BC430 - ABAP Dictionary

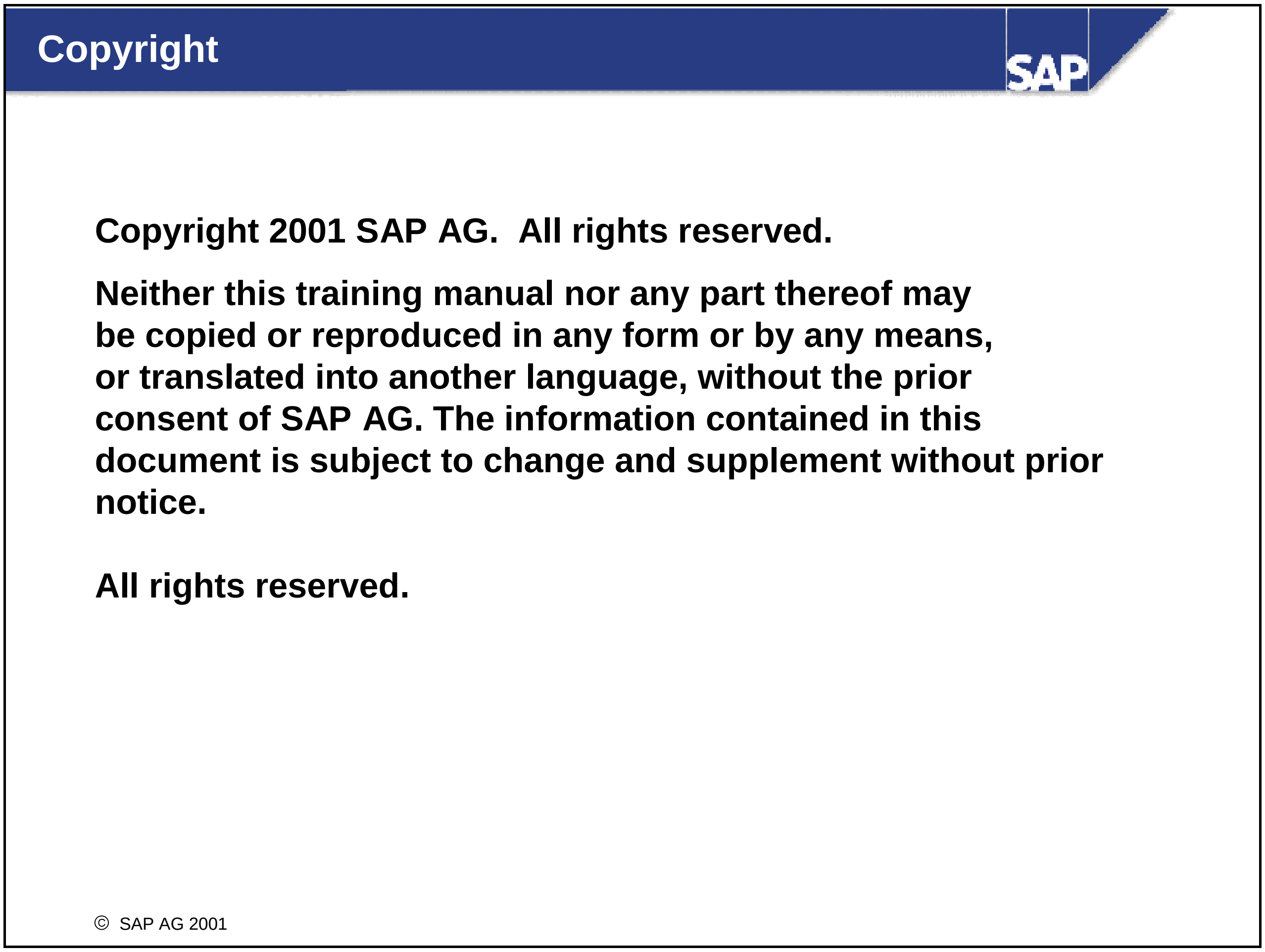
BC430

Release 46B 17.01.2003

BC430

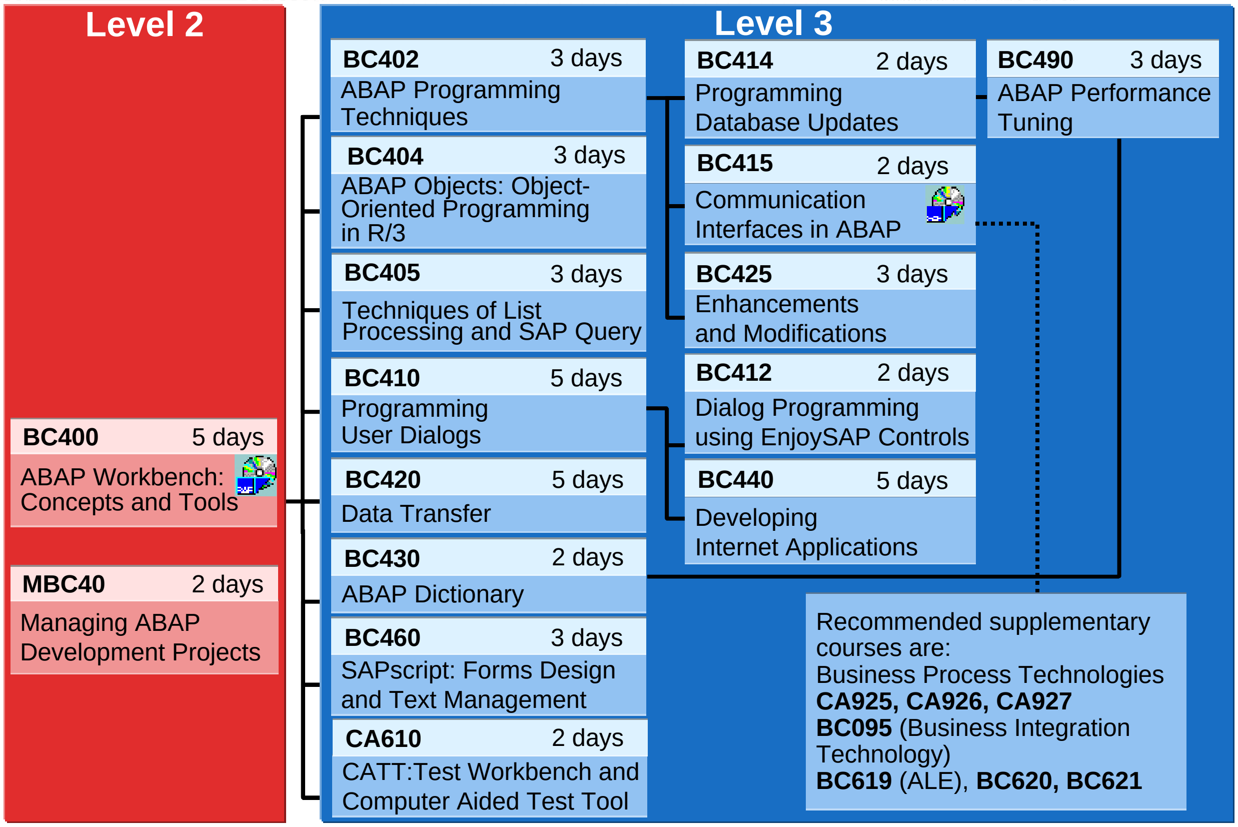
ABAP Dictionary

- R/3 System
- Release: 4.6A
- Version: January 2000
- Material number: 5003 3691



Copyright

ABAP Workbench



Level 2

BC400 5 days
ABAP Workbench: Concepts and Tools

MBC40 2 days
Managing ABAP Development Projects

Level 3

BC402 3 days
ABAP Programming Techniques

BC404 3 days
ABAP Objects: Object-Oriented Programming in R/3

BC405 3 days
Techniques of List Processing and SAP Query

BC410 5 days
Programming User Dialogs

BC420 5 days
Data Transfer

BC430 2 days
ABAP Dictionary

BC460 3 days
SAPscript: Forms Design and Text Management

CA610 2 days
CATT:Test Workbench and Computer Aided Test Tool

BC414 2 days
Programming Database Updates

BC415 2 days
Communication Interfaces in ABAP

BC425 3 days
Enhancements and Modifications

BC412 2 days
Dialog Programming using EnjoySAP Controls

BC440 5 days
Developing Internet Applications

BC490 3 days
ABAP Performance Tuning

Recommended supplementary courses are:
Business Process Technologies
CA925, CA926, CA927
BC095 (Business Integration Technology)
BC619 (ALE), **BC620, BC621**

Course Content

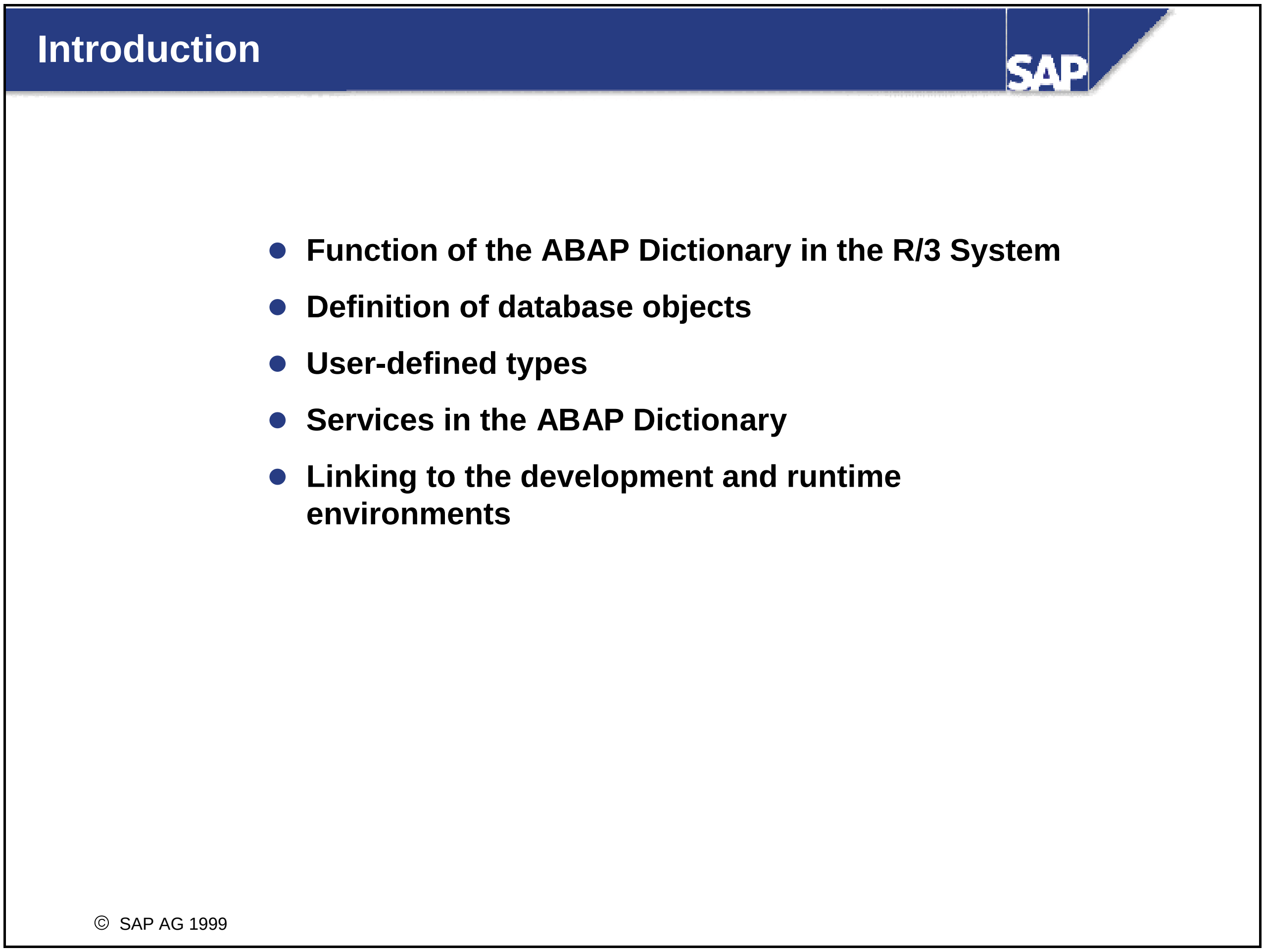
Preface

Unit 1	**Introduction**
Unit 2	**Tables in the ABAP Dictionary**
Unit 3	**Performance in Table Accesses**
Unit 4	**Consistency through Input Checks**
Unit 5	**Dependencies of ABAP Dictionary Objects**
Unit 6	**Changes to Tables**
Unit 7	**Views**
Unit 8	**Search Helps**

