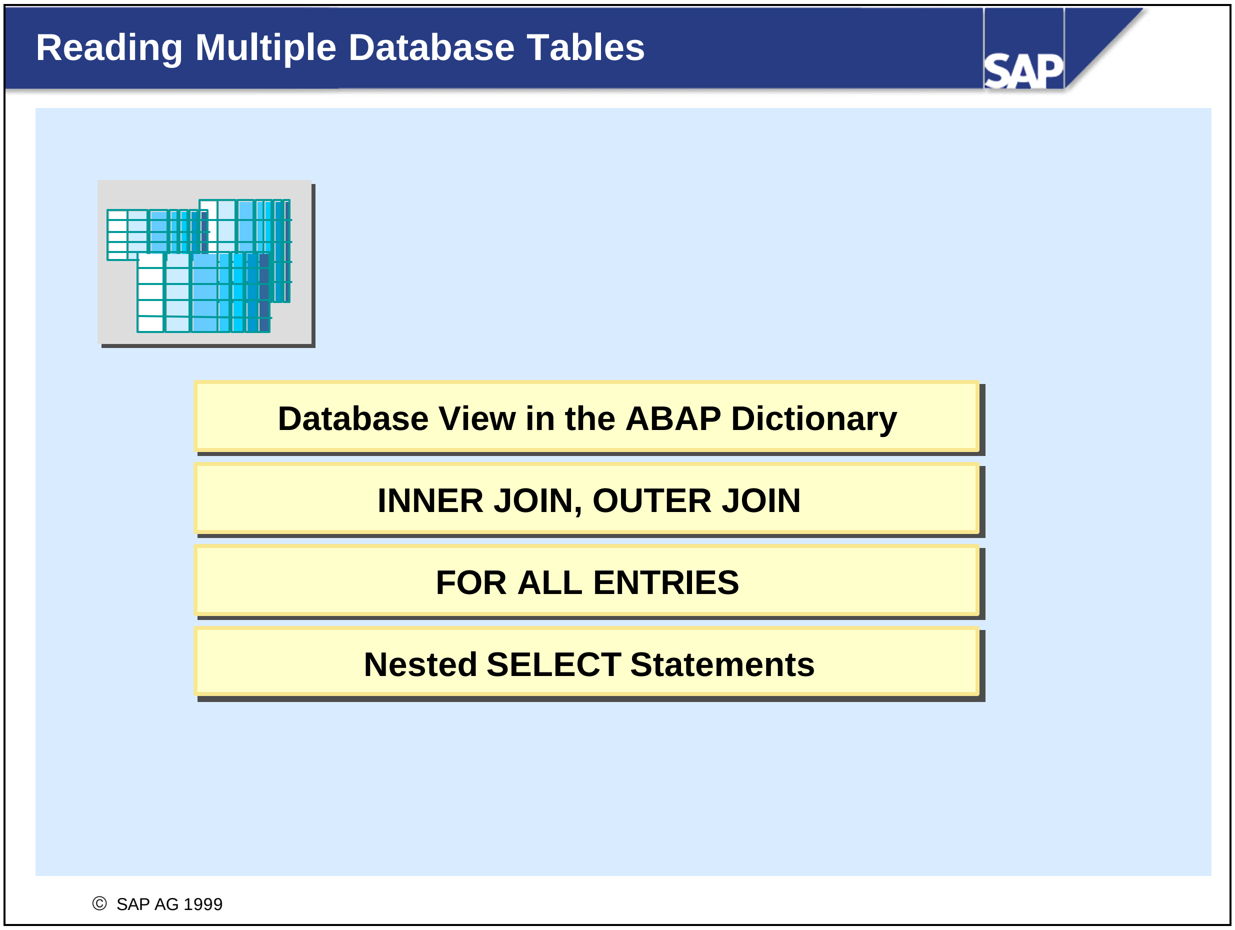
- Whenever a logical database cannot supply your program with all necessary data, you must program database access directly into the program itself. This can be done using either Open SQL or Native SQL statements.
- Open SQL statements offer several advantages. These include being able to program independent of your underlying database, access to a syntax check, and the use of a local SAP buffer.
- Native SQL statements are bound into a program using

  EXEC SQL [PERFORMING <form>].

  <Native SQL statements>.

  ENDEXEC

- Pay attention to the following when programming Native SQL:
  - Try not to use update operations **(INSERT**, **DELETE**, **UPDATE**)
  - Group **EXEC SQL** statements together (in an include) in order to be able to alter them centrally for different database systems
  - Restrict yourself to Standard SQL (ISO9075:1992)
- In order to optimize performance, choose your SQL statements carefully when accessing several (dependent) tables at a time.



Reading Multiple Database Tables

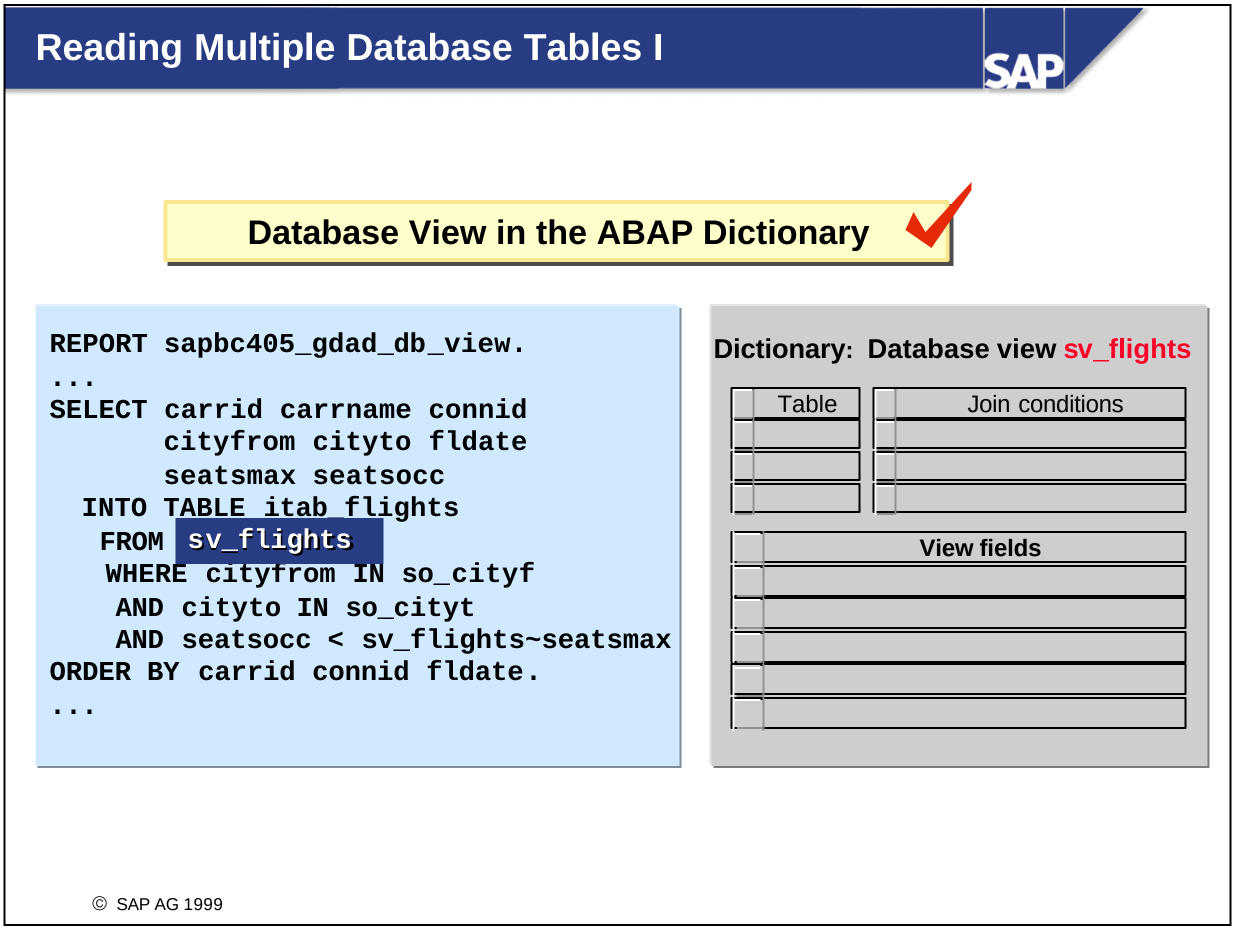
Database View in the ABAP Dictionary

INNER JOIN, OUTER JOIN

FOR ALL ENTRIES

Nested SELECT Statements

- To insure optimal database performance:
- Follow these general rules:
  - Keep the amount of selected data as small as possible (use **WHERE** conditions, for example)
  - Keep data transfer between the application server and the database to a minimum (use field lists, for example)
  - Reduce the number of database inquiries if possible (use table joins instead of nested **SELECT** statements, for example)
  - Reduce search size (this optimizes your database index)
  - Minimize database server load (use SAP buffers, for example).
- Always subject programs containing SQL statements to an SQL trace. Which processing sequence is chosen by the Optimizer? Are indices used? If so, are the right ones used? Is a FULL TABLE SCAN performed? Based on the results of this analysis, you should reprogram your SQL statements (**WHERE**) conditions, create a database index, or buffer the tables better. To start the SQL trace, use menu path GDA-1.



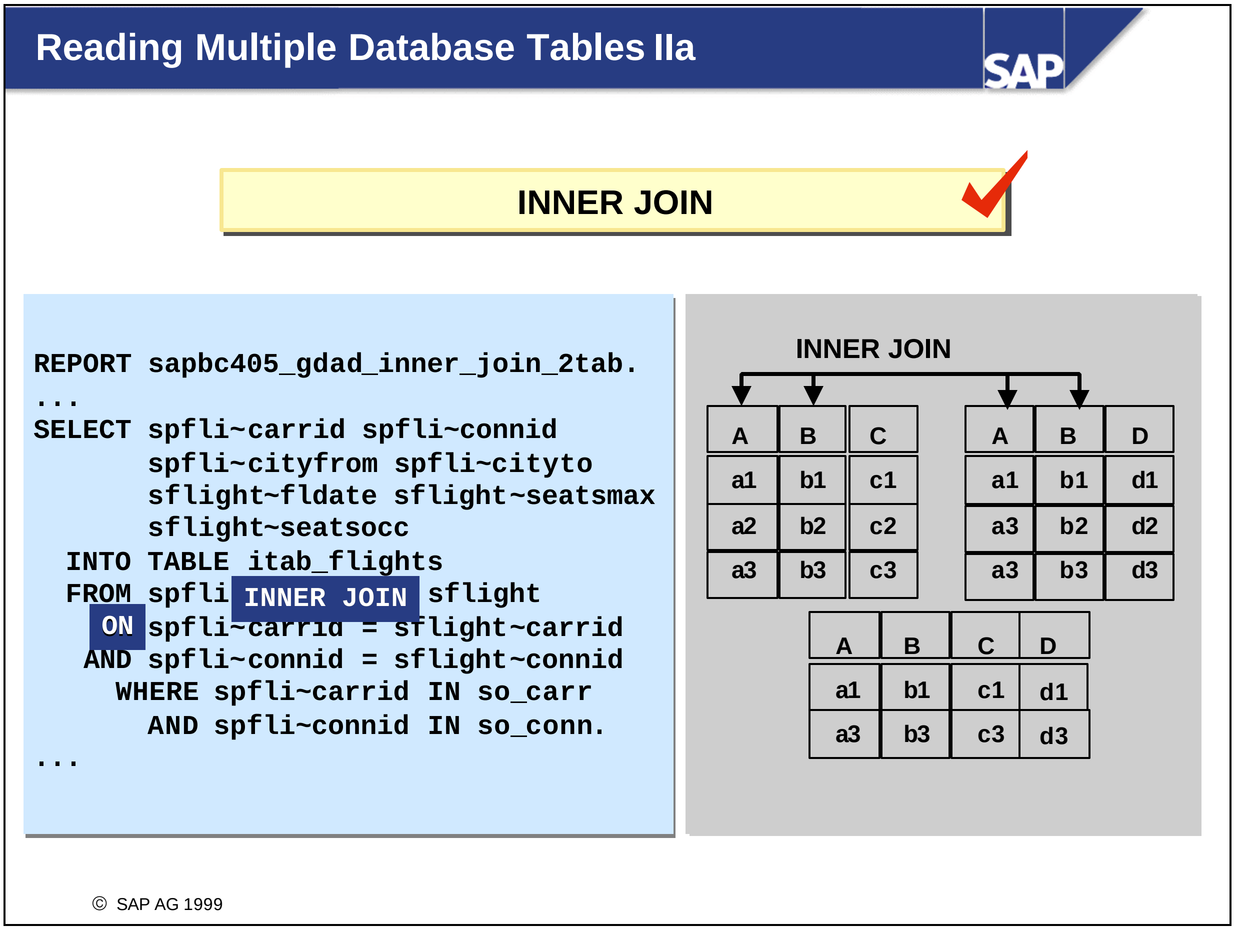
Reading Multiple Database Tables I

Database View in the ABAP Dictionary

```
REPORT sapbc405_gdad_db_view.
...
SELECT carrid carrname connid
       cityfrom cityto fldate
       seatsmax seatsocc
  INTO TABLE itab_flights
   FROM sv_flights
    WHERE cityfrom IN so_cityf
     AND cityto IN so_cityt
     AND seatsocc < sv_flights~seatsmax
ORDER BY carrid connid fldate.
...
```

Dictionary: Database view sv_flights

- You can create database views in the ABAP Dictionary. Views (aggregate objects) are application specific and allow you to work with multiple database tables. The link is mapped in an INNER JOIN LOGIC (see slide on INNER JOIN).
- From Release 4.0 you can buffer database views. You can then read from views using the SAP buffer on the relevant application server. The same rules apply when buffering views as when buffering tables.
- Database view advantages:
  - Central maintenance
  - Accessible to all users
  - Only one SELECT statement is required in the program
- One disadvantage of the view is its low flexibility.



Reading Multiple Database Tables IIa

INNER JOIN

```
REPORT sapbc405_gdad_inner_join_2tab.
...
SELECT spfli~carrid spfli~connid
       spfli~cityfrom spfli~cityto
       sflight~fldate sflight~seatsmax
       sflight~seatsocc
  INTO TABLE itab_flights
  FROM spfli INNER JOIN sflight
    ON spfli~carrid = sflight~carrid
   AND spfli~connid = sflight~connid
     WHERE spfli~carrid IN so_carr
       AND spfli~connid IN so_conn.
...
```

INNER JOIN

A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3

A	B	D
a1	b1	d1
a3	b2	d2
a3	b3	d3

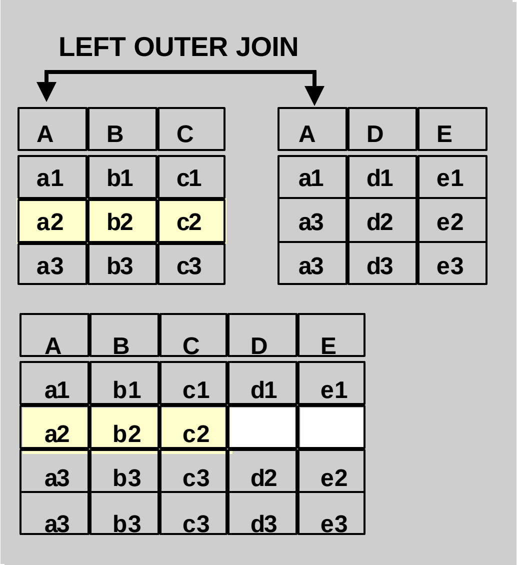
A	B	C	D
a1	b1	c1	d1
a3	b3	c3	d3

- In a join, the tables (base tables) are combined to form one results table. The join conditions are applied to this results table. The resulting composite for an inner join logic contains only those records for which matching records exist in each base table.
- Join conditions are not limited to key fields.
- If columns from two tables have the same name, then you have to ensure that the field labels are unique by prefixing the table name or a table alias.
- A table join is generally the most efficient way to read from the database. The database is responsible for deciding which table is read first and which index is used (DB Optimizer).
-

Reading Multiple Database Tables IIb

OUTER JOIN ✓

```
REPORT sapbc405_gdad_outer_join.
...
SELECT scarr~carrid scarr~carrname
       spfli~connid spfli~cityfrom
       spfli~cityto
 INTO TABLE itab_flights
 FROM  scarr LEFT OUTER JOIN spfli
   ON  scarr~carrid = spfli~carrid
   ORDER BY scarr~carrid spfli~connid.
...
```



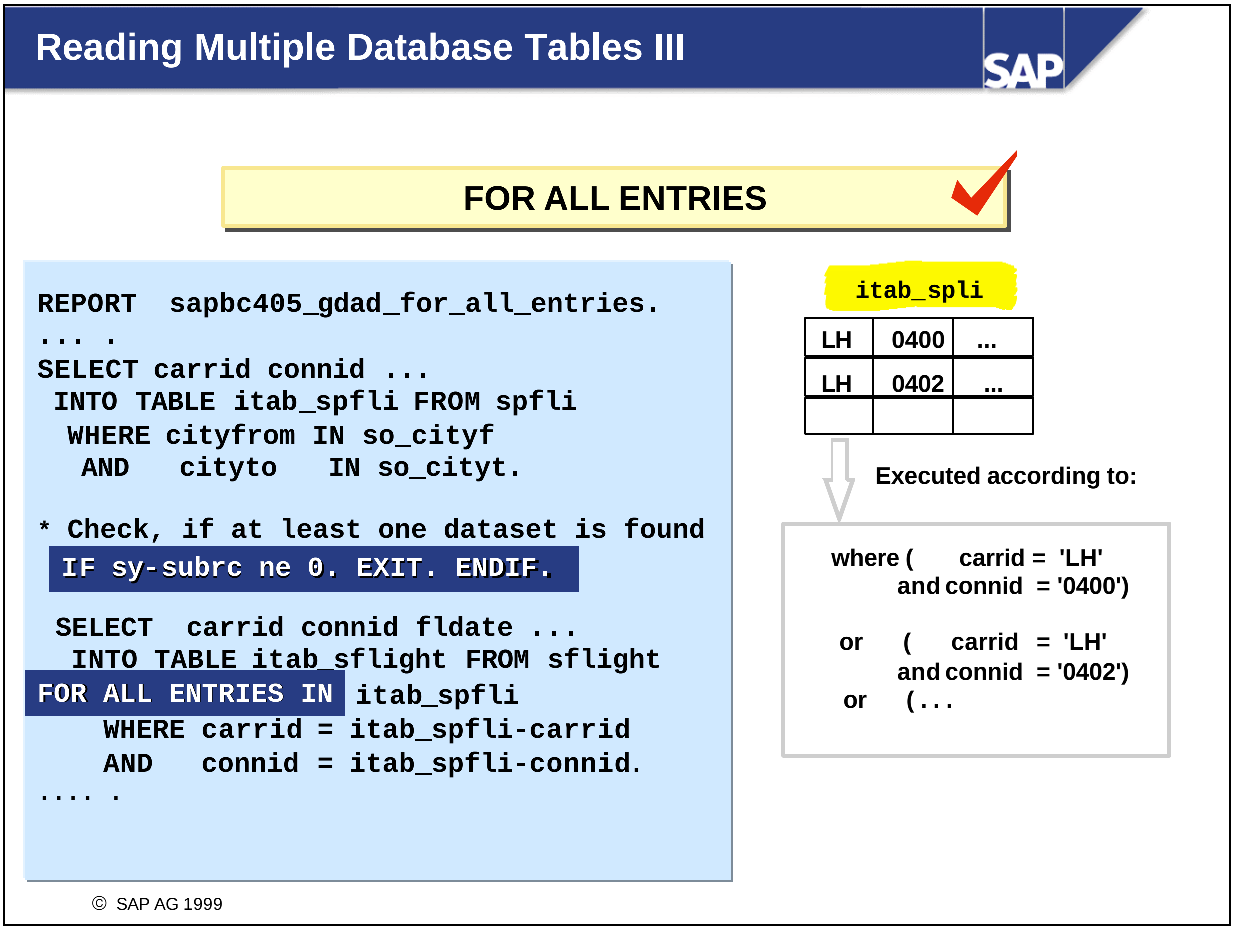
LEFT OUTER JOIN

A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3

A	D	E
a1	d1	e1
a3	d2	e2
a3	d3	e3

A	B	C	D	E
a1	b1	c1	d1	e1
a2	b2	c2		
a3	b3	c3	d2	e2
a3	b3	c3	d3	e3

- At **LEFT OUTER JOIN**, results tables can also contain entries from the designated left-hand table without the presence of corresponding data records (join conditions) from the table on the right. These table fields are filled by the database with null values and are then initialized according to ABAP type.
- It makes sense to use a **LEFT OUTER JOIN** when data from the table on the left is needed for which there are no corresponding entries in the table on the right. Example: sapbc405_gdad_outer_join: not all airlines (table scarr) have flights listed (table spfli), but all airline names are supposed to be displayed in the list.
- The following limitations apply for the Left Outer Join:
  - you can only have a table or a view to the right of the **JOIN** operator, you cannot have another join statement
  - Only **AND** can be used as a logical operator in an **ON** condition.
  - every comparison in the **ON** condition must contain a field from the table on the right.
  - if the **FROM** clause contains an Outer Join, then all **ON** conditions must contain at least one 'true' JOIN condition (a condition that contains a field from tab1 and a field from tab2).



Reading Multiple Database Tables III

FOR ALL ENTRIES

```
REPORT  sapbc405_gdad_for_all_entries.
... .
SELECT carrid connid ...
 INTO TABLE itab_spfli FROM spfli
  WHERE cityfrom IN so_cityf
   AND   cityto   IN so_cityt.

* Check, if at least one dataset is found
  IF sy-subrc ne 0. EXIT. ENDIF.

 SELECT  carrid connid fldate ...
  INTO TABLE itab_sflight FROM sflight
FOR ALL ENTRIES IN itab_spfli
    WHERE carrid = itab_spfli-carrid
    AND   connid = itab_spfli-connid.
.... .
```

itab_spli

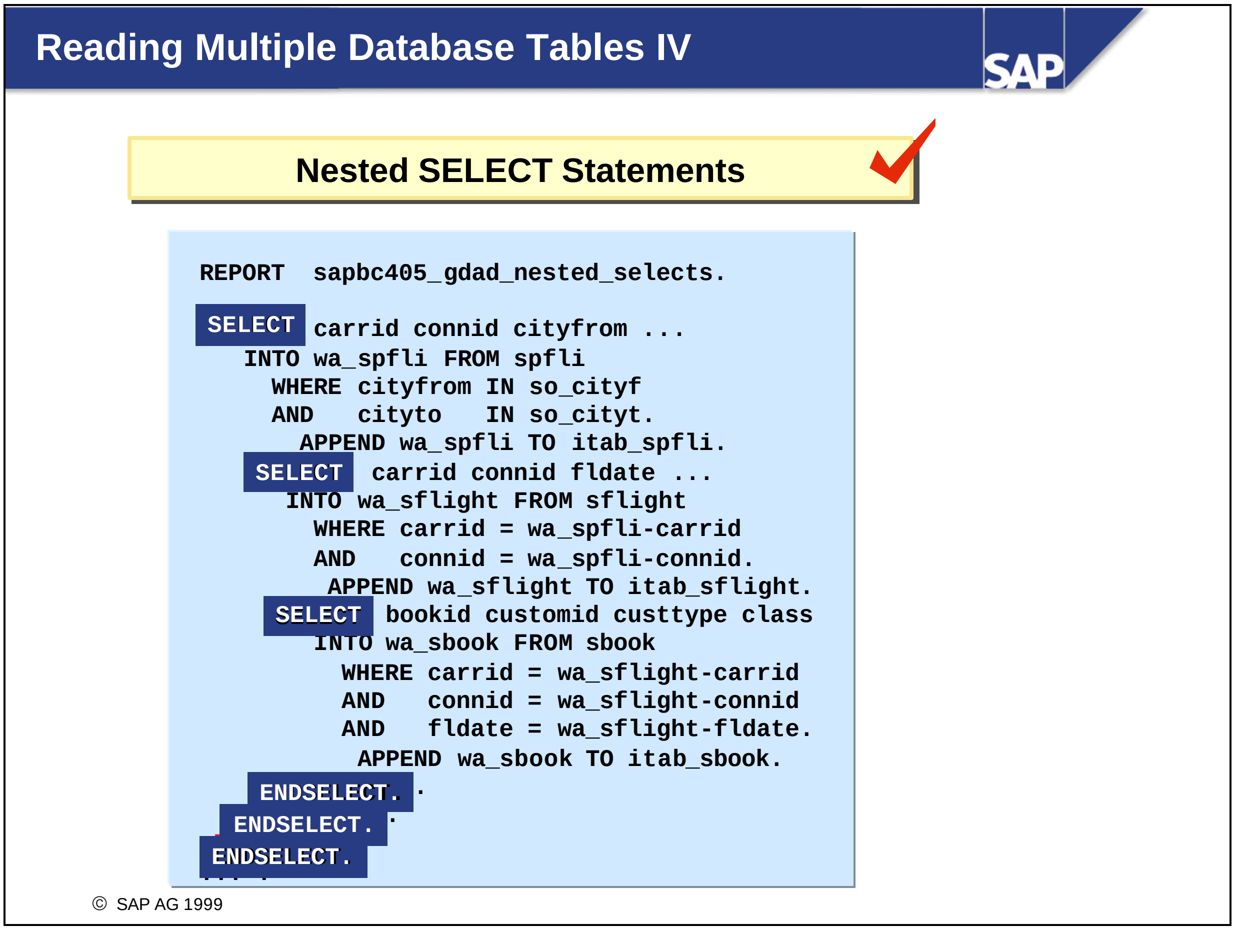
LH	0400	...
LH	0402	...

Executed according to:

```
where (     carrid =  'LH'
        and connid  = '0400')

or     (     carrid  = 'LH'
        and connid  = '0402')
or     ( . . .
```

© SAP AG 1999

- **FOR ALL ENTRIES** works with a database in a quantity-oriented manner. Initially all data is collected in an internal table. Make sure that this table contains at least one entry (query sy-subrc or **DESCRIBE**), otherwise the subsequent transaction will be carried out without any restrictions).
- **SELECT...FOR ALL ENTRIES IN <itab>** is treated like a **SELECT** statement with an external OR condition. The system only selects those table entries that meet the logical condition (**WHERE carrid = itab_sflight-carrid**), replacing the placeholders (**itab_spfli-carrid)** with values from each entry in the internal table itab_spfli. Note that **itab_spfli-carrid** is a placeholder, and not a component of the internal table. Duplicates are not allowed. The internal table can, in principle, be as large as you want it to be.
- Using **FOR ALL ENTRIES** is recommended when data is not being read from the database, that is, it is already available in the program, for example, if the user has input the data. Otherwise a join is recommended.



Reading Multiple Database Tables IV

Nested SELECT Statements

```
REPORT  sapbc405_gdad_nested_selects.

SELECT  carrid connid cityfrom ...
   INTO wa_spfli FROM spfli
     WHERE cityfrom IN so_cityf
     AND   cityto   IN so_cityt.
       APPEND wa_spfli TO itab_spfli.
    SELECT  carrid connid fldate ...
      INTO wa_sflight FROM sflight
        WHERE carrid = wa_spfli-carrid
        AND   connid = wa_spfli-connid.
         APPEND wa_sflight TO itab_sflight.
     SELECT  bookid customid custtype class
        INTO wa_sbook FROM sbook
          WHERE carrid = wa_sflight-carrid
          AND   connid = wa_sflight-connid
          AND   fldate = wa_sflight-fldate.
           APPEND wa_sbook TO itab_sbook.
    ENDSELECT.
  ENDSELECT.
ENDSELECT.
```

- The easiest technical option for reading from multiple (dependent) tables is to use nested **SELECT** statements. The biggest disadvantage of this method is that for every data record contained in the external loop a **SELECT** statement is run using the database. This leads to a considerably worse performance in client/server systems.
- From Release 4.0 you can also work with sub-queries. For more information, refer to the online documentation.

Summary

- When data in the database is read from several independent tables, it is important to optimize the performance of the database accesses.

Unit: Internal Data Collection

Topic: Inner Join

When you have completed these exercises, you will be able to:

- Use an ABAP join to read data from several different DB tables

1-1 Copy or enhance your program Z##SSC1_..., or copy the sample solution, SAPBC405_SSCS_1, to program Z##GDA1_... . Sample solution for exercise: SAPBC405_GDAS_1.

1-1-1 Replace the data collected through database view ***dv_flights*** with an internal **INNER JOIN** performed on the database.

2-1 Deactivate (mark with "*") the three SELECT statements at START-OF-SELECTION. Program INNER JOINs that fill internal table it_flights with data from tables SPFLI and SFLIGHT in the database.

Note:

The structure of the internal table ***it_flights*** does not correspond exactly to the combination of tables SPFLI and SFLIGHT. You must ensure that the fields are copied to the fields of the same name in the target table.

Unit: Programming Data Retrieval

Topic: Inner Join

```
*&---------------------------------------------------------------------*
*& Report  SAPBC405_GDAS_1                                            *
*&                                                               *
*&---------------------------------------------------------------------*
*&    Solution: Exercise 1, Internal Data Collection
*&                                                               *
*&---------------------------------------------------------------------*

INCLUDE bc405_gdas_1top.

*&---------------------------------------------------------------------*
*&   Event TOP-OF-PAGE
*&---------------------------------------------------------------------*
TOP-OF-PAGE.

* Title
  FORMAT COLOR COL_HEADING INTENSIFIED ON.
  ULINE.
  WRITE: / sy-vline,
         'Flight data'(001),
          AT line_size sy-vline.
  ULINE.

* Column header
  FORMAT COLOR COL_HEADING INTENSIFIED OFF.
  WRITE:  sy-vline, AT pos_c1 'Flight'(002).

* Fix left scroll boundary
  SET LEFT SCROLL-BOUNDARY.
  WRITE: 'Date'(003) ,
         'Departure location'(004),
         'Price'(006),
         AT line_size sy-vline.
  ULINE.



*&---------------------------------------------------------------------*
*&   Event   INITIALIZATION
*&---------------------------------------------------------------------*
INITIALIZATION.                          " OPTIONAL

* Initialize select-options for CARRID
  MOVE: 'AA' TO so_car-low,
        'QF' TO so_car-high,
        'BT' TO so_car-option,
        'I' TO so_car-sign.
  APPEND so_car.
  CLEAR so_car.
  MOVE: 'AZ' TO so_car-low,
        'EQ' TO so_car-option,
        'E' TO so_car-sign.
  APPEND so_car.
  CLEAR so_car.

*&---------------------------------------------------------------------*
*&   Event AT SELECTION-SCREEN ON BLOCK PARAM
*&---------------------------------------------------------------------*
AT SELECTION-SCREEN ON BLOCK param.    " OPTIONAL

* check country for national flights is not empty
  CHECK national = 'X' AND country = space.
  MESSAGE e003(bc405).
```

&---

***&   Event START-OF-SELECTION**

&---

START-OF-SELECTION.

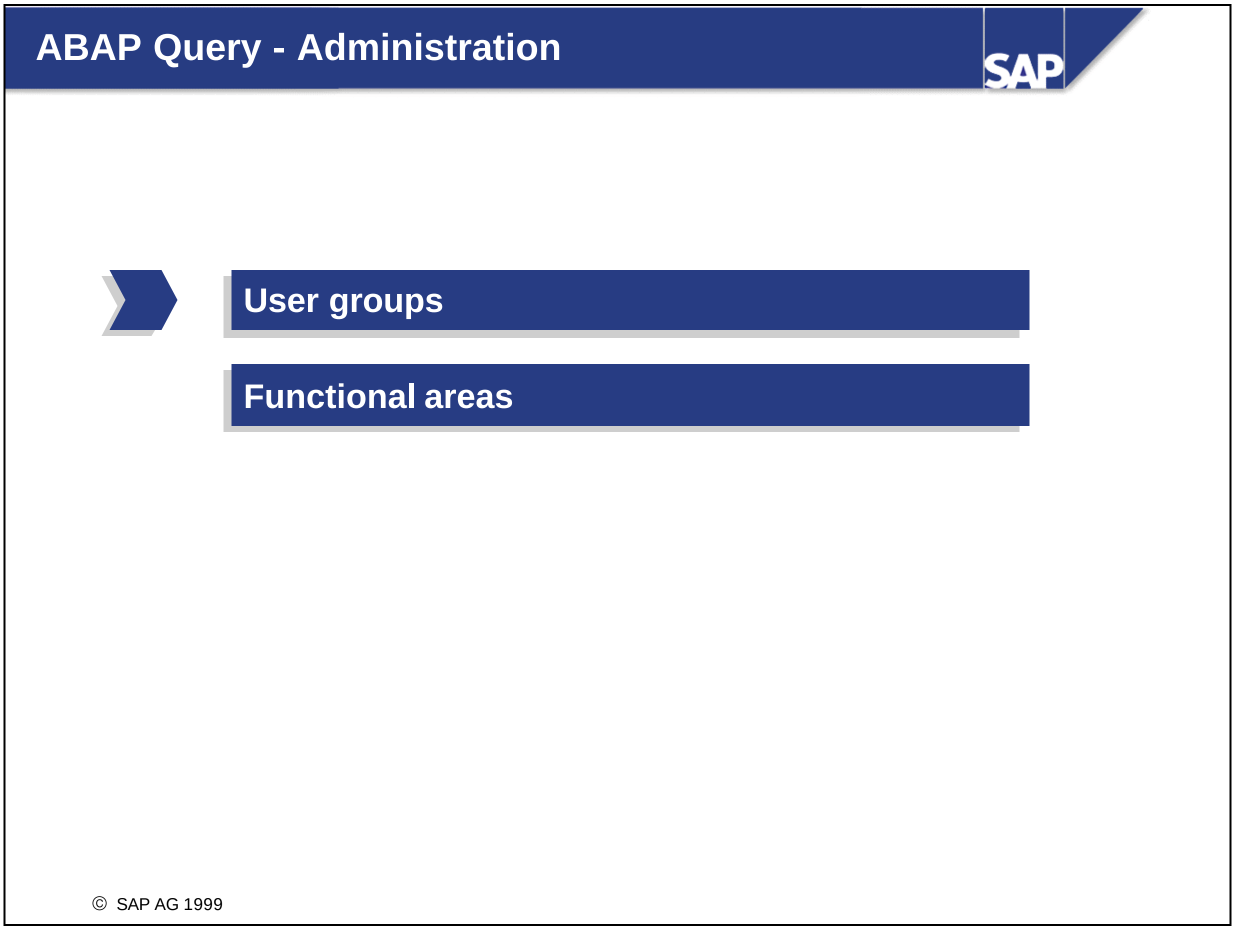
```
* Checking the output parameters
  CASE mark.
    WHEN all.
* Radiobutton ALL is marked
      SELECT * FROM spfli INNER JOIN sflight
          ON spfli~carrid = sflight~carrid
          AND spfli~connid = sflight~connid
       INTO CORRESPONDING FIELDS OF TABLE it_flights
        WHERE spfli~carrid   IN so_car
          AND spfli~connid   IN so_con
          AND sflight~fldate IN so_fdt.

      SORT it_flights BY carrid connid fldate.

    WHEN national.
* Radiobutton NATIONAL is marked
      SELECT * FROM spfli INNER JOIN sflight
             ON spfli~carrid = sflight~carrid
             AND spfli~connid = sflight~connid
       INTO CORRESPONDING FIELDS OF TABLE it_flights
        WHERE spfli~carrid   IN so_car
          AND spfli~connid   IN so_con
          AND sflight~fldate IN so_fdt
          AND spfli~countryfr = spfli~countryto
          AND spfli~countryfr = country.

      SORT it_flights BY carrid connid fldate.

    WHEN internat.
* Radiobutton INTERNAT is marked
      SELECT * FROM spfli INNER JOIN sflight
          ON spfli~carrid = sflight~carrid
          AND spfli~connid = sflight~connid
       INTO CORRESPONDING FIELDS OF TABLE it_flights
        WHERE spfli~carrid   IN so_car
          AND spfli~connid   IN so_con
          AND sflight~fldate IN so_fdt
          AND spfli~countryfr NE spfli~countryto.

      SORT it_flights BY carrid connid fldate.

  ENDCASE.

* Additional solution: dynamical WHERE condition
* PERFORM get_data.

*&---------------------------------------------------------------------*
*&   Event END-OF-SELECTION
*&---------------------------------------------------------------------*
END-OF-SELECTION.

* Data output
  PERFORM data_output.

*&---------------------------------------------------------------------*
*&      Form  DATA_OUTPUT
*&---------------------------------------------------------------------*
*       List output of flight data
*----------------------------------------------------------------------*
FORM data_output.

* Loop at the internal table for writing data
  LOOP AT it_flights INTO wa_flights.

* Get a new page if CONNID has changed.
    ON CHANGE OF wa_flights-connid.
      NEW-PAGE.
    ENDON.

* Mark international flights
    FORMAT COLOR COL_KEY INTENSIFIED ON.
    IF wa_flights-countryfr EQ wa_flights-countryto.
      WRITE: / sy-vline, icon_space AS ICON CENTERED.
    ELSE.
```

```
      WRITE: / sy-vline, icon_bw_gis AS ICON CENTERED.
    ENDIF.

* Data output
    WRITE: wa_flights-carrid,
           wa_flights-connid.
    FORMAT COLOR COL_NORMAL INTENSIFIED OFF.
    WRITE: wa_flights-fldate,
           wa_flights-cityfrom,
           wa_flights-cityto,
           wa_flights-price CURRENCY wa_flights-currency,
           wa_flights-currency,
           AT line_size sy-vline.

  ENDLOOP.

ENDFORM.                              " DATA_OUTPUT



*          <--------------   ADDITIONAL -------------->          *
*&---------------------------------------------------------------------*
*&      Form  GET_DATA
*&---------------------------------------------------------------------*
* Instead of programming three different SELECT statements, these
* SELECTs can be combined in one dynamical WHERE condition.
**--------------------------------------------------------------------
*
*FORM GET_DATA.                                    "#EC CALLED
*
*  DATA: WHERE_LINE(40),
*        WHERE_TAB LIKE TABLE OF WHERE_LINE.
*
** only national flights requested
*  IF NATIONAL NE SPACE.
*    WHERE_LINE = 'p~countryfr = p~countryto'.          "#EC NOTEXT
*    APPEND WHERE_LINE TO WHERE_TAB.
*    CONCATENATE 'AND p~countryfr ='    "#EC NOTEXT
*          '''' INTO WHERE_LINE
*          SEPARATED BY SPACE.
*    CONCATENATE WHERE_LINE COUNTRY '''' INTO WHERE_LINE.
*    APPEND WHERE_LINE TO WHERE_TAB.
*  ENDIF.
*
** only international flights requested
*  IF INTERNAT NE SPACE.
*    WHERE_LINE = 'p~countryfr NE p~countryto'.          "#EC NOTEXT
*    APPEND WHERE_LINE TO WHERE_TAB.
*  ENDIF.
*
** Close WHERE-clause by dot
*  WHERE_LINE = '.'.
*    APPEND WHERE_LINE TO WHERE_TAB.
*
** Inner join with dynamical where clause
*  SELECT P~CARRID P~CONNID
*         P~COUNTRYFR P~CITYFROM P~AIRPFROM
*         P~COUNTRYTO P~CITYTO P~AIRPTO
*         F~FLDATE F~PRICE F~CURRENCY
*    FROM SPFLI AS P JOIN SFLIGHT AS F
*         ON P~CARRID = F~CARRID AND P~CONNID = F~CONNID
*   INTO CORRESPONDING FIELDS OF TABLE IT_FLIGHTS
*    WHERE P~CARRID IN SO_CAR
*      AND P~CONNID IN SO_CON
*      AND F~FLDATE IN SO_FDT
*      AND (WHERE_TAB).
*
*ENDFORM.                              " GET_DATA
*&---------------------------------------------------------------------*
*& Include BC405_GDAS_1TOP                                             *
*&                                                                     *
*&---------------------------------------------------------------------*

REPORT   sapbc405_gdas_1 LINE-SIZE 100 NO STANDARD PAGE HEADING.
```

```
* Include for using icons
INCLUDE <icon>.

* Constants for writing position
CONSTANTS:  pos_c1 TYPE i VALUE  6,
          line_size TYPE i VALUE 100.

* Constant for CASE statement
CONSTANTS mark VALUE 'X'.

* Internal table like DDIC view DV_FLIGHTS
DATA: it_flights LIKE TABLE OF dv_flights,
     wa_flights LIKE dv_flights.

* Selections for connections
SELECTION-SCREEN BEGIN OF BLOCK conn WITH FRAME TITLE text-tl1.
SELECT-OPTIONS: so_car FOR wa_flights-carrid,
              so_con FOR wa_flights-connid.
SELECTION-SCREEN END OF BLOCK conn.

* Selections for flights
SELECTION-SCREEN BEGIN OF BLOCK flight WITH FRAME TITLE text-tl2.
SELECT-OPTIONS so_fdt FOR wa_flights-fldate NO-EXTENSION.
SELECTION-SCREEN END OF BLOCK flight.

* Output parameter
SELECTION-SCREEN BEGIN OF BLOCK param WITH FRAME TITLE text-tl3.
SELECTION-SCREEN BEGIN OF BLOCK radio WITH FRAME.
PARAMETERS: all RADIOBUTTON GROUP rbg1,
          national RADIOBUTTON GROUP rbg1,
          internat RADIOBUTTON GROUP rbg1 DEFAULT 'X'.
SELECTION-SCREEN END OF BLOCK radio.
PARAMETERS country LIKE wa_flights-countryfr.
SELECTION-SCREEN END OF BLOCK param.
```

SAP Query - Administration

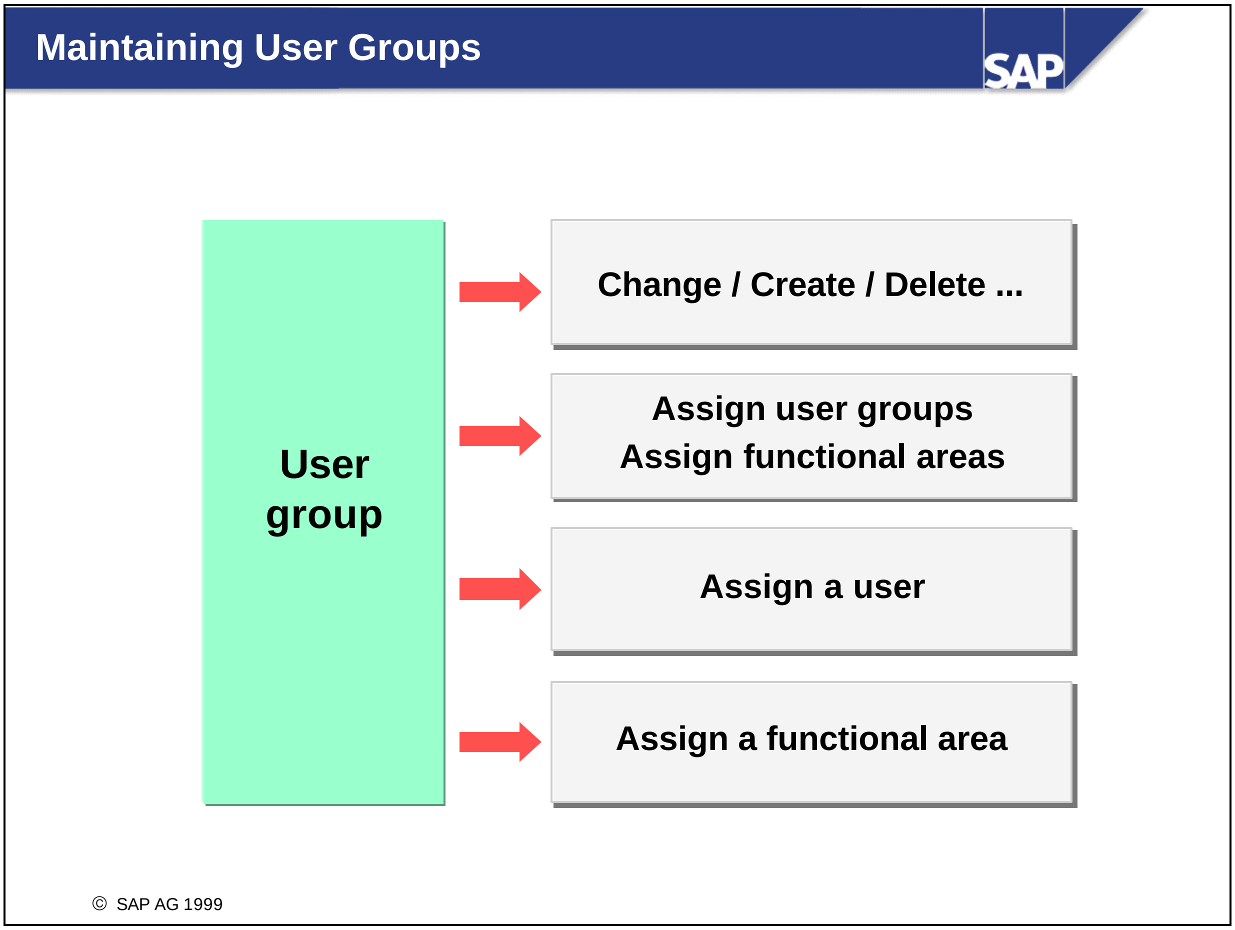
- User groups
- Functional areas



ABAP Query - Administration

User groups

Functional areas



Maintaining User Groups

User group

Change / Create / Delete ...