Exercises

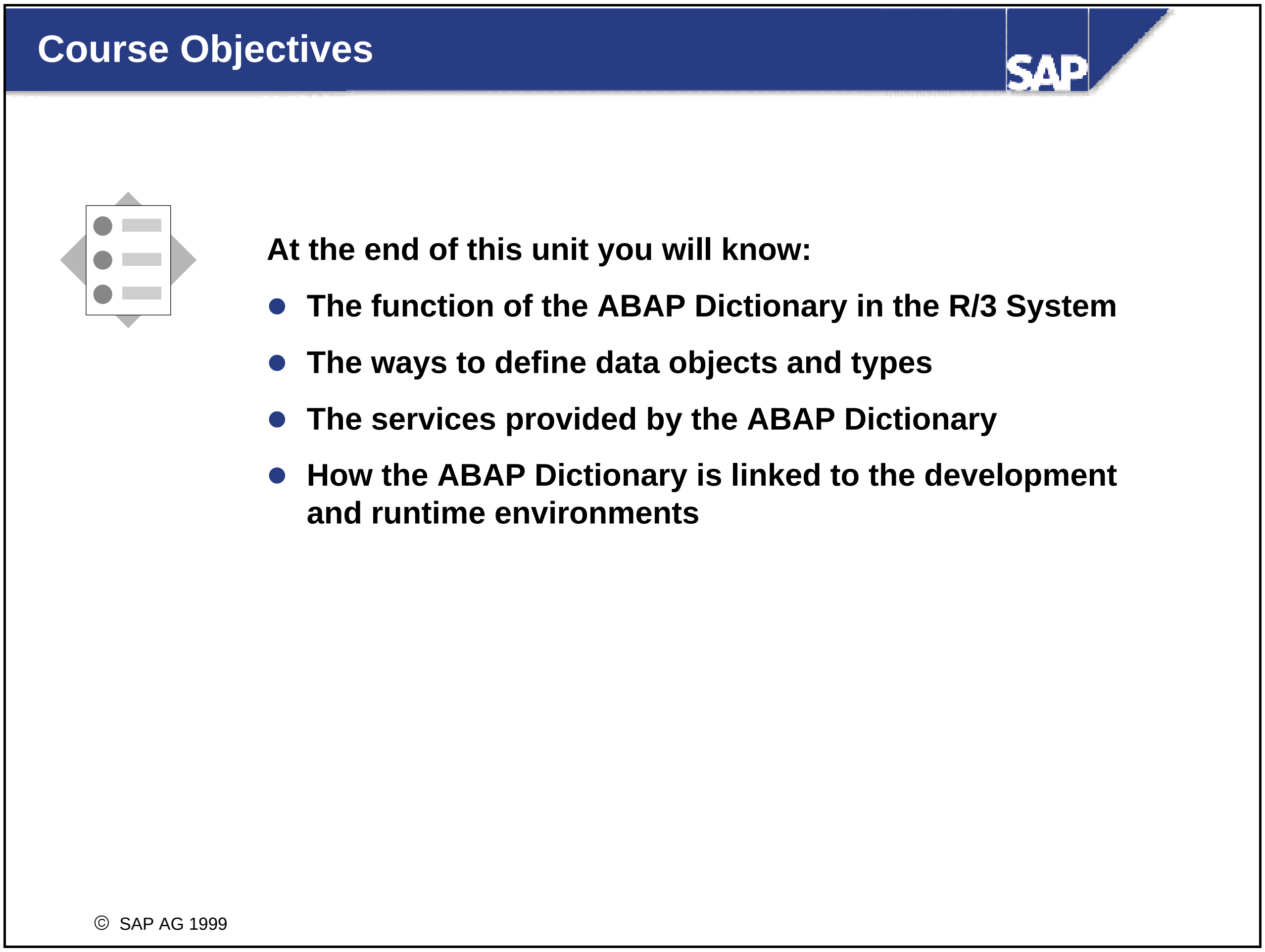
Solutions

Appendixes



Introduction

- Function of the ABAP Dictionary in the R/3 System
- Definition of database objects
- User-defined types
- Services in the ABAP Dictionary
- Linking to the development and runtime environments

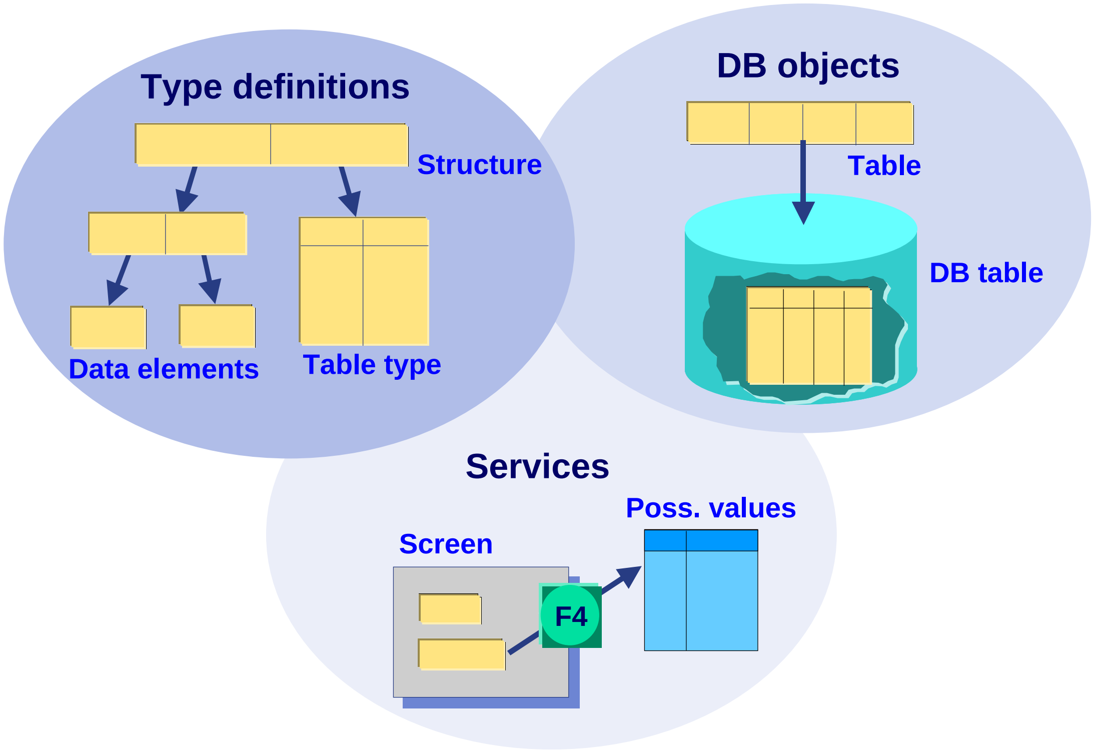


Course Objectives

At the end of this unit you will know:

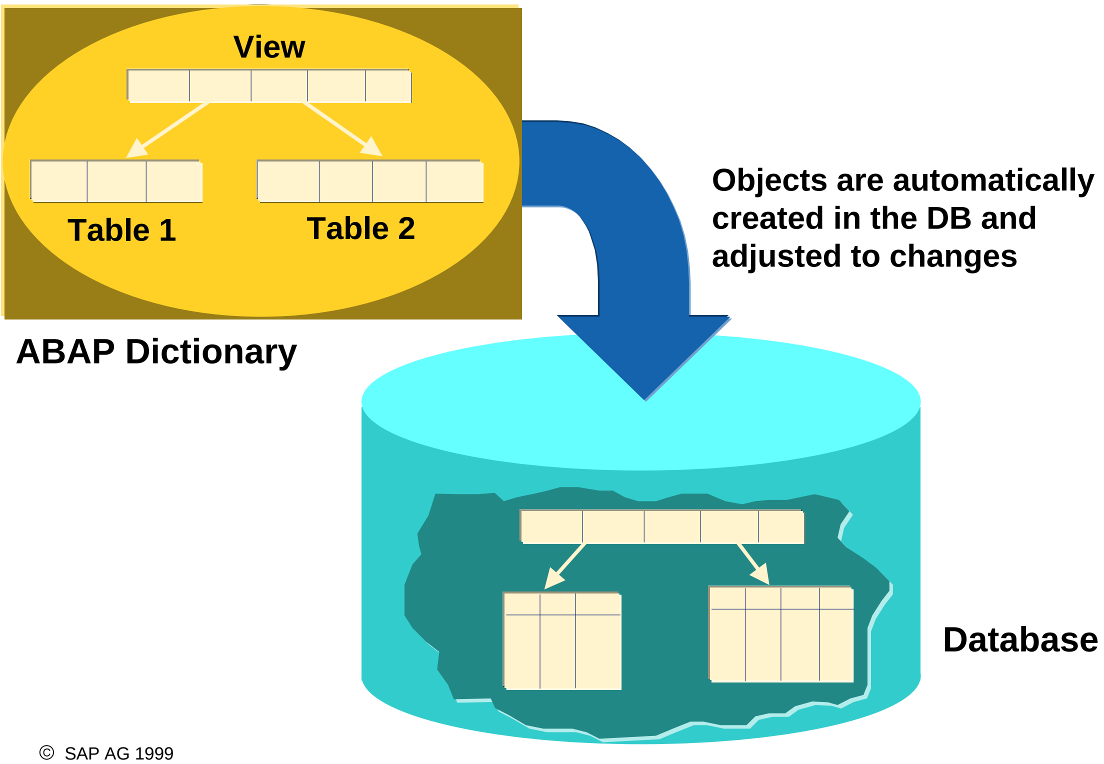
- The function of the ABAP Dictionary in the R/3 System
- The ways to define data objects and types
- The services provided by the ABAP Dictionary
- How the ABAP Dictionary is linked to the development and runtime environments