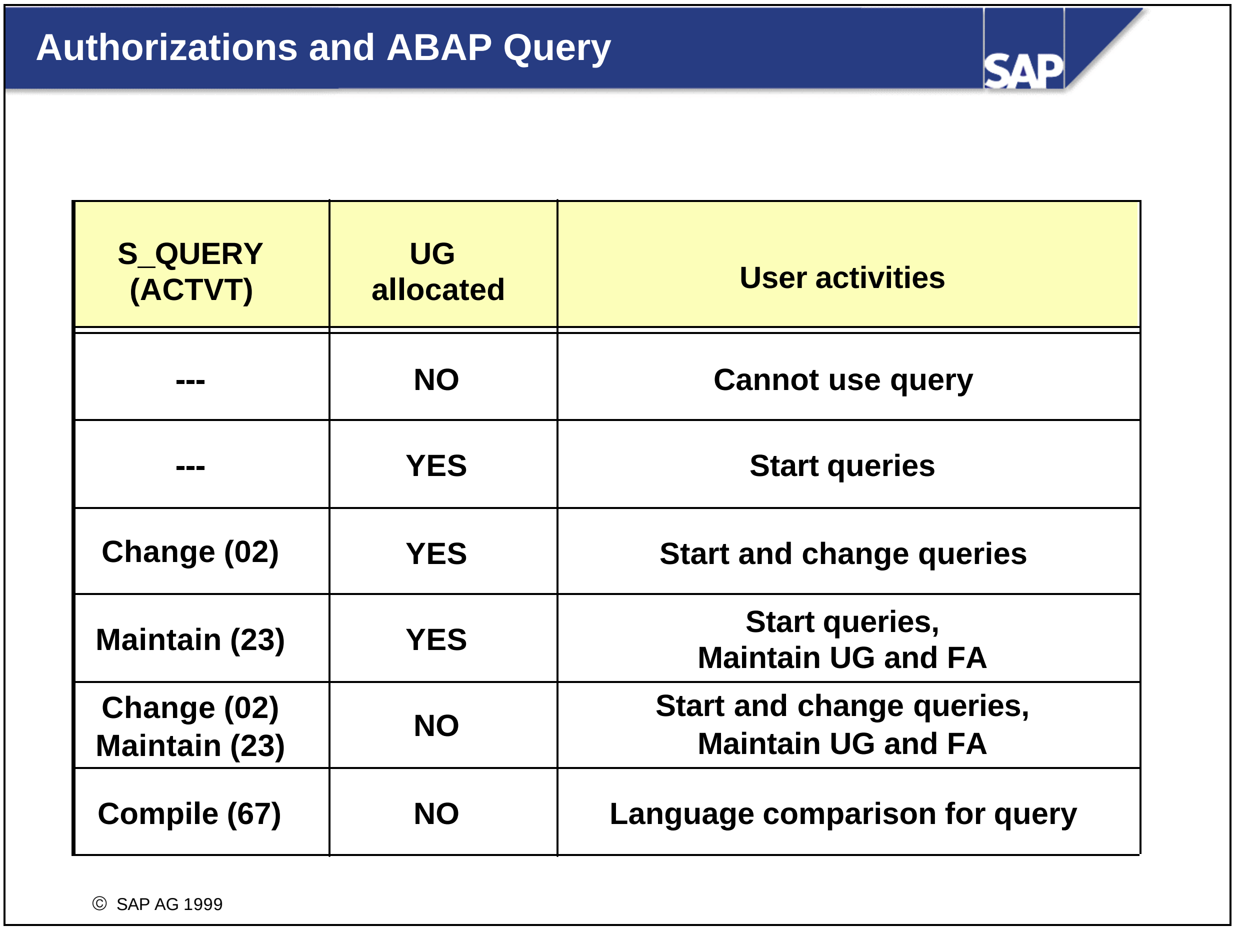
Assign user groups
Assign functional areas

Assign a user

Assign a functional area

- When you create a user group, you must assign a name and a description to the user group.
- To maintain a user group, you can use the following options:
  - You can assign users and functional areas to a user group.
  - You can assign a user to various user groups.
  - You can assign a functional area to various user groups.

- You can assign every user in the SAP System to one or more user groups.



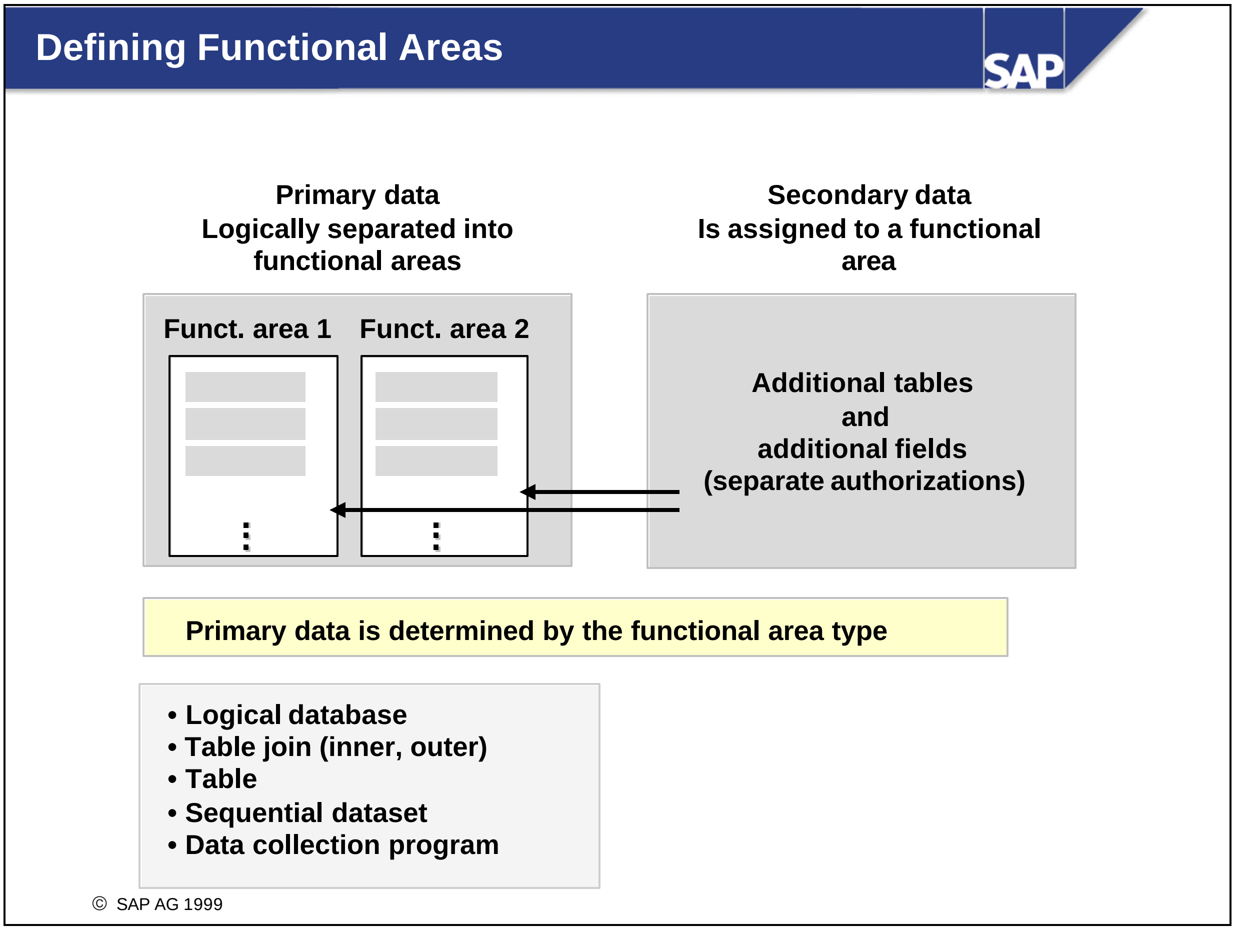
Authorizations and ABAP Query

S_QUERY (ACTVT)	UG allocated	User activities
---	NO	Cannot use query
---	YES	Start queries
Change (02)	YES	Start and change queries
Maintain (23)	YES	Start queries, Maintain UG and FA
Change (02) Maintain (23)	NO	Start and change queries, Maintain UG and FA
Compile (67)	NO	Language comparison for query

- There are basically two mechanisms for determining who can do what with a query.
- The first restriction involves the assignment to a user group. Users who have not been assigned to any user groups cannot use the query. Users who have been assigned to at least one user group can start queries.
- The second mechanism is field ACTVT in authorization object S_QUERY. You can use this field to authorize users to change queries as well (ACTVT = 02).
- ACTVT = 23 is required to maintain functional areas and user groups (typical administration tasks). An administrator does not necessarily have to be assigned to a user group.
- A separate authorization (ACTVT = 67) is required to compile the query.

ABAP Query - Administration

- User groups
- ➤ Functional areas



Defining Functional Areas

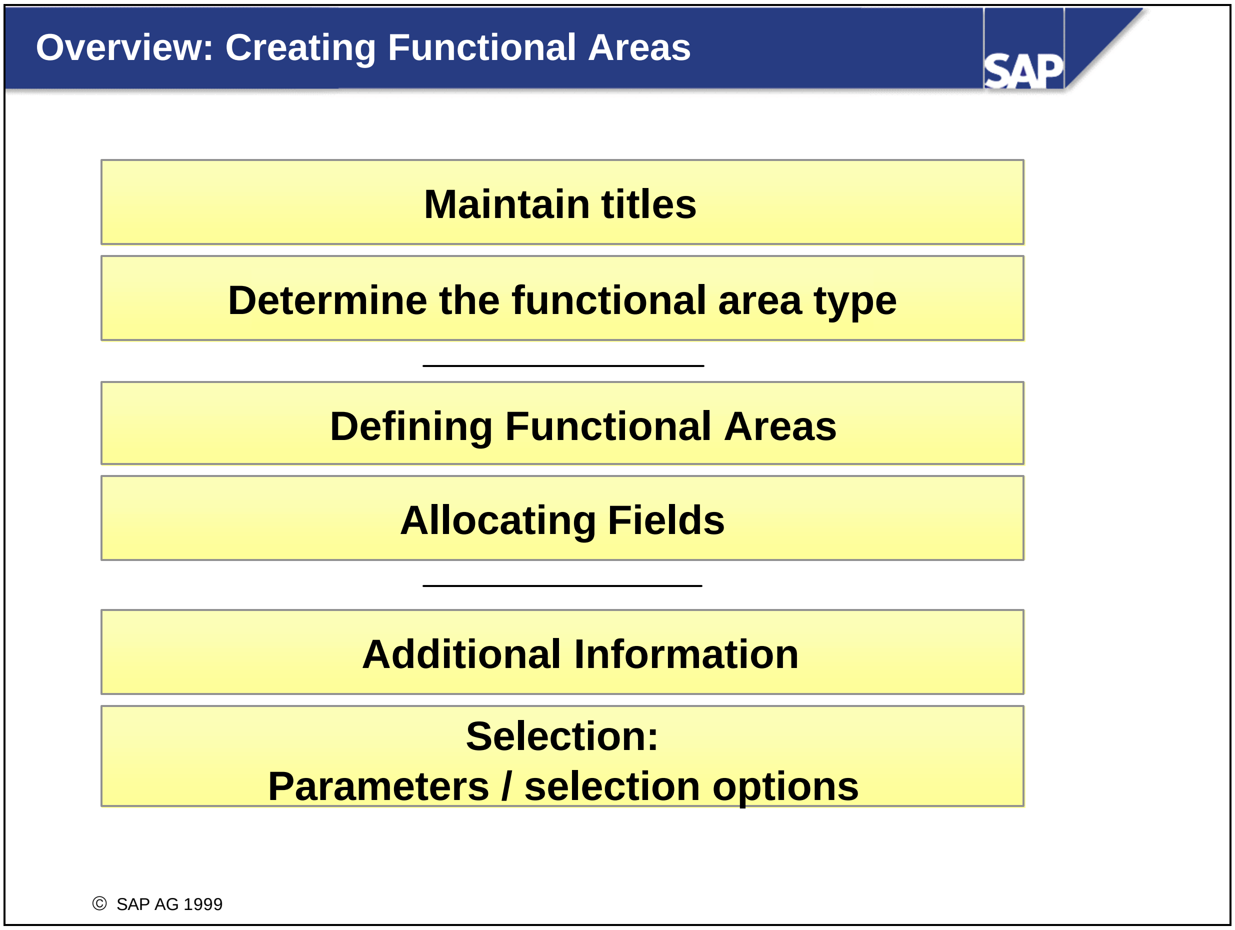
Primary data
Logically separated into
functional areas

Secondary data
Is assigned to a functional
area

Primary data is determined by the functional area type

- **Logical database**
- **Table join (inner, outer)**
- **Table**
- **Sequential dataset**
- **Data collection program**

- When you define a functional area, you must first decide on a primary dataset (functional area type).
- You can assign secondary datasets (tables, supplementary fields) to the primary dataset.
- You can choose all the fields of the primary dataset and the secondary dataset. You choose fields by assigning them to a functional group.
- Users cannot access fields that are not assigned to a functional group.
- For the end user it is irrelevant whether a field belongs to primary or secondary data.
- There are different types of functional areas:
  - Creating a functional area from a logical database provides data retrieval for logical databases (GET <node>).
  - Creating a functional area from a table join allows access to data in tables that are linked with an INNER or OUTER JOIN (SELECT).
  - Creating a functional area from a table evaluates data from a table or from a database view (SELECT).
  - Creating a functional area from a program evaluates data read by a predefined program.
  - Creating a functional area from a sequential file reads data stored in a sequential file (READ DATASET).
- For the enduser, it is irrelevant which type of functional area is used.



Overview: Creating Functional Areas

Maintain titles

Determine the functional area type

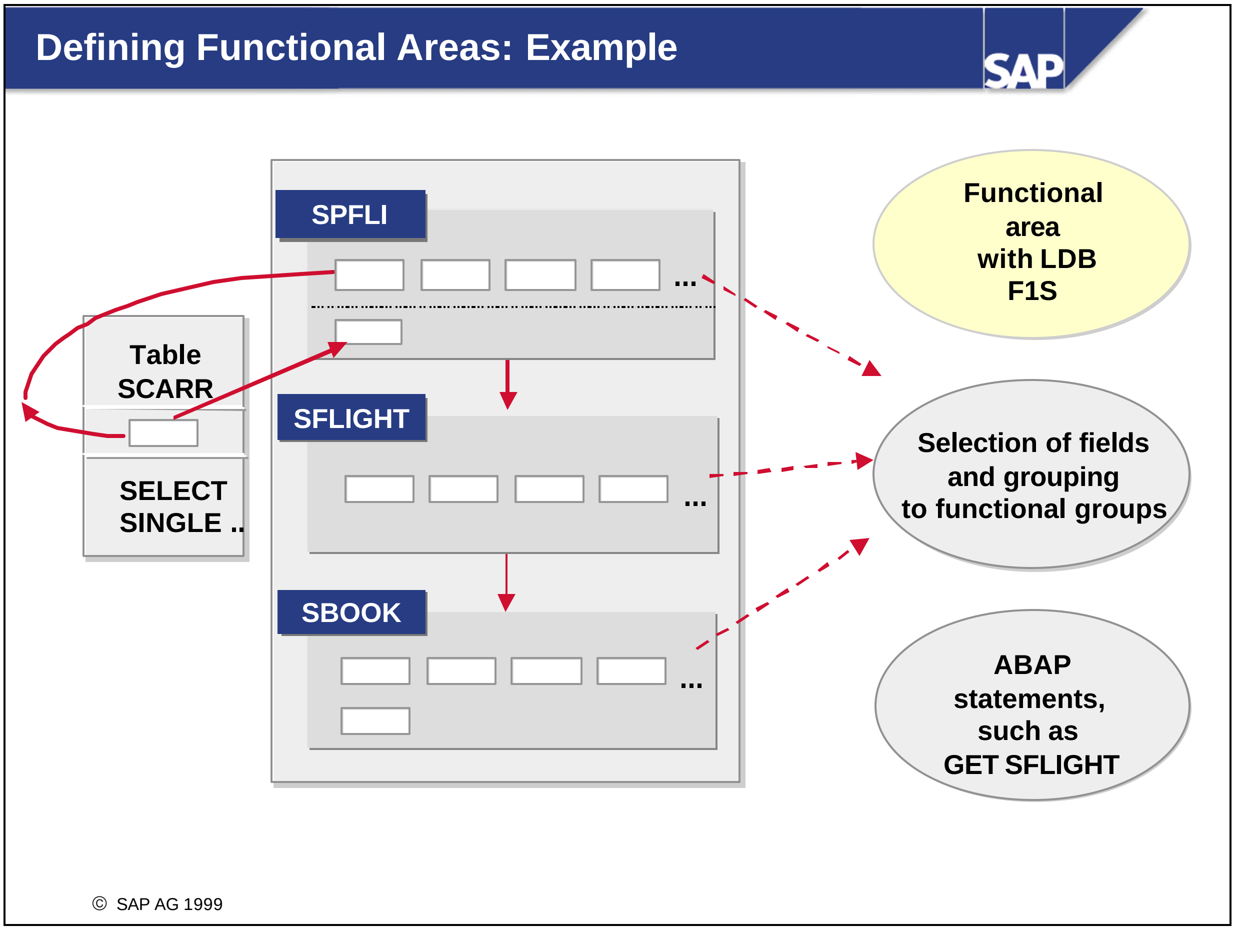
Defining Functional Areas

Allocating Fields

Additional Information

Selection:
Parameters / selection options

- When you create a functional area, you have to enter a title and select the type of functional area. You can force every query in a functional area to be allocated to an authorization group (provides protection of the generated program, object S_PROGRAM).
- After header maintenance, you can divide the available fields into functional groups.
- You can read additional tables and use additional fields.
- You can define parameters and selection options (determination of the selection screen).
- When you create a functional area, you must allocate it to the user group with which it is to work.
- The system administers every functional area in two versions: a generated version and a revised version. After you change a functional area, you must generate it so that the changes become known. Queries work only with the generated versions of functional areas.
- You can delete a functional area only when it contains no queries.
- You can user the directory function to see an overview of the available functional areas along with their contents and all queries created from a given functional area (menu path AQA-1).



Defining Functional Areas: Example

SPFLI

...

Table SCARR

SELECT SINGLE ..

SFLIGHT

...

SBOOK

...

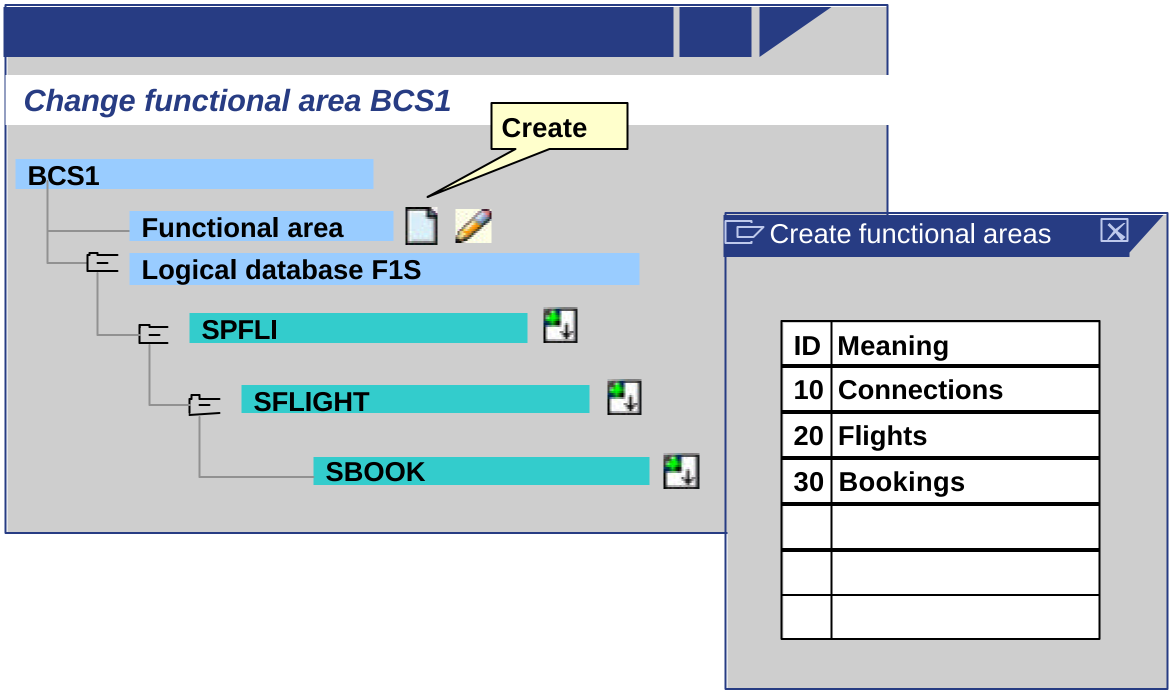
Functional area with LDB F1S

Selection of fields and grouping to functional groups

ABAP statements, such as GET SFLIGHT

- For practical purposes, retrieval of supplemental data (table accesses, supplemental fields) is an extension of the tables already present.
- When you define table accesses and supplemental fields, you must specify the time and the order (of access).
- You can also use ABAP statements to allocate various events.

Defining Functional Areas



ID	Meaning
10	Connections
20	Flights
30	Bookings

- A functional group is the grouping of several fields into a logical unit. They serve to simplify field selection for the user.
- The functional group is displayed using a two-part tree structure:
  - The first sub-tree contains the functional groups. This sub-tree is flat: all functional groups exist next to each other at the same level. Each functional group contains the fields that have been assigned to it.
  - The second sub-tree describes the structure of the data that will be read. The structure of functional areas with logical databases corresponds to the structure of the logical database and contains all the nodes and the hierarchical relationships defined between the nodes. In the case of a table join, the structure contains all the tables of the join in a flat structure.
- Functional groups can be created using the *Create* icon or from the menu. You must assign an ID code and a long text when creating a functional group.

- To allocate fields to a functional area, you must choose or select it. The system highlights the selected functional areas in color.
- To allocate a field to a functional area, you must first make the field visible by expanding the corresponding table in the second partial tree. A special icon next to the field name indicates if the field has been allocated to a functional area. If the icon is a minus sign, the field has not been allocated to a functional group. A single click on the icon allocates the field to the currently marked functional area. The icon behind the ID code of the functional group then changes from a minus sign to a plus sign.
- A single click on a plus-icon cancels the allocation. You can only cancel the field-functional area allocation if the field has not yet been used in queries.

- You can define additional tables, additional fields, and ABAP statements either at the GET event or during record processing for each table in the second sub-tree.
- You can reach these enhancements with the *Extras* function. The enhancements are summarized for each table.
- To maintain additions for a particular table, you must position the cursor on a field in the table and then call the function. A window appears that displays all additional tables, additional fields, and statements allocated to the table.
- The number in the first column indicates the order of the code sections in the generated program. This number becomes important when individual enhancements are dependent upon each other.
- Multiple allocation of additional tables (alias tables):
    - Alias tables enable repeated use of a table.
    - You can assign multiple alias names for a table. You can then address the table with its various names.

- You allocate an additional table as follows:
  - Call the *Create* function on the screen.
  - Enter the name of the additional table and set the selection button on the additional table.
  - Enter the order of the coding section and the **`WHERE`** clause in the following screen.
- A query performs a table access only when the query requires this table field.

- You allocate an additional field as follows:
  - Call the *Create* function on the screen.
  - Enter the name of the additional field and select the *Additional field* radio button.
  - On the following screen, define the code section's sequence and the field itself.
- You can enter the format specifications directly using the data type or by referring to an ABAP Dictionary field.
- This screen does not contain ABAP statements for the additional field. To maintain such statements, you must branch to the Editor using the *Editor* function. The Editor syntax check is available there. You can use include programs, external form routines, and function modules.

- Selection criteria must always refer to primary data fields or to fields that have been defined in a DATA statement in the functional area.
- When you specify the order, you also determine the output sequence of individual selection criteria and parameters. However, all the standard selections of the logical database appear in the lead positions on the selection screen.

- If selections have been defined in addition to the logical database, and these selections refer to a node that does not support free selections, then you have to use a CHECK statement to check the collected data.

Unit: SAP Query – Administration

Topic: Creating Functional Areas

When you have completed these exercises, you will be able to:

Create a functional area with a logical database

Include additional fields

Include additional tables

1-1 Create a functional area, BCS2-##, with logical database F1S in the global query area (note: ## is the group number).

1-1-1 Create the following functional groups:

Connections

Flights

Bookings

1-1-2 Assign the following fields to the functional groups:

Connections: SPFLI-CARRID

SPFLI-CONNID

SPFLI-CITYFROM

SPFLI-AIRPFROM

SPFLI-CITYTO

SPFLI-DEPTIME

SPFLI-ARRTIME

SPFLI-FLTYPE

Flights: SFLIGHT-PRICE

SFLIGHT-CURRENCY

SFLIGHT-PLANETYPE

SFLIGHT-SEATSMAX

SFLIGHT-SEATSOCC

SFLIGHT-PAYMENTSUM

SFLIGHT-FLDATE

Bookings: SBOOK-BOOKID

SBOOK-CUSTOMID

SBOOK-CUSTTYPE

SBOOK-SMOKER

SBOOK-LUGGWEIGHT

SBOOK-WUNIT

1-2 Create the additional field FREE (with the description *free seats*) for table SFLIGHT.

1-2-1 Assign the field FREE to the functional group *Flights.*

1-3 For the table SBOOK, read additional information from table SCUSTOM.

Create additional table SCUSTOM for table SBOOK.

Assign the following fields to the *Bookings* functional group:

SCUSTOM-NAME

SCUSTOM-FORM

SCUSTOM-STREET

SCUSTOM-POSTCODE

SCUSTOM-CITY

SCUSTOM-TELEPHONE

1-4 Assign the functional area to your user group BC_STUDENTS.

1-4-1 Create a query, QE2-##, for functional area BCS2-##, and test the functional area.

1-4-2 The list should have the following line structure:

Line 1: Short name of the airline, code of the

flight connection

Line 2: Departure city, arrival city, free seats

Line 3: Flight date

Line 4: Form of address, customer name, street, city, and zip code

Data Formatting and Control Level Processing

- With internal tables
- With extract datasets (see appendix)

- You can use control level processing to create structured lists. Control levels are determined by the contents of the fields that are to be displayed. there is a control level change whenever the content of a field changes. This means that there is no point in creating control levels unless the data are sorted.
- The data to be displayed must be saved temporarily if you want to use control level processing. You can also use internal tables and intermediate datasets.

- From Release 4.0, the R/3 System has included three types of tables: Standard tables (**STANDARD TABLE**), sorted tables (**SORTED TABLE**), and hashed tables(**HASHED-TABLE**).
- For information on the complete syntax of internal tables, see the online documentation.

- You can use an array fetch in a SELECT statement to fill an internal table in one go.
- You can use the **APPEND** statement to insert table entries at the end of an internal table. The variant of the **APPEND** statement on the slide is permitted only for standard or sorted tables. After an **APPEND** statement, system field **SY-TABIX** contains the index value of the newly inserted table entry.
- You use the **COLLECT** statement to generate unique or compressed datasets. The contents of the work area <wa> of the internal table are recorded as a new entry at the end of the table or are added to an existing entry. The latter occurs when the internal table already contains an entry with the same key field values as those currently in the work area. The numeric fields that do not belong to the key are added to the corresponding fields of the existing entry.
- When the **COLLECT** statement is used, all the fields that are not part of the key must be numeric.

- The **SORT** statement sorts the entries in internal table <itab> in ascending order. If the addition **BY <f1>** ..., is missing, then the key assigned when the table was defined is used.
- If addition **BY <f1> <f1> ...** is used, then fields <f1>, <f2>, ... are used as sort keys. The fields can be of any type.
- You can use the additions **ASCENDING** and **DESCENDING** with the **SORT** statement to determine whether the fields are sorted in ascending (default) or descending order.
- For more information about the **SORT** statement, please refer to appendix documentation **DAP-3**.
- You can use the loop statement **LOOP AT <itab> ... ENDLOOP** to process an internal table. The data records in the internal table are processed sequentially.
- The **CONTINUE** statement can be used to prematurely exit the current loop pass and skip to the next pass.
- The **EXIT** statement can be used to exit loop processing.
- At the end of loop processing (after **ENDLOOP**), return value sy-subrc indicates whether the loop was passed or not.

  SY-SUBRC = 0: The loop was passed at least once
  SY-SUBRC = 4: The loop was not passed because no entry was available.

- You can use special **control structures** for control level processing. All the structures begin with **AT** and end with **ENDAT**. These control structures can only be used within a **LOOP**.
- The statement blocks **AT FIRST** and **AT LAST** are run exactly once: at the first **AT FIRST** and at the last **AT LAST** loop.
- The statements within **AT NEW <f> ... ENDAT** are executed when the value of field <f> changes within the ***current*** **LOOP** (start of a control level) or the value of one of the fields in the table definition (further to the left).
- The statements within **AT END OF <f> ... ENDAT** are executed when the value of field <f> changes during the ***next*** **LOOP** (end of a control level) or the value of one of the fields in the table definition (further to the left).
- At entry of the control level (directly after **AT**),
  - all fields with the same character types after (to the right of) the current control level key
  are filled with "*"
  - all other fields after (to the right of) the current control level key are set to default values.
- When a control structure is exited (at **ENDAT**), all fields of the query area are filled with the data from the current loop pass.
- The SUM statement supplies the respective group totals in the query area of the **LOOP** in all fields of TYPE I, F and P.

- The control level structure in internal tables is static. It corresponds exactly to the sequence of columns in the internal table (from left to right). In particular, the control level structure for internal tables is independent of the criteria used to sort the internal table. The table must be sorted according to the internal table fields.
- When you implement control level processing, you must follow the sequence of individual control levels within the **`LOOP`** as illustrated in the slide. The sequence follows the sequence of fields in the internal table and is therefore also the sort sequence.
- The processing block between AT FIRST and ENDAT is executed before processing of the single lines begins. The processing block AT LAST and ENDAT is executed after all single lines have been processed.

Unit: Data Formatting and Control Level Processing

Topic: Internal Table

When you have completed these exercises, you will be able to:

- Implement control level processing with internal tables

1-1 Copy or enhance your program Z##GDA1_..., or copy the sample solution, SAPBC405_GDAS_1, to program Z##DAP1_... . Sample solution for exercise: SAPBC405_DAPS_1.

Carry out control level processing for CITYFROM, CITYTO, CARRID, CONNID. Create a list like the one in the template.

1-1-1 Create a line type in the TOP include (TYPES statement). Declare the internal table and the work area in accordance with the line type. You should include the following fields:

CARRID, CONNID, FLDATE, PRICE, CURRENCY, CITYFROM, COUNTRYFR, CITYTO, COUNTRYTO, SEATSMAX, SEATSOCC.

Note: The control level hierarchy for an internal table is established by your **line structure**.

1-1-2 Sort the internal table in accordance with the requested control level processing (event: END-OF-SELECTION).

1-2. Implement the control level processing in your output routine.

1-2-1 Output each new departure city on a separate page and with intensive display in color COL_GROUP in the list.

1-2-2 Output the city and airport for each new arrival city with a less intensive display in color COL_GROUP.

1-2-3 Ensure that the icon ICON_BW_GIS (for international flights), the airline and the flight number are displayed as required during single-record processing. Display the key fields in color COL_KEY; they should remain as hard lead columns. Display the flight date, the price, the currency, the maximum number of seats and the number occupied in color COL_NORMAL in the list with a less intensive display.

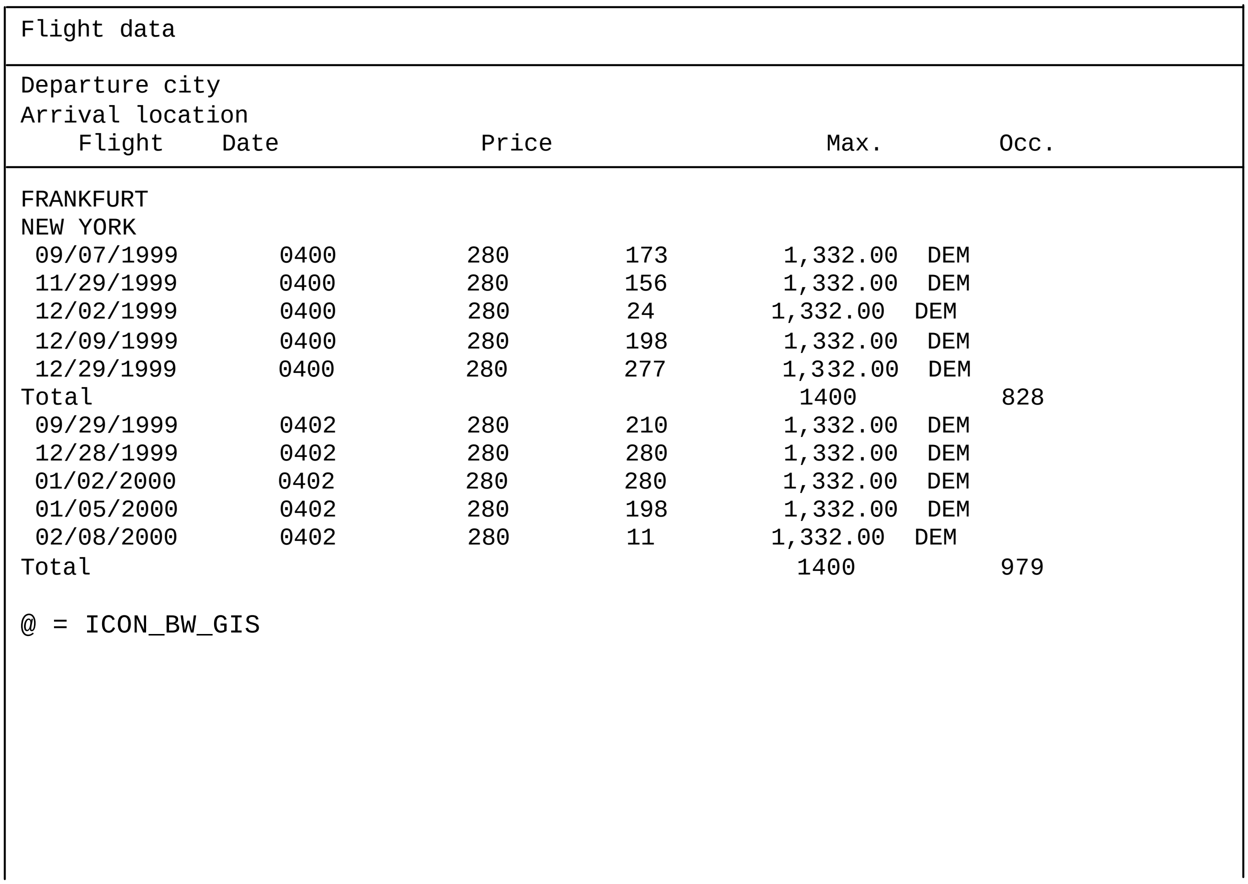
1-2-4 At the end of each flight connection, the totals for the maximum number of seats and the number occupied should be displayed in color COL_TOTAL without intensive display.

1-2-5 A solid line should appear on the list before the departure location changes.

1-2-6 Draw a frame around the list (sy-vline) .

1-2-7 Modify the column headers to fit the new list output (see the template). Use text elements to allow your texts to be translated.

Template:



```
Flight data
--------------------------------------------------------------------------------
Departure city
Arrival location
    Flight    Date              Price                    Max.        Occ.
--------------------------------------------------------------------------------

FRANKFURT
NEW YORK
 09/07/1999       0400         280         173        1,332.00  DEM
 11/29/1999       0400         280         156        1,332.00  DEM
 12/02/1999       0400         280         24        1,332.00  DEM
 12/09/1999       0400         280         198        1,332.00  DEM
 12/29/1999       0400         280         277        1,332.00  DEM
Total                                                  1400          828
 09/29/1999       0402         280         210        1,332.00  DEM
 12/28/1999       0402         280         280        1,332.00  DEM
 01/02/2000       0402         280         280        1,332.00  DEM
 01/05/2000       0402         280         198        1,332.00  DEM
 02/08/2000       0402         280         11        1,332.00  DEM
Total                                                  1400          979

@ = ICON_BW_GIS
```

Optional Exercises

Unit: Data Formatting and Control Level Processing

Topic: Extracts

When you have completed these exercises, you will be able to:

- Name field groups
- Define field groups
- Create extracts
- Perform control level processing with extracts

1-1 Copy or enhance your program Z##LDB2_..., or copy the sample solution, SAPBC405_LDBS_2, to program Z##DAP2_... . Sample solution for exercise: SAPBC405_DAPS_2.

1-1-1 Deactivate the events GET … LATE.

1-1-2 Define the following field groups in the TOP include:

HEADER

CONNECTIONS

FLIGHTS

BOOKINGS

1-1-3 Assign the following fields to the field groups:

HEADER: spfli-carrid, spfli-connid, sflight-fldate, sbook-bookid, sbook-customid

CONNECTIONS: spfli-cityfrom, spfli-airpfrom, spfli-cityto, spfli-airpto

FLIGHTS: sflight-price, sflight-currency, sflight-planetype, sflight-seatsmax, sflight-seatsocc, free_seats

BOOKINGS: sbook-bookid, sbook-customid, sbook-smoker, sbook-luggweight, sbook-wunit.

1-1-4 Fill the extract for the GET events.

1-1-5 Sort the extract according to the sequence of field group HEADER.

1-1-6 Start control level processing and create a list with the following structure:

Line 1: SPFLI-CARRID, SPFLI-CONNID

Line 2: SFLIGHT-FLDATE

Line 3: SPFLI-CITYFROM, SPFLI-AIRPFROM, SPFLI-CITYTO, SPFLI-AIRPTO

Line 4: SFLIGHT-PRICE, SFLIGHT-CURRENCY, SFLIGHT-PLANETYPE, SFLIGHT-SEATSMAX, SFLIGHT-EATSOCC, FREE_SEATS

Line 5: SBOOK-BOOKID, SBOOK-CUSTOMID, SBOOK-SMOKER, BOOK-LUGGWEIGHT, SBOOK-WUNIT.

Only output line 3 if line 4 is also output.