Function of the ABAP Dictionary



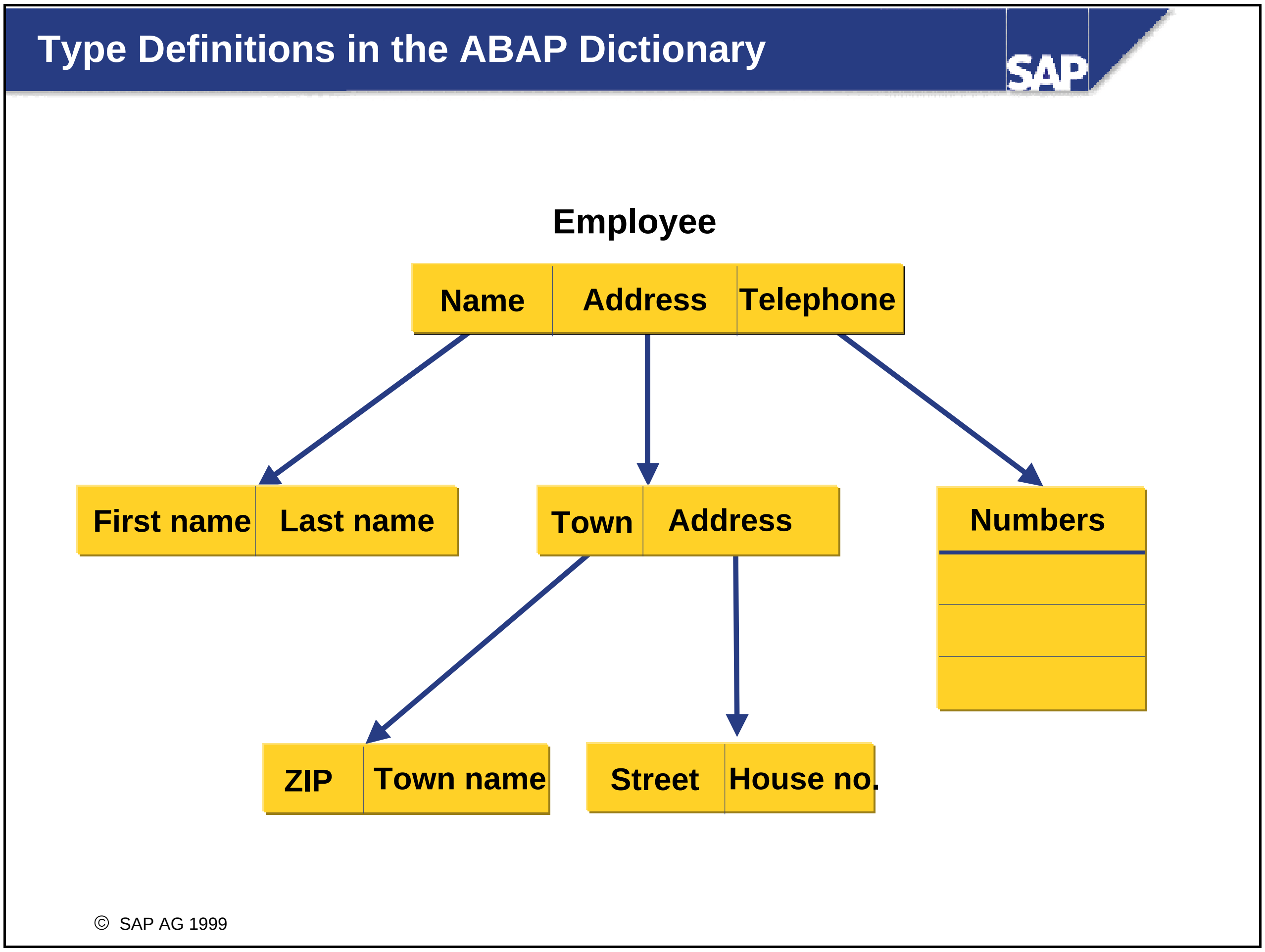
- The ABAP Dictionary permits a central management of all the data definitions used in the R/3 System.
- In the ABAP Dictionary you can create user-defined types (data elements, structures and table types) for use in ABAP programs or in interfaces of function modules. Database objects such as tables and database views can also be defined in the ABAP Dictionary and created with this definition in the database.
- The ABAP Dictionary also provides a number of services that support program development. For example, setting and releasing locks, defining an input help (F4 help) and attaching a field help (F1 help) to a screen field are supported.

Database Objects in the ABAP Dictionary



Objects are automatically created in the DB and adjusted to changes

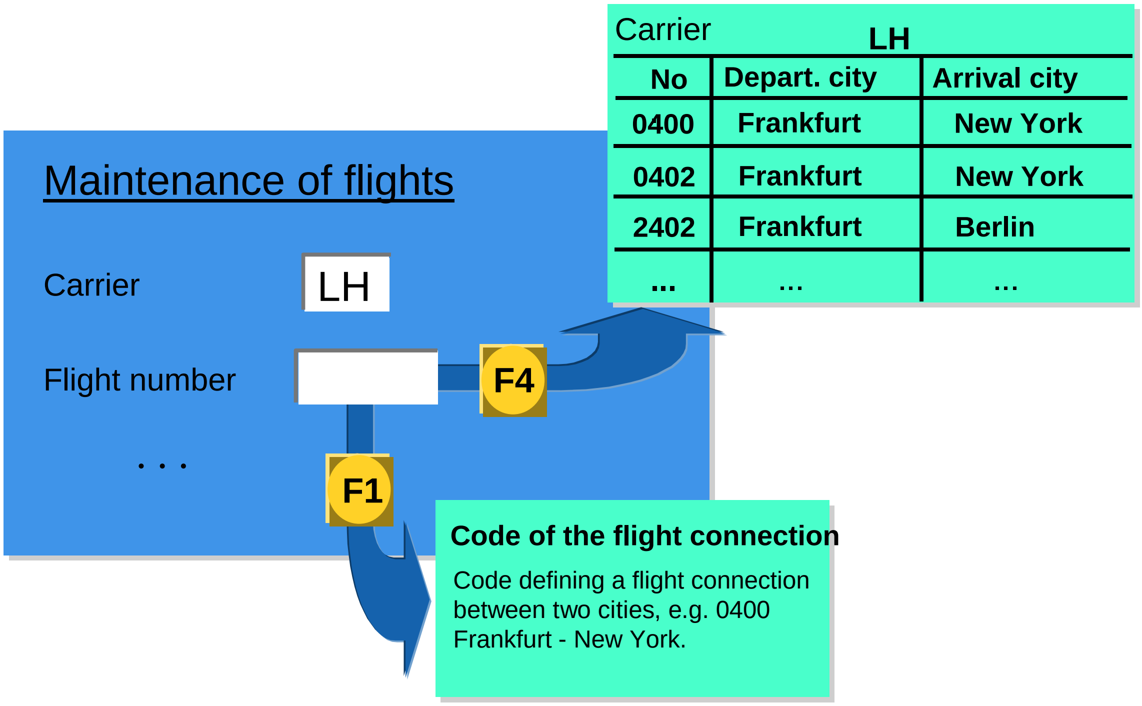
- Tables and database views can be defined in the ABAP Dictionary.
- These objects are created in the underlying database with this definition. Changes in the definition of a table or database view are also automatically made in the database.
- Indexes can be defined in the ABAP Dictionary to speed up access to data in a table. These indexes are also created in the database.



Type Definitions in the ABAP Dictionary

- There are three different type categories in the ABAP Dictionary:
  - **Data elements:** Describe an elementary type by defining the data type, length and possibly decimal places.
  - **Structures:** Consist of components that can have any type.
  - **Table types:** Describe the structure of an internal table.
- Any complex user-defined type can be built from these basic types.
- **Example:** The data of an employee is stored in a structure EMPLOYEE with the components NAME, ADDRESS and TELEPHONE. Component NAME is also a structure with components FIRST NAME and LAST NAME. Both of these components are elementary, i.e. their type is defined by a data element. The type of component ADDRESS is also defined by a structure whose components are also structures. Component TELEPHONE is defined by a table type (since an employee can have more than one telephone number).
- Types are used for example in ABAP programs or to define the types of interface parameters of function modules.

Services of the ABAP Dictionary



Maintenance of flights

Carrier: LH

Flight number:

Carrier LH

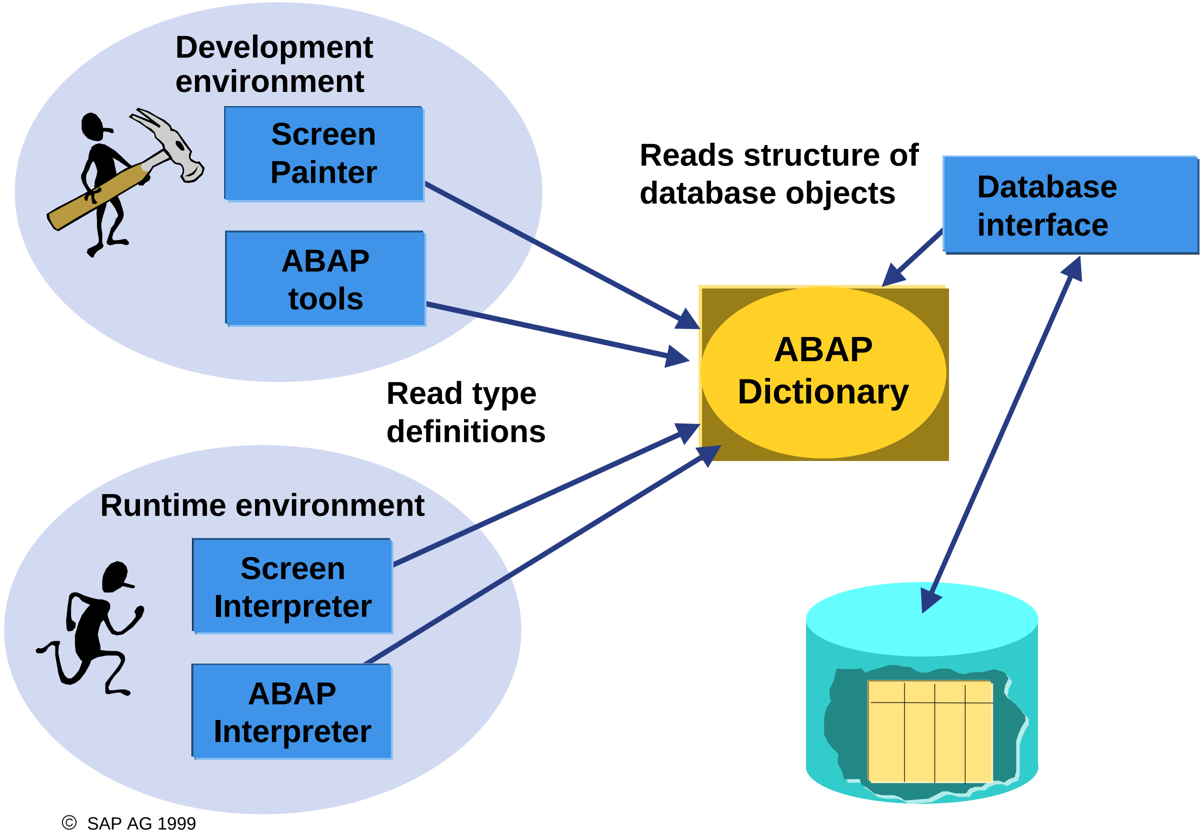
No	Depart. city	Arrival city
0400	Frankfurt	New York
0402	Frankfurt	New York
2402	Frankfurt	Berlin
...	...	...

Code of the flight connection

Code defining a flight connection between two cities, e.g. 0400 Frankfurt - New York.

- The ABAP Dictionary supports program development with a number of services:
  - Input helps (F4 helps) for screen fields can be defined with search helps.
  - Screen fields can easily be assigned a field help (F1 help) by creating documentation for the data element.
  - An input check that ensures that the values entered are consistent can easily be defined for screen fields using foreign keys.
  - The ABAP Dictionary provides support when you set and release locks. To do so, you must create lock objects in the ABAP Dictionary. Function modules for setting and releasing locks are automatically generated from these lock objects; these can then be linked into the application program.
  - The performance when accessing this data can be improved for database objects (tables, views) with buffering settings.
  - By logging, you can switch on the automatic recording of changes to the table entries.