Only output line 4 if line 5 is also output.

1-1-6 The list should also include the number of total bookings and the total weight of the luggage for each flight date in one line.

1-2 Formatting the list

1-2-1 The price and luggage weight should be output with correct format for the respective units.

1-2-2 Output each flight on a new page.

1-2-3 Output a solid line before and after the number of bookings and the total weight.

1-2-4 Use the following control structures and colors:

AT NEW spfli-connid, COL_GROUP INTENSIFIED ON

AT NEW sflight-fldate, COL_HEADING INTENSIFIED ON

AT flights WITH bookings, COL_NORMAL INTENSIFIED ON

AT bookings, COL_NORMAL INTENSIFIED OFF

AT END OF sflight-fldate, COL_TOTAL INTENSIFIED ON.

1-2-5 Position the control levels in the list in such a way that the hierarchy is apparent – for example, line 1 begins further than the left than line 2, and so on. Draw a frame around the list. Maintain the column headers (standard list header).

Unit: Data Formatting and Control Level Processing
Topic: Internal Table

```
*&---------------------------------------------------------------------*
*& Report  SAPBC405_DAPS_1                                             *
*&                                                                     *
*&---------------------------------------------------------------------*
*&   Solution; Exercise 1; Control level processing
                              With internal table
*&                                                                     *
*&---------------------------------------------------------------------*

INCLUDE bc405_daps_1top.

*&---------------------------------------------------------------------*
*&   Event TOP-OF-PAGE
*&---------------------------------------------------------------------*
TOP-OF-PAGE.

* Title
  FORMAT COLOR COL_HEADING INTENSIFIED ON.
  ULINE.
  WRITE: / sy-vline,
         'Flight data'(001),
         AT line_size sy-vline.
  ULINE.

* Column header
  FORMAT COLOR COL_HEADING INTENSIFIED OFF.
  WRITE: sy-vline, 'Departure location'(004),  AT line_size sy-vline.
  WRITE: sy-vline, 'Arrival location'(004),  AT line_size sy-vline.

  WRITE: sy-vline, AT pos_c1 'Flight'(002).
  SET LEFT SCROLL-BOUNDARY.
  WRITE: 'Date'(003),
         AT pos_c3 'Price'(006),
         AT pos_c4 'Max.'(008),
         AT pos_c5 'Occ.'(009),
         AT line_size sy-vline.
  ULINE.


*&---------------------------------------------------------------------*
*&   Event   INITIALIZATION
*&---------------------------------------------------------------------*
INITIALIZATION.                        " OPTIONAL

* Initialize select-options for CARRID
  MOVE: 'AA' TO so_car-low,
        'QF' TO so_car-high,
        'BT' TO so_car-option,
        'I' TO so_car-sign.
  APPEND so_car.
  CLEAR so_car.
  MOVE: 'AZ' TO so_car-low,
        'EQ' TO so_car-option,
        'E' TO so_car-sign.
  APPEND so_car.
  CLEAR so_car.

*&---------------------------------------------------------------------*
*&   Event AT SELECTION-SCREEN ON BLOCK PARAM
*&---------------------------------------------------------------------*
AT SELECTION-SCREEN ON BLOCK param.    " OPTIONAL

* check country for national flights is not empty
  CHECK national = 'X' AND country = space.
  MESSAGE e003(bc405).

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

```
*&   Event START-OF-SELECTION
*&---------------------------------------------------------------------*
START-OF-SELECTION.

* Checking the output parameters
 CASE mark.
  WHEN all.
* Radiobutton ALL is marked
   SELECT * FROM spfli INNER JOIN sflight
     ON spfli~carrid = sflight~carrid
     AND spfli~connid = sflight~connid
    INTO CORRESPONDING FIELDS OF TABLE it_flights
    WHERE spfli~carrid   IN so_car
     AND spfli~connid   IN so_con
     AND sflight~fldate IN so_fdt.



  WHEN national.
* Radiobutton NATIONAL is marked
   SELECT * FROM spfli INNER JOIN sflight
       ON spfli~carrid = sflight~carrid
       AND spfli~connid = sflight~connid
    INTO CORRESPONDING FIELDS OF TABLE it_flights
    WHERE spfli~carrid   IN so_car
     AND spfli~connid   IN so_con
     AND sflight~fldate IN so_fdt
     AND spfli~countryfr = spfli~countryto
     AND spfli~countryfr = country.



  WHEN internat.
* Radiobutton INTERNAT is marked
   SELECT * FROM spfli INNER JOIN sflight
     ON spfli~carrid = sflight~carrid
     AND spfli~connid = sflight~connid
    INTO CORRESPONDING FIELDS OF TABLE it_flights
    WHERE spfli~carrid   IN so_car
```

```
         AND spfli~connid   IN so_con
         AND sflight~fldate IN so_fdt
         AND spfli~countryfr NE spfli~countryto.

  ENDCASE.

* Additional solution: dynamical WHERE condition
* PERFORM get_data.

*&-------------------------------------------------------------------*
*&   Event END-OF-SELECTION
*&-------------------------------------------------------------------*
END-OF-SELECTION.

*SORT it_flights BY carrid connid fldate.

* Control Level Processing: the internal table has to be sorted
  SORT it_flights BY cityfrom cityto carrid connid.


* Data output
  PERFORM data_output.

*&-------------------------------------------------------------------*
*&      Form  DATA_OUTPUT
*&-------------------------------------------------------------------*
*       List output of flight data
*-------------------------------------------------------------------*
FORM data_output.

* Loop at the internal table for writing data
  LOOP AT it_flights INTO wa_flights.

* Group Level:  CITYFROM
    AT NEW cityfrom.
      NEW-PAGE.
      FORMAT COLOR COL_GROUP INTENSIFIED ON.
      WRITE: / sy-vline, wa_flights-cityfrom
```

```
           AT line_size sy-vline.
   FORMAT RESET.
  ENDAT.

* Group Level:  CITYTO
  AT NEW cityto.
   FORMAT COLOR COL_GROUP INTENSIFIED OFF.
   WRITE: / sy-vline, wa_flights-cityto,
        AT line_size sy-vline.
   FORMAT RESET.
  ENDAT.

* Single Record Processing
* Mark international flights
  FORMAT COLOR COL_KEY INTENSIFIED ON.
  IF wa_flights-countryfr EQ wa_flights-countryto.
   WRITE: / sy-vline, icon_space AS ICON CENTERED.
  ELSE.
   WRITE: / sy-vline, icon_bw_gis AS ICON CENTERED.
  ENDIF.
* Data output
  WRITE: wa_flights-carrid,
         wa_flights-connid.
  FORMAT COLOR COL_NORMAL INTENSIFIED OFF.
  WRITE: wa_flights-fldate,
          wa_flights-price CURRENCY wa_flights-currency,
          wa_flights-currency,
          wa_flights-seatsmax,
         wa_flights-seatsocc,
        AT line_size sy-vline.
  FORMAT RESET.

* Group Level:  CONNID
  AT END OF connid.
   SUM.
   FORMAT COLOR COL_TOTAL.
   WRITE: / sy-vline
```

```
        'Total'(007),
        wa_flights-seatsmax UNDER wa_flights-seatsmax,
        wa_flights-seatsocc UNDER wa_flights-seatsocc,
        AT line_size sy-vline.
   FORMAT RESET.
  ENDAT.

* Group Level:  CITYFROM
  AT END OF cityfrom.
   ULINE.
  ENDAT.

 ENDLOOP.

ENDFORM.                    " DATA_OUTPUT

*&---------------------------------------------------------------------*
*& Include BC405_DAPS_1TOP                                             *
*&                                                                     *
*&---------------------------------------------------------------------*

REPORT   bc405_daps_1top LINE-SIZE 100 NO STANDARD PAGE HEADING.

* Include for using icons
INCLUDE <icon>.

* Linetype of internal table
TYPES: BEGIN OF linetype,
     cityfrom LIKE spfli-cityfrom,
     cityto   LIKE spfli-cityto,
     carrid   LIKE spfli-carrid,
     connid LIKE spfli-connid,
     countryfr like spfli-countryfr,
     countryto like spfli-countryto,
     fldate     LIKE sflight-fldate,
     price  LIKE sflight-price,
     currency LIKE sflight-currency
```

```
      seatsmax LIKE sflight-seatsmax,
      seatsocc LIKE sflight-seatsocc,
     end of linetype.

* Constants for writing position
CONSTANTS:  pos_c1 TYPE i VALUE  6,
                pos_c3 TYPE i VALUE 30,
                pos_c4 TYPE i VALUE 58,
               pos_c5 TYPE i VALUE 68,
        line_size TYPE i VALUE 100.

* Constant for CASE statement
CONSTANTS mark VALUE 'X'.

* Internal table like DDIC view DV_FLIGHTS
*  DATA: it_flights LIKE TABLE OF dv_flights,
*         wa_flights LIKE dv_flights.

* Internal table type linetype
DATA: it_flights TYPE STANDARD TABLE OF linetype,
       wa_flights TYPE linetype.

* Selections for connections
SELECTION-SCREEN BEGIN OF BLOCK conn WITH FRAME TITLE text-tl1.
SELECT-OPTIONS: so_car FOR wa_flights-carrid,
          so_con FOR wa_flights-connid.
SELECTION-SCREEN END OF BLOCK conn.

* Selections for flights
SELECTION-SCREEN BEGIN OF BLOCK flight WITH FRAME TITLE text-tl2.
SELECT-OPTIONS so_fdt FOR wa_flights-fldate NO-EXTENSION.
SELECTION-SCREEN END OF BLOCK flight.

* Output parameter
SELECTION-SCREEN BEGIN OF BLOCK param
              WITH FRAME TITLE text-tl3.
SELECTION-SCREEN BEGIN OF BLOCK radio WITH FRAME
```

```
PARAMETERS: all RADIOBUTTON GROUP rbg1,
            national RADIOBUTTON GROUP rbg1,
            internat RADIOBUTTON GROUP rbg1 DEFAULT 'X'.
SELECTION-SCREEN END OF BLOCK radio.
PARAMETERS country LIKE wa_flights-countryfr.
SELECTION-SCREEN END OF BLOCK param.
```

Optional Solutions

Unit: Data Formatting and Control Level Processing

Topic: Extracts

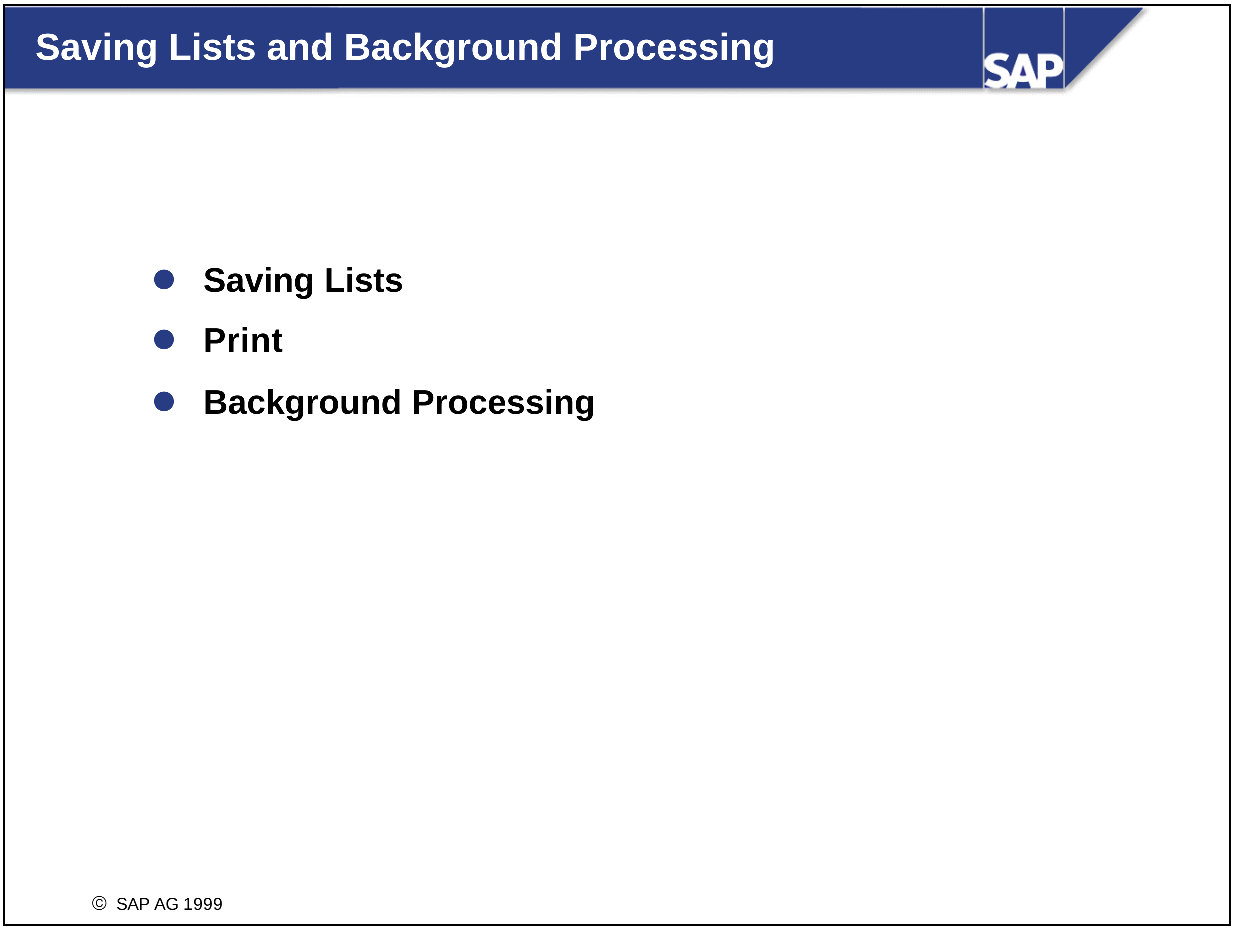
```
*&------------------------------------------------------ -----------*
*& Report  SAPBC405_DAPS_2                                *
*&                                                       *
*&-----------------------------------------------------------------*
*&                                                       *
*&                                                       *
*&-----------------------------------------------------------------*

INCLUDE bc405_daps_2top.

*&-----------------------------------------------------------------*
*&   Event GET SPFLI
*&-----------------------------------------------------------------*
GET spfli.
* Save field group:  connections
  EXTRACT connections.

*&-----------------------------------------------------------------*
*&   Event GET SFLIGHT
*&-----------------------------------------------------------------*
GET sflight.
* Calculate free seats
  free_seats = sflight-seatsmax - sflight-seatsocc.

* Save field group:  flights
  EXTRACT flights.

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

```
*&   Event GET SBOOK
*&---------------------------------------------------------------------*
GET sbook.

* Check select-option
  CHECK so_odat.

* Save field group:  bookings
  EXTRACT bookings.

*&---------------------------------------------------------------------*
*&   Event GET SPFLI LATE
*&---------------------------------------------------------------------*
*GET spfli LATE.
*  ULINE.
*  NEW-PAGE.
*&---------------------------------------------------------------------*
*&   Event GET SFLIGHT LATE
*&---------------------------------------------------------------------*
*GET sflight LATE.
*  ULINE.

*&---------------------------------------------------------------------*
*&   Event END-OF-SELECTION
*&---------------------------------------------------------------------*
END-OF-SELECTION.

* Sorting extract data according to the header fields
  SORT.

* Control level processing
  LOOP.
    AT NEW spfli-connid.
      FORMAT COLOR COL_GROUP INTENSIFIED ON.
      WRITE: / sy-vline,
               spfli-carrid,
               spfli-connid
```

```
        AT line_size sy-vline.
  ENDAT.

  AT NEW sflight-fldate.
   FORMAT COLOR COL_HEADING INTENSIFIED ON.
   WRITE: / sy-vline,
        AT pos_lev2 sflight-fldate,
        AT line_size sy-vline.
  ENDAT.

* Single record processing
  AT connections WITH flights.
   FORMAT COLOR COL_HEADING INTENSIFIED OFF.
   WRITE: / sy-vline,
        spfli-cityfrom,
        spfli-airpfrom,
        spfli-cityto,
        spfli-airpto, AT line_size sy-vline.
  ENDAT.

  AT flights WITH bookings.
   FORMAT COLOR COL_NORMAL INTENSIFIED ON.
   WRITE:  / sy-vline,
       sflight-price CURRENCY sflight-currency,
       sflight-currency,
       sflight-planetype,
       sflight-seatsmax,
       sflight-seatsocc,
       free_seats, AT line_size sy-vline.
  ENDAT.

  AT bookings.
   FORMAT COLOR COL_NORMAL INTENSIFIED OFF.
   WRITE: / sy-vline,
        AT pos_lev3 sbook-bookid,
        sbook-customid,
        sbook-smoker
       sbook-luggweight UNIT sbook-wunit,
       sbook-wunit, AT line_size sy-vline.
  ENDAT.

* Control level processing with CNT and SUM
  AT END OF sflight-fldate.
   FORMAT COLOR COL_TOTAL INTENSIFIED ON.
   ULINE.
   WRITE: sy-vline,
      'Totals:'(001),
      cnt(sbook-bookid) UNDER sbook-bookid,
      sum(sbook-luggweight) UNIT sbook-wunit
      UNDER sbook-luggweight,
      sbook-wunit,
      AT line_size sy-vline.
   ULINE.
  ENDAT.
 ENDLOOP.
```

```
*&---------------------------------------------------------------------*
*& Include BC405_DAPS_2TOP
*
*&                                                                     *
*&---------------------------------------------------------------------*

REPORT   bc405_daps_2top LINE-SIZE 83.

* Used nodes of the structure of the logical database F1S
NODES: spfli, sflight, sbook.

* Additional selections
SELECTION-SCREEN BEGIN OF BLOCK order WITH FRAME.
SELECT-OPTIONS: so_odat FOR sbook-order_date.
SELECTION-SCREEN END OF BLOCK order.

* Variables
```

```
DATA: free_seats LIKE sflight-seatsocc.

* Constants
CONSTANTS: line_size LIKE sy-linsz VALUE 83,
       pos_lev2 TYPE i VALUE 10,
       pos_lev3 TYPE i VALUE 20.

* Field groups
FIELD-GROUPS: header,              " Group Level and sorting fields
         connections,         " Fields of SPFLI
         flights,             " Fields of SFLIGHT
         bookings.            " Fields of SBOOK

* Fixing of field groups
INSERT:    spfli-carrid
        spfli-connid
        sflight-fldate
        sbook-bookid
        sbook-customid   INTO header,
        spfli-cityfrom
        spfli-airpfrom
        spfli-cityto
        spfli-airpto         INTO connections,
        sflight-price
        sflight-currency
        sflight-planetype
        sflight-seatsmax
        sflight-seatsocc
        free_seats           INTO flights,
        sbook-bookid
        sbook-customid
        sbook-smoker
        sbook-luggweight
        sbook-wunit          INTO bookings.
```



Saving Lists and Background Processing

- Saving Lists
- Print
- Background Processing

- There are three ways to save a list that you have generated:

  1.) In SAPoffice
  2.) As a local file on your PC
  3.) In the area menu

- You can always use menu sequence STL-1 in the *System* menu to save the list, or use the *List* menu in the standard list status.

- You can also create folders in SAPoffice. You can use these folders to store the lists. When a list is placed in the outbox of the personal folders, it can also be sent to other users.

- You can save a list to a PC as a local file in four different formats:

    1.) Unconverted (ASCII)

    2.) Spreadsheet format (-> Microsoft Excel)

    3.) RTF (Rich Text Format -> Microsoft Word)

    4.) HTML (Web Browser)

- You can use the program RSSOPCDR to specify the default file that the system proposes when the user chooses to save a file to the local PC.

- The area menus have been converted to tree navigation in Release 4.6A. Type 1 programs and SAP queries can now be added to the area menus in addition to the previously contained transactions. Any programs that do not have a transaction code are allocated one automatically.
- The report trees have been integrated in the area menus in Release 4.6. The report trees are now maintained using the maintenance tools for area menus. You can maintain area menus in the Workbench menu path STL-1.
- Saved lists are saved with the program itself. If the program has been integrated in the area menu, then the saved lists will also appear there. You can also use standard program RSRSSLIS to display saved lists.
- A user can access an area menu whenever that area menu has been allocated to an activity group to which the user belongs. You can use the profile generator to allocate an area menu to an activity group.

- There are 4 options for printing a list:

  1) From the selection screen

  - The list is printed when it is generated (adjusted to print format) and does not appear on the screen.
  - The list is generated in a dialog work process

  2) From within the program

  - The first two points of 1, above
  - This procedure is suitable for interactive lists: printing details lists

  3) After the list has been generated

  - The list has already been generated (visible on the screen) and can be formatted within limits. For example, the number of columns in the list cannot be changed after the list is generated.
  - The list is generated in a dialog work process

  4) In the background

  - The list can be printed after it has been generated (as described in 1 above)
  - The list is generated in a background work process. This procedure is particularly suitable for long lists, since it does not block a dialog work process during processing.

- To print a list, you must enter print parameters. The print parameters control list output and are divided into the following areas:

  1) Output device and number of copies

  2) Spool request

  3) Spool control

  4) Cover sheets

  5) Output format

- You can enter print parameters on the screen or set them directly in the program. Setting print parameters in the program is treated below (**NEW-PAGE PRINT ON**).

- You can use the function module **SET_PRINT_PARAMETERS** to set default values for printing an online list; you can execute print from the selection screen or after generation of the list.

- **NEW-PAGE PRINT ON** triggers a page break, and all the subsequent output is redirected to the spool.
- The print parameters can either be passed on to the system as a structure with the **PARAMETERS** attribute or - as shown in the above example - specified individually.
- Individual entry of print parameters is not recommended. Consider the case where the user arrives at the print parameter screen and decides not to print; the only option in this case would be to terminate the entire program.
- In contrast, if you use the **PARAMETERS** attribute, the user can cancel printing without having to terminate the program.
- If you enter parameter **NO DIALOG,** the list is placed directly in the spool without giving the user any opportunity to change the print parameters at runtime.
  If you do not enter **NO DIALOG**, the user is presented with a print parameters screen containing default values at runtime.
- **NEW-PAGE PRINT OFF** triggers a page break, ends the spool request (sy-spono is assigned), and all subsequent output is once again output on the screen.

- The structure for the **PARAMETERS** attribute of the **NEW-PAGE PRINT ON** statement **must** be filled using function module **GET_PRINT_PARAMETERS.** The structure contains an internal checksum that is calculated by **NEW-PAGE PRINT ON**. If the checksum is incorrect, the program terminates. Function module GET_PRINT_PARAMETERS calculates the checksum and returns it with out_parameters.
- Function module GET_PRINT_PARAMETERS provides users with a print parameters screen that can be used to modify the print parameters and then determine a complete new set of print parameters. The set is returned using output parameter out_parameters. In successful cases, output parameter "valid" contains the value 'X'. If the system cannot create a complete set record of print parameters, the structure transferred with out_parameters is empty and valid contains the value "space".
- You can transfer print parameters to the function module GET_PRINT_PARAMETERS. The print parameters appear as default values in the print parameters screen.
- The print parameters screen of function module GET_PRINT_PARAMETERS offers the option of canceling the filling of print parameters. In this case, the structure transferred with out_parameters is empty and valid contains the value "space".

- One application could be to send a list to several recipients.
- This has been implemented in the above example. To send a list to several recipients, you have to distribute it among several spool requests. To do this, you use parameter NEW_LIST_ID and then **NEW-PAGE PRINT OFF** to end the spool request.

- When a program converts large datasets and requires a long runtime, it makes sense to start it in the background.
- Background runs take place without user dialogs, and can take place in parallel to online operations. The dialog work processes are available for online processing. Background job runs are performed by special work processes (background processes), which enables distributed processing.
- To start a program in the background, you must first add it to a job.
- Use the job definition to determine which programs (steps) will run during this job. You can specify print parameters and set the start time for the job.
- The job overview tells you the current status of the job.

- *Define job* is located under menu path STL-2. First assign a name (of your choice) and define the priority (job class) and the destination (F4 help).
- Then determine the individual steps of the job. If you want the program to run with a selection screen, you also have to specify a variant. The list can be stored in the spool or printed immediately. This depends on the specified print parameters. When you have defined all the steps, save them and return to the initial screen of the job definition.

- Once you have defined the steps, you can determine the start date for the job. For example, you can start the job on a certain day at a certain time.
- Once you have defined the start date, save your entries and return to the initial screen of the job definition. Now save the job, which releases it to run at the specified time.
- You can also use the automated job scheduling with the function modules in function groups BTCH and BTC2. An example is available in program SAPBC405STLD_E_JOB.

Unit: Saving Lists and Background Processing
Topic: Program-Controlled Printing

When you have completed these exercises, you will be able to:

- Print using function module GET_PRINT_PARAMTERS

1-1 Copy or enhance your program Z##DAP2_..., or copy the sample solution, SAPBC405_DAPS_2, to program Z##STL1_... . Sample solution for exercise: SAPBC405_STLS_1.

1-1-1 Enhance the program with the functionality of storing the generated list in the spool. To do this, use function module GET_PRINT_PARAMETERS. Use the pattern functions available in the ABAP Editor to program the function call. Pass the following values on in the interface:

EXPORTING: copies = 2, destination = 'LP01', expiration = 3, immediately = space, line_size = 83, list_text = text-xxx (text element), no_dialog = space, release = 'X', report = 'EXAMPLE'

IMPORTING: out_parameters = print_parameter, valid = valid

1-1-2 In the TOP include, create variable print_parameter as a character field with length 1, in accordance with DDIC structure pri_params and variable valid.

1-2-1 Evaluate the return value of valid after the function call. If valid is not equal to space, store the list in the spool and suppress the print dialog. In addition, display information message 104 from message class BCTRAIN. If valid is equal to space, output information message 105 from message class BC405.

Unit: Saving Lists and Background Processing

Topic: Program-Controlled Printing

```
*&---------------------------------------------------------------------*
*& Report  SAPBC405_STLS_1                                             *
*&                                                                     *
*&---------------------------------------------------------------------*
*&                                                                     *
*&                                                                     *
*&---------------------------------------------------------------------*

INCLUDE BC405_STLS_1TOP.


*&---------------------------------------------------------------------*
*&   Event GET SPFLI
*&---------------------------------------------------------------------*
GET spfli.
* Save field group:  connections
  EXTRACT connections.

*&---------------------------------------------------------------------*
*&   Event GET SFLIGHT
*&---------------------------------------------------------------------*
GET sflight.
* Calculate free seats
  free_seats = sflight-seatsmax - sflight-seatsocc.

* Save field group:  flights
  EXTRACT flights.

*&---------------------------------------------------------------------*
*&   Event GET SBOOK
```

```
GET sbook.

* Check select-option
  CHECK so_odat.

* Save field group:  bookings
  EXTRACT bookings.

*&-------------------------------------------------------------------*
*&   Event GET SPFLI LATE
*&-------------------------------------------------------------------*
*GET spfli LATE.
*  ULINE.
*  NEW-PAGE.
*&-------------------------------------------------------------------*
*&   Event GET SFLIGHT LATE
*&-------------------------------------------------------------------*
*GET sflight LATE.
*  ULINE.

*&-------------------------------------------------------------------*
*&   Event END-OF-SELECTION
*&-------------------------------------------------------------------*
END-OF-SELECTION.

* Define print parameters via function module
CALL FUNCTION 'GET_PRINT_PARAMETERS'
  EXPORTING
       COPIES            =  2        " Number of copies
       DESTINATION       =  'LP01'    " Printer
       EXPIRATION        =  3        " Duration /d in spool
       IMMEDIATELY       =  SPACE     " print immediately
       LINE_SIZE         =  100       " Width of list
       LIST_TEXT         =  TEXT-SP1  " Title
       NO_DIALOG         =  SPACE     " Dialog is suppressed
       RELEASE           =  'X'       " delete task after print
       REPORT            =  'EXAMPLE' " Name
  IMPORTING
      OUT_PARAMETERS        = PRINT_PARAMETER
      VALID                 = VALID.

* Sending list to the SAP spool or on the screen
IF VALID NE SPACE.     " List to spool
 NEW-PAGE PRINT ON PARAMETERS PRINT_PARAMETER NO DIALOG.
  MESSAGE I653(BCTRAIN).
 ELSE.                 " List on screen
  MESSAGE I654(BCTRAIN).
ENDIF.

* Sorting extract data according to the header fields
  SORT.

* Control level processing
  LOOP.
    AT NEW spfli-connid.
      FORMAT COLOR COL_GROUP INTENSIFIED ON.
      WRITE: / sy-vline,
               spfli-carrid,
               spfli-connid,
               AT line_size sy-vline.
    ENDAT.

    AT NEW sflight-fldate.
      FORMAT COLOR COL_HEADING INTENSIFIED ON.
      WRITE: / sy-vline,
               AT pos_lev2 sflight-fldate,
               AT line_size sy-vline.
    ENDAT.

* Single record processing
    AT connections WITH flights.
      FORMAT COLOR COL_HEADING INTENSIFIED OFF.
      WRITE: / sy-vline,
               spfli-cityfrom
```

```
        spfli-airpfrom,
        spfli-cityto,
        spfli-airpto, AT line_size sy-vline.
  ENDAT.

  AT flights WITH bookings.
   FORMAT COLOR COL_NORMAL INTENSIFIED ON.
   WRITE:  / sy-vline,
       sflight-price CURRENCY sflight-currency,
       sflight-currency,
       sflight-planetype,
       sflight-seatsmax,
       sflight-seatsocc,
       free_seats, AT line_size sy-vline.
  ENDAT.

  AT bookings.
   FORMAT COLOR COL_NORMAL INTENSIFIED OFF.
   WRITE: / sy-vline,
        AT pos_lev3 sbook-bookid,
        sbook-customid,
        sbook-smoker,
        sbook-luggweight UNIT sbook-wunit,
        sbook-wunit, AT line_size sy-vline.
  ENDAT.

* Control level processing with CNT and SUM
  AT END OF sflight-fldate.
   FORMAT COLOR COL_TOTAL INTENSIFIED ON.
   ULINE.
   WRITE: sy-vline,
       'Totals:'(001),
       cnt(sbook-bookid) UNDER sbook-bookid,
       sum(sbook-luggweight) UNIT sbook-wunit
       UNDER sbook-luggweight,
       sbook-wunit,
       AT line_size sy-vline.
```

```
    ULINE.
   ENDAT.
  ENDLOOP.

*&---------------------------------------------------------------------*
*& Include BC405_STLS_1TOP                                             *
*&                                                                     *
*&---------------------------------------------------------------------*

REPORT   sapbc405_stls_1 LINE-SIZE 83.

* Used nodes of the structure of the logical database F1S
NODES: spfli, sflight, sbook.

* Additional selections
SELECTION-SCREEN BEGIN OF BLOCK order WITH FRAME.
SELECT-OPTIONS: so_odat FOR sbook-order_date.
SELECTION-SCREEN END OF BLOCK order.

* Variables
DATA: free_seats LIKE sflight-seatsocc.
DATA: print_parameter LIKE pri_params, " NEW-PAGE PRINT ON .....
    valid VALUE 'X'.

* Constants
CONSTANTS: line_size LIKE sy-linsz VALUE 83,
           pos_lev2 TYPE i VALUE 10,
           pos_lev3 TYPE i VALUE 20.

* Field groups
FIELD-GROUPS: header,              " Group Level and sorting fields
              connections,         " Fields of SPFLI
              flights,             " Fields of SFLIGHT
              bookings.            " Fields of SBOOK

* Fixing of field groups
INSERT:     spfli-carrid
```

```
spfli-connid
sflight-fldate
sbook-bookid
sbook-customid   INTO header,
spfli-cityfrom
spfli-airpfrom
spfli-cityto
spfli-airpto           INTO connections,
sflight-price
sflight-currency
sflight-planetype
sflight-seatsmax
sflight-seatsocc
free_seats              INTO flights,
sbook-bookid
sbook-customid
sbook-smoker
sbook-luggweight
sbook-wunit          INTO bookings.
```

ALV Grid Control

Contents:

- ALV Grid Control - Standard Application
- Preview of Other Techniques

- This task is performed by the SAP Control Framework.
- The R/3 System allows you to create custom controls using ABAP Objects. The application server is the Automation Client, which drives the custom controls (automation server) at the frontend.
- If custom controls are to be included on the frontend, then the SAPGUI acts as a container for them. Custom controls can be ActiveX Controls or JavaBeans.
- The system has to use a Remote Function Call (RFC) to transfer methods for creating and using a control to the front end.
- ABAP objects are used to implement the controls in programs.

- An SAP Container can contain other controls (for example, SAP ALV Grid Control, Tree Control, SAP Picture Control, SAP Splitter Control, and so on). It administers these controls logically in one collection and provides a physical area for the display.

- Every control exists in a container. Since containers are themselves controls, they can be nested within one another. The container becomes the parent of its control. SAP containers are divided into five groups:

  SAP custom container: Displays within an area defined in Screen Painter on screens or subscreens. Class: CL_GUI_CUSTOM_CONTAINER

  SAP dialog box container: Displays in a modeless dialog box or as a full screen. Class: CL_GUI_DIALOGBOX_CONTAINER

  SAP docking container: Displays as docked, resizable sub-window with the option of displaying it as a modeless dialog box. Class: CL_GUI_DOCKING_CONTAINER

  SAP splitter container: Displays and groups several controls in one area - that is, splits the area into cells Class: CL_GUI_SPLITTER_CONTAINER

  SAP easy splitter container: Displays controls in two cells, which the user can resize using a split bar. Class: CL_GUI_EASY_SPLITTER_CONTAINER.

- In the control, you can adjust the column width by dragging, or use the 'Optimum width' function to adjust the column width to the data currently displayed. You can also change the column sequence by selecting a column and dragging it to a new position.
- Standard functions are available in the control toolbar. The details display displays the fields in the line on which the cursor is positioned in a modal dialog box.
- The sort function in the ALV Control is available for as many columns as required. You can set complex sort criteria and sort columns in either ascending or descending order.
- You can use the 'Search' function to search for a string (generic search without *) within a selected area by line or column.
- You can use the 'Sum' function to create totals for one or more numeric columns. You can then use the "Subtotals" function to set up control level lists: You can use the 'Subtotal' function to structure control level lists: select the columns (non-numeric columns only) that you want to use and the corresponding control level totals are displayed.
- For 'Print' and 'Download' the whole list is always processed, not just the sections displayed on the screen.
- You can define display variants to meet your own specific requirements. For information on saving variants, see 'Advanced Techniques'.

- The ALV grid control is a generic tool for displaying lists in screens. The control offers standard functions such as sorting by any column, adding numeric columns, and fixed lead columns
- Data collection is performed in the program (with SELECT statements, for example) or by using a logical database. The data records are saved in an internal table and passed on to the ALV control along with a field description.
- The field description contains information about the characteristics of each column, such as the column header and output length. This information can defined either globally in the Dictionary (structure in the Dictionary) or in the field catalog in the program itself. You can also merge both techniques.
- The ALV link is a standard function of Query and QuickViewer. If multiline queries or QuickView lists have been defined, they will automatically be compressed to a single line and output in the ALV control as a long, single line list.

- Use Screen Painter to create a subscreen container for the ALV grid control. The control requires an area where it can be displayed in the screen. You have to create a container control that determines this area.
- Use the corresponding icon in the Screen Painter layout to create the container control. The size of area "MY_CONTROL_AREA" determines the subsequent size of the ALV control.
- The valid GUI status must be set at the PBO event in the flow logic of the ALV subscreen container. The OK_CODE processing for the cancel functions must be programmed at the PAI event.

- The reference variables for the custom container and the ALV grid control must be declared.
- To create reference variables, use ABAP statement **TYPE REF TO <class name>**.
- The global classes you need to do this are called **cl_gui_custom_container** (for the custom container control) and **cl_gui_alv_grid** (for the ALV grid control).
- The global classes are defined in the Class Builder. You can use the Class Builder to display information for the methods, their parameters, exceptions, and so on.

- Use ABAP statement **CREATE OBJECT <name>** to create the objects for the container and the ALV control. Objects Are instances of a class.
- When an object is created (**CREATE**), method CONSTRUCTOR of the corresponding class is executed. The parameters of method CONSTRUCTOR determine which parameters have to be supplied with data when the object is created. In the above example, object **`alv_grid`** is given the name of the container control (**`g_custom_container`**) in exporting parameter **`i_parent`**, which links the two controls. For information on which parameters method CONSTRUCTOR possesses and which of these parameters are required, see the Class Builder.
- Objects should only be created once during the program. To ensure that this is the case, enclose the CREATE OBJECT statement(s) in an **`IF <object_name> IS INITIAL. ... ENDIF`** clause. The objects must be generated before the control is displayed for the first time - that is, during the PBO event of the ALV subscreen container.

- To display the requested dataset in the ALV control, the data must be passed on to the control as an internal table, and a field description must exist indicating the order in which the columns will be output.
- In the simplest case, the field description can use a structure from the Dictionary. The Dictionary also determines the technical field attributes like type and length, as well as the semantic attributes like short and long texts. The ALV control uses this information to determine the column widths and headers. The column sequence is determined by the field sequence in the structure.
- If no suitable structure is active in the Dictionary, or you want to output internal program fields in the control, then you will have to define information like the output length and column header in the field catalog.
- In a typical program run, the dataset is read first (**SELECT** ....), the internal table is filled with the data to display (**... INTO TABLE ...**), and ABAP statement **CALL SCREEN <number>** is then used to call the ALV subscreen container.

- The data transfer to the ALV control takes place during the call of method set_table_for_first_display from class cl_gui_alv_grid. The method call must be programmed at the PBO event of the ALV subscreen container.
- The name of the Dictionary structure that supplies the field description is specified in exporting parameter i_structure_name. The name of the internal table that contains the data records to display is specified in changing parameter it_outtab.

- The field description for the ALV control can be taken from an active Dictionary structure (fully automatic), by passing a field catalog (manual), or through a mixture of the two options (merge).
- The field catalog is in internal table with type **lvc_t_fcat**. This type is defined globally in the Dictionary.
- Each **line** in the field catalog table corresponds to a **column** in the ALV control.

- The field characteristics (= column characteristics) are defined in the field catalog. The field catalog is in internal table with type lvc_t_fcat. Each line that is explicitly described in the ALV control corresponds to a column in the field catalog table.
- The link to the data records to output that are saved in internal table <outtab> is established through field name <outtab-field>. This name must be specified in column "fieldname" in the field catalog.
- This field can be classified through a Dictionary reference (ref_table and ref_field) or by specifying an ABAP data type (inttype).
- Column headers and field names in the detail view of an ALV control line can be determined in the field catalog in coltext and seltext, respectively.
- The position of a field during output can be determined with col_pos in the field catalog.
- If you want to hide a column, fill field no_out with an "X" in the field catalog. Hidden fields can be displayed again in a user display variant.
- Icons can be displayed in the ALV control. If you want a column to be interpreted as an icon, then the icon name must be known to the program (**include <icon>.**) and icon = "X" must be specified for this column in the field catalog.

- The above example shows a semi-automatic field description: Part of the field description comes from the Dictionary structure (sflight), while another part is explicitly defined in the field catalog (gt_fieldcat).
- The field catalog (internal table) is filled in the program and is passed on together with the name of the Dictionary structure during the method call. The information is merged accordingly in method set_table_for_first_display.

- For a user to save display variants, parameters **`is_variant`** and **`i_save`** must be passed on during method call **`set_table_for_first_screen.`** To assign display variants uniquely to a program, at least the program name must be supplied in the transferred structure (**`gs_variant`**). Program names can be up to 30 characters long.
- If you only pass on the current parameters for **`is_variant`**, then existing variants can be loaded, but no new ones can be saved. If you use parameter **`i_save`**, you must pass on a variant structure with **`is_variant`**.
- **I_SAVE = SPACE** No variants can be saved.
- **I_SAVE = 'U'** The user can only save user-specific variants.
- **I_SAVE = 'X'** The user can only save general (shared) variants.
- **I_SAVE = 'A'** The user can save both user-specific and general (shared) variants.

- You can use parameter is_layout of method set_table_for_first_display, for example, to define the header in the ALV control and the detail display.
- To do this, define a query area <gs_layout> in the program in accordance with Dictionary structure lvc_s_layo, and pass on the text to display in field <gs_layout>-grid_title or <gs_layout>-detailtitl.
- If you want to create print lists with zebra stripes, set field <gs_layout>-zebra to "X". You can display a print preview for print lists by requesting standard function "Print".

- All parameters of method SET_TABLE_FOR_FIRST_DISPLAY from global class CL_GUI_ALV_GRID are defined in the Class Builder.

- **Events** are defined in global class `cl_gui_alv_grid`; you can use these events to implement user interaction within the program. To respond to a double-click on a table line, you must respond to event DOUBLE_CLICK.
- You receive control in the program, allowing you to implement interactive reporting - such as a full-screen details list. The events for `cl_gui_alv_grid` are located in the Class Builder.

- To define an implement a local class in the program, you use a handler method. In this handler method, you program the functionality to trigger by a double-click in the output table.
- To activate a handler method at runtime, a class or an object from that class registers itself with an event using command **SET HANDLER**. The names of the IMPORTING parameters in the handler method correspond to the names of the EXPORTING parameters of the related event.
- In the above example, the local class is LCL_ILS and the handler method is ON_DBLCLICK. An object - ALV_DBLCLICK - is created and registers itself for event DOUBLE_CLICK.
- You can query parameter e_row-index to determine which output line was requested by the double-click. This parameter corresponds to the line number of the output table (internal table with the data records to output). If you need information for the selected line, you have to read it with **READ TABLE itab INDEX e_row-index**.
- This subsequent read in the output table generally corresponds to the HIDE area in conventional reporting. You first have to make sure that the user has double-clicked a line in the output table (similar to the valid line selection with the HIDE technique).

Appendix

- Additional slides
- Linking programs
- Menu paths

- A field group can contain global data objects, but not data objects that have been defined locally in a subroutine or function module.
- You can use **INSERT** to specify both fields and field symbols. This makes it possible to dynamically insert a data object referred to by a field symbol into a field group at runtime. Any field symbols that have not been assigned are ignored, which means no new field is inserted into the field group.
- The **EXTRACT** statement writes all the fields of a field group as one record to a sequential dataset (transport takes place with similarly named fields). If a **HEADER** field group is defined, then its fields are placed ahead of each record as sort keys. You can then sort the dataset with **SORT** and process it with **LOOP ...ENDLOOP**. In this case, no further **EXTRACT** is possible.

- The INSERT statement is not a declarative statement: This means field groups can also be expanded in the program flow section.
- As soon as the first dataset of a field group has been extracted with **`EXTRACT`**, that field group can no longer be expanded with **`INSERT`**. In particular, the HEADER field group cannot be expanded after the first **`EXTRACT`** (regardless of the field group).
- When the **`GET`** events are processed, the logical database automatically writes hexadecimal zeros in all the fields of a node when it returns to an upper-level node in the hierarchy. Since the HEADER normally contains sort fields for all field groups, these hexadecimal zeros in the HEADER serve as a type of hierarchy key: The more zeros there are, the further up in the control level hierarchy you go.

- The **SORT** statement sorts the extract dataset in accordance with the defined field sequence in field group HEADER. The addition **BY <f1> <f2> …** sets a new sort key. Each <fi> must be either a field of field group HEADER or a field group that consists only of fields of the field group HEADER. You can use the additions **ASCENDING** and **DESCENDING** to determine whether the fields are sorted in ascending (default) or descending order.
- Fields containing X'00' in the logical databases are always displayed before all other values during a **SORT**.
- Processing of an extract dataset always takes places within a **LOOP**. The contents of the extract dataset field are placed in program fields with the same names.
- The group change always involves the fields of the HEADER. Single record processing for extract datasets is performed using language element **AT <fg>** (<fg> = field group).
- CNT(<hf>) is not a statement, but instead a field that is automatically create d and filled when <hf> is a non-numeric field from field group HEADER and is part of the sort key. At the end of the group, CNT(<hf>) contains the number of different values that the field <hf> recorded in this group level.
- SUM(<nf>) is not a statement, but instead a field that is automatically created and filled when <nf> is a numeric field of an extract dataset. At the end of the group, SUM(<nf>) contains the control total of field <nf>.
- CUM and CNT are only available at the end of the group level or at **AT LAST**.

- Single record processing for extract datasets **AT <fg_1> WITH <fg_2>** is only performed when field group <fg_1> is immediately followed by field group <fg_2> in the temporary dataset.
- Loops over an extract dataset cannot be nested. However, several contiguous loops are permitted.

- The sequence of the control level changes within the **LOOP** must correspond to the sort sequence.
- Totals can only be calculated within control footer processing.

- Extracts allow only appends (**EXTRACT**), sorting (**SORT**) and sequential processing (**LOOP**). Once a **SORT** or **LOOP** has occurred, the intermediate dataset is frozen and cannot be expanded with **EXTRACT**. Operations that insert into or delete from EXTRACT datasets are not supported.
- Extracts allow for several record types (**FIELD-GROUPS**) with fields that can be set dynamically (**INSERT** is not a declarative statement!). Internal tables have a single, statically-defined line type.
- Internal tables use the sequence of table fields according to the declaration for the hierarchy of the control leve l. The control level structure for internal tables is therefore static, and is independent of which criteria were used to sort the internal table. Extracts do not depend on the field sequence for control level processing: a re-sort or a completely different control level process can take place. The control level structure for extract datasets is therefore dynamic. It corresponds exactly to the sort key of the extract dataset. The sort key is the sequence of fields from the field group HEADER, and is used to sort the extract dataset.
- Extracts rely on the compiler to determine which combinations of group levels and a cumulating field the control level totals desire. The desired control level totals are determined by the processing of **LOOP ... ENDLOOP** blocks. Internal tables build the control level total with the SUM statement. This procedure leads to high resource depletion for totaling control levels in internal tables.

Section: Transaction Programming

Course Overview

Contents:

- Course Goals
- Course Objectives
- Course Content
- Overview Diagram
- Main Business Scenario

■ Unit 1	**Course Overview**
■ Unit 2	Basics for Interactive Lists
■ Unit 3	**The Program Interface**
■ Unit 4	**Interactive List Techniques**
■ Unit 5	Introduction to Screen Programming
■ Unit 6	Screen Elements for Output
■ Unit 7	Screen Elements for Input/Output
■ Unit 8	Screen Elements: Subscreens and Tabstrip Controls
■ Unit 9	Screen Elements: Table Controls
■ Unit 10	Context Menus on Screens
■ Unit 11	Lists in Screen Programming
■ Unit 12	Preview: Control Framework

- **Executable program** (type 1)
  Executable programs can be run directly from the ABAP Editor. A set of processing blocks is processed in a predefined order. You can use a standard selection screen. Type 1 programs normally create and display a list.
- **Module pool** (type M)
  In order for a type M program to be executable, you **must** create at least one transaction code for it (in which you specify an initial screen). You can control the subsequent screen sequence either statically (in the screen attributes) or dynamically (in the program code).
- The following types of programs **cannot** be executed directly. They serve as "containers" for modularization units, which you can call from other programs. Whenever you load one of these modules, the system loads its entire main program into the internal session of the calling program.
  - **Function group** (type F)
    A function group can contain function modules, local data declarations, and screens.
  - **Include program** (type I)
    An include program can contain any ABAP statements.
  - **Interface pool** (type J)
    An interface pool can contain global interfaces and local data declarations.
  - **Class pool** (type J)
    A class pool can contain global classes and local data declarations.

- In the simplest case, your program will consist of a single source that contains all the necessary processing blocks. However, to make your program code easier to understand, and to enable you to reuse parts of it in other programs (for example, for data declarations), you should use include programs
- Whenever you create a program from the Object Navigator, the system proposes to create it "**With TOP include** ". Selecting this option will help you to create clearly-structured programs.
- When you create processing blocks, the system automatically asks in which include program it should place the corresponding source code.
- If you specify an include program that does not yet exist, the system creates it and inserts a corresponding **INCLUDE** statement in the main program.

Basics for Interactive Lists

Contents:

- Creating lists
- Selection screens
- Events
- User dialogs on lists
- Using the hide technique to pass data

■ Unit 1	Course Overview
■ **Unit 2**	**Basics for Interactive Lists**
■ Unit 3	The Program Interface
■ Unit 4	Interactive List Techniques
■ Unit 5	Introduction to Screen Programming
■ Unit 6	Screen Elements for Output
■ Unit 7	Screen Elements for Input/Output
■ Unit 8	Screen Elements: Sunscreens and Tabstrip Controls
■ Unit 9	Screen Elements: Table Controls
■ Unit 10	Context Menus on Screens
■ Unit 11	Lists in Screen Programming
■ Unit 12	Preview: Control Framework

- When the user starts an executable (type 1) program, the program context and memory space for data objects (variables and structures) are made available on the application server. The subsequent program flow is controlled by the ABAP runtime system.
- If the program contains a selection screen, the ABAP runtime system sends it to the presentation server at the start of the program.
- Once the user has finished entering data on the selection screen, he or she chooses 'Execute' to tell the system to start processing the rest of the program. The data entered on the selection screen is automatically placed in the corresponding data objects. The ABAP runtime system takes over control of the program.
- In this simple example, there is only one ABAP processing block to be processed by the runtime system.
- This processing block contains a read access to the database. The program sends information to the database about the records that should be read.
- The database returns the required database records and the runtime system ensures that the data is placed in the relevant data objects.
- The list output is also programmed in the processing block. After the processing block finishes, the runtime system sends the list as a screen to the presentation server.

- Selection screens allow users to enter ranges of values. They are normally used to define the set of data that needs to be read from the database.
- As well as the normal graphical elements (group boxes, checkboxes, radio buttons, and so on) that you use in screens, selection screens also have input/output fields (PARAMETERS) and special groups of input/output fields (SELECT-OPTIONS).
- You place a single input/output field on the selection screen using the PARAMETERS statement.
- You can use the SELECT-OPTIONS statement to place a group of fields on the screen that allows users to enter complex selections. The selection may be a single value, or any form of interval (discrete or continuous). You can also use patterns. (See following slides).
- You can create variants for selection screens.
- If you declare an input field with reference to an ABAP Dictionary field, any search helps defined for the Dictionary field will be available on the selection screen.
- Selection texts can be translated into other languages. They are then displayed in the user's logon language.

- Selection ranges are stored in programs using an internal table.
- The ABAP statement SELECT-OPTIONS <selname> FOR <field> declares an internal table called <selname>, containing four fields - **SIGN**, **OPTION**, **LOW**, and **HIGH**. The fields LOW and HIGH have the same type as the field <field>.
- The SIGN field can take the value 'I' (for inclusive) or 'E' (for exclusive).
- The OPTION field can contain relational operators, pattern operators, and operators that allow you to enter intervals.
- For more information about selection ranges, choose *Goto -> Selection screen help* from any selection screen.
- For more information about selection screens, refer to the online path **ILB-1** in the appendix.

- To define a selection screen, include the required **PARAMETERS** and **SELECT-OPTIONS** statements in your data declarations. If you define more selection screens than just the standard selection screen, you must enclose the additional definitions in the statmeents **SELECTION-SCREEN BEGIN OF SCREEN** <nnnn> and **SELECTION-SCREEN END OF SCREEN** <nnnn> where <nnnn> is the number of the selection screen.
- For information about other graphical elements that you can place on a sele ction screen, such as group boxes, checkboxes, radio buttons, references to input fields on other selection screens and so on, see the keyword documentation for the **SELECTION-SCREEN** statement or the online documentation (**ILB-2**). This topic also forms part of course BC405: Techniques of List Processing.
- The standard selection screen is displayed by the ABAP runtime system when the program starts. User-defined selection screens are displayed when you use the statement **CALL SELECTION-SCREEN** <nnnn>. This statement sets the return code sy-subrc to zero if the user chooses 'Execute', and to 4 if the user chooses 'Cancel'.
- You can also call a selection screen as a modal dialog box. To do this, use the syntax **CALL SELECTION-SCREEN** <nnnn> **STARTING AT** <left_col> <upper_row> **ENDING AT** <right_col> <lower_row> where <left_col> and <upper_row> are the coordinates of the top left-hand corner of the screen. <right_col> and <lower_row> are the coordinates of the bottom right-hand corner.

- Selection screen processing is event-driven. Events are ABAP processing blocks that are called by the runtime system in a particular order and processed sequentially. In the program, each event is introduced by an event keyword. The processing block ends when the next event block starts, or the definition of a subroutine or dialog module occurs.
- **AT SELECTION-SCREEN OUTPUT** is processed before the selection screen is displayed. You can use this event to modify the selection screen dynamically.
- **AT SELECTION-SCREEN ON HELP-REQUEST FOR <sel_field>** and **AT SELECTION-SCREEN ON VALUE-REQUEST FOR <sel_field>** allow you to define your own F1 and F4 help.
- **AT SELECTION-SCREEN** is processed when the user presses ENTER or chooses another function on the selection screen. You can use this event to check the values the user entered on the screen. The addition **ON...** allows you to control which fields or groups of fields should accept input again in the event of an error.

- An ABAP program consists of a sequence of processing blocks (events) that are processed by the runtime system in a particular order.
- **`LOAD-OF-PROGRAM`** is triggered directly after the system has loaded a program with type 1, M, F, or S into an internal session. The processing block is executed once only for each program in each internal session.
- **`INITIALIZATION`** is processed in executable (type 1) programs, directly before the selection screen is displayed. You can use the corresponding processing block to preassign values to the parameters and selection options on the selection screen.
- **`START-OF-SELECTION`** is processed after the selection screen has been processed. If you are working with a logical database, the corresponding `GET` events are triggered after `START-OF-SELECTION`. For further information, refer to the course BC405 'Techniques of List Processing and SAP Query' and the online documentation.
- **`END-OF-SELECTION`** is processed after all of the data has been read, and before the list is displayed.
- **`TOP-OF-PAGE`** is an event in list-processing. The processing block is always executed when you start a new page in the list.

- Once the basic list has been displayed, you can react to possible user actions. Detail lists allow you to distribute the information you want to display across several lists.
- This makes the lists easier for the user to understand, and improves performance, since you can delay reading extra information from the database until the user actually requests it.
- You can also use additional selection screens to allow the user to enter further restrictions.
- For each basic list you can use up to 20 detail lists. Each list is stored in its own list buffer. When the user chooses 'Back' (green arrow) or 'Cancel' (red cross), he or she returns to the previous list. This action initializes the list buffer of the list level the user just left.
- When the user chooses 'Exit' (yellow arrow), the system terminates the list processing and returns to the standard selection screen.

- The events START-OF-SELECTION, GET, END-OF-SELECTION, TOP-OF-PAGE and END-OF-PAGE can be used only to create basic lists.
- To create detail lists, use the events **`AT LINE-SELECTION`** or **`AT USER-COMMAND`**.
- Use **`TOP-OF-PAGE DURING LINE-SELECTION`** for page headers on detail lists.
- Each detail list event exists only once in the program and is shared by all detail lists. You must therefore ensure yourself, within the processing block, that the correct list is created. To do this, use a CASE structure that uses the system field sy-lsind. This system field contains the list index of the list that you are currently generating.

- Use the statement `HIDE global_field` to store the contents of the global data field `global_field` for the current line.
- If the user selects the line, the data field is automatically filled with the value that you retained for the line.
- You do not have to display the field on the list in order to retain its value using `HIDE`.
- The field can be a structure. However, deep structures (structures containing internal tables as components) are not supported.

- When the user selects a line on an interactive list, all of the global data fields whose values you stored using the HIDE statement while you were creating the basic list are filled with those values.
- The line selection is based on the cursor position when the `AT LINE-SELECTION` and `AT USER-COMMAND` events occur. (system field `sy-lilli`).
- If you choose a line using the `READ LINE...` statement, . the values are placed back in the original fields according to the line numbers.

- To check whether the user selected a valid line, you can use the fact that the hide area only contains data for valid lines. When you have finished creating the list, initialize a suitable test field. This allows you to check before you create the detail list whether a value from the hide area has been placed in the test field.
- Once you have created the detail list, re-initialize the test field to ensure that the user cannot choose an invalid line once he or she returns from the detail list and attempts to select another line for a new detail list

Unit: Basics for Interactive Lists

Theme: Creating a simple list

At the conclusion of these exercises, you will be able to:

- Create a simple program to create a list with an include structure for more complex applications.

The first step in your application is to write a program that displays a list of flights. The program should have a selection screen to allow the user to restrict the amount of data read and displayed.

2-1 Start your development project.

2-1-1 Create a development class **Z##BC410** (where ## is your group number) and assign it to your change request. You will use this development class for all of the Repository objects you create this week.

2-2 Write a simple program to create a list, which will serve as a basis for your further work. Call your program **Z##BC410_SOLUTION** (where ## is your group number). The program should display a list of flights.
The data you want to display is contained in the table SFLIGHT. You can use the model solution for orientation: SAPBC410ILBS_SIMPLE_LIST.

2-2-1 Your main program should consist of three include programs:
Z##BC410_SOLUTIONTOP Top include
Z##BC410_SOLUTIONE01 Event include
Z##BC410_SOLUTIONF01 Subroutine include.
Create the includes.

2-2-2 In the top include, declare a work area **wa_sflight** with type SFLIGHT; and a corresponding (standard) internal table **it_sflight**.

2-2-3 Create a selection screen with selection options for **wa_sflight-carrid** and **wa_sflight-connid**. Place these in a group box with the title “Flight”, and maintain the selection texts.

2-2-4 In the event include, write two events START-OF-SELECTION and END-OF-SELECTION. In the START-OF-SELECTION event, call a subroutine **read_flights**, in which you use an array fetch to read the data from table SFLIGHT into your internal table. Remember to take the user’s selections into account when you read the data. Create the subroutine in your subroutine include using forward navigation. In the END-OF-SELECTION event, call a subroutine **display_flights**, in which you display the data on a list. Display the data as shown on the model list. Ensure that the price is displayed **appropriately to its currency**.

2-2-5 Maintain standard headings for the list as on the model list.

You can program an array fetch as follows:

```
SELECT * INTO TABLE it_sflight
 FROM SFLIGHT ... .
```

```
Flights
------------------------------------------------------------------------

Flight          Date                Price                      Seats
max.            occ.
------------------------------------------------------------------------
AA  0017        12/21/1999          513.69       USD      660      10
AA  0017        12/19/1999          513.69       USD      660      16
AA  0017        11/29/1999          513.69       USD      660      51
AA  0017        11/22/1999          513.69       USD      660      95
AA  0017        11/19/1999          513.69       USD      660       0
AA  0017        09/30/1999          513.69       USD      660       8
AA  0017        08/28/1999          513.69       USD      660      34
LH  0400        12/31/1999          1,332.00     DEM      280      42
LH  0400        12/29/1999          1,332.00     DEM      107      76
LH  0400        12/09/1999          1,332.00     DEM      280      20
LH  0400        12/02/1999          1,332.00     DEM      280       0
LH  0400        11/29/1999          1,332.00     DEM      280      41
LH  0400        10/10/1999          1,332.00     DEM      280       1
LH  0400        09/07/1999          1,332.00     DEM      280     183
AZ  0555        12/21/1999          360,202      ITL      220      89
AZ  0555        19.12.1999          360.202      ITL      220     205
AZ  0555        29.11.1999          360.202      ITL      220       0
AZ  0555        22.11.1999          360.202      ITL      220      66
AZ  0555        19.11.1999          360.202      ITL      220     140
```

Unit: Basics for Interactive Lists

Theme: Creating a simple list

2-2 Model solution: SAPBC410ILBS_SIMPLE_LIST

Main program

```
*&---------------------------------------------------------------------*
*& program             SAPBC410ILBS_SIMPLE_LIST                        *
*&                                                                     *
*&---------------------------------------------------------------------*
INCLUDE bc410ilbs_simple_listtop.
INCLUDE bc410ilbs_simple_liste01.
INCLUDE bc410ilbs_simple_listf01.
```

Top include

```
*----------------------------------------------------------------------*
*   INCLUDE      BC410ILBS_SIMPLE_LISTTOP                               *
*----------------------------------------------------------------------*
PROGRAM sapbc410ilbs_simple_list.

workarea and internal table for flights
DATA: wa_sflight TYPE sflight,
it_sflight LIKE TABLE OF wa_sflight.

selection screen for choosing connections
SELECTION-SCREEN BEGIN OF BLOCK conn WITH FRAME TITLE text-001.
SELECT-OPTIONS: so_car FOR wa_sflight-carrid,
so_con FOR wa_sflight-connid.
SELECTION-SCREEN END OF BLOCK conn.
```

Event include

```
*---------------------------------------------------------------------*
*   INCLUDE     BC410ILBS_SIMPLE_LISTE01                              *
*----------------------------------------------------------------------*
START-OF-SELECTION.
PERFORM read_flights.
```

```
END-OF-SELECTION.
PERFORM display_flights.
```

Subroutine include

```
*-------------------------------------------------------------------*
***INCLUDE  BC410ILBS_SIMPLE_LISTF01.
*-------------------------------------------------------------------*
*&------------------------------------------------------------------*
*&      Form  READ_FLIGHTS
*&------------------------------------------------------------------*
FORM read_flights.
SELECT * INTO TABLE it_sflight FROM sflight
WHERE carrid IN so_car
AND connid IN so_con.
ENDFORM.

*&------------------------------------------------------------------*
*&      Form  DISPLAY_FLIGHTS
*&------------------------------------------------------------------*
FORM display_flights.
LOOP AT it_sflight INTO wa_sflight.

WRITE: / wa_sflight-carrid,
wa_sflight-connid,
wa_sflight-fldate,
wa_sflight-price CURRENCY wa_sflight-currency,
wa_sflight-currency,
wa_sflight-seatsmax,
wa_sflight-seatsocc.

ENDLOOP.
ENDFORM.                    " DISPLAY_FLIGHTS
```

The Program Interface

Contents:

- Overview: GUI title and GUI status
- Creating a GUI status
- Using a GUI status

- Unit 1 Course Overview
- Unit 2 Basics for Interactive Lists
- **Unit 3 The Program Interface**
- Unit 4 Interactive List Techniques
- Unit 5 Introduction to Screen Programming
- Unit 6 Screen Elements for Output
- Unit 7 Screen Elements for Input/Output
- Unit 8 Screen Elements: Subscreens and Tabstrip Controls
- Unit 9 Screen Elements: Table Controls
- Unit 10 Context Menus on Screens
- Unit 11 Lists in Screen Programming
- Unit 12 Preview: Control Framework

- A **GUI status** is made up of a **menu bar**, a **standard toolbar**, an **application toolbar**, and of **function key settings**. Each screen can have one or more GUI statuses. For example, an editor program might have two statuses - one for display mode and one for change mode.
- The elements of a GUI status allow users to choose functions using the mouse.
- Menus are control elements that allow the user to choose which functions will be processed by an application program. Menus can also contain submenus. The 'System' and 'Help' menus are present on every screen in the R/3 System. They always have identical functions and cannot be changed or hidden.
- The application toolbar contains icons for frequently-used functions. The standard toolbar, which is the same on every screen in the R/3 System, contains a set of icons, each of which has a fixed assignment to a corresponding function key. If a function in the standard toolbar is not available on the current screen, the icon is grayed out.
- The application toolbar allows the user to choose frequently-used functions by clicking the corresponding pushbutton.
- You use the function key settings to assign functions such as Find, Replace, or Cut to the function keys.
- All of a program's GUI titles and statuses taken together make up its user interface. Whenever you change or add a new title or status, you must regenerate the user interface.

- There are three ways to create a title: from the object list in the Object Navigator, from the Menu Painter, or by forward navigation from the ABAP Editor.
- The name of a title can be up to 20 characters long.
- You should set an appropriate title for each screen in your application.
- You can use variables in titles that are set dynamically at runtime by including the ampersand character (&) as a placeholder. At runtime, the ampersand is replaced by a value that you specify. You can use up to nine variables by placing digits after the ampersand.
  To set a title that contains variables, use the statement:
  ```
       SET TITLEBAR <title_name> WITH <&1> ... <&10>.
  ```
- A title bar remains in place until you set another one. At runtime, the system variable sy-title contains the current title. Title bars are also known as GUI titles.

- From a technical point of view, a status is a **reference** to a menu bar, to certain key assignments and to an application toolbar.
- A single component (such as a menu bar) can be used by more than one GUI status.
- GUI statuses are ABAP program objects that can be displayed on screens and lists.
- You should set a status for every screen in your application.

- A status is a reference to a menu bar, a key setting, and an application toolbar.
- A menu bar is made up of individual menus.
- Key assignments and application toolbars are sub-objects of the function key settings.
- You can create a set of application toolbars for a single key setting. Refer to the online documentation path in appendix reference **GUI-1**. In order to include a function in an application toolbar, the function must be assigned to a function key. Each status contains a single application toolbar.
- All program menus and key assignments refer to a particular function list. This list can be reached using F4 help. Application toolbars refer to the function list indirectly by way of the key assignment.
- A function within a status can be either active or inactive. Inactive functions are not displayed in the application toolbar.

- Functions are identified by their *function codes*.
- The attribute *function type* determines the intended purpose of a function. You can use the function types ' ' (space), 'E', and 'P' for pushbuttons that you place on a screen using the Screen Painter, and for tab titles. Function types 'S and 'H' are reserved for internal use by SAP. Function type 'T' indicates a transaction code. When the user chooses a function with this type, the system leaves the current program (without performing any checks) and calls the new transaction.
- Functions can be created with static texts or dynamic texts.
- If a function has a static text, you can assign an icon to it (*Icon name* attribute). If the function is already assigned to a pushbutton, an icon is displayed instead of the static text. The static text is used when you assign the function to a menu entry. The function text belonging to the function is used as "quick info". The contents of the *Infotext* attribute appear in the status bar of the screen when the user chooses the function. If you want to display text as well as the icon, enter the text in the *Icon text* attribute.
- The *Fastpath* attribute allows you to define a letter code, which users can enter to choose the function without using the mouse.
- For further information, refer to the online documentation path in appendix reference **GUI-2**.

- Functions can be assigned to individual function keys or pushbuttons.
- Function key settings consist of a key assignment and an application toolbar pushbutton.
- A function key assignment's type (possible values: screen or dialog box) only serves to define the technical use of the key assignment.
- Key assignments consist of *Reserved Functions Keys*, *Recommended Functions Keys* and *Freely Assigned Function Keys*. *Reserved Functions Keys* are function keys whose assigned values cannot be changed in the SAP system. You may activate and deactivate their functions, however, the icons and texts assigned to them cannot be changed. Activated *Reserved Functions Keys* appear in the standard toolbars of both screens and lists.
- *Recommended Functions Keys* are assigned suggested values that satisfy SAP usability (ergonomic) norms.
- Functions that have been assigned to function keys can also be assigned to pushbuttons in the application toolbar.
- An application toolbar can contain up to 35 pushbuttons.

- A menu can contain up to 15 entries.
- Possible entries are functions, separators, and menus (cascading menus).
- Menus can be up to three levels deep. The third level may only contain functions and separators.
- Menus can be created with static or dynamic text. You must assign a field name to menus with dynamic text, whose contents will be displayed as the menu text at runtime.
- The menu type *Include menu* allows you to reference menus in other programs. When you do this, you must specify the name of the program and status from which you want to include the menu next to the *Short documentation* field.
- Include menus can only be accessed using the menu bar.
- A menu bar can contain up to eight different menus. Up to six of these can be freely assigned. The system automatically adds both the *System* menu and the *Help* menu to every menu bar.

- There are three ways to create a title: from the object list in the Object Navigator, from the Menu Painter, or by forward navigation from the ABAP Editor.
- The status type indicates the technical attributes of the status. You can choose between a dialog status (status for fullscreen), or a dialog box status (for use with modal dialog boxes). Context menus are special collections of functions that can be displayed when the user clicks the right-hand mouse button. We will deal with them separately in the **Context Menus** unit.
- You can create your own statuses by initially generating new settings, using existing ones (reference technique), or by combining both of these procedures. If you want to create an entirely new status, you must then create your own menu bars, menu functions, and other elements. Changes to a status only affect that status.
- When you use the reference technique, you create menu bars, application toolbars, and function key assignments as independent elements. You then create your own status and refer to the menu bar, application toolbar, and any function key assignment you want. The Menu Painter stores and maintains these references so that any changes in the menu bar, application toolbar or function key assignments automatically take effect in all statuses referring to them.
- The linking technique is particularly effective for ensuring consistency in very large applications that use several statuses. The links ensure that the user can access functions in the same way whatever status is set.

- The *Adjust template* function in the *Extras* menu allows you to add standardized function codes to your own statuses. This function allows you to merge objects from any status into the current status. In particular, it allows you to use standards for list statuses or selection screens, or to take a status from another ABAP program.
- You can also choose to display standard proposals for the menu bar, which you can then modify.

- In the Menu Painter, you can include in a status key settings, application toolbars, or menu bars that you have already defined elsewhere. If you do this, work from the bottom upwards. If there is more than one application toolbar defined for your key setting, you can choose the appropriate one.
- Initially, all functions are inactive. You only have to activate the functions that are relevant in the current status.
- When you create new functions, you can decide whether you want to change all of the statuses that use the same object. The new functions are initially inactive in all other statuses in which you include them.

- In a key setting, you assign individual functions to function keys and pushbuttons. Function key settings consist of a key assignment and a set of application toolbars.
- A function key assignment's type (possible values: Screen, Dialog box, List, List in a dialog box) determines where the key can be used.
- You can attach functions to reserved function keys, recommended function keys, and freely-assigned function keys. You should observe SAP's ergonomic guidelines. There is a series of examples that you can display from within the Menu Painter.
- *Recommended Functions Keys* are assigned suggested values that satisfy SAP usability (ergonomic) norms.
- If a function is important, and you have already assigned it to a function key, you can also assign it to a pushbutton in the application toolbar. The application toolbar may contain up to 35 pushbuttons.

- When you assign a function to the standard toolbar, it is also automatically assigned to a reserved function key.
- To find out the function keys to which these functions are assigned in the current status, click the Information icon in the Menu Painter.
- For more information about how key combinations such as Ctrl-P are converted into internal function key numbers (for example, for batch input), follow menu path **GUI-3** in the Menu Painter.

- You can only use a function in the application toolbar if you have already assigned it to a function key.
- Functions in the application toolbar are identified by their function code. The *Function type* attribute identifies the purpose of the function. If you want to process a function in the program, use the type ' ' (space). If you assign type 'T' to a function, the current program terminates when the user chooses the function, and the system starts the transaction assigned to the function.
- If you assign an icon to a function with a static text (*Icon name* attribute), the system displays the icon instead of the text in the application toolbar. The function text belonging to the function is used as "quick info". The contents of the attribute *Info. text* appear in a screen's status line whenever the event is triggered. If you want to display additional text with an icon, it should be entered in the attribute *Icon text.*
- You can use the *Fastpath* attribute to specify the letters that allow you to choose a function without using the mouse.
- To insert a separator in the application toolbar, use the *Insert* menu in the Menu Painter.
- If you use the 'Fixed positions' attribute for the application toolbar, pushbuttons for inactive functions are grayed out instead of being hidden. To set this attribute, double-click the open padlock symbol next to the application toolbar description.

- A menu entry can be a function, a separator, or another menu (cascading menu)
- To add a function to a menu, enter its function code in the left-hand column. If the function already exists in the function list and has a text assigned to it, this is entered automatically in the text field. If not, double-click the right-hand field to maintain a text.
- To insert a separator, use the 'Insert' menu or fill the function text field with minus signs at the appropriate position.

- Status and title names can be up to 20 characters in length and must be entered all in capital letters. A status stays active until a new one is set.
- You can use up to nine variables in a GUI title using the syntax `SET TITLEBAR <title> WITH <f1> ...<f9>.`
- If no GUI interface has been set, a standard user interface is displayed. Use SET PF-STATUS SPACE to deactivate previously entered statuses and activate the default list status. You can deactivate functions at runtime with the EXCLUDING <FCODE> addition. If you want to deactivate several function codes at the same time, you must first transfer these to the system using an internal table.

- You should work with an interactive event and centrally control various user actions in the program, handling the actions independently of each other.
- You program AT USER-COMMAND as an interactive event and evaluate the system field sy-ucomm in a CASE control structure. This field contains the current function code.
- Data is restored from the hide area to the corresponding global data fields for the line on which the cursor was positioned.

Unit: The Program Interface

Theme: Creating a GUI status and GUI title

At the conclusion of these exercises, you will be able to:

- Create a user interface for a program and use it for navigation.

Continue developing your application by creating a GUI status for your program. You should, in particular, create a function in your interface that allows the user to display a list of bookings for a flight. You will also make it easier for the user to recognize where he or she is in your application by using GUI titles.

3-1 Add a GUI status and GUI title to your program and create a detail list containing the bookings for a flight.

3-1-1 Extend your program **Z##BC410_SOLUTION** from the *Basics for Interactive Lists* unit, or copy the corresponding model solution **SAPBC410ILBS_SIMPLE_LIST.** You can use the model solution **SAPBC410GUIS_LIST_GUI** for orientation.

3-1-2 Create a GUI status and a GUI title for your program with the following attributes. The status should be a list status with the standards included. Create the function book, assigning it to a pushbutton in the application toolbar and to a menu entry.

GUI Status	BASE	**Type:** Dialog status
Function	BOOK	**Function key:** F5 **Text:** Bookings **Icon:** ICON_ICON_LIST **Infotext:** Booking list **Function type:** ` ´
GUI Title	BASE	**Text:** Flights

3-1-3 Set the status and title for the basic list.

3-1-4 In the AT USER-COMMAND event, create a detail list if the user chooses the BOOK function. Read the data from the database table SBOOK for the line in which the cursor is positioned. Encapsulate the database access in a subroutine read_bookings. You will need to pass the airline, flight number, date, and cancellation flag to it as parameters. Read the data into a global

internal table it_sbook_read with the line type SBOOK.   Make sure that you only read the flights that have not been canceled (cancellation flag: **canceled = ` ´**.)
Append the lines of **it_sbook_read** to another internal table **it_sbook** with the same type. Sort it_sbook by airline, flight number, flight date, and booking number. Write another subroutine **display_bookings** to display the data from the internal table **it_sbook** on the detail list. Display the following booking data as it appears on the model list: booking number (**bookid**), customer number (**customid**), customer type (**custtype**), luggage weight (**luggweight**), weight unit (**wunit**), class (**class**), and booking date (**order_date**). To do this, create a global work area **wa_sbook** with type SBOOK for the internal table **it_sbook**. Display the luggage weight with the correct unit. Use the `UNIT` addition in the `WRITE` statement.

3-1-5 Create a GUI status BOOK and a title BOOK for the booking list. Status BOOK should reference the status BASE for its menu bar, function key setting, and application toolbar. However, you should deactivate the BOOK function. Set the status and title for the booking list.

3-1-6 Ensure that the detail list is not displayed if the user does not select a valid line.

3-1-7 Use the TOP-OF-PAGE DURING LINE-SELECTION event to create list headings as shown in the model list.

Template:

```
Flight:   AZ   0555

Date   09/30/1999

00000536 00000195  B     10,3000  KG  C    10/28/1998
00000537 00000074  B           0  KG  C    10/28/1998
00000538 00000274  B     11,2000  KG  C    11/03/1998
00000539 00000140  B           0  KG  C    11/01/1998
00000540 00000141  B           0  KG  C    10/28/1998
00000541 00000270  B      6,1000  KG  C    10/23/1998
00000542 00000206  B           0  KG  C    11/03/1998
00000543 00000206  B     10,7000  KG  C    11/01.1998
00000544 00000201  B           0  KG  C    10/28/1998
00000545 00000201  B      1,3000  KG  C    10/23/1998
00000546 00000165  B           0  KG  C    11/03/1998
00000547 00000072  B           0  KG  C    11/01/1998
00000548 00000072  B      5,1000  KG  C    10/28/1998
00000549 00000168  P           0  KG  F    10/23/1998
```

Unit: The Program Interface

Theme: Creating a GUI status and GUI title

3-1 Model solution SAPBC410GUIS_LIST_GUI

Add the coding in bold type to your program. Create the new subroutines using forward navigation.

Top include

```
workarea and internal tables for bookings
DATA: wa_sbook LIKE sbook,
it_sbook_read LIKE TABLE OF wa_sbook,
it_sbook LIKE TABLE OF wa_sbook.
```

Event include

```
AT USER-COMMAND.
CHECK NOT wa_sflight-carrid IS INITIAL.
CASE sy-ucomm.
WHEN 'BOOK'.
REFRESH it_sbook.
PERFORM read_bookings
USING wa_sflight-carrid
wa_sflight-connid
wa_sflight-fldate
' '.
APPEND LINES OF it_sbook_read TO it_sbook.
SORT it_sbook BY carrid connid fldate bookid.
PERFORM display_bookings.
SET PF-STATUS 'BOOK'.
SET TITLEBAR 'BOOK'.
CLEAR wa_sflight-carrid.
ENDCASE.
```

```
CHECK sy-ucomm = 'BOOK'.
FORMAT COLOR COL_HEADING.
ULINE.
WRITE: / 'Flight:'(t01), wa_sbook-carrid, wa_sbook-connid,
AT sy-linsz space,
/ 'Date:'(t02), wa_sbook-fldate, AT sy-linsz space.
ULINE.
```

Subroutine include

```
FORM display_flights.
LOOP AT it_sflight INTO wa_sflight.

WRITE: / wa_sflight-carrid,
wa_sflight-connid,
wa_sflight-fldate,
wa_sflight-price CURRENCY wa_sflight-currency,
wa_sflight-currency,
wa_sflight-seatsmax,
wa_sflight-seatsocc.
HIDE: wa_sflight.
ENDLOOP.
ENDFORM.                               " DISPLAY_FLIGHTS
*&-----------------------------------------------------------------*
*&      Form  READ_BOOKINGS
*&-----------------------------------------------------------------*
FORM read_bookings USING    p_carrid LIKE wa_sbook-carrid
p_connid LIKE wa_sbook-connid
p_fldate LIKE wa_sbook-fldate
p_cancelled LIKE wa_sbook-cancelled.

SELECT * INTO TABLE it_sbook_read FROM sbook
WHERE carrid = p_carrid
AND connid = p_connid
AND fldate = p_fldate
AND cancelled = p_cancelled.

ENDFORM.                               " READ_BOOKINGS
```

```
*&-----------------------------------------------------------------*
*&      Form  DISPLAY_BOOKINGS
*&-----------------------------------------------------------------*
FORM display_bookings.

LOOP AT it_sbook INTO wa_sbook.

WRITE: / wa_sbook-bookid,
wa_sbook-customid,
wa_sbook-custtype,
wa_sbook-luggweight UNIT wa_sbook-wunit,
wa_sbook-wunit,
wa_sbook-class,
wa_sbook-order_date.

ENDLOOP.

ENDFORM.                               " DISPLAY_BOOKINGS
```

Interactive List Techniques

Contents:

- Selecting multiple lines
- Sorting lists
- Controlling the list sequence and messages

Unit 1	Course Overview
Unit 2	Basics for Interactive Lists
Unit 3	The Program Interface
Unit 4	**Interactive List Techniques**
Unit 5	Introduction to Screen Programming
Unit 6	Screen Elements for Output
Unit 7	Screen Elements for Input/Output
Unit 8	Screen Elements: Subscreens and Tabstrip Controls
Unit 9	Screen Elements: Table Controls
Unit 10	Context Menus on Screens
Unit 11	Lists in Screen Programming
Unit 12	Preview: Control Framework

- When a user chooses a function from a list, it triggers that function's function code. This function code, in turn, triggers a corresponding event.
- Some function codes are reserved for use by the system and therefore do not trigger an interactive event when the user chooses them (that is, the system does not return to the program). Instead these codes trigger a corresponding system function.
- All function codes with the exception of "PICK" as well as all codes reserved for system use trigger the event AT USER-COMMAND. For more information, refer to the *Program Interface* unit.
- You can find a list of those function codes reserved for system use in the Menu Painter under the appendix **ILS-1.**

- At a READ statement all values for that line that have been stored in the hide area are inserted into their corresponding fields and thus made available to the program.
- With the addition INDEX <i> you can read the lines of a particular list level <i>. If you omit this addition, the system refers to the list last displayed.
- If you use the addition FIELD VALUE <f1> INTO <g1>, the system reads field <f1> from the corresponding line of the list buffer and places the contents in field <g1>. If you leave out INTO <g1>, only field <f1> is filled.
  Caution: All lines in the list buffer are stored as character strings (type C). Thus, values inserted in field <f1> are automatically converted to type C.
- If you use the addition LINE VALUE INTO <wa>, the system places the entire line in the work area <wa>.
- For more on READ statement variants, refer to the online documentation path in appendix reference **ILS-2**.

- The statement `MODIFY LINE <l>` modifys the lth line of the list. The values stored in the hide area for this line are placed in the corresponding fields, and are thus available in the program.
- The statement `MODIFY CURRENT LINE` changes the last line to have been chosen by line selection or the `READ LINE` statement (even if it was in a different list level).
- If you use the `LINE FORMAT` addition, the selected line is formatted according to the specifications `<fm1>, <fm2>`, ...
- The addition `FIELD VALUE` replaces the field contents of `<f1>, <f2>`, ... in the list line with the current values of `<g1>, <g2>`, ... (all values are converted to type C).
  The contents of `<f1>`, `<f2>`, ... themselves are not replaced.
  If a field from the line being modified is displayed more than once, that line will only be modified the first time it is displayed.
- The `LINE VALUE FROM <wa>` addition allows you to replace the entire line being modified with the current contents of field `<wa>`.
- For more information about the `MODIFY` statement, refer to the online documentation path in appendix reference **ILS-3**.

- The `FIELD <fieldname>` statement, allows you to find out the name of the field in which the cursor is positioned. The name of the variable from which the value comes is placed in the field `<fieldname>`. However, for the sort criterion, you only use the name of the field as it appears in the table definition. You therefore need to use an offset specification to find out the field name. The offset is the length of the structure name plus one character for the hyphen.
- The name of the output field is provided in the field specified in the `FIELD` parameter. The output value is contained in the field specified in the `VALUE` parameter.
- The operation sets the return code `sy-subrc`.
  - sy-subrc = 0: The cursor was positioned on a field.
  - sy-subrc = 4: The cursor was not positioned on a field.
- Caution: Do not use the value from the `VALUE` parameter as a selection criterion in a `SELECT` statement. If it is not a character field, the system will convert its type, which could lead to undesired results. It is better to use the hide technique instead.
- You can find out more about what additions can be used with `GET CURSOR`. Refer to the online documentation path in appendix reference **ILS-4**.

- Before you find out the sort field, check that the user placed the cursor on a valid line. If this is not the case, you should display an appropriate message.
- Decreasing the list level (changing the value of sy-lsind) should always be the last action before you display the list buffer. This system field determines the list level at which the new list is displayed. The hide area and list buffer of any higher list levels are automatically initialized.

- You can use system field `sy-lsind` to determine the list level at which the list is displayed. In the example above, list level 2 where the list is sorted according to the number of unoccupied seats is being displayed. The statement `sy-lsind = 1` causes the list to be displayed at list level 1, thus replacing the list sorted according to date, which would normally be displayed first.
- You cannot assign a value to `sy-lsind` that is greater than the current value of the field assigned by the system. This means you cannot bypass list levels in ascending direction.
- You should only change `sy-lsind` in the last statement before the list is displayed, since changing the value does not always lead to an immediate change of list level. The new list level is assigned to the list at the very end, after the entire list buffer has been displayed. If you are not acquainted with this behavior, you could program your lists incorrectly.