Linking to the Development and Runtime Environment



Development environment

Screen Painter

ABAP tools

Reads structure of database objects

Database interface

ABAP Dictionary

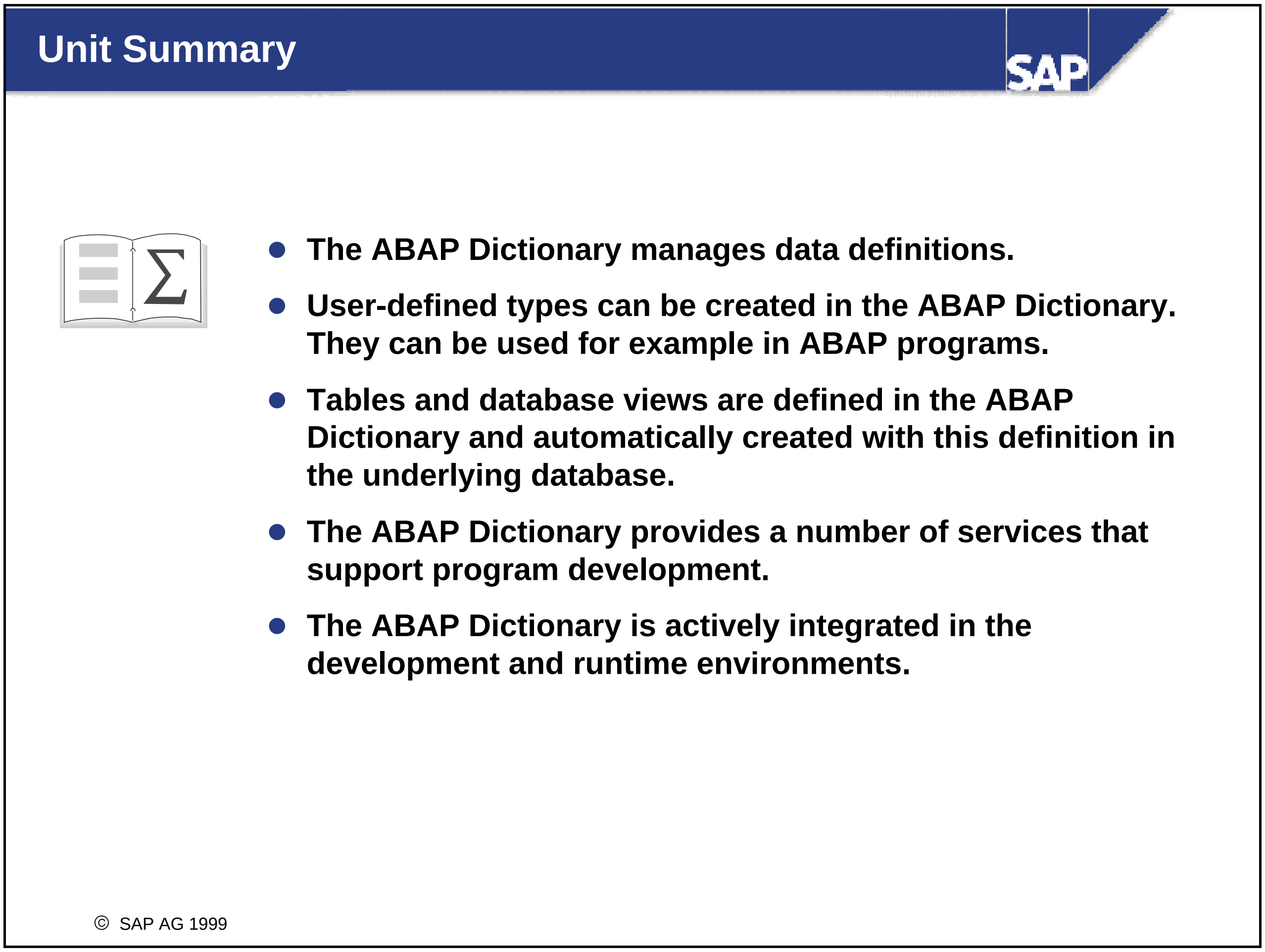
Read type definitions

Runtime environment

Screen Interpreter

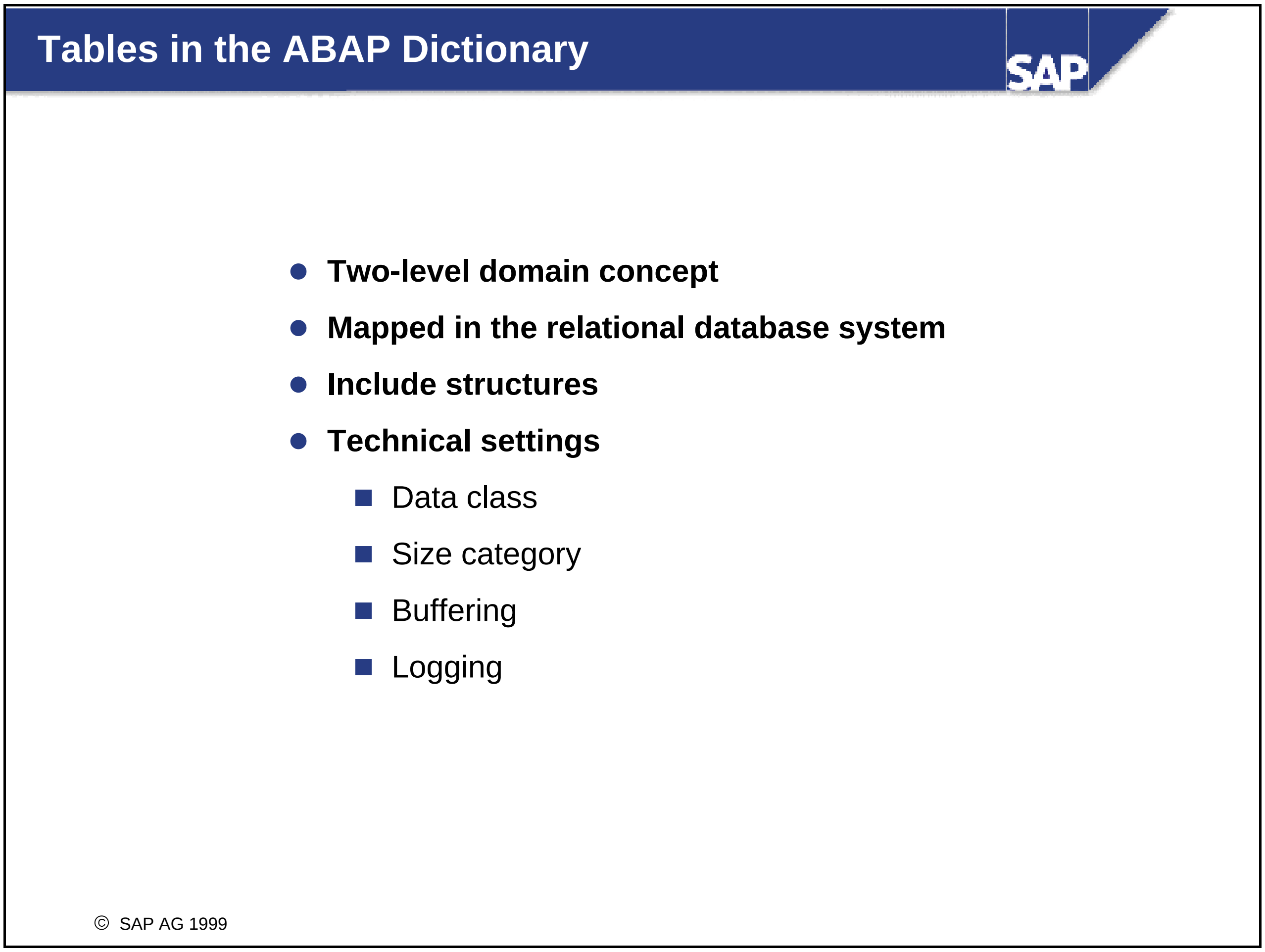
ABAP Interpreter

- The ABAP Dictionary is actively integrated in the development and runtime environments. Each change takes immediate effect in the relevant ABAP programs and screens.
- **Examples:**
  - When a program or screen is generated, the ABAP interpreter and the screen interpreter access the type definitions stored in the ABAP Dictionary.
  - The ABAP tools and the Screen Painter use the information stored in the ABAP Dictionary to support you during program development. An example of this is the *Get from Dictionary* function in the Screen Painter, with which you can place fields of a table or structure defined in the ABAP Dictionary in a screen.
  - The database interface uses the information about tables or database views stored in the ABAP Dictionary to access the data of these objects.



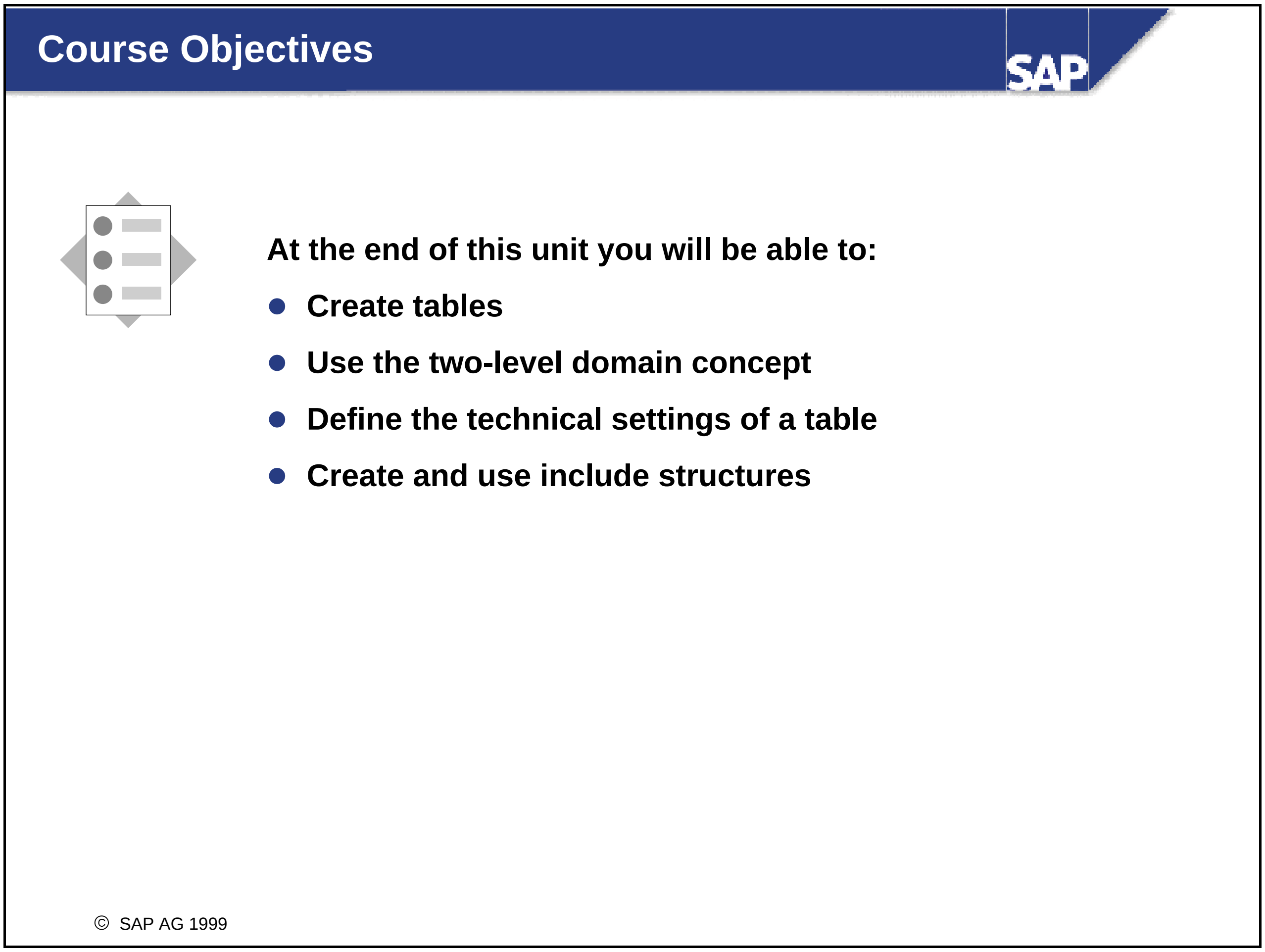
Unit Summary

- The ABAP Dictionary manages data definitions.
- User-defined types can be created in the ABAP Dictionary. They can be used for example in ABAP programs.
- Tables and database views are defined in the ABAP Dictionary and automatically created with this definition in the underlying database.
- The ABAP Dictionary provides a number of services that support program development.
- The ABAP Dictionary is actively integrated in the development and runtime environments.



Tables in the ABAP Dictionary

- **Two-level domain concept**
- **Mapped in the relational database system**
- **Include structures**
- **Technical settings**
  - Data class
  - Size category
  - Buffering
  - Logging

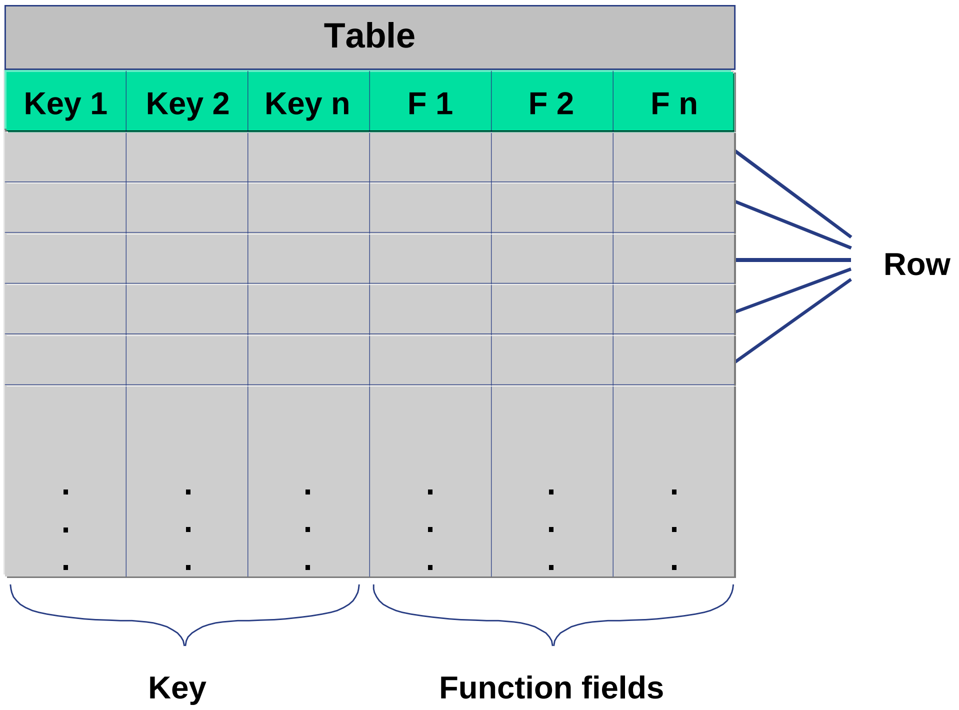


Course Objectives

At the end of this unit you will be able to:

- Create tables
- Use the two-level domain concept
- Define the technical settings of a table
- Create and use include structures

Tables and Fields



Table

Key 1 | Key 2 | Key n | F 1 | F 2 | F n

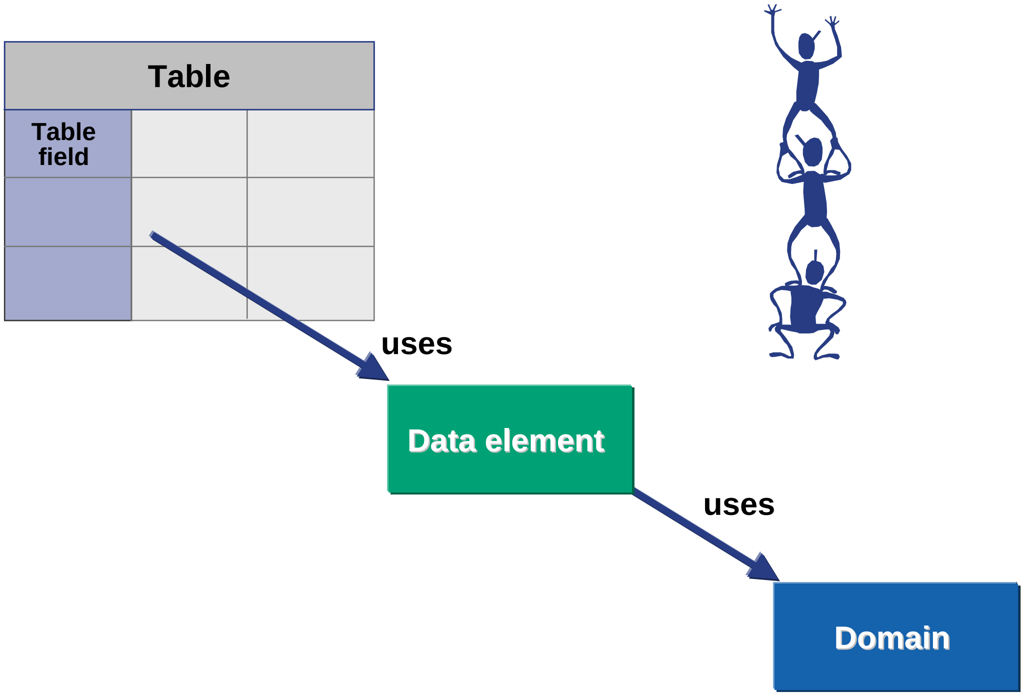
Row

Key

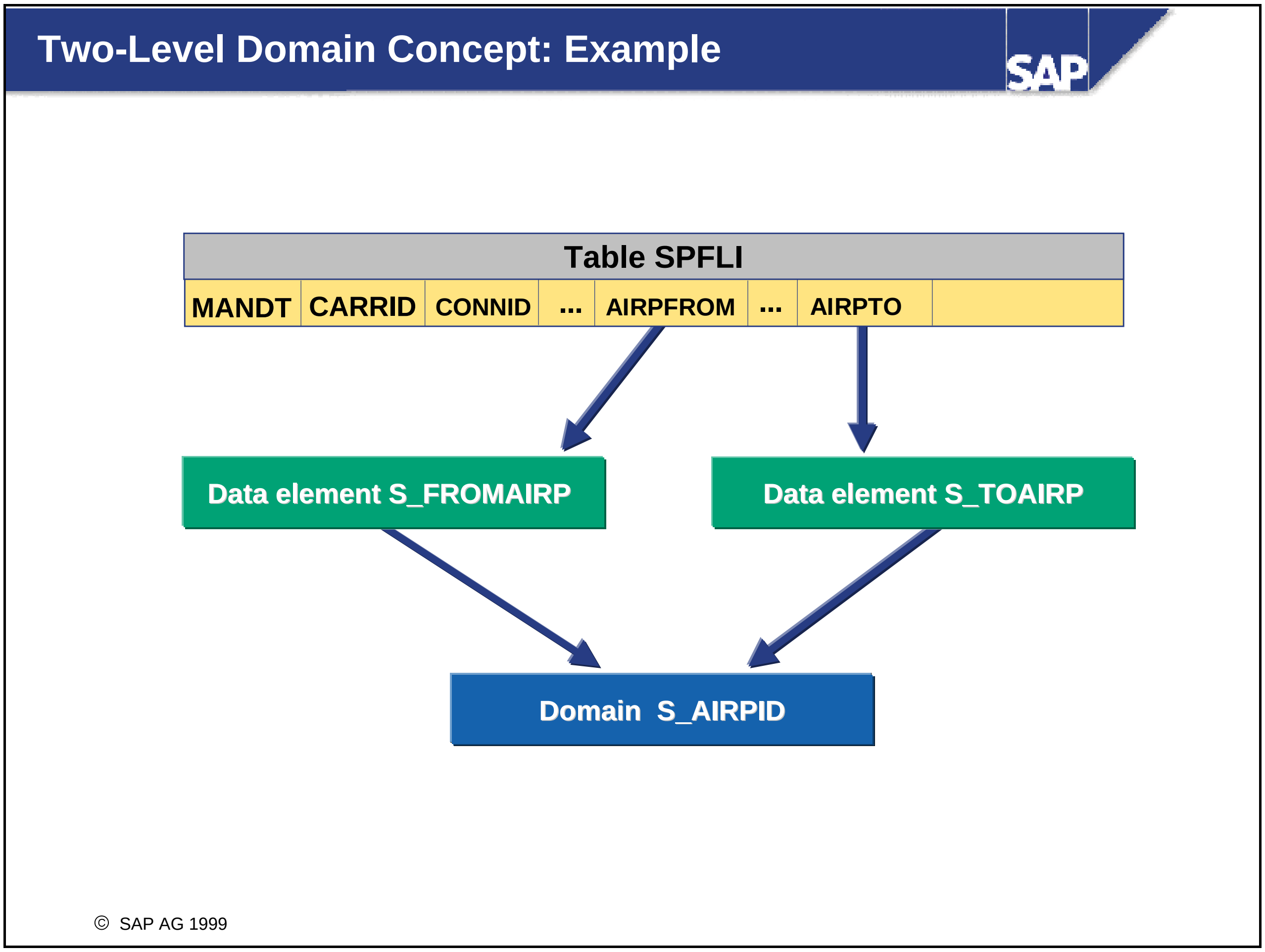
Function fields

- The structure of the objects of application development are mapped in tables on the underlying relational database.
- The attributes of these objects correspond to fields of the table.
- A table consists of columns (fields) and rows (entries). It has a name and different attributes, such as delivery class and maintenance authorization.
- A field has a unique name and attributes; for example it can be a key field.
- A table has one or more key fields, called the primary key.
- The values of these key fields uniquely identify a table entry.
- You must specify a *reference table* for fields containing a currency (data type CURR) or quantity (data type QUAN). It must contain a field (*reference field*) with the format for currency keys (data type CUKY) or the format for units (data type UNIT). The field is only assigned to the reference field at program runtime.

Basic Objects of the ABAP Dictionary



- The basic objects for defining data in the ABAP Dictionary are tables, data elements and domains. The domain is used for the technical definition of a table field (for example field type and length) and the data element is used for the semantic definition (for example short description).
- A **domain** describes the value range of a field. It is defined by its data type and length. The value range can be limited by specifying fixed values.
- A **data element** describes the meaning of a domain in a certain business context. It contains primarily the field help (F1 documentation) and the field labels in the screen.
- A field is not an independent object. It is table-dependent and can only be maintained within a table.
- You can enter the data type and number of places directly for a field. No data element is required in this case. Instead the data type and number of places is defined by specifying a **direct type**.
- The data type attributes of a data element can also be defined by specifying a **built-in type**, where the data type and number of places is entered directly.