- Message types have the following effects on list processing:
  - Type E messages discard the curent detail lists and return to the list level previously displayed.
  - Type W messages are always displayed as error messages (type E).
  - While the basic list is being created, type W and type E messages always lead to program termination (corresponds to type A).
- For full details of how messages behave in a particular event, refer to the online documentation for the **`MESSAGE`** keyword.

Unit: Interactive List Techniques

Theme: Multiple line selection and dynamic list sorting

At the conclusion of these exercises, you will be able to:

- Display detail information for a set of lines in a list.
- Sort an existing list dynamically.

Extend your application to allow the user to display the bookings for a set of flights from the bookings list. You should also allow the user to sort the booking list by the column on which the cursor is positioned.

4-1 Enable multiple line selection.

4-1-1 Extend your program **Z##BC410_SOLUTION** from the *Program interface* unit, or copy the corresponding model solution **SAPBC410GUIS_LIST_GUI**. You can use the model solution **SAPBC410ILSS_INTERACTIVE_LIST1** for orientation.

4-1-2 At the beginning of each line of the basic list, display a selection field **mark** as a checkbox (subroutine **display_flights**).

4-1-3 In the AT USER-COMMAND event, establish the lines that the user chose and, if necessary, retrieve the corresponding booking data using the subroutine **read_bookings**. The effort with the two internal tables **it_sbook_read** and **it_sbook** was not worth it after all.

4-1-4 Extend the subroutine **display_bookings** so that there is a page break before each new flight. To do this, create a global structure key_sflight with the fields **carrid**, **connid**, and **fldate** (table SFLIGHT). Whenever the data in the booking data record changes, generate a page break.

You could use the following construction:

```
IF wa_sbook-carrid NE key_sflight-carrid
    OR ... .
MOVE-CORRESPONDING wa_sbook TO key_sflight.
...
ENDIF.
```

4-1-5 Add the functions SELECT and DESELECT to the status BASE. Use this to allow the user to select or deselect all of the list entries in a single step.

Implement the functions in the application toolbar and in the menu.

Function	SELECT	**Function key:** F6 **Text:** Select all **Icon:** ICON_SELECT_ALL **Function type:** ` ´
Function	DESELECT	**Function key:** F7 **Text:** Deselect all **Icon:** ICON_DESELECT_ALL **Function type:** ` ´

4-1-6 Program the SELECT and DESELECT functions in the AT USER-COMMAND event.

4-2 Allow the user to sort the booking list.

4-2-1 Extend your program **Z##BC410_SOLUTION** from the previous exercise (or copy the model solution **SAPBC410ILSS_INTERACTIVE_LIST1**). You can use the model solution **SAPBC410ILSS_INTERACTIVE_LIST2** for orientation.

4-2-2 Add the functions SRTU and SRTD to the status BOOK. Allow the user to sort the booking list in either ascending or descending order. To do this, sort the internal table it_sbook by the airline, flight number, flight date, and the field on which the cursor is positioned. Ensure that the user can only sort the table when the cursor is positioned on a valid line. Implement the functions in the application toolbar and in the menu.

Function	SRTU	**Function key:** F8 **Text:** Sort ascending **Icon:** ICON_SORT_UP **Function type:** ` ´
Function	SRTD	**Function key:** F9 **Text:** Sort descending **Icon:** ICON_SORT_DOWN **Function type:** ` ´

4-2-3 Make sure that the list level is not increased in each sort, and that the correct list header is displayed.

Unit: Interactive List Techniques

Theme: Multiple line selection and dynamic list sorting

4-1 Model solution SAPBC410ILSS_INTERACTIVE_LIST1

Add the coding in bold type to your program. Create the new subroutines using forward navigation.

Top include

```
workarea and internal table for flights
DATA: mark,
wa_sflight type sflight,
it_sflight LIKE TABLE OF wa_sflight.
sflight key for testing changes
DATA: BEGIN OF key_sflight,
carrid LIKE wa_sflight-carrid,
connid LIKE wa_sflight-connid,
fldate LIKE wa_sflight-fldate,
END OF key_sflight.
```

Event include

```
AT USER-COMMAND.
CHECK NOT wa_sflight-carrid IS INITIAL.
CASE sy-ucomm.
WHEN 'BOOK'.
REFRESH it_sbook.
DO.
READ LINE sy-index FIELD VALUE mark.
IF sy-subrc NE 0. EXIT. ENDIF.
CHECK NOT mark IS INITIAL.
PERFORM read_bookings
USING wa_sflight-carrid
wa_sflight-connid
wa_sflight-fldate
' '.
APPEND LINES OF it_sbook_read TO it_sbook.
ENDDO.
SORT it_sbook BY carrid connid fldate bookid
```

```
PERFORM display_bookings.
SET PF-STATUS 'BOOK'.
SET TITLEBAR 'BOOK'.
CLEAR wa_sflight-carrid.
WHEN 'SELECT'.
DO.
READ LINE sy-index.
IF sy-subrc NE 0. EXIT. ENDIF.
MODIFY CURRENT LINE FIELD VALUE mark FROM 'X'.
ENDDO.
WHEN 'DESELECT'.
DO.
READ LINE sy-index.
IF sy-subrc NE 0. EXIT. ENDIF.
MODIFY CURRENT LINE FIELD VALUE mark FROM space.
ENDDO.

ENDCASE.
---------------------------------------------------------------------------
```

Subroutine include

```
FORM display_bookings.

LOOP AT it_sbook INTO wa_sbook.

IF key_sflight-carrid NE wa_sbook-carrid
OR  key_sflight-connid NE wa_sbook-connid
OR  key_sflight-fldate NE wa_sbook-fldate.
MOVE-CORRESPONDING wa_sbook TO key_sflight.
NEW-PAGE.
ENDIF.

WRITE: / wa_sbook-bookid,
wa_sbook-customid,
wa_sbook-custtype,
wa_sbook-luggweight UNIT wa_sbook-wunit,
wa_sbook-wunit,
wa_sbook-class,
wa_sbook-order_date.

ENDLOOP.
```

```
ENDFORM.                               " DISPLAY_BOOKINGS
```

4-2 **Model solution** SAPBC410ILSS_INTERACTIVE_LIST2

Add the coding in bold type to your program. Create the new subroutines using forward navigation.

Top include

```
field name for GET CURSOR
DATA  fieldname(50).
```

Event include

```
AT USER-COMMAND.
CASE sy-ucomm.
WHEN 'BOOK'.
...
CLEAR: wa_sflight-carrid,
wa_sbook-bookid.
WHEN 'SRTU'.
CHECK NOT wa_sbook-bookid IS INITIAL.
GET CURSOR FIELD fieldname.
fieldname = fieldname+9.
SORT it_sbook BY carrid connid fldate (fieldname).
PERFORM display_bookings.
sy-lsind = sy-lsind - 1.
CLEAR wa_sbook-bookid.
WHEN 'SRTD'.
CHECK NOT wa_sbook-bookid IS INITIAL.
GET CURSOR FIELD fieldname.
fieldname = fieldname+9.
SORT it_sbook BY carrid connid fldate (fieldname) DESCENDING.
PERFORM display_bookings.
sy-lsind = sy-lsind - 1.
CLEAR wa_sbook-bookid.

ENDCASE.

TOP-OF-PAGE DURING LINE-SELECTION.
CHECK sy-ucomm = 'BOOK' OR sy-ucomm = 'SRTD' OR sy-ucomm = 'SRTU'.
...
```

Subroutine include

```
FORM display_bookings.

LOOP AT it_sbook INTO wa_sbook.

IF key_sflight-carrid NE wa_sbook-carrid
OR  key_sflight-connid NE wa_sbook-connid
OR  key_sflight-fldate NE wa_sbook-fldate.
MOVE-CORRESPONDING wa_sbook TO key_sflight.
NEW-PAGE.
ENDIF.

WRITE: / wa_sbook-bookid,
wa_sbook-customid,
wa_sbook-custtype,
wa_sbook-luggweight UNIT wa_sbook-wunit,
wa_sbook-wunit,
wa_sbook-class,
wa_sbook-order_date.
HIDE wa_sbook-bookid.
ENDLOOP.

ENDFORM.                               " DISPLAY_BOOKINGS
```

Introduction to Screen Programming

Contents:

- Principles of screen programming
- Screen objects
- Dynamic screen modifications
- Screen processing
- GUI status for screens

■ Unit 1	Course Overview
■ Unit 2	Basics for Interactive Lists
■ Unit 3	The Program Interface
■ Unit 4	Interactive List Techniques
■ **Unit 5**	**Introduction to Screen Programming**
■ Unit 6	Screen Elements for Output
■ Unit 7	Screen Elements for Input/Output
■ Unit 8	Screen Elements: Subscreens and Tabstrip Controls
■ Unit 9	Screen Elements: Table Controls
■ Unit 10	Context Menus on Screens
■ Unit 11	Lists in Screen Programming
■ Unit 12	Preview: Control Framework

- Screens allow you to enter and to display data.
- One of their strengths is that they can combine with the ABAP Dictionary to allow you to check the consistency of the data that a user has entered.
- Screens allow you to create user-friendly dialogs with pushbuttons, tabstrip controls, table controls, and other graphical elements.

- Let us look at a simple dialog program with a selection screen as its initial screen and a screen for displaying information for a selected data record.
- When the program starts, the system loads its program context and prepares memory space for the program data objects. The selection screen is displayed.
- The user enters data on the selection screen and chooses *Execute*.
- In a processing block, the program reads data from the database. To do so, it passes information about the data requested by the user to the database. The database fills a structure with the required data record.
- The processing logic then calls a screen. This triggers a processing block belonging to the screen called Process Before Output (or **PBO**). Once the PBO has been processed, the data is transferred to a structure that serves as an interface to the screen. It is then transferred to the screen and displayed.
- Any user action on the screen (pressing enter, choosing a menu entry, clicking a pushbutton, ...) returns control to the runtime system. The screen fields are then transported into the structure that serves as the interface between screen and program, and the runtime system triggers another processing block belonging to the screen, which is always processed after a user interaction, and is called Process After Input (or **PAI**).

- In this course, you will learn about screen objects. A screen object is any screen element in the R/3 System that allows users to interact with an ABAP program.
- On the following pages, screen objects are presented from an object-oriented viewpoint, that is, their **attributes** are described, along with the **methods** you can use to work with them.

- You can use the *User settings* icon (on the far right hand side of the standard toolbar) to configure the R/3 window according to your own preferences.
- You can:
  - Change the colors of various elements of the R/3 interface
  - Change the font of texts displayed in the system
  - Modify the R/3 window (for example, hide the standard toolbar or restore the default window size
  - Show or hide grid lines in lists
  - Change the cursor behavior
- The changes you make to the user settings are stored on your presentation server, not in the R/3 System.
- For more detailed information, refer to the online documentation path in appendix reference **DIA-1**.

- The screen objects *text field*, *input/output field*, *status icon*, *group box*, *radio button*, *checkbox*, and *pushbutton* all have general attributes, Dictionary attributes, program attributes, and display attributes.
- The objects *subscreen*, *tabstrip control* and *table control* have general attributes, and special attributes relating to the respective object type.
- We can divide the attributes of an object into
    - **Statically definable** attributes that **cannot be changed dynamically**
    - **Statically definable** attributes that **can be changed dynamically**
    - Attributes that **can only be changed dynamically**
- For complete documentation of the attributes of screen objects, refer to the online documentation path in appendix reference **DIA-2**.

- At the beginning of the PBO, the runtime system reads the statically-created and dynamically-modifiable attributes of each screen object on the screen into a system table with the line type SCREEN.
- The slide shows the assignment of the fields in the system table SCREEN to the names of the statically created attributes of the screen objects.

- When a screen is processed, the system table SCREEN contains an entry for each screen object that has been created in the Screen Painter for that screen.
- It is initialized in the PBO of each screen, and is filled with the screen objects belonging to that screen.
- You can change the dynamically-modifiable attributes of the elements on the screen using the construction
  LOOP AT SCREEN. ... ENDLOOP.
  in a PBO module. To do this, you use the structure SCREEN, which is created automatically by the system, and filled with the values of each successive line of the system table in the LOOP. Active attributes have the value '1', inactive attributes have the value '0'. To change the system table, use MODIFY SCREEN. within the LOOP.
- To find the object whose attributes you want to modify, you can use a LOOP on the SCREEN table, and query one of the following fields: SCREEN-NAME, SCREEN-GROUP1 to SCREEN-GROUP4. There is further information about modification groups on the next page.
- For further information about the SCREEN table, see the description of the structure of SCREEN or the documentation for the LOOP statement.

- Dynamic changes to the attributes of screen objects are **temporary**.
- Using this technique to modify the attributes of a screen object (for example, to change whether an input/output field is ready for input), you can replace long sequences of separate screens, which are more costly in terms of both programming time and runtime.

- If you want to change the attributes of several attributes at once at runtime, you can include them in a modification group. To do this, enter the same three-character group name in one of the fields `SCREEN-GROUP1` through `SCREEN-GROUP4` of each element.
- Each object can belong to up to four modification groups. You assign the group names in the element list or layout editor in the Screen Painter.

- You must program your screen modifications in a module that is processed during the PROCESS BEFORE OUTPUT processing block.
- You use a loop throught the table SCREEN to change the attributes of an object or a group of objects. (LOOP AT SCREEN WHERE . . . and READ TABLE SCREEN are not supported.)
- To activate and deactivate attributes, assign the value 1 (active) or 0 (inactive), and save your changes using the MODIFY SCREEN statement.
- Note that objects you have defined statically in the Screen Painter as invisible cannot be reactivated with SCREEN-ACTIVE = 1. However, objects that you have statically defined as visible in the Screen Painter can dynamically be made invisible. SCREEN-ACTIVE = 0 has the same effect as the following three statements:
  SCREEN-INVISIBLE = 1, SCREEN-INPUT = 0, SCREEN-OUTPUT = 0.

- Screens are freely-definable objects that you can use to display or enter information.
- They are a form of dialog between the user and the ABAP program.

- A screen consists of the input/output mask (layout), the screen attributes, and the screen flow logic. For further information about how to program screen flow logic, refer to the ABAP User's Guide.
- Screens have four components: the screen mask, the screen attributes, the element list, and the flow logic. The flow logic contains flow logic code (not ABAP statements).
- Screens are containers for other screen objects.

- Each screen has a set of administration attributes that specify its type, size, and the subsequent screen. It also has settings that influence other properties of the screen and of its components.
- The administration attributes *Program* and *Screen number* identify the screen by its number and the program to which it belongs.
- Screen numbers greater than 9000 are reserved for SAP customers. Screen numbers 1000 through 1010 are reserved for the maintenance screens of ABAP Dictionary tables and the standard selection screens of reports.
- The screen type identifies the purpose of the screen. Certain other special attributes of a screen and its components depend on this attribute.
- The "Next screen" attribute allows you to specify the screen that should be processed after the current screen in a fixed sequence.
- For a full list of screen attributes with their meanings, refer to the online documentation path in appendix reference **DIA-3**.

- When you create a screen, you must:
  - Set the general screen attributes (on the attribute screen)
  - Design the screen layout (in the layout editor)
  - Set the field attributes (in the field list)
  - Write the flow logic (in the flow logic editor).

- To create a screen from the object list in the Object Navigator, create a new development object with the type *Screen*. Position the cursor on *Screens* and right-click.
- The Object Navigator automatically opens the Screen Painter.
- When you create a screen, you first have to enter its attributes. Enter a screen number, a short text, and a screen type. You will normally use the screen type *Normal*. You can specify the number of the next screen in the *Next screen* field.
- If you enter 0 (or nothing) for the next screen, the system resumes processing from the point at which the screen was called once it has finished processing the screen itself.
- You can also create a screen by writing a CALL SCREEN <nnnn> statement in the ABAP Editor and then double-clicking the screen number <nnnn>.

- To allow you to set the attributes of all screen elements, the Screen Painter contains an element list with six views. You can also display all of the attributes for a single element from any of the lists (*Attributes*). You can also maintain the attributes for an element from the layout editor using the *Attributes* function.
- Within the Screen Painter, you work with external data types. These correspond to the types defined in the ABAP Dictionary. For fields that you have chosen that are defined in the ABAP Dictionary, the system displays the external data type in the *Format* column. For elements (templates) that do not have an ABAP Dictionary reference, you must enter an external data type yourself.
- To find out the corresponding external data type for an internal data type (ABAP data type), see the keyword documentation for the ABAP **TABLES** statement. For example:

ABAP Dictionary Data Type	ABAP Data Type
CHAR	C
NUMC	N

- You usually define screen fields by adopting the corresponding field descriptions from the ABAP Dictionary. However, you can also use field descriptions that you have defined in your program. In order to do this, you must generate the program first.
- You can use the key word texts and templates either together or separately.
- The graphical layout editor provides an easy way of defining the various screen elements (such as input/output fields, key word texts, boxes, and so on). You simply choose the element you require, and position it on the screen using the mouse.
- To delete a screen element, select it, and choose *Delete*.
- You can move elements on the screen by dragging and dropping them with the mouse.

Note:

The graphical layout editor is available under Windows NT, Windows 95 and UNIX.
If you use a different operating system, you must use the alphanumeric Screen Painter.

- Screens have their own set of keywords that you use in the PBO and PAI events of the flow logic.
- In the flow logic, you write `MODULE` calls. The modules are components of the same ABAP program. They contain the ABAP statements that you want to execute.
- You can create a module by double-clicking the module name in the flow logic Editor.
- To create a module from the object list in the Object Navigator, choose the development module 'PBO module' or 'PAI module'.
- You can call the same module from more than one screen. If the processing depends on the screen number, you can retrieve the current screen number from the system field `sy-dynnr`.
- Note that the modules you call in the PBO processing block must be defined using the `MODULE OUTPUT` statement; modules that you define using the statement `MODULE... INPUT` can only be called in the PAI event.

- In order for a screen and its ABAP program to be able to communicate, the fields on the screen and the corresponding fields in the program **MUST HAVE IDENTICAL NAMES**.
- **After** it has processed all of the modules in the **PBO** processing block, the system copies the contents of the fields in the ABAP work area to their corresponding fields in the screen work area.
- **Before** it processes the first module in the **PAI** processing block, the system copies the contents of the fields in the screen work area to their corresponding fields in the ABAP work area.
- You should use your own structures (SDYN_CONN, …) for transporting data between the screen and the ABAP program. This ensures that the data being transported from the screen to the program and vice versa is exactly the data that you want.

- You can establish a static sequence of screens by entering a value in the *Next screen* field of the screen attributes.
- If you enter 0 (or no value) as the next screen, the system resumes processing from the point at which the screen was initiated, once it has finished processing the screen itself.

- The `SET SCREEN <nnnn>` statement **temporarily** overwrites the *Next screen* attribute.
- The screen `<nnnn>` must belong to the same program.
- The next screen is processed either when the current screen processing ends, or when you terminate it using the `LEAVE SCREEN` statement.
- To specify the next screen and leave the current screen in a single step, use the `LEAVE TO SCREEN <nnnn>` statement.

- To interrupt processing of the current screen and branch to a new screen (or sequence of screens), use the `CALL SCREEN <nnnn>` statement. The screen `<nnnn>` must belong to the same program.
- In the program, the system constructs a stack. The stack has to be destroyed before the end of the program.
- To return to the statement following the `CALL SCREEN` statement, you can use either `SET SCREEN 0`, `LEAVE SCREEN`, or `LEAVE TO SCREEN 0`. The screen that called the other screen is then processed further.
- If you use the above statements outside of a call chain, the program terminates, and control returns to the point from which it was called. You can also terminate a program using the ABAP statement `LEAVE PROGRAM`.

- In the `CALL SCREEN` statement, you can use the `STARTING AT` and `ENDING AT` additions to specify the position and size of the screen that you are calling. The screen in the `CALL SCREEN` statement must be defined as a modal dialog box.
- If you omit the `ENDING AT` statement, the size of the dialog box is determined by the *Used size* in its screen attributes.
- If you use the `ENDING AT` addition, the system displays as much of the dialog box as will fit into the available space. If there is not enough room to show the entire dialog box, it appears with scrollbars.

- The starting position (origin) of every SAP window is its top left-hand corner.
- The values that you pass to lc, ur, rc, and lr in the statement
  `CALL SCREN STARTING AT lc ur ENDING AT rc lr`
  refer to the R/3 window in which you display the dialog box (on the slide, screen 100).

- When the system displays a screen, it automatically places the cursor in the first input field. If you want the cursor always to appear in a different field, you can enter the corresponding object name in the Cursor position field of the screen attributes.
- You can also tell the system in the PBO event to position the cursor in a particular field. This makes your application more user-friendly.
- You can set the field in which the cursor should appear in the program.
  To do this, use the ABAP statement

```
SET CURSOR FIELD <field_name> OFFSET <position>.
```

- `<field_name>` can be a unique name in quotation marks, or a variable containing the object name. To place the cursor at a certain position within a field, use the `OFFSET` parameter, specifying the required position in `<position>`.
- The system then places the cursor at the corresponding offset position, counting from the beginning of the field.

- A **GUI status** is made up of a **menu bar**, a **standard toolbar**, an **application toolbar**, and of **function key settings**. A single screen can have more than one status. You should use a module set_status_<nnnn> in the PBO (Process Before Output) event of each of your screens to assign a GUI status and a GUI title to it.
- There are three ways to create a status: from the object list in the Objet Navigator, from the Menu Painter, or by forward navigation from the ABAP Editor. The status type describes the technical attributes of the status. Choose Dialog status if you want use the status with a screen in fullscreen mode, and dialog box, if you are going to use it with a dialog box. Context menus are special menus that you can attach to the right-hand mouse button. They are described in a separate unit.
- When you change a status, you must activate it before the changes become visible.

- To ensure consistency, you should reuse existing menu bars, application toolbars, and key settings wherever possible. The Menu Painter administers the links you establish between these objects so that any changes apply to all other statuses that use them. There is also a set of standard menu entries that you can use as a template and modify.
- When you assign functions to the reserved function keys in the standard toolbar, you should adhere to the SAP standards. This makes your program easier for users to understand and for you to maintain. For further information, refer to the *SAP Style Guide*.

- When the user triggers a function with type ' ' using a pushbutton, menu entry, or function key, the system places the relevant function code in the OK_CODE field of the screen.
- To allow you to process this field in the PAI event, you must assign a name to the field, which you enter in the element list in the Screen Painter. You must then create a field in your ABAP program with the same name. During the automatic field transport at the beginning of the PAI event, the function code is passed from the screen to the corresponding field in the program.
- To avoid the function code le ading to unexpected processing steps on the next screen (ENTER does not usually change the OK_CODE field), you should initialize the function code field in the ABAP program before leaving the screen, otherwise it will be transported back to the screen automatically in the PBO event.

Unit: Introduction to Screen Programming

Theme: Creating a screen and using it in an executable program

At the conclusion of these exercises, you will be able to:

- Create screens and use them in your programs.

Create a maintenance screen for your program. Design an interface for it. The user should see the screen after double-clicking a line on the basic list.

5-1 Create a screen and include it in your program.

5-1-1 Extend your program **Z##BC410_SOLUTION** from the previous exercise (or copy the model solution **SAPBC410ILSS_INTERACTIVE_LIST2**). You can use the model solution **SAPBC410DIAS_DYNPRO** for orientation.

5-1-2 In the AT LINE-SELECTION event, call screen 100.

5-1-3 Create the following program object:

Screen	0100	**Short description:** Maintenance screen **Type:** Normal **Next screen:** 100

5-1-4 In the PBO event of screen 100, call a module **status**. Use forward navigation to create the module in a new include.
Z##BC410_SOLUTIONO01 PBO module include. In this module, set the GUI status STATUS_100 and GUI title TITEL_100 (Flight data (&)) and pass “Display” to the title as a parameter. Use a text element for the parameter, to ensure that it can be translated. You can create the status and title by forward navigation.
Assign the type *Dialog status* to the status. Activate the standard function BACK (F3) with the function type ‘ ’ (space).

5-1-5 Assign the name **ok_code** to the function code field on your screen, and create a corresponding variable in the top include of your program.

5-1-6 In the PROCESS AFTER INPUT event of screen 100, call the modules **save_ok_code** and **user_command_100**. Use forward navigation to create the modules in a new include
Z##BC410_SOLUTIONI01 PAI module include. Ensure that

the user can return from screen 100 to the basic list if he or she chooses BACK (F3).

5-1-7 Make sure that you have inserted the necessary `INCLUDE` statements in your main program.

Unit: Introduction to Screen Programming

Theme: Creating a screen and using it in an executable program

5-1 **Model solution** SAPBC410DIAS_DYNPRO

Add the coding in bold type to your program. Create the new modules using forward navigation.

Flow logic for screen 100

```
PROCESS BEFORE OUTPUT.

MODULE status.

PROCESS AFTER INPUT.

MODULE save_ok_code.
MODULE user_command_100.
```

Main program

```
*&-----------------------------------------------------------------*
*& program       SAPBC410DIAS_DYNPRO                               *
*&                                                                 *
*&-----------------------------------------------------------------*

INCLUDE BC410DIAS_DYNPROTOP.
INCLUDE BC410DIAS_DYNPROE01.
INCLUDE BC410DIAS_DYNPROF01.
INCLUDE BC410DIAS_DYNPROO01.
INCLUDE BC410DIAS_DYNPROI01.
```

Top include

```
fields for ok_code processing
DATA: ok_code LIKE sy-ucomm,
save_ok LIKE ok_code.
```

Event include

```
*&---------------------------------------------------------------*
*&   Event AT LINE-SELECTION.
*&---------------------------------------------------------------*
AT LINE-SELECTION.
CALL SCREEN 100.
```

PBO module include

```
*----------------------------------------------------------------*
***INCLUDE  BC410DIAS_DYNPROO01.
*----------------------------------------------------------------*
*&---------------------------------------------------------------*
*&      Module  STATUS  OUTPUT
*&---------------------------------------------------------------*
MODULE status OUTPUT.
SET PF-STATUS 'STATUS_100'.
SET TITLEBAR 'TITLE_100' WITH 'View'(m01).
ENDMODULE.                             " STATUS  OUTPUT
```

PAI module include

```
*----------------------------------------------------------------*
***INCLUDE BC410DIAS_DYNPROI01.
*----------------------------------------------------------------*
*&---------------------------------------------------------------*
*&      Module  USER_COMMAND  INPUT
*&---------------------------------------------------------------*
MODULE USER_COMMAND_100 INPUT.
CASE SAVE_OK.
WHEN 'BACK'.
LEAVE TO SCREEN 0.
ENDCASE.
ENDMODULE.                 " USER_COMMAND  INPUT

*&---------------------------------------------------------------*
*&      Module  SAVE_OK_CODE  INPUT
*&---------------------------------------------------------------*
MODULE SAVE_OK_CODE INPUT.
SAVE_OK = OK_CODE.
CLEAR OK_CODE
```

```
ENDMODULE.                 " SAVE_OK_CODE  INPUT
```

Screen Elements for Output

Contents:

- Using, creating, and modifying
  - Text fields
  - Status icons
  - Group boxes
- Example: Dynamic screen modifications

- Unit 1 Course Overview
- Unit 2 Basics for Interactive Lists
- Unit 3 The Program Interface
- Unit 4 Interactive List Techniques
- Unit 5 Introduction to Screen Programming
- **Unit 6 Screen Elements for Output**
- Unit 7 Screen Elements for Input/Output
- Unit 8 Screen Elements: Subscreens and Tabstrip Controls
- Unit 9 Screen Elements: Table Controls
- Unit 10 Context Menus on Screens
- Unit 11 Lists in Screen Programming
- Unit 12 Preview: Control Framework

- A text field is a rectangular area on a screen in which the system displays text.
- Text fields contain labels for other elements. These labels (sometimes called "keywords"), are purely for display - they cannot be changed at runtime by the user. Text fields are displayed in a fixed position on the screen.
- Text fields can also contain literals, lines, icons, and other static elements. They can contain any alphanumeric characters, but may not begin with an underscore (_) or a question mark (?) . If you use a text as a label for a checkbox or radio button, it must have the same object name as the checkbox or radio button it accompanies.
- If your text consists of more than one word, use underscore characters as separators. This enables the system to recognize that the different words in fact belong together. The system interprets spaces as separators between two different text fields.
- Text fields can be translated. They then appear in the user's logon language. To do this, follow the menu path under **OUT-1**.

- At runtime, you can change the size (*visible length*) of a text field and the display attributes *Bright* and *Invisible*. To do this, use the fields SCREEN-LENGTH, SCREEN-INTENSIFIED, and SCREEN-INVISIBLE (or SCREEN-ACTIVE).

- You can create text fields in either of the following ways:

  Directly in the layout editor, by placing a text field object in the work area and entering the text in the *Object text* attribute.

  By using the accompanying text of a data element from the ABAP Dictionary.

- When you use fields from ABAP Dictionary structures on the screen, the system normally displays the data element text as well as the template for the input/output fields on the screen.

- You can make text fields invisible at runtime.
- If you make an object invisible that is enclosed in a box, the box is not displayed either. (For further information, refer to the documentation for the screen attribute *Switch off runtime compression*).

- At PBO, the system table with the line type SCREEN is initialized by the runtime environment, and filled with the static attributes from the Screen Painter.
- To hide a text field at runtime, you modify the system table. Use LOOP AT SCREEN. ... MODIFY SCREEN
  ENDLOOP.
- To make a text field invisible, use SCREEN-INVISIBLE = 1 or SCREEN-ACTIVE = 0.

- To ensure that the field TEXTFIELD1 is **not** displayed on the screen, you can call a module in the PROCESS BEFORE OUTPUT processing block that sets the invisible attribute for that field.
- To do this, set the contents of the field SCREEN-INVISIBLE to 0.
- You can process the SCREEN table like an internal table with header line (LOOP AT SCREEN. … MODIFY SCREEN. ENDLOOP.)
- The system does not support the statements LOOP AT SCREEN WHERE… and READ TABLE SCREEN.

- A status icon is an output field that contains an icon. You choose the relevant icon at runtime. Icons allow you to indicate a status in your application. They are predefined in the system, and take up between two and four characters.
- For information about the available icons, see the online documentation (reference **OUT-2**).

- Status icons are special output fields that display icons. The system sets the attributes 'Output field' and '2 dimensional', and these cannot be changed. The default data format is CHAR.
- You can change the *Visible length*, *Intensified*, and *Invisible* attributes of a status icon dynamically.

- You can only define a status field in the graphical layout editor. A status field is an output field with an icon. You use them to display an icon, which you specify dynamically at runtime.
- To assign an icon to an output field dynamically, use the function module **ICON_CREATE.** The internal length of the output field must be at least 13 (icon without text). To ensure that you can display quickinfos that might be longer, define the field with defined length 132 and visible length 2.
- In the ABAP program, define a field with the same name as the screen field using the field TEXT from the structure ICONS.

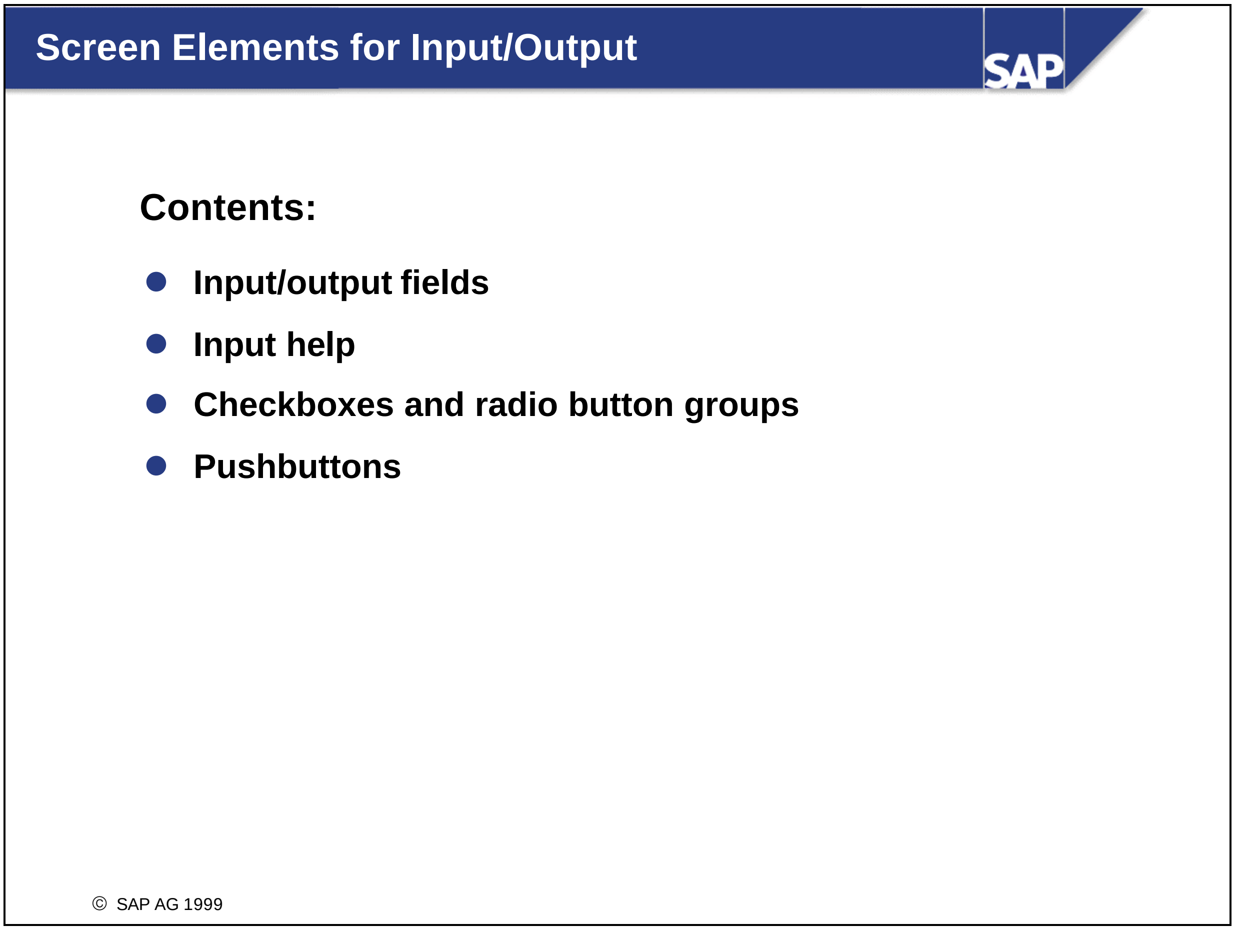
- You select the icon you want to display from the ABAP program. Before the screen is displayed, you need to find out the technical name of the icon. You do this by calling a module in the PBO event.
- You retrieve the technical name of an icon using the function module ICON_CREATE. You must pass the name of the icon you want to display to the function module. You can also pass a text to be displayed with the icon. The function module returns the technical name of the icon.
- For further details about this function module, refer to its documentation.

- Group boxes enclose a selection of elements that belong together (for example, a group of fields or a radio button group). They are purely display elements, and help the user to identify which elements on the screen belong together in a group.
- You can use group boxes to make sure that all fields within a box have the same context menu assigned to them. For further information, refer to the *Context Menus on Screens* unit.
- Group boxes may have a title.

- You can change the *Visible length* and *Invisible* attributes using the system table SCREEN.
- A group box may contain other screen objects.
- At runtime, if the box contains only invisible elements and the screen attribute *Runtime compression* is set, the box itself is not displayed.

- You define a group box in the layout editor. The object must have a name, and you may also assign a heading to the box.
- You can change the group box text dynamically. To do this, you should activate the *output field* attribute and create a global data field in the ABAP program with the same name. Because the Screen Painter field and the program field have the same name, any changes to the field contents will be immediately visible on the screen (similarly to input/output fields).

- Output objects are for improving the layout of your screens.
- Text fields allow you to label input/output fields. In this case, you should use the same name for the text field as for the input/output field. If you deactivate the input/output field, the text label is then automatically deactivated as well.
- Status icons allow you to provide the user with a quick graphical overview of information.
- Group boxes allow you to make a group of fields that logically belong together. Runtime compression ensures that empty boxes cannot be displayed.
- Static texts on a screen can be translated, so that they appear on the screen in the language in which the user is logged on. To make dynamically-assigned text accessible to translators, you must use text elements in your ABAP programs.



Screen Elements for Input/Output

Contents:

- Input/output fields
- Input help
- Checkboxes and radio button groups
- Pushbuttons

- Unit 1 Course Overview
- Unit 2 Basics for Interactive Lists
- Unit 3 The Program Interface
- Unit 4 Interactive List Techniques
- Unit 5 Introduction to Screen Programming
- Unit 6 **Screen Elements for Output**
- Unit 7 Screen Elements for Input/Output
- Unit 8 Screen Elements: Subscreens and Tabstrip Controls
- Unit 9 Screen Elements: Table Controls
- Unit 10 Context Menus on Screens
- Unit 11 Lists in Screen Programming
- Unit 12 Preview: Control Framework

- An input field is a rectangular screen element in which users can enter data.
- An output field is a rectangular screen element in which the system displays text or other data.
- Input/output fields are also known as templates.
- Input fields can have automatic input checks based on thir data type (for example, a date field will only allow you to enter a valid date).
- Input fields that you create with reference to ABAP Dictionary fields may have built-in data consistency checks (foreign key checks, value sets).
- Input fields may have possible values help (F4)
- For further information about input/output fields, see the online documentation **INP-1**.

- You can temporarily change the object attributes marked in gray using the system table SCREEN.
- It may not be possible to activate all possible combinations of attributes. This depends on the format of the input/output field.
  Example: You cannot activate the *Leading zeros* attribute for a field with the data format CHAR, since it is only relevant for numeric fields.
- For further information about the "Data format" attribute, refer to the online documentation path in appendix reference **INP-2**.

- You can create input/output fields in two ways:

  By entering them directly in the layout editor. You determine the size of the field by the number of underscore characters in the object text attribute. For numeric values, you can specify a comma as a separator, and a period as a decimal point. As the last character in the input/output field, you can enter 'V' as a placeholder for a plus or minus sign.

  By using a template from the ABAP Dictionary. Choose *Dict/Program fields* to do this.

- If you want to use the contents of an input/output field in your ABAP program, you must declare the field globally using the `DATA` or `TABLES` statement.

- You can save values in SAP memory using a parameter ID. These are user and terminal-session specific, but available to **all internal sessions**.
- `SET PARAMETER` copies the corresponding field contents into SAP memory in the PAI processing block.
- `GET PARAMETER` copies the corresponding field contents from SAP memory at the end of the PBO processing block (after data has been transferred from the program), but only if the screen field still has its initial value.

- You can link an input/output field to an area of SAP memory in the ABAP Dictionary.
- When you use an input/output field that is defined in the ABAP Dictionary, its parameter ID is displayed in the Dictionary attribute PID in the Screen Painter.
- The SPA and GPA attributes allow you to enable the set and get parameter functions separately.
- You can define parameter IDs in table TPARA.

- After the screen has been displayed, but before the PAI modules are processed, the system automatically checks the values that the user has entered on the screen.
- The first check is to ensure that all required fields have been filled.
- The system can only perform a foreign key check if a screen field refers back to a ABAP Dictionary field for which a check table has been defined and the Foreign key check attribute has been set.
- The F4 help function is also active. This enables users to display possible entries.

- If the automatic field input checks are insufficient for your requirements, you can program your own in the PAI event. To do this, use the `FIELD` statement with the `MODULE` addition. This means that the module you specify is only processed for the field specified in the `FIELD` statement.
- If an error or warning message occurs during the module, the system sends the screen again, but without processing the PBO module. The message is displayed, and only the field to which the check was applied is ready for input.
- Note: It is the `FIELD` statement that is responsible for making the field ready for input again. If you use a message in a module that is not called from within a `FIELD` statement, the system displays the message, but does not make the field ready for input again.

- If you want to ensure that more than one field is ready for input following an error dialog, you must list all of the relevant fields in the `FIELD` statement, and include both that and the `MODULE` statement in a `CHAIN` ... `ENDCHAIN` block.
- You can include individual fields in more than one `CHAIN` ... `ENDCHAIN` block.
- Note that the `FIELD` statement does not only make the field ready for input again; it also means that field contents changed during the current PAI processing are only visible if the field in question was also included in the `FIELD` statement of the current `CHAIN` block.

- If the system sends an error or warning message, the current screen is sent again, but the PBO is not processed again.
- Only the fields to which the module is assigned are ready for input again.
- After the user has entered new values, the `PROCESS AFTER INPUT` module is not completely reprocessed, but restarted somewhere within the processing block.
- The system finds out which field the user changed and resumes processing at the first corresponding `FIELD` statement.
- If the user merely confirms a warning message (without changing the fields contents), the system restarts the PAI processing after the `MESSAGE` statement where the error was triggered.

- There are six different categories of message: A, X, E, W, I, and S:

A Termination The processing terminates and the user must restart the transaction.

X Exit Like message type A, but with short dump MESSAGE_TYPE_X.

E Error Processing is interrupted, and the user **must** correct the entry.

W Warning Processing is interrupted and the user can correct the entries (works like an E message). However, it is also possible to confirm the existing entries by pressing ENTER (works like an I message)

I Information Processing is interrupted, but continues when the user has confirmed the message (pressed ENTER).

S Success Displays information on the next screen

- The system transports data from screen fields into the ABAP fields with the same name in the PAI processing block. First, it transports all fields that are not contained in any `FIELD` statements. The remaining fields are transported when the system processes the relevant `FIELD` statement.
- If an error or warning message occurs in a module belonging to a `FIELD` statement, the current values of all fields in the same `CHAIN` structure are automatically transported back into their corresponding screen fields.

- Field input checks usually require access to the database. Consequently, avoiding them where possible improves the performance of your program.
- If the user "strayed" onto the screen by mistake, he or she will not usually be able to make a consistent set of entries that will satisfy the input checks. You should therefore make it possible for a user to leave a screen without the field checks taking place.
- To protect the user from losing data that he or she has already entered if they leave the screen unintentionally, you should program security prompts.