Two-Level Domain Concept: Example

Table SPFLI

MANDT	CARRID	CONNID	...	AIRPFROM	...	AIRPTO	

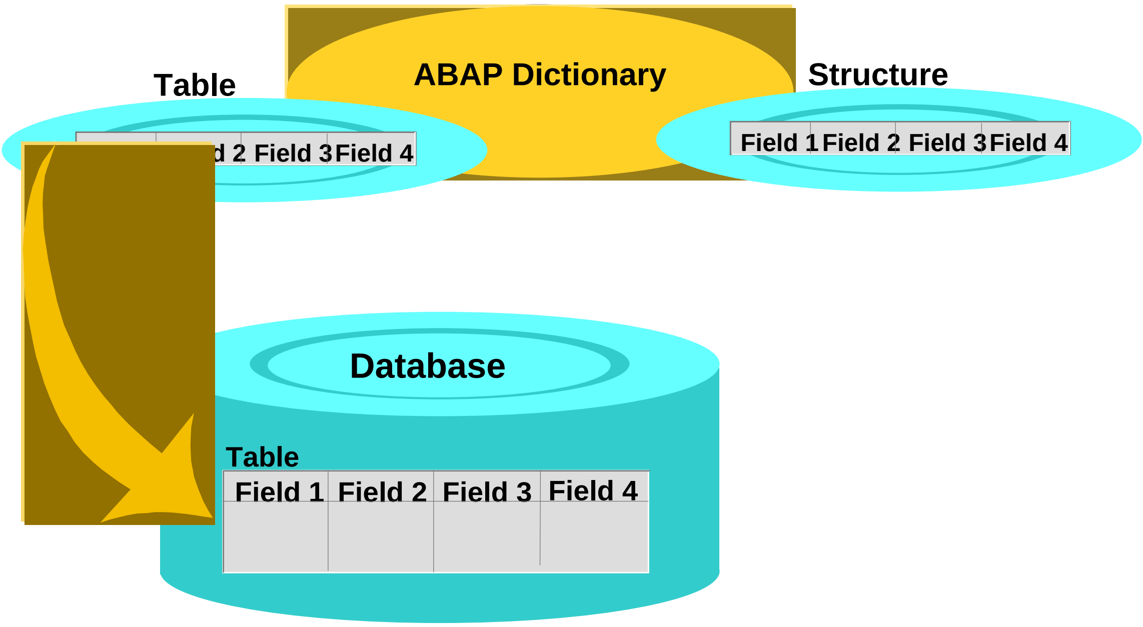
Data element S_FROMAIRP

Data element S_TOAIRP

Domain S_AIRPID

- The flight schedule is stored in table SPFLI. Table fields AIRPFROM (departure airport) and AIRPTO (arrival airport) have the same domain S_AIRPID. Both fields use the same domain because both fields contain airport IDs and therefore have the same technical attributes. They have a different semantic meaning, however, and use different data elements to document this. Field AIRPFROM uses data element S_FROMAIRP and field AIRPTO uses data element S_TOAIRP.

Transparent Tables and Structures


ABAP Dictionary
Table
d 2 Field 3 Field 4
Structure
Field 1 Field 2 Field 3 Field 4
Database
Table
Field 1 Field 2 Field 3 Field 4

Physical definition of the table

- A transparent table is automatically created on the database when it is activated in the ABAP Dictionary. At this time the database-independent description of the table in the ABAP Dictionary is translated into the language of the database system used.
- The database table has the same name as the table in the ABAP Dictionary. The fields also have the same name in both the database and the ABAP Dictionary. The data types in the ABAP Dictionary are converted to the corresponding data types of the database system.
- The order of the fields in the ABAP Dictionary can differ from the order of the fields on the database. This permits you to insert new fields without having to convert the table. When a new field is added, the adjustment is made by changing the database catalog (ALTER TABLE). The new field is added to the database table, whatever the position of the new field in the ABAP Dictionary.
- ABAP programs can access a transparent table in two ways. One way is to access the data contained in the table with OPEN SQL (or EXEC SQL). With the other method, the table defines a structured type that is accessed when variables (or more complex types) are defined.
- You can also create a structured type in the ABAP Dictionary for which there is no corresponding object in the database. Such types are called **structures**. Structures can also be used to define the types of variables.

Include Structures

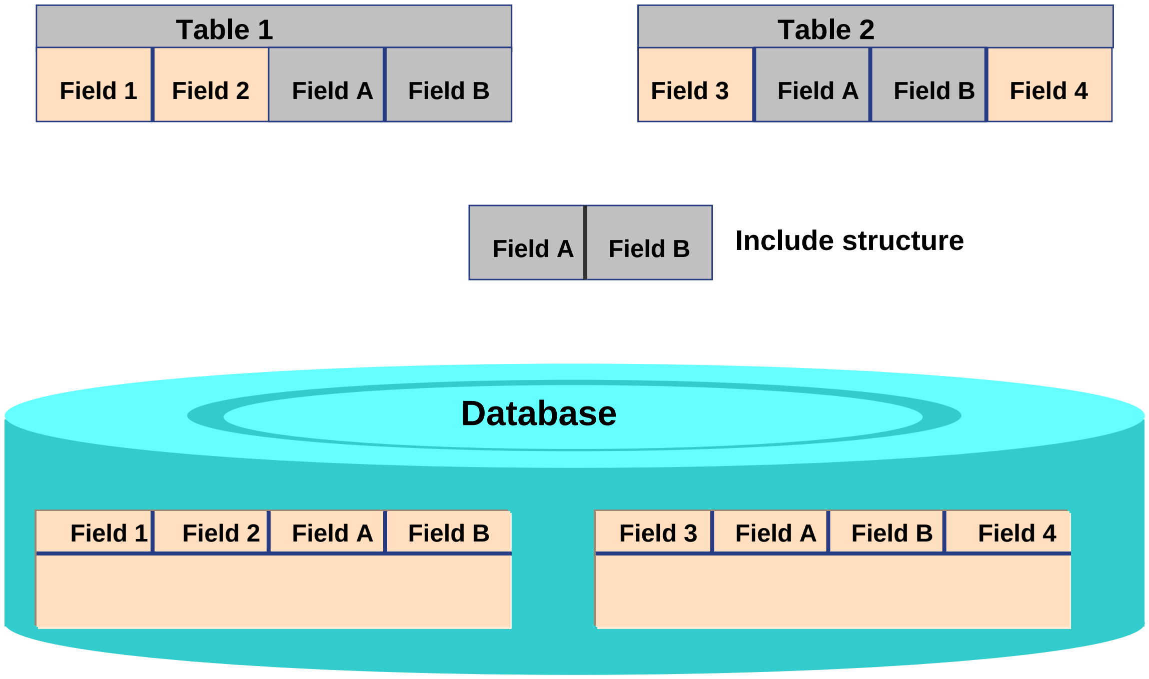


Table 1

Field 1 | Field 2 | Field A | Field B

Table 2

Field 3 | Field A | Field B | Field 4

Field A | Field B

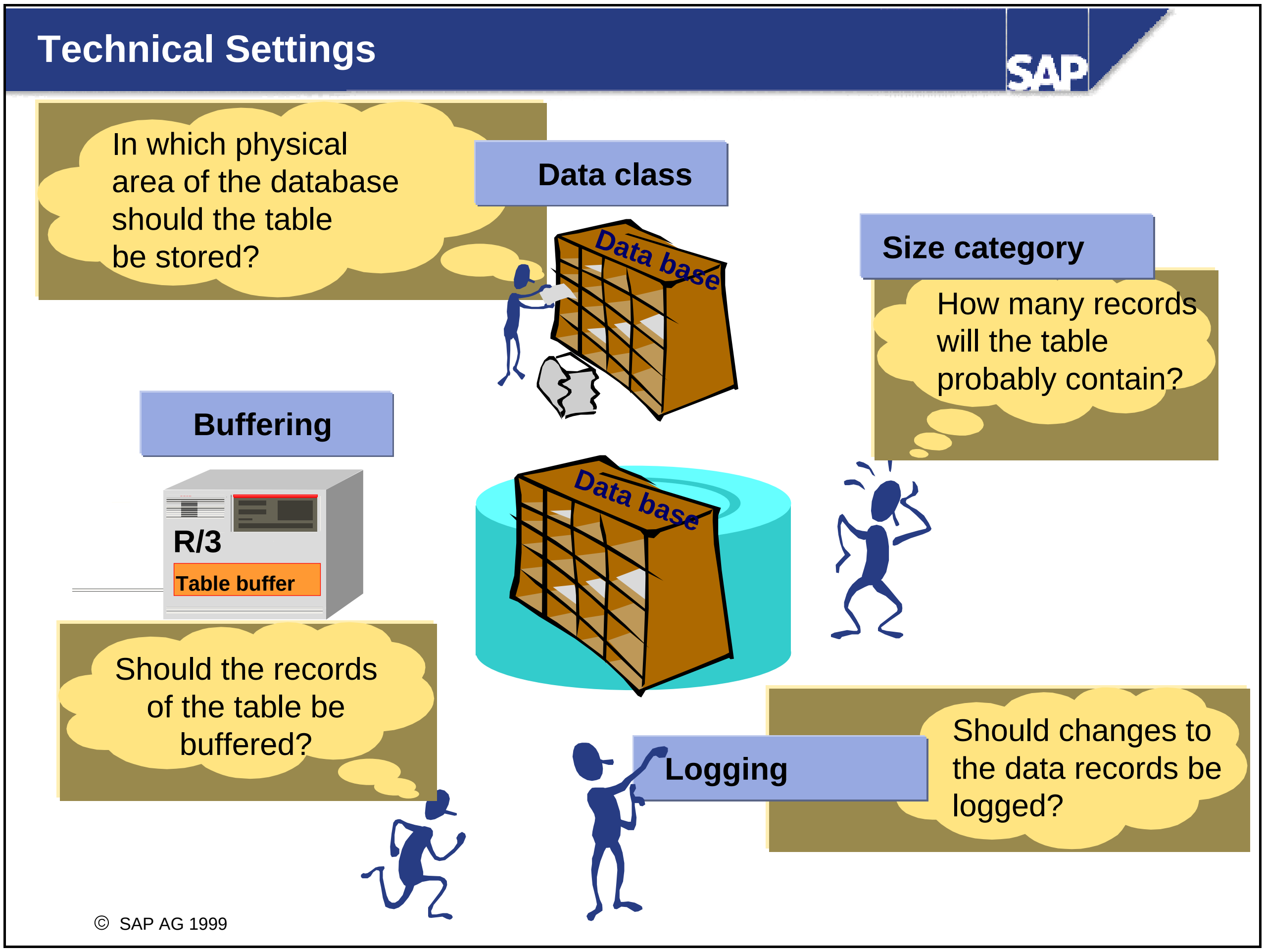
Include structure

Database

Field 1 | Field 2 | Field A | Field B

Field 3 | Field A | Field B | Field 4

- Structures can be included in tables or other structures to avoid redundant structure definitions.
- A table may only be included as an entire table.
- A chain of includes may only contain one database table. The table in which you are including belongs to the include chain. This means that you may not include a transparent table in a transparent table.
- Includes may contain further includes.
- Foreign key definitions are generally imparted from the include to the including table. The attributes of the foreign key definition are passed from the include to the including table so that the foreign key depends on the definition in the include.



Technical Settings

Data class

In which physical area of the database should the table be stored?

Size category

How many records will the table probably contain?

Buffering

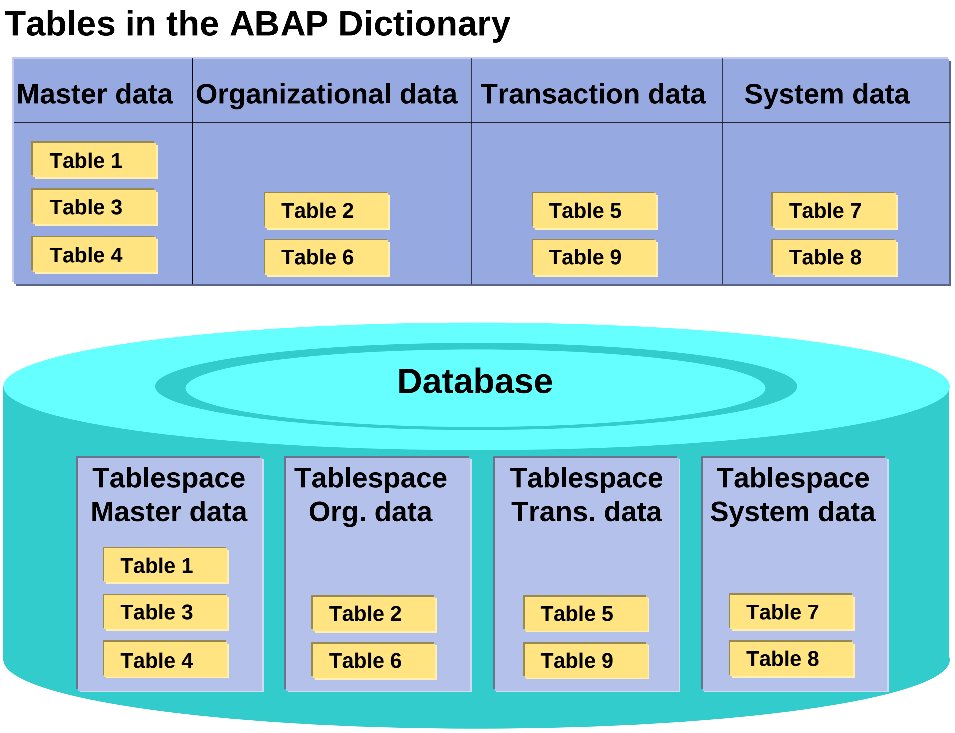
Should the records of the table be buffered?