- If you use the `ON INPUT` addition in a `MODULE` statement after `FIELD`, the module is only called if the field contents have changed from their initial value.
- Within a `CHAIN` block, you must use the `ON CHAIN-INPUT` addition. The module is then called if the contents of at least one screen field within the `CHAIN` block have changed from their initial value.
- You may only use the `ON INPUT` addition if the `MODULE` statement is contained in a `FIELD` statement.

- If you use the `ON REQUEST` addition in a `MODULE` statement after `FIELD`, the module is only calle d if the user has entered a new value in that field.
- Within a `CHAIN` block, you must use the `ON CHAIN-REQUEST` addition. The module is then called if the user has changed the contents of at least one screen field within the `CHAIN` block.
- You may only use the `ON REQUEST` addition if the `MODULE` statement is contained in a `FIELD` statement.

- The module with the addition `AT EXIT-COMMAND` is processed before the automatic field input checks. You can use it for navigation. You may only use the `AT EXIT-COMMAND` addition with one module. It may not have an associated `FIELD` statement.
- If you do not leave the screen from this module, the automatic field checks are processed after it, followed by the rest of the PAI event.
- Ensure that you process the contents of the `OK_CODE` field appropriately.

- The SAP Style Guide contains details of how you should set the navigation functions *Back*, *Exit*, and *Cancel.*
- The `BACK` function (green left arrow) should lead one logical level backwards. From screens on the same level as the initial screen, it leads back to the initial screen. From screens that contain detailed information, it leads back to the screen that called the current screen.
- The `CANCEL` function differs from `BACK` in its dialog behavior. For details, see the next page.
- The `EXIT` function should return to where the processing unit was called.
- On the initial screen of a program, all three functions *Back*, *Exit*, and *Cancel* lead back to the screen from which the current program was called.
- For further information, refer to the online documentation path in appendix reference **INP-3**.

- If the user has entered data on the screen (`sy-datar` = 'X' or your own flag), you can avoid accidental loss of data by using a predefined security prompt.
- As well as specifying the targets of the *Back*, *Exit*, and *Cancel* functions, the SAP Style Guide also contains information about the dialogs you should conduct with the user, and the sequence of dialogs and automatic field checks.
- For the *Exit* and *Cancel* functions, you should first send a dialog box to the user. Then (in the case of the *Exit* function), the system checks the input on the screen. The functions in question must have function type 'E'.
- In the case of the *Back* function, the input checks come before the dialog.
- The R/3 System contains a series of function modules that you can use for the user dia logs.
- These are listed above. For further information, refer to the online documentation path in appendix reference **INP-4**.

- You can help the user with input by using dropdown list boxes containing the possible entries.
- Input help (F4 help) is a standard function in the R/3 System. It allows the user to display a list of possible entries for a screen field. If the field is ready for input, the user can place a value in it by selecting it from the list.
- If a field has input help, the possible entries button appears on its right hand side. The button is visible whenever the cursor is placed in the field. You can start the help either by clicking the button or choosing F4.
- As well as the possible entries, the input help displays relevant additional information about the entries. This is especially useful when the field requires a formal key.
- Since the input help is a standard function, it should have the same appearance and behavior throughout the system. There are utilities in the ABAP Workbench that allow you to assign standardized input help to a screen field.
- The precise description of the input help of a field usually arises from its semantics. Consequently, input help is usually defined in the ABAP Dictionary.

- Dropdown list boxes allow the user to choose an entry from a pull-down list containing the possible entries. The user cannot enter values freely, but must choose a value from the list.
- To create a dropdown list box for an input field, you must do the following in the Screen Painter:
  - Set the *Dropdown* attribute to *list box*.
  - Change the *visLength* attribute to the displayed length of the descriptive text.
  - Set the Value list attribute to ' ' to use value help from the ABAP Dictionary.
  - If required, set the function code for the selection. Like a menu entry, this function code triggers the PAI, and you can interpret it using the OK_CODE field.
- Important: The **visible length of the field determines the width of the field** (including button) and the selection list, and you must normally change it when you convert the field to a dropdown box.
- The values are filled automatically using the search help assigned to the ABAP Dictionary field. The Dictionary field must have a search help (check table) with two columns or a table of fixed values.

- Various things are required of input help for a screen field (the **search field**):
- The input help must take into account information that the system already knows (the context). This includes both information that the user has entered on the current screen, and information from previous dialog steps. The input help normally uses the context to restrict the set of possible values.
- The input help must find out the values that it will then present to the user for selection. It must also determine the data that will be displayed as additional information in the list of possible values. In determining the possible values, it must take into account restrictions that arise from the context, as well as those entered by the user as specific search conditions.
- The input help must conduct a user dialog. This involves (at least) displaying the possible values with additional information, and allowing the user to choose a value from it. In many cases, the input help will also contain an input screen on which the user can specify conditions to restrict the number of possible entries displayed.
- When the user selects a value, the input help must place it into the search field. In many cases, there are extra fields on the input screen (often only output fields), containing extra information about the search field. The input help should also update the contents of these fields.

- The ABAP Dictionary object **search help** is a description of an input help. Its definition contains the information that the system requires to meet the user's needs.
- The **interface** of the search help controls the data that is passed between the input screen and the F4 help. The interface determines the context data that is required and the data that can be placed back on the input screen when the user chooses a value.
- The **internal behavior** of the search help describes the actual F4 process. This contains the **selection method**, which retrieves the values for display, and the **dialog behavior**, which describes the interaction with the user.
- Similarly to function modules, search helps have an interface, which defines their capacity to exchange data with other software components, and an internal behavior (which, in the case of a function module, is its source code).
- It is only worth defining a search help if there is a mechanism that allows you to address it from a screen. This mechanism is called a **search help connection**, and is described later.
- Like the function module editor, the search help editor also allows you to test your objects. This allows you to check how a search help behaves before you assign it to a screen field.

- A search help describes the process of an input help. In order for it to work, we need a mechanism that assigns the search help to the field. This is called the **search help connection**.
- Connecting a search help to a field affects its behavior. It is therefore regarded as part of the field definition.
- The semantic and technical attributes of a screen field (type, length, F1 help) are normally not defined directly when you define the screen. Normally, you use a reference in the Screen Painter to an existing field in the ABAP Dictionary. The screen field then inherits the attributes of the ABAP Dictionary field.
  The same principle applies when you define input help for a screen field. The link between the search help and the search field is established using the ABAP Dictionary field, not the screen field.
- When you assign a search help, its interface parameters are asssigned to the screen fields that are filled by the search help, or which pass information to it from the screen. The search field must be assigned to an EXPORT parameter of the search help. You should also make the search field an IMPORT parameter, so that the search help can take into account a search pattern already entered in the field by the user.
- A field can have input help even if it does not have a search help - there are other mechanisms for F4 help (for example, fixed values for a domain).

- There are three ways to link a search help to a field in the ABAP Dictionary.
- It can be assigned directly to a field of a structure or table. You define this link in very much the same way as you would define a foreign key. You should define the assignment here (between the interface parameters of the search help and the structure field). The system generates a proposal.
- If the field has a check table, its contents are automatically proposed as possible values in the input help. The key fields of the check table are displayed. If the check table has a text table, the first non-key character field is also displayed.
  If the default display is insufficient for your requirements, you can attach a search help to the check table. This is then used for all fields that have that check table. When you link the search help, you must define the assignment between the search help interface and the key of the check table.
- The semantics of a field and its possible values are defined by its data element. You can therefore also link a search help to a data element. The search help is then used by all fields that are based on that data element. When you link the search help, you must specify a single EXPORT parameter, which will be used to transfer the data.
- Attaching a search help to a check table (or data element) increases its reusability. However, it does restrict your options for passing extra values to the search help interface.

- To allow as many fields as possible to carry useful input help, the R/3 System contains a wide range of mechanisms with which you can define it. If it is possible to use more than one of these for a particular field, the one highest in the hierarchy is used.
- As well as defining the input help for a field in the ABAP Dictionary (as we have already seen), you can also define it in the screen field. This method has the disadvantage that you cannot reuse it automatically.
- The screen event POV (PROCESS ON VALUE-REQUEST) allows you to program input help for a field yourself. You can make this help appear in standard form by using the function modules `F4IF_FIELD_VALUE_REQUEST` or `F4IF_INT_TABLE_VALUE_REQUEST`.
  However, you should first check to see whether you cannot program your own input help better using a search help exit.
- You can also attach a search help to a screen field in the Screen Painter. However, the functional scope of this technique is more restricted in comparison to attaching a search help in the ABAP Dictionary.
- You should no longer use input checks programmed directly in the flow logic (and from which input help can be derived).
- In the context menu (right-click) for the hit list, there is a function *Technical info*. This tells you which mechanism is being used in a particular case.

- Use radio buttons when you want to allow a user to choose only a single element from a group of fields.
- Use checkboxes when you want to allow the user to choose one or more elements from a group of fields.
- Wtih radio buttons, one selection rules out all other options within the group. When the user selects one, all of the others are automatically deselected.
- When the user selects a radio button, control is not immediately passed back to a work process on the application server. As with checkboxes (but in contrast to pushbuttons), it is still possible to make further entries before pressing a pushbutton or choosing a menu option.
- Checkboxes allow the user to select more than one element at once. Control is not returned to a work process on the application server. This does not happen until the user chooses a pushbutton or menu entry.

- Checkboxes and radio buttons must have an object name.
- As well as the input/output field, you can display text and icons for them. The text is contained in the *Object text* field in the attributes. To display an icon, enter its name in the Icon name attribute. You can enter quick info for the icon in the appropriate field.
- You can change the *Input field* and *Invisible* attributes using the system table SCREEN.

- You create checkboxes in the fullscreen editor of the Screen Painter. To do this, choose the checkbox object from the object list and place it on the screen. You must assign names to checkboxes. In the ABAP program, create a field with the same name, type C, and length 1.
- You can find out whether a user has chosen a checkbox in the ABAP program by querying the field contents. If a checkbox is not selected, its field value is initial.
- You can assign a function code and function type to a checkbox. When the user selects it, the PAI event is triggered as though the user had chosen a menu entry.

- You create radio buttons in the layout editor of the Screen Painter. There are two steps involved:
    - Create the radio buttons as individual elements. Choose "radio button" from the object list and place it on the screen. You must assign names to radio buttons. In the ABAP program, create a field with the same name, type C, and length 1.
    - Combine a collection of radio buttons into a radio button group. To do this, select the radio buttons in the layout editor and then choose *Edit* -> *Group* -> *Radio button group* -> *Define*.
- You can find out which radio button a user has chosen by querying the field contents in the ABAP program. If a radio button is not selected, the field value is initial.
- You can assign a function code and function type to a radio button group. When the user selects one of its radio buttons, the PAI event is triggered as though the user had chosen a menu entry.

- A pushbutton triggers a particular function. When the user chooses it, the system tells the program which function has been chosen. At this point, control of the program passes back to a work process on the application server, which processes the PAI processing block.

- Pushbuttons may contain text (*Object text* attribute), an icon, or both. You can either specify an icon statically, or dynamically, using the function module ICON_CREATE.
- You can change the *visible length*, *output field*, and *invisible* attributes dynamically using the system table SCREEN.
- You can change the text on a pushbutton dynamically. To do this, you must have set the *Output field* attribute in the Screen Painter to active, and created a global field with the same name in your ABAP program. Because the Screen Painter field and the program field have the same name, any changes to the field contents will be immediately visible on the screen (similarly to input/output fields).

- When you create a pushbutton, you must:

  Create the pushbutton itself. Choose the Pushbutton object from the Screen Painter object list, place it on the screen, and assign a name to it in the "Object name" attribute. You can enter a static text in the "Object text" attribute. Enter a function code for the pushbutton in the "Function code" attribute. This is placed in the OK_CODE field automatically when the user chooses the pushbutton on the screen.

  Activate the OK_CODE field on the screen by assigning a name to the OK_CODE field object in the Screen Painter field list and creating a field in your ABAP program that has the same name. You m,ust give the field a name in the element list of the Screen Painter, then declare an identically-named field in the ABAP program with reference to the system field `sy-ucomm`.

- When the user chooses a function on the screen, the system places the corresponding function code into the OK_CODE field. You can then query the field and use the result to trigger the appropriate processing block.

- Pushbuttons have a function code and a function type.
- If the user chooses a pushbutton that has the function type ' ', the PAI event is processed. The system places the function code that has been triggered into the OK_CODE field, which you can then query in the module.
- If the user chooses a pushbutton whose function has the function type "E", the system processes a module with the addition AT EXIT-COMMAND. This happens before the automatic field transport and the field input checks. The system places the function code that has been triggered into the OK_CODE field, which you can then query in the module.
- After the AT EXIT-COMMAND module, the system continues processing the screen normally (field input checks, followed by PAI processing).

Unit: Screen Elements for Input/Output

Theme: Input/output fields on screens, input help, mode selection using a radio button group

At the conclusion of these exercises, you will be able to:

- Create input/output fields for screens
- Make input checks
- Use input helps in your programs
- Create radio button groups and program the relevant logic
- Make dynamic changes to screens.

- Add input/output fields to your program for flight information. When the user arrives on the screen, it should display data for the line that he or she chose. The *Airline*, *flight number*, and *flight date* fields should be ready for input.
- Support the user by checking the entries and providing input help.
- Allow the use to switch between different program modes. These are:
- Display mode
- Flight data maintenance mode (the user can change the aircraft type)
- Maintain bookings (you will use this later)
- The current mode should be indicated in the title. If the user changes the aircraft type, he or she should be able to save the changed value. If this is the case, you must update the basic list.
- **Additional task:** Use a standard dialog to warn the user if data will be lost when he or she leaves the screen. You should also provide the opportunity at this point to save the data.

7-1 Add the input/output fields to the screen, implement the input checks, and extend the navigation options on the screen to include the *Cancel* and *Exit* functions.

7-1-1 Extend your program **Z##BC410_SOLUTION** from the previous exercise (or copy the model solution **SAPBC410DIAS_DYNPRO**). You can use the model solution **SAPBC410INPS_INPUT_FIELD** for orientation.

7-1-2 Copy the ABAP Dictionary structure **SDYN_CONN00** to the structure **ZDYN_CONN##.** (## is your group number). Use the TABLES statement to create a structure with the same name (for transporting data between the program and the screen).

7-1-3 Create the following fields on the screen. Use the facility for using fields from the ABAP Dictionary.

Screen 100	**I/O fields, text fields:** ZDYN_CONN## -CARRID -CONNID -FLDATE	**For each field: Input:** on **Output:** On **Required:** On
	I/O fields, text fields: ZDYN_CONN## -PRICE -CURRENCY -PLANETYPE -SEATSMAX -SEATSOCC -PAYMENTSUM	**For each field: Input:** Off **Output:** On

7-1-4 In the PBO event of screen 100, call a module **get_sflight_data**. Create the module using forward navigation. Copy the relevant fields of your work area **wa_sflight** into your screen data transfer structure **zdyn_conn##** Ensure that all of the required fields have been restored from the hide area.

7-1-5 Check the combination of airline, flight number, and flight date if the user changed any of these details. To do this, try to read the corresponding data record from table SFLIGHT, and analyze the return code **sy-subrc**. If the data record does not exist, display message **007** from class **BC410** as an error message. Make sure that the fields are ready for input again. If the input checks are successful, update the ABAP work area wa_sflight.

7-1-6 Assign the function codes EXIT and CANCEL to the standard keys Shift-F3 (exit) and F12 (cancel). Ensure that these functions are processed **before** the automatic input checks. If the user chooses *Exit*, leave the program. If, on the other hand, the user chooses *Cancel*, return to the basic list, but without checking the field values. In the case of *Cancel*, remember to initialize the OK_CODE field.

7-2 Make the user's job easier by providing input help.

7-2-1 Extend your program **Z##BC410_SOLUTION** from the previous exercise or copy the corresponding model solution **SAPBC410INPS_INPUT_FIELD**. You can use the model solution **SAPBC410INPS_HELP_FOR_INPUT** for orientation.

7-2-2 On screen 100, set the *Dropdown* attribute to *List box* for the input/output field **zdyn_conn##-carrid**. Make sure that the program attribute *Value list* is set to ' ' (from ABAP Dictionary).

7-2-3 In the ABAP Dictionary, attach the search help SDYN_CONN_CONNID to the field **zdyn_conn##-connid** and the search help SDYN_CONN_FLDATE to the field **zdyn_conn##-fldate**.

7-2-4 Test the input help to see if it is context-sensitive.

7-3 Create a radio button group to allow the user to choose one of a range of program modes.

7-3-1 Extend your program **Z##BC410_SOLUTION** from the previous exercise (or copy the model solution **SAPBC410INPS_HELP_FOR_INPUT**). You can use the model solution **SAPBC410INPS_RADIOBUTTON_GROUP** for orientation.

7-3-2 On screen 100, create a radio button group with the buttons **view**, **maintain_flights**, and **maintain_bookings**. Make sure that the function code MODE (with type ' ') is triggered when the user chooses a different mode. Create a group box around the radio button group called **frame** and assign it the text "Mode". Declare the relevant data fields in your top include.

7-3-3 Set the GUI title according to the mode chosen by the user.

7-3-4 Program the *Maintain flight data* mode. In this mode, the input/output field **zdyn_conn##-planetype** should be ready for input. Assign the modification group ADM to the field, and create a module **modify_screen** to make the corresponding dynamic screen modification.
If the user enters a new aircraft type, check whether the number of seats booked is greater than the maximum number of seats. To do this, update the field **zdyn_conn##-seatsmax** from table **SAPLANE**. If an error occurs, display message **109** from class **BC410** as an error message. If an error occurs, transport the maximum number of seats back to the screen.

7-3-5 Assign the function code SAVE (function type ' ') to the standard key Ctrl-S. If the user chooses this function, save the new data record in the database. To do this, write a subroutine update_sflight containing a direct database update in the form:

```
UPDATE sflight FROM wa_sflight.
       IF sy-subrc = 0.
         CLEAR dataloss.
         MESSAGE s009(bc410).
       ELSE.
         MESSAGE a008(bc410).
       ENDIF.
```

(This process would normally use a suitable SAP lock, but we have omitted it here for simplicity.)

7-3-6 Ensure that the basic list is updated if the user has changed the aircraft type. To do this, use the subroutine read_flights to read the data from the database and then use the subroutine display_flights to display the list again at list level 0.

7-4 Additional task:

7-4-1 Use the function modules `popup_to_confirm_step` and `popup_to_confirm_loss_of_data` to ensure that the user cannot inadvertently lose his or her changes by leaving the screen. Use the system field **sy-datar**. You can use it in the AT EXIT-COMMAND module to find out whether the user changed data on the current screen. You will also need a flag of your own- call it **dataloss**.

Unit: Screen Elements for Input/Output

Theme: Input/output fields on screens, input help, mode selection using a radio button group

7-1 Model solution SAPBC410INPS_INPUT_FIELDS

Add the coding in bold type to your program. Create the new modules and subroutines using forward navigation.

Flow logic for screen 100

```
PROCESS BEFORE OUTPUT.

MODULE status.
MODULE get_sflight_data.

PROCESS AFTER INPUT.

MODULE exit AT EXIT-COMMAND.
CHAIN.
FIELD: sdyn_conn00-carrid,
sdyn_conn00-connid,
sdyn_conn00-fldate MODULE check_sflight ON CHAIN-REQUEST.
ENDCHAIN.
MODULE save_ok_code.
MODULE user_command_100.
```

Top include

```
structures for dynpro processing
TABLES  SDYN_CONN00.
```

PBO module include

```
*&------------------------------------------------------------------*
*&      Module  GET_SFLIGHT_DATA  OUTPUT
*&------------------------------------------------------------------*
MODULE get_sflight_data OUTPUT.
MOVE-CORRESPONDING wa_sflight TO sdyn_conn00.
ENDMODULE.                             " GET_SFLIGHT_DATA  OUTPUT
```

PAI module include

```
*&------------------------------------------------------------------*
*&      Module  CHECK_SFLIGHT  INPUT
*&------------------------------------------------------------------*
MODULE check_sflight INPUT.
CHECK sdyn_conn00-carrid NE wa_sflight-carrid
OR sdyn_conn00-connid NE wa_sflight-connid
OR sdyn_conn00-fldate NE wa_sflight-fldate.
SELECT SINGLE * INTO wa_sflight FROM sflight
WHERE carrid = sdyn_conn00-carrid
AND connid = sdyn_conn00-connid
AND fldate = sdyn_conn00-fldate.
CHECK sy-subrc NE 0.
MESSAGE e007(bc410).
ENDMODULE.                             " CHECK_SFLIGHT  INPUT

*&------------------------------------------------------------------*
*&      Module  EXIT  INPUT
*&------------------------------------------------------------------*
MODULE exit INPUT.
CASE ok_code.
WHEN 'CANCEL'.
CLEAR ok_code.
LEAVE TO SCREEN 0.
WHEN 'EXIT'.
LEAVE PROGRAM.
ENDCASE.
ENDMODULE.                             " EXIT  INPUT
```

7-3 **Model solution** SAPBC410INPS_RADIOBUTTON_GROUP

Add the coding in bold type to your program. Create the new modules using forward navigation.

Flow logic for screen 100

```
PROCESS BEFORE OUTPUT.
MODULE status.
MODULE get_sflight_data.
MODULE modify_screen.

PROCESS AFTER INPUT.
MODULE exit AT EXIT-COMMAND.
...
CHAIN.
FIELD: sdyn_conn-planetype,
sdyn_conn-seatsmax MODULE check_planetype ON CHAIN-REQUEST.
ENDCHAIN.
MODULE trans_from_100.
MODULE save_ok_code.
MODULE user_command_100.
```

Top include

```
fields for mode choice
DATA: view VALUE 'X', maintain_flights, maintain_bookings,
mode(20).
flags for update
DATA:  planetype_changed.
```

Event include

```
AT LINE-SELECTION.
CALL SCREEN 100.
update list of flights if necessary
IF NOT PLANETYPE_CHANGED IS INITIAL.
CLEAR PLANETYPE_CHANGED.
```

```
PERFORM READ_FLIGHTS.
PERFORM DISPLAY_FLIGHTS.
SY-LSIND = SY-LSIND - 1.
ENDIF.
```

Subroutine include

```
*&-----------------------------------------------------------------*
*&      Form  UPDATE_SFLIGHT
*&-----------------------------------------------------------------*
FORM update_sflight.
UPDATE sflight FROM wa_sflight.
IF sy-subrc = 0.
MESSAGE s009(bc410).
ELSE.
MESSAGE a008(bc410).
ENDIF.
ENDFORM.                               " UPDATE_SFLIGHT
```

PBO module include

```
MODULE status OUTPUT.
SET PF-STATUS 'STATUS_100'.
CASE 'X'.
WHEN view.
mode = 'view'(m01).
WHEN maintain_flights.
mode = 'maintain flights'(m02).
WHEN maintain_bookings.
mode = 'maintain bookings'(m03).
ENDCASE.
SET TITLEBAR 'TITLE_100' WITH mode.
ENDMODULE.                             " STATUS  OUTPUT

*&----------------------------------------------------------------*
*&      Module  MODIFY_SCREEN  OUTPUT
*&----------------------------------------------------------------*

MODULE modify_screen OUTPUT.
CHECK NOT maintain_flights IS INITIAL.
CHECK screen-group1 = 'ADM'.
screen-input = 1.
MODIFY SCREEN.
ENDLOOP.
ENDMODULE.                             " MODIFY_SCREEN  OUTPUT
```

PAI module include

```
MODULE user_command_100 INPUT.
CASE save_ok.
WHEN 'BACK'.
...
WHEN 'SAVE'.
PERFORM update_sflight.
ENDCASE.
ENDMODULE.                             " USER_COMMAND  INPUT
*&-----------------------------------------------------------------*
*&      Module  CHECK_PLANETYPE  INPUT
*&-----------------------------------------------------------------*
MODULE check_planetype INPUT.
CLEAR planetype_changed.
SELECT SINGLE seatsmax INTO sdyn_conn-seatsmax FROM saplane
WHERE planetype = sdyn_conn-planetype.
planetype_changed = 'X'.
CHECK sdyn_conn-seatsmax < sdyn_conn-seatsocc.
MESSAGE e109(bc410).
ENDMODULE.                             " CHECK_PLANETYPE  INPUT
*&-----------------------------------------------------------------*
*&      Module  TRANS_FROM_100  INPUT
*&-----------------------------------------------------------------*
MODULE trans_from_100 INPUT.
MOVE-CORRESPONDING sdyn_conn TO wa_sflight.
ENDMODULE.                             " TRANS_FROM_100  INPUT
```

7-4 **Model solution** SAPBC410INPS_RADIOBUTTON_GROUP

Add the coding in bold type to your program. Create the new modules using forward navigation.

Top include

```
flags for update
DATA:  planetype_changed, dataloss.
```

Subroutine include

```
FORM update_sflight.
UPDATE sflight FROM wa_sflight.
IF sy-subrc = 0.
CLEAR dataloss.
MESSAGE s009(bc410).
ELSE.
MESSAGE a008(bc410).
ENDIF.
ENDFORM.                               " UPDATE_SFLIGHT
```

PAI module include

```
MODULE user_command_100 INPUT.
CASE save_ok.
WHEN 'BACK'.
IF dataloss IS INITIAL.
LEAVE TO SCREEN 0.
ELSE.
CALL FUNCTION 'POPUP_TO_CONFIRM_STEP'
EXPORTING
textline1 = text-e05
titel     = text-e06
IMPORTING
answer    = answer.
CASE answer.
WHEN 'J'.                    " J = Yes
PERFORM update_sflight.
... release database locks
LEAVE TO SCREEN 0.
WHEN 'N'.                    " N = No
CLEAR dataloss.
```

```
LEAVE TO SCREEN 0.
ENDCASE.
ENDIF.
...
WHEN 'SAVE+EXIT'.
PERFORM update_sflight.
LEAVE PROGRAM.
ENDCASE.
ENDMODULE.                             " USER_COMMAND  INPUT

MODULE exit INPUT.
IF sy-datar IS INITIAL AND dataloss IS INITIAL.
CASE ok_code.
WHEN 'CANCEL'.
CLEAR ok_code.
LEAVE TO SCREEN 0.
WHEN 'EXIT'.
LEAVE PROGRAM.
ENDCASE.
ELSE.
CASE ok_code.
WHEN 'EXIT'.
CALL FUNCTION 'POPUP_TO_CONFIRM_STEP'
EXPORTING
textline1 = text-e01
titel     = text-e02
IMPORTING
answer    = answer.

CASE answer.
WHEN 'J'.                    " J = Yes
Do not save here, but during "normal" ok-code processing.
Remember that all screen checks will be executed after this
module. Saving and LEAVE TO SCREEN 0 must be coded after all checks!
ok_code = 'SAVE+EXIT'.
WHEN 'N'.                    " N = No
... release all database locks
LEAVE PROGRAM.

WHEN OTHERS.                 " Cancel cancelled :-)
CLEAR ok_code.
```

```
ENDCASE.

WHEN 'CANCEL'.
CALL FUNCTION 'POPUP_TO_CONFIRM_LOSS_OF_DATA'
EXPORTING
textline1 = text-e03
titel     = text-e04
IMPORTING
answer    = answer.

IF answer = 'J'.               " J = Yes
Release all database locks
CLEAR dataloss.
CLEAR ok_code.
LEAVE TO SCREEN 0.
ENDIF.
ENDCASE.
ENDIF.
ENDMODULE.                             " EXIT  INPUT

MODULE check_planetype INPUT.
CLEAR planetype_changed.
SELECT SINGLE seatsmax INTO sdyn_conn-seatsmax FROM saplane
WHERE planetype = sdyn_conn-planetype.
planetype_changed = 'X'.
dataloss = 'X'.
CHECK sdyn_conn-seatsmax < sdyn_conn-seatsocc.
MESSAGE e109(bc410).
ENDMODULE.                             " CHECK_PLANETYPE  INPUT
```

Screen Elements: Subscreens and Tabstrip Controls

Contents:

- Subscreens
- Tabstrip controls

■ Unit 1	Course Overview
■ Unit 2	Basics for Interactive Lists
■ Unit 3	The Program Interface
■ Unit 4	Interactive List Techniques
■ Unit 5	Introduction to Screen Programming
■ Unit 6	Screen Elements for Output
■ Unit 7	Screen Elements for Input/Output
■ Unit 8	**Screen Elements: Subscreens and Tabstrip Controls**
■ Unit 9	Screen Elements: Table Controls
■ Unit 10	Context Menus on Screens
■ Unit 11	Lists in Screen Programming
■ Unit 12	Preview: Control Framework

- A subscreen area is a reserved rectangular area on a screen, into which you place another screen at runtime. Subscreen areas may not contain any other screen elements. To use a subscreen, you create a second screen (with the type subscreen), and display it in the subscreen area you defined on the main screen.
- A subscreen is an independent screen that you display within another screen. You may want to use a subscreen as a way of displaying a group of objects in certain circumstances, but not in others. You can use this technique to display or hide extra fields on the main screen, depending on the entries the user has made.

- A second use for subscreens is that different programs can use the same subscreens. To do this, you must execute other screen programs within your main program.
- You can include more than one subscreen on a single main screen. You can also determine the subscreens dynamically at runtime.
- You can use subscreens in the following circumstances:
  - In screen enhancements (screen exits),
  - Within other screen objects (tabstrip control)
  - In the Modification Assistant
  - In Web transactions.

- If the subscreen is larger than the subscreen area in which it is called, the system only displays as much of it as will fit onto the screen. However, you can use the *Scrollable* attribute to ensure that, if the screen is too big, the system will display scrollbars.
- The resizing attributes control whether the size of a subscreen area can be changed vertically and horizontally. You should set these attributes if you want the size of the subscreen area to change with the size of the whole window. You can use the minimum size attribute to set a lower limit beyond which the subscreen area cannot be resized.
- The *Context menu* attribute allows you to assign a context-sensitive menu to the output fields on the subscreen screen.
- The following restrictions apply to subscreens:

  `CALL SUBSCREEN ...` is not allowed between `LOOP` and `ENDLOOP` or between `CHAIN` and `ENDCHAIN`.

  A subscreen may not have a named `OK_CODE` field.

  Object names must be unique within the set of all subscreens called in a single main screen.

  Subscreens may not contain a module with the `AT EXIT-COMMAND` addition.

  You cannot use the `SET TITLEBAR`, `SET PF-STATUS`, `SET SCREEN`, or `LEAVE SCREEN` statements in the modules of a subscreen.

- To create a subscreen area, choose subscreen from the object list in the Screen Painter and place it on the screen. Fix the top-left hand corner of the table control area, and then drag the object to the required size.
- In the *Object text* field, enter a name for the subscreen area. You need this to identify the area when you call the subscreen.

- To use a subscreen, you must call it in both the PBO and PAI sections of the flow logic of the main screen. The `CALL SUBSCREEN <subarea>` statement tells the system to execute the PBO and PAI processing blocks for the subscreen as components of the PBO and PAI of the main screen. You program the ABAP modules for subscreens in the same way as for a normal screen (apart from the restrictions already mentioned).

- If the subscreen is not in the same module pool as the "main program", the global data of the main program is not available to the subscreen, and the data from the screen will not be transferred back to the program. You must program the data transfer yourself (for example, using a function module that exports and imports data, with an appropriate MOVE statement in the subscreen coding).

- If you want to use subscreens in the screens of several different programs, you should encapsulate the subscreens in a function group and use function modules to transport data between the program in which you want to use the subscreen and the function group.
- You can pass data between the calling program and the function group using the interfaces of the function modules.
- This is the technique used for customer subscreens (screen enhancements).

- You use function modules to transport data between the calling program and the function group.
- To declare the data from the calling program to the subscreen of the function group, use a module before the subscreen call. This should call a function module whose interface you can use to pass the required data to the function group.
- The function module call must occur before the subscreen call. This ensures that the data is known in the function group before the `PROCESS BEFORE OUTPUT` processing block of the subscreen is called.
- In the PAI module of the calling screen, the sequence is reversed: You call the `PROCESS AFTER INPUT` processing block of the subscreen before calling a function module to pass the data from the function group back to the calling program.

- For the data from the calling program to be available globally in the function group, you must transfer the interface parameters from the function module into global data fields of the function group.
- The function module that you use to transfer the data from the calling program into the function group must copy its interface parameters into the global data in the function group.
- The function module that you use to transfer data from the function group to the calling program must copy the corresponding data from the global data of the function group into its interface parameters.

- Tabstrip controls provide you with an easy, user-friendly way of displaying different components of an application on a single screen and allowing the user to navigate between them. Their intuitive design makes navigation much easier for end users.
- Tabstrip controls are a useful way of simplifying complex applications. You can use tabstrip controls wherever you have different components of an application that form a logical unit. For example, you might have a set of header data which remains constant, while underneath it, you want to display various other sets of data.
- You should **not** use tabstrip controls if

  You need to change the screen environment (menus, pushbuttons, header data, and so on) while processing the application components. The screen surrounding the tabstrip must remain constant.

  The components must be processed in a certain order. Tabstrips are designed to allow users to navigate freely between components.

  The components are processed dynamically, that is, if user input on one tab page leads to other tab pages suddenly appearing.
- Tabstrip controls are compatible with batch input processing.

- A tabstrip control consists of individual pages. These consist of the page area and the tab title.
- The tab may only have one row of tab titles.
- If the tabstrip control contains too many pages, it will not be possible for all of the tab titles to be displayed at once. If this happens, the system displays a scrollbar with which you can scroll through the remaining tab pages. In the top right-hand corner of the tab is a pushbutton. If the user clicks this, a list of all of the tab titles is displayed. The active tab title is marked with a tick.

- A tab page consists of a tab title, a subscreen area, and a subscreen.
- From a technical point of view, the system handles tab titles like pushbuttons.
- The contents of tab pages are displayed using the subscreen technique. You assign a subscreen area to each tab page, for which you can then call a subscreen.

- As well as the general "Object name", "Starting position" and static size attributes, tabstrip controls also have special tabstrip attributes.
- For details of these special attributes, see the section on *subscreen* attributes.

- You create a tabstrip control in the following three steps:
  - Define the tab area
  - Define the tab titles and, if necessary, add further tab titles
  - Assign a subscreen area to each tab page.

- To create a tabstrip area, choose Tabstrip from the object list in the Screen Painter and place it on the screen. Fix the top-left hand corner of the table control area, and then drag the object to the required size.
- Assign a name to the tabstrip control in the "Object name" attribute. You need this name to identify your tabstrip control.
- In your ABAP program use the `CONTROLS` statement to declare an object with the same name. Use `TABSTRIP` as the type.
- The type `TABSTRIP` is defined in the type pool `CXTAB`. The field `ACTIVETAB` contains the function code of the tab title of the currently active tab page. The other fields are reserved for internal use.
- The default number of tab pages for a tabstrip control is two.

- Technically, tab titles are treated in the same way as pushbuttons. They have an object name, a text, a function code, and a function type. You enter these in the "Object name", "Object Text", "FctCode" and "FctType" fields of the obje ct attributes.
- A tab title can have the function type ' ' (space) or 'P'. If the function type is ' ' (space), the PAI processing block is triggered when the user chooses that tab, and the function code of the tab title is placed in the `OK_CODE` field. If the function type is 'P', the user can scroll between the different tab pages with the same type without the PAI processing block being triggered. For further details, refer to the following pages.
- If you want your tabstrip control to have more than two pages, you must create further tab titles. To do this, choose *Pushbutton* from the object list in the Screen Painter and place it in the tab title area.

- You must assign a subscreen area to each tab page.
- The subscreen area assigned to a tab page is automatically entered as the "Reference object" (in the *Dictionary* attributes) for the **tab title** of that page.
- To assign a subscreen area to one or more tab pages, choose the relevant tab title in the fullscreen editor, choose the *Subscreen* object, and place it on the tab page.
- Alternatively, you can assign a single subscreen area to several tab pages by entering the name of the subscreen area directly in the "Reference object" field of the attributes of the relevant tab pages.

- If you have assigned a different subscreen area to each tab page in a tabstrip control, you can scroll between the pages locally at the frontend.
- To do this, you must send all of the subscreens to the front end when you send the main screen itself. All of the tab titles in the tabstrip control must also have function type 'P'.
- Now, when you scroll between the different tab pages, there is no communication between the presentation server and the application server.
- When the user chooses a function on the screen that triggers PAI processing, the system processes the PAI blocks of **all** of the subscreens as well. This means that **all of the field checks** are run. In this respect, you could regard the tabstrip control as behaving like a single large screen.
- Local scrolling in tabstrip controls is more appropriate for display transactions.

- To program a tabstrip control to scroll locally at the front end, you must:
  - Assign a separate subscreen area to each tab page; a subscreen will be sent to each of these when the screen is processed.
  - Call all of the subscreens from the flow logic.
  - Assign function code type 'P' to all of the tab titles.
- The system hides any tab page whose subscreen contains no elements that can be displayed.
- If there are no tab pages containing elements that can be displayed, the system hides the entire tabstrip control.
- For further information about tabstrip controls, follow the appendix documentation path **SUB-2**.

- If all of the tab pages share a single subscreen area, the program analyzes the function code of the chosen tab title to determine which screen is displayed.
- There are two steps in this process:
  - In the PAI processing block, the program determines which tab page needs to be active, based on the tab title chosen by the user.
  - When the PBO processing block is processed again, the program displays the corresponding screen.
- During this process, the system only checks the fields of the subscreen that is actually displayed.

- If you want the application program to process scrolling in a tabstrip control,
  - All of the tab pages must share a common subscreen area
  - All of the tab titles must have the function code type `' '` (space), and
  - In the flow logic, you must use a variable to call the screen that is to be displayed in the subscreen area.
- In the PAI block, you must call a module in which the function code of the active tab title is placed in the field `ACTIVETAB` of the structure you created in your program with type `TABSTRIP`. In the example above, this is `MY_TAB_STRIP`.
- The PBO processing block must contain a module, before the subscreen is called, in which you place the number of the subscreen in the corresponding variable. In order for the screen to be processed the first time (before the user has had a chance to choose a tab title), you must assign an initial value to this field.
- You can hide a tab page at runtime by setting the corresponding tab title to inactive using the system table `SCREEN` (`SCREEN-ACTIVE = '0'`). You should do this before processing the tabstrip control for the first time, to ensure that the screen environment remains constant.

- You can now create tabstrip controls on selection screens. They allow you to create logical groups of fields, and make large selection screens more user-friendly.

- The following requirements must be met if you are to use selection screens with tabstrip controls in your R/3 System:

  - GUI version 4.0 or higher

  - Frontend: Motif, Windows 95, MacOs, NT 3.51 or higher.

- For a selection screen with tabstrips, you must define:

  - A subscreen area on the selection screen to accommodate the tabstrip control

  - The individual tab titles

  - Selection screens as subscreens for the individual tab pages

- Since it is possible to define selection screens as subscreens, you can include selection fields that you create in this way in any other screens. Selection screens as subscreens are processed similarly to other screens.

- You define a selection screen as a subscreen as follows:

```
SELECTION-SCREEN BEGIN OF SCREEN <scrn> AS SUBSCREEN
    [NESTING LEVEL <m>] [NO INTERVALS].
    ...
SELECTION-SCREEN END OF SCREEN <scrn>.
```

  Optional additions: `[NESTING LEVEL <m>].` Each box around a tabstrip control increases the `NESTING LEVEL` by one.
  `[NO INTERVALS].` This option hides the HIGH fields for any selection criteria defined using SELECT-OPTIONS on the screen.

- You define a subscreen area for a tabstrip control on a selection screen as follows: SELECTION-SCREEN BEGIN OF TABBED BLOCK <blockname> FOR <n> LINES.
  SELECTION-SCREEN END OF BLOCK <blockname>.
  The size of the subscreen area in lines is defined by <n>.

- The system automatically generates a CONTROLS statement (CONTROLS: TABSTRIP_BLOCKNAME TYPE TABSTRIP.) You must not write your own CONTROLS statement. If you try to do so, a syntax error results.

- You define the individual tab pages as follows:
  SELECTION-SCREEN TAB (length) <name> USER-COMMAND <ucomm> [DEFAULT [PROGRAM <prog>/SCREEN <dynnr>]].
  Optional additions: [DEFAULT [PROGRAM <prog>/SCREEN <dynnr>]].
  Assign the selection screen to a tab page. If you use the DEFAULT addition, you must also use the SCREEN addition. The PROGRAM addition is optional. You only need it if the screen comes from another program.

- You can delay specifying the link between the tab title and the selection screen until runtime. You can also change an existing assignment at runtime. To do this, fill the structure blockname. This is created automatically for every tabstrip block. The structure has the same name as the tabstrip block, and contains the fields PROG, DYNNR, and ACTIVETAB. For further information, refer to the online documentation in appendix reference **SUB-2**.

- If you define a selection screen as a subscreen, you can display it on a normal screne or in a tabstrip control that is embedded in a normal screen.
- All you need to do is define the selection screen as a subscreen with the relevant selection options and input parameters. You can then define a subscreen area on the screen and embed the subscreen screen in it, by calling the subscreen screen in the PBO event and, if necessary, in the PAI event as well.

Unit: Subscreen and Tabstrip Control

Theme: Creating subscreens and tabstrip controls

At the conclusion of these exercises, you will be able to:

- Use subscreens and tabstrips on screens and selection screens in your programs

Display additional information on your screen, depending on the mode in which the user is working.
Extend the display to allow users to switch between the additional information using a tabstrip control.

8-1 Extend the “Flight data” screen (100) to display flight information and the aircraft type. Use a subscreen to do this.

8-1-1 Extend your program **Z##BC410_SOLUTION** from the previous exercise (or copy the model solution **SAPBC410DIAS_DYNPRO**). You can use the model solution **SAPBC410SUBS_SUBSCREEN** for orientation.