Logging

Should changes to the data records be logged?

- You must maintain the technical settings when you define a transparent table in the ABAP Dictionary.
- The technical settings are used to individually optimize the storage requirements and accessing behavior of database tables.
- The technical settings can be used to define how the table should be handled when it is created on the database, whether the table should be buffered and whether changes to entries should be logged.
- The table is automatically created on the database when it is activated in the ABAP Dictionary. The storage area to be selected (tablespace) and space allocation settings are determined from the settings for the data class and size category.
- The settings for buffering define whether and how the table should be buffered.
- You can define whether changes to the table entries should be logged.

Data Class



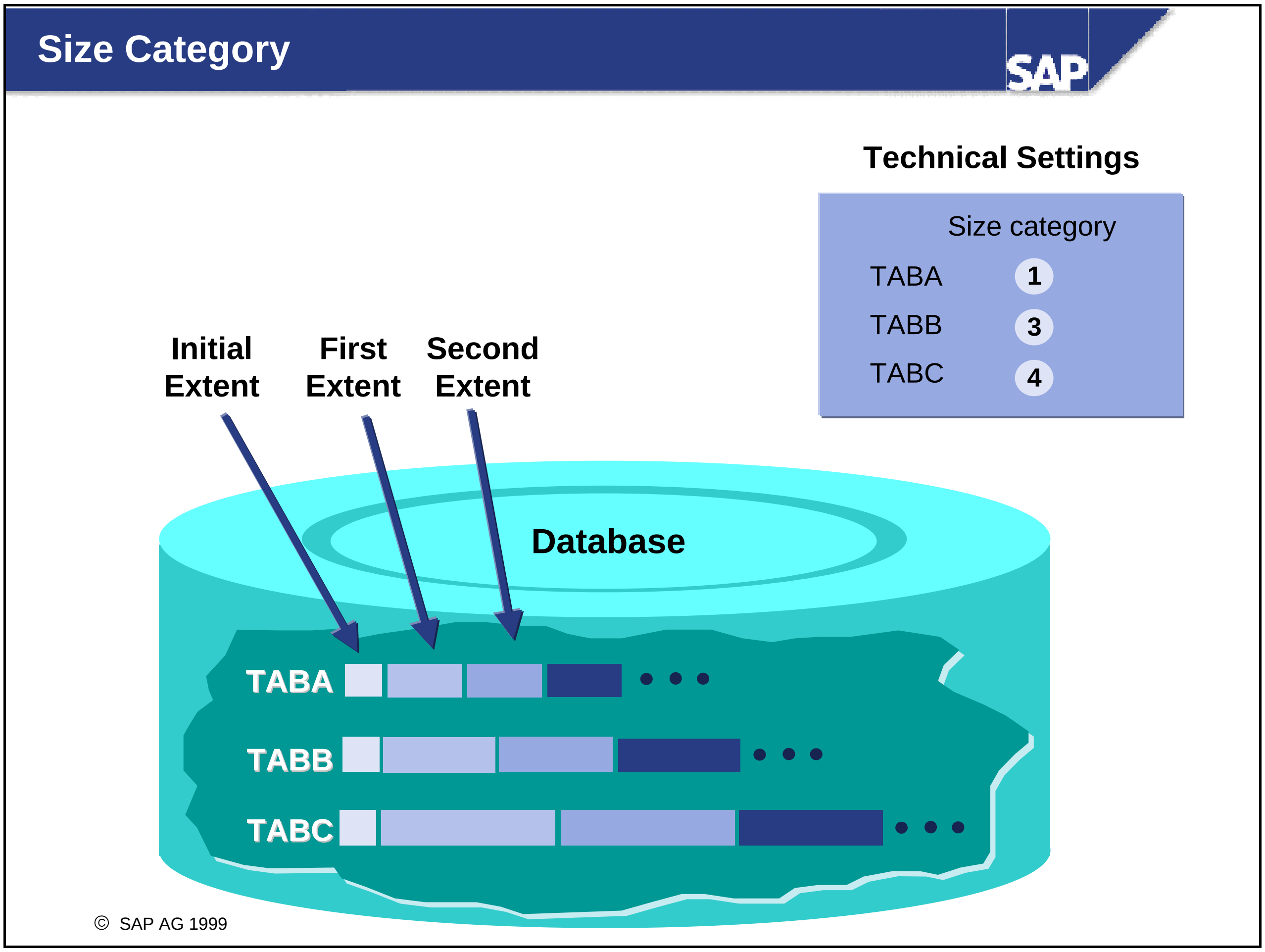
Tables in the ABAP Dictionary

Master data	Organizational data	Transaction data	System data
Table 1 Table 3 Table 4	Table 2 Table 6	Table 5 Table 9	Table 7 Table 8

Database

Tablespace Master data	Tablespace Org. data	Tablespace Trans. data	Tablespace System data
Table 1 Table 3 Table 4	Table 2 Table 6	Table 5 Table 9	Table 7 Table 8

- The data class logically defines the physical area of the database (for ORACLE the tablespace) in which your table should be stored. If you choose the data class correctly, the table will automatically be created in the appropriate area on the database when it is activated in the ABAP Dictionary.
- The most important data classes are **master data**, **transaction data**, **organizational data** and **system data**.
- Master data is data that is rarely modified. An example of master data is the data of an address file, for example the name, address and telephone number.
- Transaction data is data that is frequently modified. An example is the material stock of a warehouse, which can change after each purchase order.
- Organizational data is data that is defined during customizing when the system is installed and that is rarely modified thereafter. The country keys are an example.
- System data is data that the R/3 System itself needs. The program sources are an example.
- Further data classes, called customer data classes (USR, USR1), are provided for customers. These should be used for customer developments. Special storage areas must be allocated in the database.



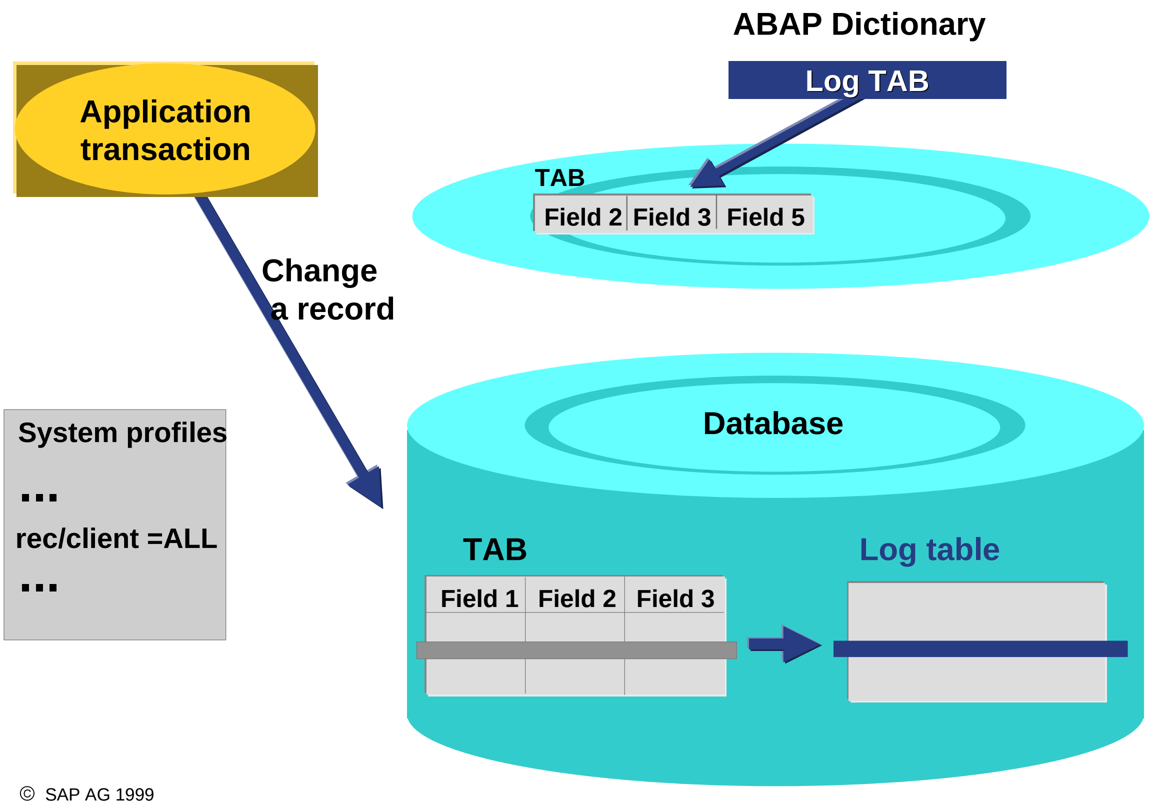
Size Category

Technical Settings

Size category	
TABA	1
TABB	3
TABC	4

- The size category describes the expected storage requirements for the table on the database.
- An initial extent is reserved when a table is created on the database. The size of the initial extent is identical for all size categories. If the table needs more space for data at a later time, extents are added. These additional extents have a fixed size that is determined by the size category specified in the ABAP Dictionary.
- You can choose a size category from 0 to 4. A fixed extent size, which depends on the database system used, is assigned to each category.
- Correctly assigning a size category therefore ensures that you do not create a large number of small extents. It also prevents storage space from being wasted when creating extents that are too large.

Logging



ABAP Dictionary

Log TAB

TAB

Field 2	Field 3	Field 5

Application transaction

Change a record

System profiles

...

rec/client =ALL

...

Database

TAB

Field 1	Field 2	Field 3

Log table

- Modifications to the entries of a table can be recorded and stored using logging.
- To activate logging, the corresponding field must be selected in the technical settings. Logging, however, only will take place if the R/3 System was started with a profile containing parameter *'rec/client'*. Only selecting the flag in the ABAP Dictionary is not sufficient to trigger logging.
- Parameter *'rec/client'* can have the following settings:

  rec/client = ALL All clients should be logged.
  rec/client = 000[...] Only the specified clients should be logged.
  rec/client = OFF Logging is not enabled on this system.
- The data modifications are logged independently of the update. The logs can be displayed with the Transaction *Table History* (SCU3).
- **Logging creates a 'bottleneck' in the system:**
  - Additional write access for each modification to tables being logged.
  - This can result in lock situations although the users are accessing different application tables!

Unit Summary

- All the business-oriented data is administered in the form of tables whose definition is stored in the ABAP Dictionary.
- A two-level domain concept is used for defining the tables. The semantic definition is implemented with data elements and the technical definition with domains.
- The fields of include structures can be included in tables.
- The technical settings of a table define how the table should be stored in the database (tablespace, extent size) and whether changes to the data records should be logged.

Exercise Data

Explanation of the Symbols in the Exercises and Solutions

?	**Exercises**
📝	**Solutions**
📋	Course Objectives
➤	Business Scenario
💡	Tips & Tricks
⚡	Warning or Caution

Data in the Exercises

Type of data	Data in the training system
Data model BC_TRAVEL	yes
All objects in development class BC_DATAMODEL	yes
All objects in development class BC430	yes

When creating ABAP Dictionary objects in this course, you should adhere to the following conventions:

- Your object names for tables, data elements and domains should begin with Z and end with your two-digit group number (xx).
- Use both your own data elements or domains (Z<Objectname>xx) and standard SAP objects for the table fields.
- All objects should be created as local objects (development class **$tmp**).