8-1-2 On the “Flight data” screen (100), create a subscreen area with the following attributes:

Subscreen	SUB	**Attributes:** Vert. And horiz. resizing: On

8-1-3 Create two screens 110 and 120, each with the type subscreen and the following attributes:

Screen 110	**I/O fields, text fields:** ZDYN_CONN## - COUNTRYFROM - COUNTRYTO - CITYFROM - CITYTO - AIRPFROM - AIRPTO - DEPTIME - ARRTIME	**For each field:** **Input:** Off **Output:** On
Screen 120	**I/O fields, text fields:** SAPLANE - PLANETYPE	**For each field:** **Input:** Off **Output:** ON

	- PRODUCER - SEATSMAX - TANKCAP - CAP_UNIT - WEIGHT - WEI_UNIT - OP_SPEED - SPEED_UNIT	**Output only:** On

8-1-4 **a)** In your TOP include, create a field **DYNNR** that you can use in the flow logic to determine which subscreen should appear in the subscreen area.

8-1-5 Call the subscreen screens in the flow logic of screen 100. Before the call, write a PBO module to determine which of the subscreens will appear. If the user is in “Display” mode, call subscreen screen 110 with the flight information. If the user is in “Maintain flight data” mode, call subscreen screen 120 with the aircraft information.

8-1-6 In the flow logic of screen 110, read the flight information from table SPFLI using the key field values.

8-1-7 In the flow logic for screen 120, read the information for the aircraft information from table SAPLANE using the value you have for the aircraft type.

8-2 Create a tabstrip control on screen 100 for displaying extra flight information and details of the aircraft type.

8-2-1 Extend your program **Z##BC410_SOLUTION** from the previous exercise (or copy the model solution **SAPBC410SUBS_SUBSCREEN**). You can use the model solution **SAPBC410SUBS_TABSTRIP** for orientation.

8-2-2 Create the tabstrip control: Remove the subscreen area on screen 100 and create a tabstrip control with the following attributes:

Tabstrip control	**Name:** MY_TABSTRIP	**Attributes:** Vert. And horiz. resizing: On
Pushbutton (Tab title 1)	**Name:** P1	**Attributes:** **Text:** Display flight data **Function code:** FC1 **Function type:** <blank> **Reference field:** SUB
Pushbutton (Tab title 2)	**Name:** P2	**Attributes:** **Text:** Display technical data for aircraft **Function code:** FC2

		Function type: <blank> **Reference field:** SUB
Pushbutton (Tab title 3)	**Name:** P3	**Attributes:** **Text:** Maintain bookings **Function code:** FC3 **Function type:** <blank> **Reference field:** SUB

In the TOP include of your program, create a data object for the tabstrip control using the following statement:
`CONTROLS MY_TABSTRIP ...`

8-2-3 In the flow logic of screen 100, implement the call for the subscreen screen in the tabstrip control.

8-2-4 Before calling the subscreen, write a PBO module in which you determine which of the subscreens is to be called (regardless of the mode in which the user is working). Additionally, determine which subscreen screen you want to set the first time the screen is displayed, and assign the corresponding function code to the field **`MY_TABSTRIP-ACTIVETAB`**.

8-2-5 Extend your function code processing for screen 100 to include scrolling logic for the first two pages of the tabstrip control. Do this by assigning the relevant value to **`MY_TABSTRIP-ACTIVETAB`**.

8-2-6 You will create the subscreen for the third tab page and program its scrolling logic in a later exercise.

Unit: Subscreen and Tabstrip Control

Theme: Creating subscreens and tabstrip controls

8-1 **Model solution SAPBC410SUBS_SUBSCREEN**

Add the coding in bold type, and create new modules where appropriate using forward navigation.

SCREEN 100

```
PROCESS BEFORE OUTPUT.
  MODULE STATUS.
  MODULE GET_SFLIGHT_DATA.
  MODULE MODIFY_SCREEN.
  MODULE CHOOSE_SUBSCREEN_DYNPRO.
  CALL SUBSCREEN SUB INCLUDING SY-CPROG DYNNR.

PROCESS AFTER INPUT.
  ...
  MODULE USER_COMMAND_100.
```

SCREEN 110

```
PROCESS BEFORE OUTPUT.
  MODULE GET_SPFLI.

PROCESS AFTER INPUT.
```

SCREEN 120

```
PROCESS BEFORE OUTPUT.
  MODULE GET_SAPLANE.

PROCESS AFTER INPUT.
```

Module pool

Add the following to the ABAP program:

Top include

```
TABLES:  sdyn_conn, saplane.
* screen for subscreen
DATA  dynnr LIKE sy-dynnr.
```

PBO modules

```
*&---------------------------------------------------------*
*&      Module  CHOOSE SUBSCREEN DYNPRO  OUTPUT
```

```
*&---------------------------------------------------*
MODULE CHOOSE_SUBSCREEN_DYNPRO OUTPUT.
 CASE 'X'.
   WHEN VIEW.
     DYNNR = '0110'.
   WHEN MAINTAIN_FLIGHTS.
     DYNNR = '0120'.
 ENDCASE.
ENDMODULE.            " CHOOSE_SUBSCREEN_DYNPRO  OUTPUT

*&---------------------------------------------------*
*&      Module  GET_SPFLI  OUTPUT
*&---------------------------------------------------*
MODULE GET_SPFLI OUTPUT.
  SELECT SINGLE * INTO CORRESPONDING FIELDS OF
    SDYN_CONN FROM SPFLI
    WHERE CARRID = WA_SFLIGHT-CARRID
      AND CONNID = WA_SFLIGHT-CONNID.
ENDMODULE.                 " GET_SPFLI  OUTPUT

*&---------------------------------------------------*
*&      Module  GET_SAPLANE  OUTPUT
*&---------------------------------------------------*
MODULE GET_SAPLANE OUTPUT.
  SELECT SINGLE * FROM SAPLANE
    WHERE PLANETYPE = WA_SFLIGHT-LANETYPE.
ENDMODULE.            " GET_SAPLANE  OUTPUT

-------------------------------------------------------
-------------------------------------------------------
```

8-2 Model solution SAPBC410SUBS_TABSTRIP

Add the coding in bold type, creating new modules where appropriate using forward navigation.

SCREEN 100

```
PROCESS BEFORE OUTPUT.
  MODULE STATUS.
  MODULE GET_SFLIGHT_DATA.
  MODULE MODIFY_SCREEN.
  MODULE FILL_DYNNR.
  CALL SUBSCREEN SUB INCLUDING SY-CPROG DYNNR.

PROCESS AFTER INPUT.
  ...
  MODULE USER_COMMAND_100.

------------------------------------------------------
```

Module pool

Add the following to your ABAP program:

Top include

```
* definition of tabstrip control structure
CONTROLS my_tabstrip TYPE TABSTRIP.
```

PBO modules

```
*&-------------------------------------------------*
*&      Module  FILL_DYNNR  OUTPUT
*&-------------------------------------------------*
MODULE FILL_DYNNR OUTPUT.
  CASE MY_TABSTRIP-ACTIVETAB.
    WHEN 'FC1'.
      DYNNR = '0110'.
    WHEN 'FC2'.
      DYNNR = '0120'.
*    WHEN 'FC3'.
*      DYNNR = '0130'.
    WHEN OTHERS.
      MY_TABSTRIP-ACTIVETAB = 'FC1'.
      DYNNR = '0110'.
  ENDCASE.
ENDMODULE.                   " FILL_DYNNR  OUTPUT
```

PAI modules

```
*&-------------------------------------------------*
*&      Module  USER_COMMAND  INPUT
*&-------------------------------------------------*
MODULE USER_COMMAND_100 INPUT.
  CASE SAVE_OK.
    WHEN 'FC1' OR 'FC2'.           "OR 'FC3'.
      MY_TABSTRIP-ACTIVETAB = SAVE_OK.
  ENDCASE.
ENDMODULE.               " USER_COMMAND  INPUT
```

Screen Element: Table Controls

Contents:

- Table control overview
- Creating a table control
- Processing a table control
- Other techniques

- Unit 1 Course Overview
- Unit 2 Basics for Interactive Lists
- Unit 3 The Program Interface
- Unit 4 Interactive List Techniques
- Unit 5 Introduction to Screen Programming
- Unit 6 Screen Elements for Output
- Unit 7 Screen Elements for Input/Output
- Unit 8 Screen Elements: Subscreens and Tabstrip Controls
- Unit 9 **Screen Elements: Table Controls**
- Unit 10 Context Menus on Screens
- Unit 11 Lists in Screen Programming
- Unit 12 Preview: Control Framework

- A table control is an area on the screen in which the system displays data in tabular form. It is processed using a loop. The top line of a table control is the header line, which is distinguished by a gray separator.
- Within a table control, you can use table elements, key words, templates, checkboxes, radio buttons, radio button groups, and pushbuttons. A line may have up to 255 columns; each column may have a title.

- You can display or enter single structured lines of data using a table control.
- Features:
  - Resizeable table for displaying and editing data.
  - The user or program can change the column width and position, save the changes, and reload them later.
  - Check column for marking lines. Marked lines are highlighted in a different color.
  - Line selection: Single lines, multiple lines, all lines, and deselection
  - Column headings double as pushbuttons for marking columns.
  - Scrollbars for horizontal and vertical scrolling.
  - You can fix any number of key (leading) columns.
  - Cell attributes are variable at runtime.

- Users can save display variants for table controls. These variants can be saved by each user, along with the basic setting, as the current display setting or as the default display setting.

- The table control contains a series of attributes that are controlled entirely at the presentation server: These are:
  - Horizontal scrolling using the scrollbar in the table control
  - Swapping columns
  - Changing column widths
  - Marking columns
  - Marking lines
- The PAI processing block is triggered when you scroll vertically in the table control or save the user configuration.

- As well as the normal "Object name", "Start position on screen" and "Static size" attributes, table controls also have special table control attributes.
- The "Special table control attributes" determine the table type and display options for a table control, as well as whether it can be configured by the user. The fields `stepl` and `loopc` of structure `syst` contain information about the loop processing used with table controls (see following pages).
- For further information about the static attributes, refer to the online documentation.
- For more information about the dynamically changeable attributes, refer to the online documentation in appendix reference **TAB-1**.

- When you create a table control, you must create:
  - A table control area.
  - Table control fields.

- To create a table control area, choose the table control object from the object list in the Screen Painter and place it in the screen work area. Fix the top-left hand corner of the table control area, and then drag the object to the required size.
- In the "Object name" attribute, assign a name to your table control. In the ABAP program, declare a structure with the same name, containing the dynamically changeable attributes of the table control.
- The `CONTROLS` statement declares a complex data object with the type `TABLEVIEW` (corresponding to the type `CXTAB_CONTROL`, declared in type group `CXTAB` in the ABAP Dictionary). At runtime, the data object (`my_control`) contains the static attributes of the table control.
- You maintain the initial values (static attributes) in the Screen Painter. The `USING SCREEN` addition in the `CONTROLS` statement determines the screen whose initial values are to be used for the table control.
- You can reset a table control to its initial attributes at any time using the statement **`REFRESH CONTROL <ctrl> FROM SCREEN <scr>`**. `<scr>` does not have to be the same as the initial screen of the table control.

- You create fields in a table control using the *Dict./Program fields* function. This involves the following steps:
  - Enter the name of the structure whose fields you want to use in the table control and press ENTER.
  - In the field list, choose the fields that you want to use and choose OK.
  - Position the cursor in the table control area and click the left mouse button.

    The system places all of the selected fields in the table control. If the fields have data element texts, the system uses these as column headings.
- Alternatively, you can position individual input/output fields in the table control area, each of which generates a single column.

- When you create a table control, the system automatically proposes one with a selection column.
- The selection column behaves like a checkbox. It must therefore be a field with length 1 and data type CHAR. You must enter the field name in the attributes of the table control.
- The selection column is a field of the structure used for transport between the screen and the ABAP program.

- The runtime attributes of a table control, which are stored in the structure declared using the CONTROLS statement, can be divided into **general attributes** and **column attributes**.
- The **general attributes** contain information about the attributes of the table control as a whole, such as, for example, the number of fixed columns.
- The **column attributes** are stored in an internal table (one entry per column of the internal table). For each column, it maintains the attributes stored in the structure SCREEN, plus the special table control column attributes *column position*, a *selection* and a *visibility* flag, and a field for the *displayed width*.
- For information about the names of the attributes and their precise meanings, refer to the keyword documentation in the ABAP Editor for the *CONTROLS* statement (then choose *Tableview* -> *CXTAB_CONTROL*), and the online documentation in appendix reference **TAB-1**.

- You can change a table control dynamically by modifying the contents of the fields in the table control structure declared in your program.
- The fields of the table control structure also provide information about user interaction with the table control. For example, you can use the selected field to determine whether the user has selected a particular column.

- For performance reasons, you read the data for the table control once from the database and store it in an internal table using an array fetch.
- The system fills the table control lines from this internal table.

- Before you can display data from an internal table in a table control, you must first fill the table. Make sure that you do not fill the internal table in every PBO event, but only when the key fields change (in the above example, airline and flight number).
- For the table control processing, you need to know how far the user can scroll vertically (the size of the internal table). You should therefore use the `DESCRIBE TABLE` statement to find out the number of entries in the internal table, and save this in the `LINES` field of the table control.
- There is only one work area for processing lines in the table control. For this reason, you need a `LOOP` … `ENDLOOP`. structure in both the PBO and PAI events for each table control.
- In the PBO processing block, you must fill one line of the table control with the corresponding line from the internal table in each loop pass.
- Similarly, in the PAI processing block, you must pass the changes made in the table control back to the correct line of the internal table.
- When you process functions, you must distinguish between those that should only apply to individual lines of a table control, and those that should apply to the entire screen.

- There are three steps involved in displaying buffered data from the internal table in the table control:
  - The system loops through the lines of the table control. The lines of the screen table are processed one by one. For each line, the system carries out the following steps:

    The current line of the internal table is placed in the work area of the internal table. (Note that it is possible to scroll in the table on the screen.)

    The data from the work area of the internal table is copied into the relevant line of the table control.

- When you use table controls on a screen, the field transport sequence changes.
- In the PBO processing block, data is transferred from the ABAP program to the screen after each loop pass in the flow logic. The rest of the screen fields are filled, as normal, at the end of the PBO.

- The loop statement in the flow logic `LOOP AT <itab> INTO <wa_itab> WITH CONTROL <tc_name>`
  starts a loop through the screen table, and reads the line of the internal table corresponding to the current line of the screen table, placing it in `<wa_itab>`.
  `<itab>` is the name of the internal table containing the data, `<wa_itab>` is the name of the work area for the internal table, and `<tc_name>` is the name of the table control on the screen.
- If the fields in your table control have the same structure and name as those in the work area `<wa_itab>`, the system can transport data between the ABAP program and the screen automatically (step 3).
- If you are not using the same structure for the table control fields and the work area of the internal table, you must call a module between `LOOP` and `ENDLOOP` that moves the data from the work area `<wa_itab>` into the screen fields (`MOVE-CORRESPONDING <wa_itab> TO` ...) .
- The system calculates the value of `<ctrl>-TOP_LINE` when you scroll, but not when you scroll a page at a time outside the table control.

- Transferring changed values from the table control back to the internal table involves the following three steps:
  - The system loops through the lines of the table control. The lines of the screen table are processed one by one. For each line, the system carries out the following steps:

    The data from the current line of the table control is copied into the header line of the internal table.

    The data in the work area must then be placed in the line of the internal table corresponding to the line of the table control that is being processed. (Note that it is possible to scroll in the table on the screen.)

- In the PAI processing block, all screen fields that do not belong to a table control and that are not listed in a `FIELD` statement are transported back to the work fields in the ABAP program first.
- The contents of the table control are transported line-by-line to the corresponding work area in the ABAP program in the appropriate loop.
- Lastly, the fields that occur in `FIELD` statements are transported directly before the corresponding statement.

- The `LOOP AT <itab>. ... ENDLOOP` block processes a loop through the lines of the table on the screen.
- If the fields on your screen have the same names as the fields in the internal table, you must return the data from the header line of the internal table to the body of the table itself. You do this using the field `<control>-current_line`.
- If the fields on your screen do not have the same names as the fields in the internal table, you must first copy the data into the header line of the internal table. You can then copy the data back into the internal table itself. You can also use the field `<control>-current_line` to do this.

- You can modify the attributes of a table control by overwriting the field contents of the structure created in the CONTROLS statement.
- To change the attributes of individual cells temporarily (!), change the table SCREEN in a PBO module that you processes between LOOP and ENDLOOP in the flow logic (LOOP AT SCREEN, MODIFY SCREEN).
- In the LOOP, the runtime system initializes the attributes set statically for the table control in the Screen Painter. You can only change these in a module called from a loop through the table control.

- You can change a table control dynamically by modifying the contents of the fields of its structure.
- The fields of the table control structure also provide information about user interaction with the table control. For example, you can use the selected field to determine whether the user has selected a particular column.

- You can change a table control dynamically by modifying the contents of the fields of its structure.
- The fields of the table control structure also give you information about user interaction with the table control. For example, you can use the selected field to determine whether the user has selected a particular column.

- It is possible to change the attributes of table control fields temporarily. These changes are only effective while the current screen is being processed.
- To do this, you call a module from within the table control loop in the flow logic, in which you change the attributes of the current line.
- To change the attributes of the fields of a line in the table control, use a `LOOP AT SCREEN. ... ENDLOOP.` block to loop through the fields of the current line. Within this loop, you can change the attributes of the fie lds of the current line of the table control.

- You can easily sort the table control display by a particular column using the table control attribute `<ctrl>-selected` and `<ctrl>-screen-name`

- You can scroll a page at a time in a table control using the table control attribute `<ctrl>-top_line`.
- In the PAI processing block, you need to know the current number of lines in the corresponding table control.
- The system field `sy-loopc` contains the number of table control lines in the PBO processing block. However, in the PAI, it contains the number of filled lines.
- `Sy-loopc` is only filled between `LOOP` and `ENDLOOP`, since it always refers to the current loop.
- Note that you must catch any overflow or shortfall yourself (see processing above for 'F22').
- See also the function module `SCROLLING_IN_TABLE`.

- The `LINE` parameter in the `GET` or `SET` statement refers to the system field `sy-stepl`, the special loop index in the flow logic.
- You calculate the internal table line that corresponds to the selected table control line as follows: `Line = <ctrl>-TOP_LINE + cursor position - 1.`
- The `GET CURSOR` statement sets the return code as follows: `sy-subrc` = 0. The cursor was on a field. `sy-subrc` = 4: The cursor was not positioned on a field.
- If you use a step loop on your screen, you can place the cursor on a particular element within the step loop block. To do this, use the `LINE` parameter and enter the line on which the cursor should be positioned: `SET CURSOR FIELD <field_name> LINE <line>.`
- You can also use the `OFFSET` and `LINE` parameters together.

Unit: Table control

Theme: Creating a table control

At the conclusion of these exercises, you will be able to:

- Use a table control and its processing logic in your program

On the third page of your tabstrip control, create a table control in which you can maintain bookings for your flight.

9-1 For the third page of your tabstrip control, create a table control containing the booking information for a flight.

9-1-1 Extend your program **Z##BC410_SOLUTION** from the previous exercise (or copy the model solution **SAPBC410SUBS_TABSTRIP**). You can use the model solution **SAPBC410SUBS_TABLE_CONTROL1** for orientation.

9-1-2 Create a subscreen screen 130: Call the subscreen screen for the third page of your tabstrip control.

9-1-3 **Creating the table control area:**
On subscreen screen 130, create a table control with the following attributes:

Table control	**Name:** **`MY_TABLE_CONTROL`**	**Attributes:** **Vert. And horiz. Resizing:** ON **Vert. and horiz. separators:** ON **Column selection:** SINGLE **Line selection:** MULTIPLE **Column headers:** On **Configurable:** ON **Selection column:** **`SDYN_BOOK-MARK`** **No. of fixed columns:** 2

Note: You cannot set the *number of fixed columns* attribute for the table control until you have created all of its columns
In the TOP include, create a complex data object for the attributes of your table control: **`CONTROLS: MY_TABLE_CON`**

Creating the table control columns: Use the following structure fields to create the columns in the table control:

Input/output field
Text field (in table control)
	Name:
	`SDYN_BOOK`
	`- BOOKID`
	`- CUSTOMID`
	`- CUSTTYPE`
	`- SMOKER`
	`- LUGGWEIGHT`
	`- WUNIT`
	`- INVOICE`
	`- CLASS`
	`- FORCURAM`
	`- FORCURKEY`
	`- LOCCURAM`
	`- LOCCURKEY`
	`- ORDER_DATE`
	`- COUNTER`
	`- AGENCYNUM`
		Attributes:
		Input: Off
		Output: On

Now enter the *number of fixed columns* in the table control attributes.

9-1-4 **Declaring the internal table:** In the TOP include of your program, create an internal table **`IT_SDYN_BOOK`**. This will buffer the bookings that you are going to display in the table control. Create the internal table with type **`STANDARD`** and no header line. Declare a suitable work area for the internal table. Use the line type **`SDYN_BOOK`** to declare both the internal table and the work area.

9-1-5 **Reading the data:** In the flow logic of screen 130, create a PBO module in which you read all of the bookings for the flight selected from the basic list that have not been canceled. The data should only be read if the flight selection from the basic list has changed. Include an appropriate query for this in the module. The **`LINES`** field in your table control requires the number of lines in your internal table. To find out the value, use the **`DESCRIBE TABLE ...`** statement.

9-1-6 **Implementing the table control:** Program a loop for the table control in both the PBO and PAI events of screen 130 (reads the entries in the internal table). **`LOOP ... ENDLOOP.`** In the PBO loop, call a module to copy data from the work area of the internal table into the screen fields In the PAI loop, call a module to copy the data from the screen into the internal table. The module should only be called for lines that have been selected on the screen (**`FIELD ... MODULE ... ON REQUEST.`**).

Note: You can test whether your changes are transferred to and from the screen properly by scrolling in the table control (selecting a line represents a change). Select a line, then scroll down and up again in the table control. If the selected entry is still selected after you have scrolled, your changes have been copied correctly from the internal table to the table control.

9-2 Implement functions for canceling a booking, selecting all unmarked table control lines, and deselecting all marked table control lines.

9-2-1 Extend your program **Z##BC410_SOLUTION** from the previous exercise (or copy the model solution **SAPBC410TABS_TABLE_CONTROL1**). You can use the model solution **SAPBC410TABS_TABLE_CONTROL2** for orientation.

9-2-2 **Creating Pushbuttons:** Create the following pushbuttons on screen 130:

Pushbutton	**Name:** SELECT_ALL	**Function code:** SELE **Function type:** <blank> **Icon:** ICON_SELECT_ALL
Pushbutton	**Name:** DESELECT_ALL	**Function code:** DSELE **Function type:** <blank> **Icon:** ICON_DESELECT_ALL
Pushbutton	**Name:** P_DELETE	**Function code:** DELE **Function type:** <blank> **Icon:** ICON_DELETE **Input:** Off **Output:** Off **Invisible:** On

9-2-2 **Implementing the functions:** Extend the `OK_CODE` processing for screen 100 to implement cancellation function. The pushbutton for canceling a booking should only appear if the user is in "Maintain bookings" mode. To do this, create a PBO module for screen 130 in which you change the attributes of pushbutton `P_DELETE` at `runtime (LOOP AT SCREEN ...)`. Ensure that the function code for the "Maintain bookings" checkbox is set to trigger PAI.
In the TOP include of your program, create a new internal table `IT_SDYN_BOOK_UPD` with type `STANDARD` and no header line, as well as a work area. Use the line type `SBOOK` to declare both the internal table and the work area. Copy the selected entries from the internal table `IT_SDYN_BOOK` into the table `IT_SBOOK_UPD`, copying the selection column flag to the field `CANCELLED`. Pass both internal tables to the function module `BC_GLOBAL_UPDATE_BOOK`. The function module makes the database changes for table `SBOOK` and the resulting changes in table `SFLIGHT`. Initialize the table `IT_SDYN_BOOK_UPD` and return to the basic list. Ensure that the changed data is displayed on the basic list straight away.

9-2-3 Extend the function code processing of screen 100 or 130 to include the functions "Select all" and "Deselect all".

9-3 In the table control, implement the functions "Scroll using standard toolbar" and "Sort".

9-3-1 Extend your program **Z##BC410_SOLUTION** from the previous exercise (or copy the model solution **SAPBC410TABS_TABLE_CONTROL2**). You can use the model solution **SAPBC410TABS_TABLE_CONTROL3** for orientation.

9-3-2 **Sorting the table control** You can sort the table control by a selected column as follows:
Creating Pushbuttons: Create the following pushbuttons on screen 130:

Pushbutton	**Name:** P_SRTU	**Function code:** SRTU **Function type:** <blank> **Icon:** ICON_SORT_UP
Pushbutton	**Name:** P_SRTD	**Function code:** SRTD **Function type:** <blank> **Icon:** ICON_SORT_DOWN

9-3-3 **Implementing the functions:** Extend the **`OK_CODE`** processing for screen 130 to implement the two sort functions. Use the table control structure **`MY_TABLE_CONTROL`** to find out the column in the table control selected by the user. You will need to write a loop for the internal table **`MY_TABLE_CONTROL-COLS`**. This means that you will also need a work area for **`MY_TABLE_CONTROL-COLS`**. Create this in the TOP include of your program (suggested name: **`WA_COLS`**). ii) Use the fields **`MY_TABLE_CONTROL-COLS-SELECTED`** and **`WA_COLS-SCREEN-NAME`** to find out the name of the column selected by the user. Since the field **`WA_COLS-SCREEN-NAME`** contains the name of the screen field (**`SDYN_BOOK-<fname>`**), you will need to find out the field name using an offset specification. Sort the internal table containing the data for the tree control by the selected field in the chosen direction.

9-3-4 **Scrolling in the table control (screen):** You can allow the user to scroll through a table control using pushbuttons as follows:
Creating the buttons in the standard toolbar: Assign the following function codes to the functions in the standard toolbar:

Button in standard toolbar	**Icon** First page	**Function code:** P-- **Function type:** <blank>
Button in standard toolbar	**Icon** Previous page	**Function code:** P- **Function type:** <blank>
Button in standard toolbar	**Icon** Next page	**Function code:** P+ **Function type:** <blank>
Button in standard toolbar	**Icon** Last page	**Function code:** P++ **Function type:** <blank>

9-3-5 **Implementing the functions:** Extend the **`OK_CODE`** processing for screen 100 to implement the scroll functions.
In order to scroll through the table control using pushbuttons, your program

needs to know how many lines can currently be displayed within the control (if the screen is of variable size, the user may resize the control). The system field **SY-LOOPC** contains this information between **LOOP** and **ENDLOOP** in the PBO. Store this value in a global field in your program, which you should declare in the TOP include (suggested name: LOOPLINES) in a module between **LOOP** and **ENDLOOP** in the PBO processing of screen 130. To find out the new value of **MY_TABLE_CONTROL-TOP_LINE**, use the function module **SCROLLING_IN_TABLE**. Pass the value of the fields **MY_TABLE_CONTROL-TOP_LINE**, **MY_TABLE_CONTRL-LINES**, **LOOPLINES** and the function code of the scrolling pushbutton to the function module. It returns the new value for **MY_TABLE_CONTROL-TOP_LINE**.

Unit: Table control

Theme: Creating a table control

9-1 **Model solution SAPBC410TABS_TABLE_CONTROL1**

Add the coding in bold type, and create new modules where appropriate using forward navigation.

SCREEN 100

```
PROCESS BEFORE OUTPUT.
  MODULE STATUS.
  MODULE GET_SFLIGHT_DATA.
  MODULE MODIFY_SCREEN.
  MODULE FILL_DYNNR.
  CALL SUBSCREEN SUB INCLUDING SY-CPROG DYNNR.
*
PROCESS AFTER INPUT.
  MODULE EXIT AT EXIT-COMMAND.
  CHAIN.
    FIELD: SDYN_CONN-CARRID,
           SDYN_CONN-CONNID,
           SDYN_CONN-FLDATE MODULE CHECK_SFLIGHT
           ON CHAIN-REQUEST.
  ENDCHAIN.
  CHAIN.
    FIELD: SDYN_CONN-PLANETYPE,
           SDYN_CONN-SEATSMAX MODULE CHECK_PLANETYPE
           ON CHAIN-REQUEST.
  ENDCHAIN.
  MODULE TRANS_FROM_100.
  CALL SUBSCREEN SUB.
  MODULE SAVE_OK_CODE.
  MODULE USER_COMMAND_100.
```

SCREEN 130

```
PROCESS BEFORE OUTPUT.
 MODULE GET_SBOOK.
  LOOP AT IT_SDYN_BOOK INTO WA_SDYN_BOOK
          WITH CONTROL MY_TABLE_CONTROL.
    MODULE MOVE_TO_DYNP.
  ENDLOOP.
*
PROCESS AFTER INPUT.
  LOOP AT IT_SDYN_BOOK.
    FIELD SDYN_BOOK-MARK MODULE UPDATE_ITAB ON REQUEST.
  ENDLOOP.
```

Module pool

Add the following coding to your ABAP program.

Top include

```
TABLES sdyn_book.
* workarea and internal table for table control
DATA: wa_sdyn_book TYPE sdyn_book,
      it_sdyn_book LIKE TABLE OF wa_sdyn_book.
* definition of table control structure
CONTROLS my_table_control TYPE TABLEVIEW
                          USING SCREEN '0130'.
```

PBO module

```
*&---------------------------------------------------------------------*
*&      Module  FILL_DYNNR  OUTPUT
*&---------------------------------------------------------------------*
MODULE FILL_DYNNR OUTPUT.
  CASE MY_TABSTRIP-ACTIVETAB.
    WHEN 'FC1'.
      DYNNR = '0110'.
    WHEN 'FC2'.
      DYNNR = '0120'.
    WHEN 'FC3'.
      DYNNR = '0130'.
    WHEN OTHERS.
      MY_TABSTRIP-ACTIVETAB = 'FC1'.
      DYNNR = '0110'.
  ENDCASE.
ENDMODULE.          " FILL_DYNNR  OUTPUT

*&---------------------------------------------------------------------*
*&      Module  GET_SBOOK  OUTPUT
*&---------------------------------------------------------------------*
MODULE GET_SBOOK OUTPUT.
  IF SDYN_CONN-CARRID <> KEY_SFLIGHT-CARRID OR
     SDYN_CONN-CONNID <> KEY_SFLIGHT-CONNID OR
     SDYN_CONN-FLDATE <> KEY_SFLIGHT-FLDATE.

    MOVE-CORRESPONDING SDYN_CONN TO KEY_SFLIGHT.

    SELECT * INTO CORRESPONDING FIELDS OF TABLE
             IT_SDYN_BOOK FROM SBOOK
         WHERE CARRID   = WA_SFLIGHT-CARRID
         AND   CONNID   = WA_SFLIGHT-CONNID
         AND   FLDATE   = WA_SFLIGHT-FLDATE
         AND CANCELLED = ' '.
    DESCRIBE TABLE IT_SDYN_BOOK LINES
             MY_TABLE_CONTROL-LINES.
  ENDIF.
ENDMODULE.            " GET_SBOOK  OUTPUT

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

```
*&------------------------------------------------------*
MODULE MOVE_TO_DYNP OUTPUT.
  MOVE-CORRESPONDING WA_SDYN_BOOK TO SDYN_BOOK.
ENDMODULE.           " MOVE_TO_DYNP  OUTPUT
```

PAI module

```
*&------------------------------------------------------*
*&      Module  USER_COMMAND  INPUT
*&------------------------------------------------------*
MODULE USER_COMMAND_100 INPUT.
  CASE SAVE_OK.
    WHEN 'FC1' OR 'FC2' OR 'FC3'.
      MY_TABSTRIP-ACTIVETAB = SAVE_OK.
  ENDCASE.
ENDMODULE.            " USER_COMMAND  INPUT

*&------------------------------------------------------*
*&      Module  UPDATE_ITAB  INPUT
*&------------------------------------------------------*
MODULE UPDATE_ITAB INPUT.
  MOVE SDYN_BOOK-MARK TO WA_SDYN_BOOK-MARK.
  MODIFY IT_SDYN_BOOK FROM WA_SDYN_BOOK INDEX
                      MY_TABLE_CONTROL-CURRENT_LINE.
ENDMODULE.            " UPDATE_ITAB  INPUT
```

1-2 **Model solution SAPBC410TABS_TABLE_CONTROL2**

Add the coding in bold type, and create new modules where appropriate using forward navigation.

SCREEN 130

```
PROCESS BEFORE OUTPUT.
 MODULE MODIFY_BUTTON.
 MODULE GET_SBOOK.
  LOOP AT IT_SDYN_BOOK INTO WA_SDYN_BOOK
          WITH CONTROL MY_TABLE_CONTROL.
    MODULE MOVE_TO_DYNP.
  ENDLOOP.
*
PROCESS AFTER INPUT.
  LOOP AT IT_SDYN_BOOK.
    FIELD SDYN_BOOK-MARK MODULE UPDATE_ITAB ON REQUEST.
  ENDLOOP.
 MODULE USER_COMMAND_0130.
```

Module pool

Add the following coding to your ABAP program.

Top include

```
* workarea and internal table for update sbook
DATA: wa_sdyn_book_upd TYPE sbook,
      it_sdyn_book_upd LIKE TABLE OF wa_sdyn_book_upd.

* flag for update
DATA  upd_flag.
```

PBO module

```
*&------------------------------------------------------*
*&      Module  MODIFY_BUTTON  OUTPUT
*&------------------------------------------------------*
MODULE MODIFY_BUTTON OUTPUT.
  IF NOT MAINTAIN_BOOKINGS IS INITIAL.
    LOOP AT SCREEN.
      IF SCREEN-NAME = 'P_DELETE'.
        SCREEN-INVISIBLE = 0.
      ENDIF.
      MODIFY SCREEN.
    ENDLOOP.
  ENDIF.
ENDMODULE.             " MODIFY_BUTTON  OUTPUT

*&------------------------------------------------------*
*&      Module  GET_SBOOK  OUTPUT
*&------------------------------------------------------*
MODULE GET_SBOOK OUTPUT.
  IF SDYN_CONN-CARRID <> KEY_SFLIGHT-CARRID OR
     SDYN_CONN-CONNID <> KEY_SFLIGHT-CONNID OR
     SDYN_CONN-FLDATE <> KEY_SFLIGHT-FLDATE OR
     UPD_FLAG = 'X'.


    MOVE-CORRESPONDING SDYN_CONN TO KEY_SFLIGHT.

    SELECT * INTO CORRESPONDING FIELDS OF TABLE
             IT_SDYN_BOOK FROM SBOOK
         WHERE CARRID   = WA_SFLIGHT-CARRID
         AND   CONNID   = WA_SFLIGHT-CONNID
         AND   FLDATE   = WA_SFLIGHT-FLDATE
         AND CANCELLED = ' '.
    DESCRIBE TABLE IT_SDYN_BOOK LINES
                   MY_TABLE_CONTROL-LINES.
    CLEAR UPD_FLAG.
  ENDIF.
ENDMODULE.                 " GET_SBOOK  OUTPUT
```

PAI module

```
*&------------------------------------------------------*
*&      Module  USER_COMMAND  INPUT
*&------------------------------------------------------*
MODULE USER_COMMAND_100 INPUT.
  CASE SAVE_OK.
    WHEN 'DELE'.
      PERFORM UPDATE_SBOOK.
      LEAVE TO SCREEN 0.
  ENDCASE.
ENDMODULE.             " USER_COMMAND  INPUT

*&------------------------------------------------------*
*&      Module  USER_COMMAND_0130  INPUT
*&------------------------------------------------------*
MODULE USER_COMMAND_0130 INPUT.
  CASE OK_CODE.
*    WHEN 'DELE'.  used in USER_COMMAND_0100 because of
*    SET SCREEN
    WHEN 'DSELE'.
      LOOP AT IT_SDYN_BOOK INTO WA_SDYN_BOOK
                           WHERE MARK = 'X'.
        WA_SDYN_BOOK-MARK = ' '.
        MODIFY IT_SDYN_BOOK FROM WA_SDYN_BOOK.
      ENDLOOP.
    WHEN 'SELE'.
      LOOP AT IT_SDYN_BOOK INTO WA_SDYN_BOOK
                           WHERE MARK = ' '.
        WA_SDYN_BOOK-MARK = 'X'.
        MODIFY IT_SDYN_BOOK FROM WA_SDYN_BOOK.
      ENDLOOP.
  ENDCASE.
ENDMODULE.            " USER_COMMAND_0130  INPUT
```

FORM routines

```
*&------------------------------------------------------*
*&      Form  UPDATE_SBOOK
*&------------------------------------------------------*
FORM UPDATE_SBOOK.
* Check if entries have to be updated
  READ TABLE IT_SDYN_BOOK WITH KEY MARK = 'X'
                          TRANSPORTING NO FIELDS.
  IF SY-SUBRC NE 0.
    MESSAGE S022.
*   No update (no booking to cancel)
    LEAVE TO SCREEN 0.
  ENDIF.

* Copy cancelled bookings to update table
  LOOP AT IT_SDYN_BOOK INTO WA_SDYN_BOOK
                       WHERE MARK = 'X'.
    MOVE-CORRESPONDING WA_SDYN_BOOK TO WA_SDYN_BOOK_UPD.
    MOVE WA_SDYN_BOOK-MARK TO WA_SDYN_BOOK_UPD-CANCELLED.
    APPEND WA_SDYN_BOOK_UPD TO IT_SDYN_BOOK_UPD.
  ENDLOOP.

* Call function for update table SBOOK.
  CALL FUNCTION 'BC_GLOBAL_UPDATE_BOOK'
       TABLES
            BOOKING_TAB     = IT_SDYN_BOOK
            BOOKING_TAB_UPD = IT_SDYN_BOOK_UPD
       EXCEPTIONS
            OTHERS      = 1.
  CLEAR: WA_SDYN_BOOK_UPD, IT_SDYN_BOOK_UPD.
  UPD_FLAG = 'X'.
ENDFORM.                 " UPDATE_SBOOK
```

Events

```
*&---------------------------------------------------------------------*
*&   Event AT LINE-SELECTION.
*&---------------------------------------------------------------------*
AT LINE-SELECTION.
  CALL SCREEN 100.
* update list of flights if necessary
  IF NOT PLANETYPE_CHANGED IS INITIAL.
    CLEAR PLANETYPE_CHANGED.
    PERFORM READ_FLIGHTS.
    PERFORM DISPLAY_FLIGHTS.
  ELSEIF NOT UPD_FLAG IS INITIAL.
    PERFORM READ_FLIGHTS.
    PERFORM DISPLAY_FLIGHTS.
  ENDIF.
  SY-LSIND = SY-LSIND - 1.
```

1-3 Model solution SAPBC410TABS_TABLE_CONTROL3

Add the coding in bold type, and create new modules where appropriate using forward navigation.

SCREEN 130

```
PROCESS BEFORE OUTPUT.
 MODULE MODIFY_BUTTON.
 MODULE GET_SBOOK.
  LOOP AT IT_SDYN_BOOK INTO WA_SDYN_BOOK
          WITH CONTROL MY_TABLE_CONTROL.
    MODULE GET_LOOPLINES.
    MODULE MOVE_TO_DYNP.
  ENDLOOP.
*
PROCESS AFTER INPUT.
  LOOP AT IT_SDYN_BOOK.
    FIELD SDYN_BOOK-MARK MODULE UPDATE_ITAB ON REQUEST.
  ENDLOOP.
 MODULE USER_COMMAND_0130.
```

Module pool

Add the coding below to your ABAP program

TOP include

```
* sy-loopc at PBO
DATA  looplines LIKE sy-loopc.
* workarea for Table Control structure COLS
DATA     wa_cols LIKE LINE OF my_table_control-cols.
```

PBO module

```
*&------------------------------------------------------*
*&      Module  GET_LOOPLINES  OUTPUT
*&------------------------------------------------------*
MODULE GET_LOOPLINES OUTPUT.
  LOOPLINES = SY-LOOPC.
ENDMODULE.              " GET_LOOPLINES  OUTPUT
```

PAI module

```
*&------------------------------------------------------*
*&      Module  USER_COMMAND  INPUT
*&------------------------------------------------------*
MODULE USER_COMMAND_100 INPUT.
  CASE SAVE_OK.
    WHEN 'P--' OR 'P-' OR 'P+' OR 'P++'.
      PERFORM TABLE_PAGING
              USING SAVE_OK MY_TABLE_CONTROL-TOP_LINE
                    MY_TABLE_CONTROL-LINES LOOPLINES.
  ENDCASE.
ENDMODULE.               " USER_COMMAND  INPUT

*&------------------------------------------------------*
*&      Module  USER_COMMAND_0130  INPUT
*&------------------------------------------------------*
MODULE USER_COMMAND_0130 INPUT.
  CASE OK_CODE.
    WHEN 'SRTU'.
      READ TABLE MY_TABLE_CONTROL-COLS INTO WA_COLS
                 WITH KEY SELECTED = 'X'.
      IF SY-SUBRC = 0.
        SORT IT_SDYN_BOOK BY (WA_COLS-SCREEN-NAME+10)
                             ASCENDING.
      ENDIF.
*  second method
*     LOOP AT MY_TABLE_CONTROL-COLS INTO WA_COLS WHERE
*             SELECTED = 'X'.
*       SORT IT_SDYN_BOOK BY (WA_COLS-SCREEN-NAME+10)
*                            ASCENDING.
*     ENDLOOP.

* determine fieldname dynamically
* 1. 'slower' version
*         DATA SORTFIELD LIKE SCREEN-NAME.
*         IF WA_COLS-SCREEN-NAME CS '-'.
*            POS = SY-FDPOS + 1.
*            SORTFIELD = TC_COL-SCREEN-NAME+POS.
*            SORT IT_SDYN_BOOK BY (SORTFIELD).
*         ENDIF.
* 2. 'faster' version
*         FIELD_SYMBOLS <sort_field>.
*         IF WA_COLS-SCREEN-NAME CS '-'.
*            POS = SY-FDPOS + 1.
*            ASSIGN WA_COLS-SCREEN-NAME+POS(*) TO
*                   <SORT_FIELD>.
*            SORT IT_SDYN_BOOK BY (<SORT_FIELD>).
*         ENDIF.
    WHEN 'SRTD'.
      READ TABLE MY_TABLE_CONTROL-COLS INTO WA_COLS
                 WITH KEY SELECTED = 'X'.
      IF SY-SUBRC = 0.
        SORT IT_SDYN_BOOK BY (WA_COLS-SCREEN-NAME+10)
                             DESCENDING.
      ENDIF.
*  second method
*     LOOP AT MY_TABLE_CONTROL-COLS INTO WA_COLS WHERE
*                      SELECTED = 'X'.
*       SORT IT_SDYN_BOOK BY (WA_COLS-SCREEN-NAME+10)
*                            DESCENDING.
*     ENDLOOP.
  ENDCASE.
ENDMODULE.           " USER_COMMAND_0130  INPUT
```

FORM routines

```
*&-------------------------------------------------
-----*
*&      Form  TABLE_PAGING
*&-------------------------------------------------
-----*
*       Implementation of table paging
*--------------------------------------------------
-----*
*      -->P_SAVE_OK                       function
code
*      -->P_MY_TABLE_CONTROL_TOP_LINE  table index
*                                      (TOP_LINE)
*      -->P_MY_TABLE_CONTROL_LINES     itab rows
*                                      (maximum
index)
```

```
*      -->P_LOOPLINES                          screen rows
*                                              (SY-LOOPC at
PBO)
*---------------------------------------------------
-----*
FORM TABLE_PAGING USING    P_SAVE_OK

P_MY_TABLE_CONTROL_TOP_LINE
                           P_MY_TABLE_CONTROL_LINES
                           P_LOOPLINES.

   CALL FUNCTION 'SCROLLING_IN_TABLE'
        EXPORTING
             ENTRY_ACT     =
P_MY_TABLE_CONTROL_TOP_LINE
             ENTRY_TO      =
P_MY_TABLE_CONTROL_LINES
             LOOPS         = P_LOOPLINES
             OK_CODE       = P_SAVE_OK
        IMPORTING
             ENTRY_NEW     =
P_MY_TABLE_CONTROL_TOP_LINE
        EXCEPTIONS
             NO_ENTRY_OR_PAGE_ACT  = 1
             NO_ENTRY_TO           = 2
             NO_OK_CODE_OR_PAGE_GO = 3
             OTHERS                = 4.
   IF SY-SUBRC <> 0.
*     not required
   ENDIF.
ENDFORM.                    " TABLE_PAGING
```

Context Menus on Screens

Contents:

- Creating, using and modifying context menus

- Unit 1 Course Overview
- Unit 2 Basics for Interactive Lists
- Unit 3 The Program Interface
- Unit 4 Interactive List Techniques
- Unit 5 Introduction to Screen Programming
- Unit 6 Screen Elements for Output
- Unit 7 Screen Elements for Input/Output
- Unit 8 Screen Elements: Subscreens and Tabstrip Controls
- Unit 9 Screen Elements: Table Controls
- Unit 10 **Context Menus on Screens**
- Unit 11 Lists in Screen Programming
- Unit 12 Preview: Control Framework

- Context menus (right mouse key, SHIFT F10) are shortcuts for functions that are frequently used.
- They can be used to display context-sensitive menus. The context is defined by the position (cursor for SHIFT F10, mouse location for right mouse key) where the user called the context menu. If needed, you can specify the context more precisely with the displayed contents. This permits the user to select functions that are relevant for the current context using the context menu.
- You define whether a context menu should be offered when you create a screen object (screens, input fields, table controls, boxes, ...). When the user selects a context menu on an object, an event mechanism (as understood by ABAP objects) calls a certain subroutine in the application program. The program is assigned a menu reference. The program uses this menu reference to build the display menu. Menus defined with the Menu Painter and dynamic menus can be used here.
  After the user executes a menu function, the application program regains control and can react to the user input.
- Context menus are assigned to output fields. When you assign a context menu to a box, table control or screen (normal or subscreen), all the subordinate output fields that do not have a context menu inherit that one.

- You can create a context menu from within the object list of the Object Navigator. Position the cursor on *GUI status* and right-click. The Object Navigator automatically opens the Menu Painter.
- Of course you can also create a context menu directly in the Menu Painter.
- A context menu is a special GUI status. Assign it a name, a descriptive text and status type C*ontext menu.*

- In a context menu you can link any function codes and function texts. In particular, you can take advantage of your screen pushbuttons. The functions already provided in the interface can be used as an F4 input help.
- The link technique ensures consistent context menus in large applications.
- You should observe the following rules when designing context menus.
  - Do not use any functions that cannot be found elsewhere in the system (pushbuttons or interface).
  - Avoid using more than two hierarchy levels in context menus.
  - Do not use more than 10 entries, but map all the available pushbuttons.
  - Use separators to structure the context menu optically.
  - Place object-specific statements at the beginning of the menu.

- Pressing the right mouse key triggers a callback routine in your program. You can create this callback routine in your application program with forward navigation. It is named `ON_CTMENU_<name>.` You define which callback routine is called in the Screen Painter.
- You can directly assign a callback routine to input/output fields, text fields and status icons. Checkboxes, radio buttons and pushbuttons do not have their own callback routines. However, these fields can inherit context menus from boxes or screens.
- If you assign a callback routine to a table control, it is triggered for all the fields of the table control that do not have their own callback routine.
- The callback routine has the following form:

```
FORM ON_CTMENU_<name> USING p_menu TYPE REF TO cl_ctmenu.
     <definition of the context menu>.
ENDFORM.
```

- The context menu is built with a method call for the instance of class `cl_ctmenu` that was passed.

- Clicking with the right mouse key on an output field triggers the corresponding callback routine.
- You can now use the static method `load_gui_status` of class `cl_ctmenu` to load a context menu that was predefined in the Menu Painter. Using other methods of class cl_ctmenu (see next slide) you can also completely rebuild the context menu or modify a loaded menu.
- If the user triggers a function in the context menu, the corresponding function code is placed in the command field and triggered depending on function type PAI of the screen.

- The class `cl_ctmenu` provides a number of other methods in addition to the static method `load_gui_status.` You can use them to adjust the context menu at runtime (e.g. using the values in data fields).
- The corresponding methods are called within the callback routine.
- You can find further information in the documentation for class `cl_ctmenu` in the Class Builder.

Unit: Context Menus on Screens

Theme: Creating and Using a Context Menu

At the conclusion of these exercises, you will be able to:

Use context menus in your programs.

Make the functions for your table control available in a context menu.

10-1 Create a GUI status with type context menu and use it for the output fields on screen 130.

10-1-1 Extend your program **Z##BC410_SOLUTION** from the previous exercise (or copy the model solution **SAPBC410TABS_TABLE_CONTROL3**). You can use the model solution **SAPBC410CONS_CONTEXTMENU** for orientation.

10-1-2 Create the GUI status **sub130** with type *context menu* and the short description *Table control subscreen*. Assign the following functions to the menu:

Context menu	**Function code:**	**Function text**
	SELE	Select all
	DSELE	Deselect all
	DELE	Delete line
	SRTU	Sort ascending
	SRTD	Sort descending
	Separator	First page
	P- -	Previous page
	P-	Next page
	P+	Last page
	P+ +	

10-1-3 Assign function type ‘ ’ (space) to all of the functions. Deactivate the function DELE.

10-1-4 In the screen attributes of 130, declare that you want to use subroutine **on_ctmenu_sub130** to create the context menu.

10-1-5 Write the subroutine to create the context menu.

10-1-6 Additional task:
Activate the DELE function at runtime if the user is in booking maintenance mode. Note that you must pass the function code to the method in a table with type **ui_functions**.

Unit: Context Menus on Screens

Theme: Creating and Using a Context Menu

10-1 Model solution: SAPBC410CONS_CONTEXTMENU

Add the coding in bold type to your program, and create the subroutine.

Subroutine include

```
*&------------------------------------------------------------------*
*&      Form  ON_CTMENU_SUB130
*&------------------------------------------------------------------*
FORM on_ctmenu_sub130 USING p_menu TYPE REF TO cl_ctmenu.
data fcodes type ui_functions.
load of the context menu defined in the menu painter

CALL METHOD cl_ctmenu=>load_gui_status
EXPORTING program = sy-cprog
status = 'SUB130'
menu = p_menu.

activate DELE for deleting bookings in maintain booking mode
CHECK NOT maintain_bookings IS INITIAL.
append 'DELE' to fcodes.
CALL METHOD p_menu->enable_functions EXPORTING fcodes = fcodes.

ENDFORM.                               " ON_CTMENU_SUB130
```

Lists in Screen Programming

Contents:

- Lists on screens

■ Unit 1	Course Overview
■ Unit 2	Basics for Interactive Lists
■ Unit 3	The Program Interface
■ Unit 4	Interactive List Techniques
■ Unit 5	Introduction to Screen Programming
■ Unit 6	Screen Elements for Output
■ Unit 7	Screen Elements for Input/Output
■ Unit 8	Screen Elements: Subscreens and Tabstrip Controls
■ Unit 9	Screen Elements: Table Controls
■ Unit 10	Context Menus on Screens
■ Unit 11	**Lists in Screen Programming**
■ Unit 12	Preview: Control Framework

- A list is generally used to output mass data. It can be output on either the screen or a printer.
- List may contain colors, symbols and icons as well as text.
- There is a standard GUI status for lists. Lists may also have a header and up to four lines of column headers. These are independent program objects, and are translatable.
- You can also program interactive lists, which allow users to select lines or particular values. A selection triggers further processing. This might, for example, generate a further list containing a detail list.
- To find out more about list processing, refer to the units *Basics for Interactive Lists, The Program Interface* and *Interactive List Techniques*.

- You fill the corresponding basic list buffer with `WRITE` statements at PBO or PAI. You can create your own list and column headers by programming a `TOP-OF-PAGE` event. This event will be triggered whenever a new page is created in the list buffer (`NEW-PAGE`) .
- You can direct the output directly to the spool with the `NEW-PAGE PRINT ON` statement.

- To create interactive lists on screens, you can use the list events AT LINE-SELECTION, AT USER-COMMAND, TOP-OF-PAGE, END-OF-PAGE and TOP_OF_PAGE DURING LINE-SELECTION.

- There is no common list buffer outside of a `CALL` level.
- The list display is processed at the end of the screen in which `LEAVE TO LIST-PROCESSING` was programmed at PBO or PAI.
- To direct the output to the spool, use the `NEW-PAGE PRINT ON` statement, but not `LEAVE TO LIST-PROCESSING.`

- To create a list that is displayed on a screen, use the ABAP statement `LEAVE TO LIST-PROCESSING.` This sets a switch that ensures that the contents of the list buffer are output once the current screen has been processed. The `SET PF-STATUS SPACE` statement ensures that the list is displayed with the standard GUI status for lists.

- Once the screen has been fully processed and `LEAVE TO LIST- PROCESSING` was executed, the list is displayed on list screen 120 (screen for a basis program).

- You can also use the following form: `LEAVE TO LIST-PROCESSING AND RETURN TO SCREEN 0. SET PF-STATUS SPACE. WRITE ...  LEAVE SCREEN`.

- When the system exits list processing (user presses F3, or ABAP statement `LEAVE LIST-PROCESSING`), the system carries on processing the program with the screen following the one from which the list processing was started. You can override this by using the `AND RETURN TO SCREEN <scr>` addition in the `LEAVE TO LIST-PROCESSING` statement.

- If you include the ABAP statement `SUPPRESS DIALOG` in a PBO module, the current screen is not displayed.

- If you want to display a list in a dialog box within a transaction, you must call a screen, but include the `SUPPRESS DIALOG` statement in its PBO processing block.
- To return to the calling screen after leaving the list, declare: `LEAVE TO LIST-PROCESSING AND RETURN TO SCREEN 0.`

Unit: Lists in Screen Programming

Theme: Displaying a list on a screen

At the conclusion of these exercises, you will be able to:

Use lists on screens in your programs.

Extend your flight maintenance screen to display a booking list. Allow the user to display the bookings by choosing a pushbutton or menu entry. Make it possible to sort the list dynamically.

Program the booking list.

Extend your program **Z##BC410_SOLUTION** from the previous exercise (or copy the model solution **SAPBC410CONS_CONTEXTMENU**). You can use the model solution **SAPBC410LISS_LISTS_ON_DYNPROS** for orientation.

Create a function BOOK in status STATUS_100, assigning it to function key F5, a pushbutton, and a menu entry. The function is already in the function list of your interface, so you can use the F4 help.

Call screen 200 if the user chooses the BOOK function. Create the screen (type: normal). In the PBO event of the screen, call a module in which you create the list. Screen 200 is only a container – you should not actually display it. Use your subroutines to read and display the booking data. Set the GUI status and GUI title BOOK and start list processing.

Use the TOP-OF-PAGE to create list headers in the same way you would in TOP-OF-PAGE DURING LINE-SELECTION. Make sure that the headings are not displayed on the flight list as well. Stop the standard list header from being displayed on the booking list. (`NEW-PAGE NO-TITLE NO-HEADING` before the first display.) If there are no bookings for the selection data, display message 186 from message class BC410 as an information message.

Test your program. Good work for just a week, isn't it!

Unit: Lists in Screen Programming

Theme: Displaying a list on a screen

Model solution: SAPBC410LISS_LISTS_ON_DYNPROS

Add the coding in bold type to your program, and create the module.

Flow logic for screen 200

```
PROCESS BEFORE OUTPUT.

MODULE list.
*
PROCESS AFTER INPUT.
```

Top include

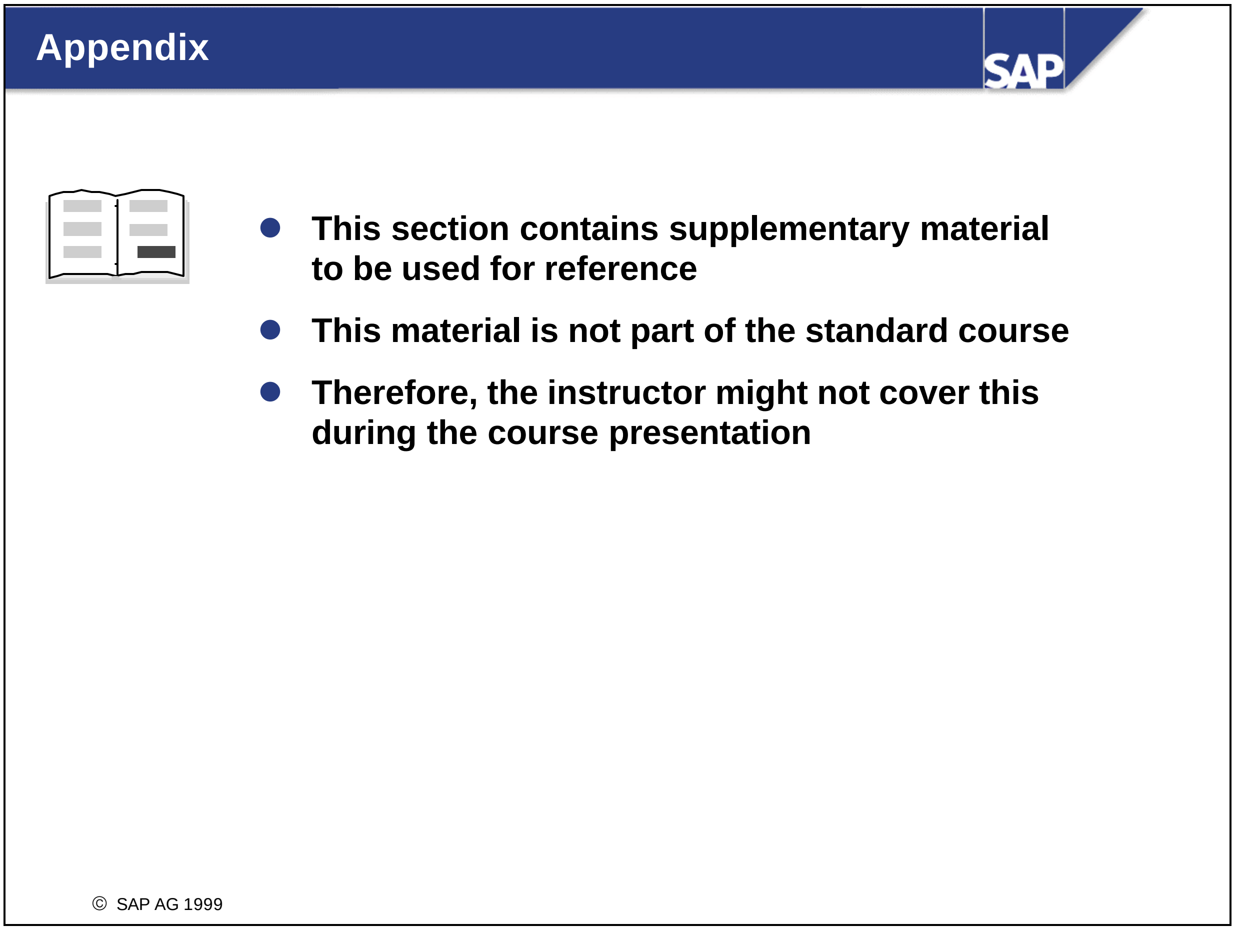
```
number of lines of an internal table
DATA  lines type i.
```

Event include

```
TOP-OF-PAGE.
CHECK sy-dynnr = 200.
FORMAT COLOR COL_HEADING.
ULINE.
WRITE: / 'Flight:'(t01), wa_sbook-carrid, wa_sbook-connid,
AT sy-linsz space,
/ 'Date:'(t02), wa_sbook-fldate, AT sy-linsz space.
ULINE.
```

PBO module include

```
*&-----------------------------------------------------------------*
*&      Module  LIST  OUTPUT
*&-----------------------------------------------------------------*
CHECK NOT wa_sflight-carrid IS INITIAL.
CHECK NOT wa_sflight-connid IS INITIAL.
CHECK NOT wa_sflight-fldate IS INITIAL.
REFRESH it_sbook.
PERFORM read_bookings
USING wa_sflight-carrid
wa_sflight-connid
wa_sflight-fldate
' '.
APPEND LINES OF it_sbook_read TO it_sbook.

DESCRIBE TABLE it_sbook LINES lines.
IF lines = 0.
MESSAGE i186(bc410).
ELSE.
SORT it_sbook BY carrid connid fldate bookid.
NEW-PAGE NO-TITLE NO-HEADING.
PERFORM display_bookings.
SET PF-STATUS 'BOOK'.
SET TITLEBAR 'BOOK'.
ENDIF.
CLEAR: wa_sbook-bookid.
LEAVE TO LIST-PROCESSING AND RETURN TO SCREEN 0.
SUPPRESS DIALOG.
ENDMODULE.                             " LIST  OUTPUT
```



Appendix

- This section contains supplementary material to be used for reference
- This material is not part of the standard course
- Therefore, the instructor might not cover this during the course presentation

▪ *Documentation Links*

Ref. number	Path in documentation
ILB-1	*SAP Library ® Basis Components ® ABAP Programming and Runtime Environment ® BC - ABAP Programming ® ABAP User Dialogs ® Selection Screens*
ILB-2	*SAP Library ® Basis Components ® ABAP Programming and Runtime Environment ® BC - ABAP Programming ® ABAP User Dialogs ® Selection Screens ® Defining Selection Screens*

GUI-1	In the Menu Painter: *Goto ® Interface objects;*
	Function key settings ® <name> ® Pushbutton settings;
	Interface ® Subobject ® Create

GUI-2	*BC - Basis ® ABAP Workbench ® BC - ABAP Workbench: Tools ® ABAP Workbench: Tools ® Menu Painter ® Functions*
GUI-3	In the Menu Painter: *Utilities ® Help texts ® Internal key numbers*
ILS-1	In the Menu Painter: *Utilities ® Help texts ® Standards/proposals*
ILS-2	In the ABAP Editor: *Utilities ® Help on...* , ABAP term: *READ*
ILS-3	In the ABAP Editor: *Utilities ® Help on...* , ABAP term: *MODIFY*
ILS-3	In the ABAP Editor: *Utilities ® Help on...* , ABAP term: *GET CURSOR*
DIA-1	*SAP Library ® Getting Started with the SAP System ® Layout Menu*
DIA-2	*SAP Library ® Basis ® ABAP Workbench ® BC - ABAP Workbench: Tools ® ABAP Workbench: Tools ® Screen Painter ® Working with element attributes*
DIA-3	*SAP Library ® Basis ® ABAP Workbench ® BC - ABAP Workbench: Tools ® ABAP Workbench: Tools ® Screen Painter ® Creating screens.*

OUT-1	In the Screen Painter: *Goto ® Translation*
OUT-2	*SAP Library ® Basis Components ® ABAP Workbench ® BC - SAP Style Guide ® R/3 Icons and symbols ® Icons ® Icons as status displays.*
INP-1	*SAP Library ® Basis Components ® ABAP Workbench ® BC – SAP Style Guide ® Interface elements ® Input/output fields*
INP-2	*SAP Library ® Basis Components ® ABAP Workbench ® BC - ABAP Workbench: Tools ® ABAP Workbench: Tools ® Screen Painter ® Defining the Element Attributes ® Choosing Field Formats*
INP-3	*SAP Library ® Basis Components ® ABAP Workbench ® BC - SAP Style Guide ® Functions – General guidelines ® Navigation functions - Overview.*
INP-4	*SAP Library ® Basis Components ® ABAP Workbench ® BC - SAP Style Guide ® Functions – General guidelines ® Navigation Functions – Overview ® Comparison of Exit, Back, and Cancel.*
SUB-1	*SAP Library ® Basis Components ® ABAP Programming and Runtime Environment ® BC - ABAP Programming® ABAP User Dialogs ® Screens ® Complex Screen Elements ® Tabstrip Controls*
SUB-2	*SAP Library ® Basis Components ® ABAP Programming and Runtime Environment ® BC - ABAP Programming® ABAP User Dialogs ® Selection Screens ® Subscreens and Tabstrip Controls on Selection Screens*
TAB-1	*SAP Library ® Basis Components ® ABAP Programming and Runtime Environment ® BC - ABAP Programming® ABAP User Dialogs ® Screens ® Complex Screen Elements ® Table Controls*

Each entry in the glossary contains a reference to the application component to which it belongs. You can use this path in the R/3 Library to find further information. For example, for more information about ABAP Dictionary, look under ABAP Workbench (application component BC-DWB).

A

ABAP Workbench (BC-DWB)

Central and redundancy-free storage facility for all data used in the R/3 System. The ABAP Dictionary describes the logical structure of application development objects and their representation in the structures of the underlying relational database. All runtime environment components such as application programs or the database interface, get information about these objects from the ABAP Dictionary. The ABAP Dictionary is an active data dictionary and is fully integrated into the ABAP Workbench.

ABAP Workbench (BC-DWB)

ABAP Native SQL allows you to include database-specific SQL statements in an ABAP program. Most ABAP programs containing database-specific SQL statements do not run with different databases. If different databases are involved, use Open SQL. To execute ABAP Native SQL in an ABAP program, use the statement EXEC.

ABAP Workbench (BC-DWB)

Subset of standard SQL statements.

To avoid conflicts between database tables and to keep ABAP programs independent from the database system used, SAP has generated its own set of SQL statements known as Open SQL.

Using Open SQL allows you to access all database tables available in the R/3 System, regardless of the manufacturer.

ABAP Workbench (BC-DWB)

Program written in the ABAP programming language.

An ABAP program consists of a collection of processing locks, which are processed sequentially as soon as they are called by the runtime system.

There are two main kinds of ABAP program:

- Report programs (ABAP reports)
- Dialog programs

Basis Services/Communication Interfaces (BC-SRV)

ABAP Workbench tool that allows users without knowledge of the ABAP programming language, or table or field names, to define and execute their own reports.

To determine the structure of reports in ABAP Query, users only have to enter texts, and select fields and options. Fields are selected from functional areas and can be assigned a sequence by numbering.

There are three types of report available:

- Basic lists
- Statistics
- Ranked lists

ABAP Workbench (BC-DWB)

ABAP program that reads and analyzes the data in database tables without modifying the database.

ABAP report programs are defined as type '1' programs and are linked to a particular logical database. Both of these values are specified in the program attributes.

When you execute an ABAP report program, you can display the resulting output list - also known as a report - on the screen or send it to a printer.

ABAP Workbench (BC-DWB)

SAP's integrated graphical programming environment.

The ABAP Workbench supports the development and modification of R/3 client/server applications written in

You can use the tools of the ABAP Workbench to

- write ABAP code
- design screens
- create user interfaces
- use predefined functions
- get access to database information
- control access to development objects
- test applications for efficiency
- debug applications

ABAP Workbench (BC-DWB)

Process that makes an object available at runtime.

When you activate an object, the system generates a load version that application programs and screens can access and use.

Graphical User Interface (BC-FES-GUI)

Dialog box, allowing the user to work on one screen without the previous screen first being closed.

Business Navigator (BC-BE-NAV)

Organizational tool for displaying all of the business applications in the R/3 System.

The application hierarchy has a user interface similar to that of a file manager, with a hierarchical structure. You can display either the standard applications delivered with the system, or a company-specific hierarchy.

Integration Technology ALE (CA-BFA-ALE)

Application Link Enabling (ALE) refers to the creation and operation of distributed applications.

The basic idea is to guarantee a distributed, but integrated, R/3 installation. This involves business-controlled message exchange with consistent data across loosely linked SAP applications.

Application integration is achieved not via a central database, but via synchronous and asynchronous communication.

ALE comprises the following three layers:

- application services
- distribution services

communication services

Graphical User Interface (BC-FES-GUI)

Element of the graphical user interface,

The application toolbar is situated below the standard toolbar on the screen. It contains pushbuttons, which allow users quick access to application-specific functions, and occupies the whole of the primary window.

Before you can assign a function to a pushbutton, you must assign it to a function key.

Computing Center Management System (BC-CCM)

Authority to perform a particular action in the R/3 System.

Each authorization refers to one authorization object and defines one or more permissible values for each authorization field listed in the authorization object.

Authorizations are combined in profiles which are entered in a user's master record.

Computing Center Management System (BC-CCM)

Element of an authorization object.

In authorization objects, authorization fields represent values for individual system elements which are supposed to undergo authorization checking to verify a user's authorization.

Computing Center Management System (BC-CCM)

Element of the authorization concept.

Authorization objects allow you to define complex authorizations.

An authorization object groups together up to 10 authorization fields in an AND relationship in order to check whether a user is allowed to perform a certain action.

To pass an authorization test for an object, the user must satisfy the authorization check for each field in the object.

Computing Center Management System (BC-CCM)

Element of the authorization concept.

An authorization profile gives a user access to the system. It contains authorizations, identified by the name of an authorization object. Users have all of the authorizations contained in each profile entered in their user master record.

B

Basis Services/Communications Interfaces (BC-SRV)

Interface allowing you to import large amounts of data into an R/3 System.

You use batch input to import legacy data into your new R/3 System, and for periodic imports of external data.

Basis Services/Communications Interfaces (BC-SRV)

Set of transactions supplied with data by a program.

The transactions are stored as a stack. You can then run the session later in dialog mode. The database changes are not made until you have run the session.

This method allows you to import large quantities of data into an R/3 System in a short time.

C

ABAP Workbench (BC-DWB)

Information folder in the Workbench Organizer and Customizing Organizer for entering and administrating all changes to Repository objects and Customizing settings made during a development project.

ABAP Workbench (BC-DWB)

A change request that can be transported into other systems once it has been released.

Business Engineer (BC-BE)

In commercial, organizational and technical terms, a self-contained unit in an R/3 System with separate master records and its own set of tables.

See also the glossary entry for "logical system".

Graphical User Interface (BC-FES-GUI)

Memory resource that stores a copy of the last information to be copied with the 'Copy' function, or cut with the 'Cut' function.

You can use the 'Paste' function to copy data stored in the clipboard to the current program.

The clipboard is managed by the operating system.

Graphical User Interface (BC-FES-GUI)

Input field in the standard toolbar to the right of the ENTER pushbutton.

You can enter fastpaths or transaction codes in this field, to choose menu entries or call transactions respectively.

ABAP Workbench (BC-DWB)

Point at which the system changes from one control group to another in a report.

The control group change represents a change in the value of whichever field is currently most significant. In an ABAP program, you trigger it with the AT NEW statement.

D

ABAP Workbench (BC-DWB)

Language used to define all the attributes and properties of a database management system

The query language SQL (Structured Query Language) consists of two kinds of statements:

- DDL (data definition language)
- DML (data manipulation language)

ABAP Workbench (BC-DWB)

Sequential dataset in the memory area of a report.

ABAP Workbench (BC-DWB)

Language for processing data in a database management system.

The query language SQL (Structured Query Language) consists of two kinds of statements:

- DDL (data definition language)

DML (data manipulation language)

Data Model (BC-RMC-DMO)

Structured description of data objects, their attributes, and the relationships between them.

There are different types of data model, depending on the types of data structure you want to define (for example, relational data model).

ABAP Workbench (BC-DWB)

Physical unit used by a program.

Each data object has a certain data type, which defines how ABAP processes it. All data object occupy memory space.

ABAP Workbench (BC-DWB)

Attribute of a Data Object

Data types describe the technical attributes of data objects. They are purely descriptions, and occupy no memory space.

ABAP Workbench (BC-DWB)

In a database commit, all of the database update requests from the current logical unit of work (LUW) are written to the database.

In the R/3 System, database commits are either triggered automatically or manually, using the ABAP statement COMMIT WORK (or, in Native SQL, the database-specific equivalent).

ABAP Workbench (BC-DWB)

If you discover an error within an LUW, you can undo all of the update requests in the LUW (that is, since the last commit) using a database rollback.

In the R/3 System, database rollbacks are either triggered automatically or manually, using the ABAP statement ROLLBACK WORK (or, in Native SQL, the database-specific equivalent).

ABAP Workbench (BC-DWB)

A type of view in the ABAP Dictionary.

Database views are implemented using an equivalent view in the underlying database system.

ABAP Workbench (BC-DWB)

Group of logically related development objects. A development class contains all the objects which must be corrected and transported as a whole. The objects which make up a transaction usually belong to one development class. Customer development classes begin with 'Y' or 'Z'.

E

Electronic Data Interchange.

Business-to-business electronic data interchange (for example, sales documents). The business partners may be in different countries, and might be using different hardware, software, and communication services. The data is formatted according to fixed standards.

In addition, SAP ALE enables companies to exchange data internally.

ABAP Workbench (BC-DWB)

Type of ABAP keyword.

An event keyword defines a processing block in an ABAP program. The processing block is processed when the particular event occurs.

Examples. GET, START-OF-SELECTION, AT SELECTION-SCREEN.

F

ABAP Workbench (BC-DWB)

Group of functions that logically belong together and use a shared program context at runtime.

The function group is a container program for the function modules that it contains. Functions that work with the same data are usually all included in the same function group.

Function groups are an administrative unit within the Function Builder.

ABAP Workbench (BC-DWB)

Reusable function.

Function modules are external subroutines that you maintain centrally in the Function Builder, and which can be called from any ABAP program. This allows you to avoid redundancy in your coding and makes programming more efficient.

Unlike normal subroutines, function modules have a defined interface.

Basis Services/Communications Interfaces (BC-SRV)

You can use ABAP Query to define reports without any previous programming knowledge. When you create a query, you must assign it to a functional area, which determines the tables and fields that the query can use. Functional areas in ABAP Query are usually subsets of logical databases.

Basis Services/Communications Interfaces (BC-SRV)

Element of ABAP Query

A functional group is a collection of fields that forms a logical unit. You use them to provide users with a selection of fields so that he or she does not need to sort through all of the fields in a logical database in order to create a query.

You must assign a field to a functional group in order for it to be used later in a query.

G

Graphical User Interface (BC-FES-GUI)

Main element of the graphical user interface.

A GUI status consists of:

- A menu bar with menus
- A standard toolbar
- An application toolbar
- Functions, and function key settings

H

ABAP Workbench (BC-DWB)

Main memory area for storing key fields of a line in a report list.

If you want to select further data based on a line selection, the system can find the key fields that it requires in the hide area.

You must place the key fields into the hide area yourself using the HIDE statement.

I

Graphical User Interface (BC-FES-GUI)

Graphical representation of an object or functions. Icons are small colored bitmaps that are used for pushbuttons, checkboxes, and radio buttons, either with or without text.

Unlike symbols, icons always have the same size, which is one of two, selected automatically by the system according to the font size.

IMG (BC-BE-IMG)

The R/3 International Demonstration and Education System.

IDES contains several fictional companies that model the different business processes in the R/3 System. Simple user guides and sample master and transaction data allow you to simulate a wide range of scenarios. This makes IDES a useful tool for training your project team.

ABAP Workbench (BC-DWB)

Data Modeler (BC-RMC-DMO)

Passing of attributes from one data object to another.

Attributes can either be passed generally (all attributes), or by copying individual characteristics.

Workflow (BC-BMT-WFM)

„is a“ relationship between object types in which shared attributes and methods of supertypes are passed automatically to subtypes.

Subtypes usually have the same key fields as the supertype, but a more wide-ranging function.

ABAP Workbench (BC-DWB)

Temporary data structure that exists during the runtime of a program.

Internal tables are one of two structured data types in ABAP. They consist of any number of table lines, each of which has the same structure. They may or may not have a header line.

The header line is a structure, and serves as a work area for the internal table. The data type of the line can be either elementary or structured.

L

Graphical User Interface (BC-FES-GUI)

Standard function in the R/3 System used to display lists.

ABAP Workbench (BC-DWB)

First line of the screen in a list.

The list header is often the same as the title of the program. However, you can maintain it independently of the program title.

ABAP Workbench (BC-DWB)

Special ABAP program that combines the contents of certain database tables.

You can attach a logical database to an ABAP report program as one of the program attributes. It supplies the report with a set of hierarchically-structured table lines, which can come from different database tables. This saves the programmer from having to retrieve the data him- or herself.

The term „logical database“ applies not only to the program, but also to the data itself.

ABAP Workbench (BC-DWB)

Inseparable sequence of database operations, working on the all-or-nothing principle, where the operations are either all carried out or all canceled.

From the point of view of the database system, logical units of work (LUWs) are crucial to the integrity of the data in the database.

M

Graphical User Interface (BC-FES-GUI)

Graphical element for choosing functions.

Menus are graphical elements that present the user with a series of options, each of which triggers a function in the system. This can include opening a submenu.

There are two types of menu:

- Menu bars
- Action menus

To choose a menu entry, single-click it with the mouse, or position the cursor on it using the arrow keys and press ENTER.

Graphical User Interface (BC-FES-GUI)

Element of the graphical user interface.

The menu bar appears directly below the title bar in the primary window.

When you choose an entry in the menu bar, the system opens the corresponding action menu below the entry. You can put up to 6 menus in the menu bar, to which the system automatically adds the ‘System’ and ‘Help’ menus.

ABAP Workbench (BC-DWB)

Placeholder in the standard system for customers’ own menu entries.

Menu exits allow you to link your own functions to menu entries reserved in the standard system as part of the enhancement concept (Transaction CMOD). You use an associated function module exit to implement the function.

ABAP Workbench (BC-DWB)

Development tool in the ABAP Workbench for designing the graphical user interface of an ABAP program. Each GUI consists of a title and a GUI status.

The GUI status contains the following elements:

- Menu bar with menus
- Standard toolbar
- Application toolbar

Functions, assigned to function keys.

ABAP Workbench (BC-DWB)

Collection of messages that are used by a particular application.

Each ABAP program is linked to a message class. The name of the class can be up to 20 characters.

ABAP Workbench (BC-DWB)

Data that describes other data.

Metadata are data definitions, usually stored in a data dictionary.

Graphical User Interface (BC-FES-GUI)

Dialog box that must be processed or canceled before the screen behind it can be processed further.

ABAP Workbench (BC-DWB)

Customer-specific change to an R/3 Repository object.

When you upgrade the system, you need to check, and possibly update, your modified objects.

P

ABAP Workbench (BC-DWB)

Group of program statements that are processed together as a unit at a particular point.

ABAP is an event-oriented language (the flow of a program is controlled by events). Program sections are therefore grouped into processing blocks, which are assigned to particular events. Events are triggered in the program using event keywords.

A processing block consists of all of the statements between two event keywords or between an event keyword and a FORM statement.

ABAP Workbench (BC-DWB)

A set of statements that provides a solution to a task.

A program consists of a set of statements that are interpreted and executed by a computer.

Q

Basis Services/ Communication Interfaces (BC-SRV)

Report that users without programming expertise can generate using ABAP Query.

There are three different types of query:

- Basic list
- Statistic

Ranked list

R

ABAP Workbench (BC-DWB)

Central store for development objects in the ABAP Workbench.

Development objects include ABAP programs, screens, documentation, and so on.

ABAP Workbench (BC-DWB)

Information system that enables you to find information about al of the development objects in the R/3 System and the relationships between them.

The user interface of the R/3 Repository Information System displays objects in a hierarchical structure similar to a file manager.

The R/3 Repository Information allows you to:

- Create lists of programs, tables, fields, data elements, and domains.
- Find out where tables and fields are used in ABAP programs and screens.

Display foreign key relationships.

ABAP Workbench (BC-DWB)

Remote Function Call.

RFC is an SAP interface protocol based on CPI-C. This simplifies the process of programming communication between systems.

RFC allows you to call and execute predefined functions in a remote system. They have built-in communication control, parameter passing, and error handling.

S

ABAP Workbench (BC-DWB)

Global, user-specific memory.

You address the SAP memory using SPA/GPA parameters.

ABAP Workbench (BC-DWB)

A screen (in the sense of a 'dynpro' or DYNamic PROgram) consists of a screen and its underlying flow logic.

The main components of a screen are:

- attributes (e.g. screen number, next screen)
- layout (the arrangement of texts, fields, and other elements)
- field attributes (definition of the properties of individual fields)
- flow logic (calls the relevant ABAP modules)

Graphical User Interface (BC-FES-GUI)

Screen in an ABAP report program.

You use the selection screen to enter the selection criteria by which the system should retrieve data from the database.

ABAP Workbench (BC-DWB)

Internal table containing selection criteria.

The system creates a selection table for each SELECT-OPTIONS statement that you use in an ABAP report program. They allow you to save complex selections in a standard format.

T

ABAP Workbench (BC-DWB)

Tabular collection of data. The definition is stored in the ABAP Dictionary, the contents are stored in the database.

A table consists of columns (sets of data values with the same type) and lines (data records).

Each line of a table can be identified uniquely using a field or a combination of fields.

ABAP Workbench (BC-DWB)

Data object created in an ABAP program using the TABLES statement.

A table work area is a structure with the same construction as the corresponding table in the ABAP Dictionary.

ABAP Workbench (BC-DWB)

Information carrier in the Workbench Organizer for entering and managing all changes to Repository objects and Customizing settings performed by employees within a development project.

A task is assigned to a change request

ABAP Workbench (BC-DWB)

Text constant that you create and maintain outside programs.

You use text symbols instead of text literals to make texts easier to maintain and translate.

Each text symbol is identified by a three-character code.

Graphical User Interface (BC-FES-GUI)

Element in the graphical user interface.

The title bar is the top line of every primary window and dialog box in the R/3 System.

It contains the title of the window, and icons that allow you to control the window size.

ABAP Workbench (BC-DWB)

A logical process in the R/3 System.

From the user's point of view, a transaction is a logical unit (for example, to generate a list of customers, change a customer's address, create a reservation for a flight, or run a program). From the programmer's point of view, it is a complex object, consisting of a module pool and a set of screens. You start transactions using a transaction code.

After logging onto the R/3 System, there are three levels - the SAP level, work area level, and application level. A transaction is a process at application level. To start the transaction, you can either use the menus or enter a four-character transaction code. Using the transaction code saves you having to remember the menu path.

To start a program from the ABAP Workbench, you can either choose Tools → ABAP Workbench → ABAP Editor, or enter SE38 in the command field.

ABAP Workbench (BC-DWB)

Sequence of up to twenty characters that identifies an SAP transaction.

When you enter a transaction code in the command field, the corresponding transaction is started in the R/3 System.

For example, the transaction code SM31 identifies the transaction „Display Table“.

ABAP Workbench (BC-DWB)

Table type in the ABAP Dictionary.

You define transparent tables in the ABAP Dictionary. They are created in the database.

Computing Center Management System (BC-CCM)

Document for copying corrections between different kinds of system.

Released corrections are collected in a transport request. When you release the request, it is transported.

For example, you can transport corrections from an integration system into a consolidation system.

ABAP Workbench (BC-DWB)

Function in the ABAP Editor that enables you to avoid unnecessary type conversions in ABAP programs.

When the function is called, the system analyzes the parameters in the PERFORM statements and searches in the FORM statements for formal parameters with similar technical attributes (type and length). Whenever it finds two parameters that correspond, it suggests a type for the formal parameter in the FORM statement. You can then change your coding accordingly.

Basis Services/Communication Interfaces (BC-SRV)

Object in ABAP Query.

The assignment to a user group determines which queries a user is allowed to execute and/or maintain.

ABAP Workbench (BC-DWB)

Technical features and functions available to the user to exchange information with the computer system.

In the R/3 System, you design the user interface in the ABAP Workbench with the Screen Painter and the Menu Painter.

V

ABAP Workbench (BC-DWB)

Application-specific view of different tables in the ABAP Dictionary.

When you create a table, you assign a key according to technical criteria. However, the key fields may be insufficient for solving certain problems, or some of them may be irrelevant. In this case, you can use a view to access part of a table or a series of tables.

W

Web Basis (CA-B-WEB)

WebRFC applicat allowing Internet users to access information in the R/3 System.

Users can access SAP reports, display lists, and navigate through reporting trees using URLs.