The appendix contains information about the flight data model used in the training courses.

Exercises: Tables in the ABAP Dictionary

Unit: Tables in the ABAP Dictionary

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

- Create tables and use the two-level domain concept
- Define the technical settings sensibly
- Document fields
- Create and use include structures

In these exercises the tables of the flight model will be enhanced with employee management. This employee management will enable the airlines to enter and evaluate data about their employees (e.g. name, personnel number, salary, department, etc.) and about assignments within the organization (airline departments).

In this exercise, two tables will be created for the employee data and the airline departments.

These tables will be enhanced step by step in the following exercises.

2-1 Create two transparent tables ZEMPLOYxx and ZDEPMENTxx and define their key fields. Define the technical settings when you activate the tables.

Note the following:

Data is maintained for three airlines. An airline has 20,000 employees and between 10 and 30 departments. Do not buffer or log the data. Buffering will be discussed in the exercises for the next unit.

2-1-1 Create table ZEMPLOYxx. The data for the employees is maintained here. The names and addresses of the employees and their salaries is stored here.

Table ZEMPLOYxx

Field name	Data element	Domain	Type, Length
Client	S_MANDT	MANDT	
Carrier	S_CARR_ID	S_CARR_ID	
Personnel number	own	own	NUMC, 10
First name	S_FNAME	S_FNAME	
Last name	S_LNAME	S_LNAME	
Department code	own	own	CHAR, 4
Area	own	own	CHAR, 1
Salary	own	own	CURR, 10 2 decimal places
Currency	S_CURRCODE	S_CURR	

2-1-2 Create table ZDEPMENTxx. This table contains the departments of the airline. Each department can be reached with a telephone and fax number.

Table ZDEPMENTxx

Field name	Data element	Domain	Type, Length
Client	S_MANDT	MANDT	
Carrier	S_CARR_ID	S_CARR_ID	
Department code	own	own	CHAR, 4
Telephone	own	S_PHONE	CHAR, 30
Fax	own	S_PHONE	CHAR, 30

2-2 Document fields *Personnel number* and *Department code.*

2-3 Changes to tables ZEMPLOYxx and ZDEPMENTxx are critical and therefore must be recorded. The maintenance transaction must note who last changed a table entry. This can be done by appending fields for the personnel number of the last person to change the entry and the date of the last entry to tables ZEMPLOYxx and ZDEPMENTxx. Make sure that the same fields are available in both tables for recording the changes by adding these fields to both tables with a substructure ZCHANGExx. Create a new data element for field *Lastchangedby*, but use an existing domain. Use S_CHDATE as data element for the date of the last change.

What actions are executed on activation in the database?

Note: In a 'real' application, the above enhancement would always cause the standard table maintenance for the two tables to be deactivated. Your own maintenance transactions would instead be created for the table in which the fields for change logging would be filled internally by the program and not directly by the user.
Creation of such transactions goes beyond the scope of this course. In this course we will therefore assume that all users themselves (correctly) fill these fields in the standard table maintenance routine.

Start Program BC430_CHECK in Transaction SE38. This program checks whether your solutions are correct and fills the new tables ZEMPLOYxx and ZDEPMENTxx with sample data needed for later exercises.

Solutions: Tables in the ABAP Dictionary

Unit: Tables in the ABAP Dictionary

2-1 The path

Tools → **ABAP Workbench** → **Development** → **Dictionary** or Transaction SE11 takes you to the overview screen of the ABAP Dictionary.

2-1-1 To create table ZEMPLOYxx:

1) Mark *Database table* and enter table name ZEMPLOYxx in the corresponding input field.

2) Choose *Create*.

3) Enter a short text in the maintenance screen for the table.

4) Choose delivery class A and mark *Table maintenance allowed*.

5) Now click on tab page *Fields* to go to the maintenance screen for the field definitions. Enter the field names there (they need not lie in the customer namespace).

6) Use the given data elements for fields *Client, Carrier, First name, Last name* and *Currency* by entering the names of the data elements in column *Field type*. Save your entries.

7) Create your own data elements for fields *Personnel number, Department code, Area* and *Salary*. Enter a name (Z<object>xx) for the data element in column *Field type*. Double-click on the name of the data element. The data element definition appears.

8) Enter a short text (component of the F1 help). Now click on tab page *Field label* and store the texts for the field labels there.

9) You also have to assign the data element a technical description (domain). Click on tab page *Definition* and enter a name (Z<object>xx) for your domain there. If the domain is predefined, activate the data element and return to the maintenance screen for the table fields (F3 or ←).. Otherwise double-click on the domain name. The domain definition appears.

10) Define the short description, data type (for example NUMC) and number of characters (for example 10) there. Activate the domain.

11) Navigate back one screen (F3 or ←) to the data element definition and activate your data element.

12) Navigate back one more screen to the field definition. Start again there with 7) until all the table fields are defined. Save your table.

13) Define the *reference table* and *reference field* for the salary field. Double-click on the field name and enter the following in the next dialog box:
Reference table:
ZEMPLOYxx
Reference field: *Currency*

14) Define the key fields for table ZEMPLOYxx. Fields ***Client, Carrier*** and ***Personnel number*** uniquely identify an entry. They must therefore be marked as key fields. You can do this by marking the *Key* column following the field name. The key fields must be at the beginning of the field list in this order.

15) Activate the table. The maintenance screen for the technical settings appears automatically:

Since the contents of table ZEMPLOYxx do not change frequently, you must choose data class APPL0 (master data). The expected number of records in table ZEMPLOYxx is 60,000, so you must choose size category 2. The table should not be either buffered or logged.

ZEMPLOYxx	
Data class	APPL0 (master data)
Size category	*2*
Buffering	Not allowed
Logging	No logging

Save the technical settings. Go back to the maintenance screen of the table (F3 or ←). The table is activated.

2-1-2 To create table ZDEPMENTxx:

1) to 12) see Solution 2-1-1

13) Define the key fields for table ZDEPMENTxx. Fields ***Client, Carrier*** and ***Department code*** uniquely identify an entry. They must therefore be marked as key fields. You can do this by marking the *Key* column following the field name.

14) Activate your table and define the technical settings:

Since the contents of table ZDEPMENTxx do not change frequently, you must choose data class APPL0 (master data). The expected number of records in table ZDEPMENTxx is defined to be at most 90 entries, so you must choose size category 0. The table should not be either buffered or logged.

ZDEPMENTxx	
Data class	APPL0 (master data)
Size category	*0*
Buffering	Not allowed
Logging	No logging

2-2 To document fields *Personnel number* and *Department code*:

1) Double-click to go to the data element in the data element definition. With *Display <-> Change* switch to change mode. Press *Documentation* or choose *Goto → Documentation*.

2) Enter a text for the fields and save your entries.

2-3 Create structure ZCHANGExx as follows:

1) In the initial screen of the ABAP Dictionary, mark Data type and enter ZCHANGExx in the corresponding field. Choose Create.

2) Mark *Structure* in the next dialog box.

3) Enter the field names in column *Component* and the corresponding data elements in column *Component type*. Create one field for the personnel number and another one for the date of change. Use data element S_CHDATE for the second field. Create your own data element for the first field, as in Exercise 2-1. Use the domain you created for the personnel number in table ZEMPLOYxx.

4) Activate structure ZCHANGExx.

Now insert ZCHANGExx as an include in tables ZEMPLOYxx and ZDEPMENTxx as follows:

1) Go to the maintenance screen for table ZEMPLOYxx.

2) Choose *New rows* and position the cursor on the first new field.

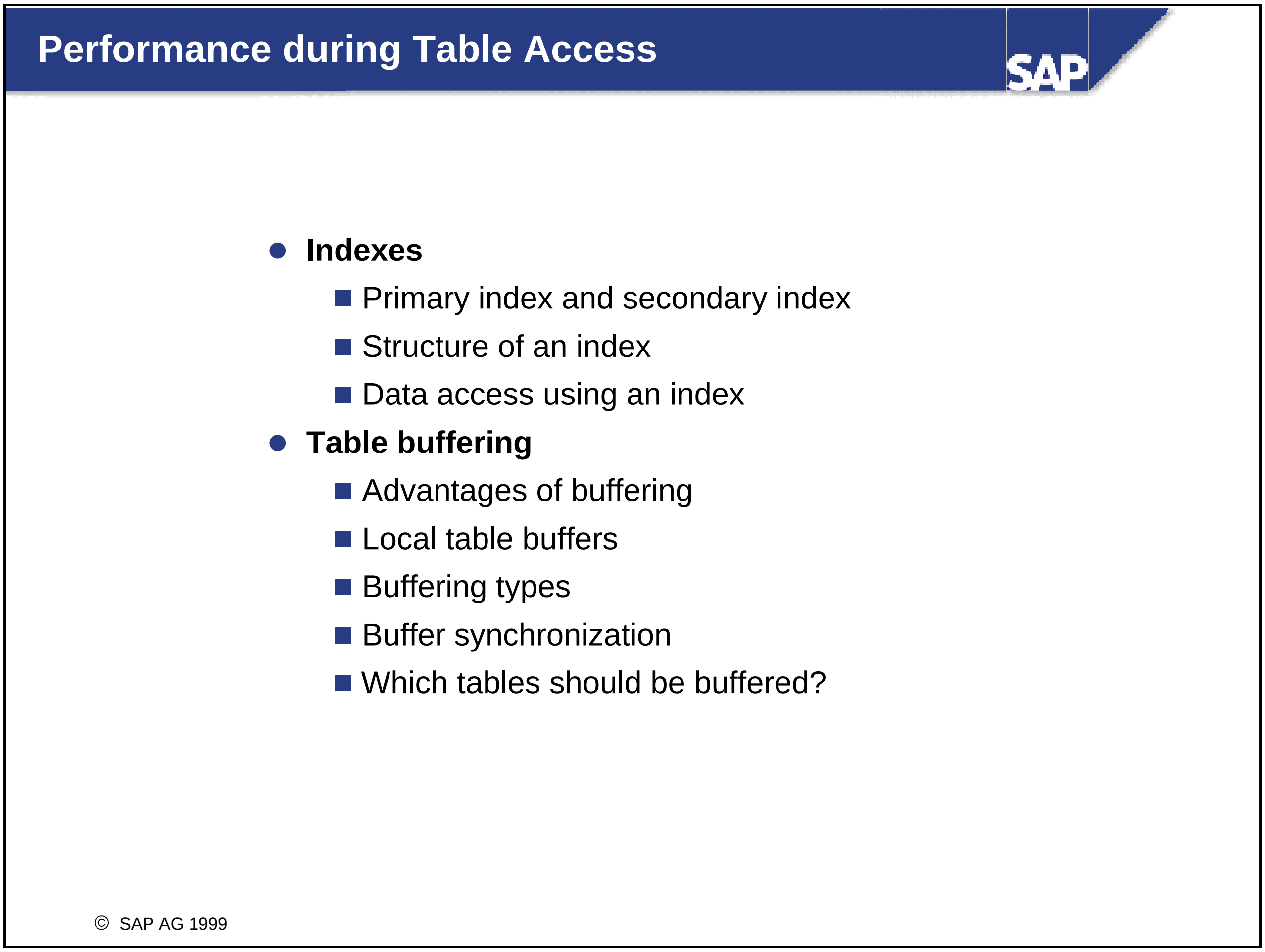
3) Choose *Edit* → *Include* →*Insert.*

4) In the next dialog box, enter the name ZCHANGExx and choose Continue.

5) Activate the table.

6) You can display the actions that were performed in the database with *Utilities* ? *Activation log*.

7) Start with step 1) to insert substructure ZCHANGExx in table ZDEPMENTxx.



Performance during Table Access

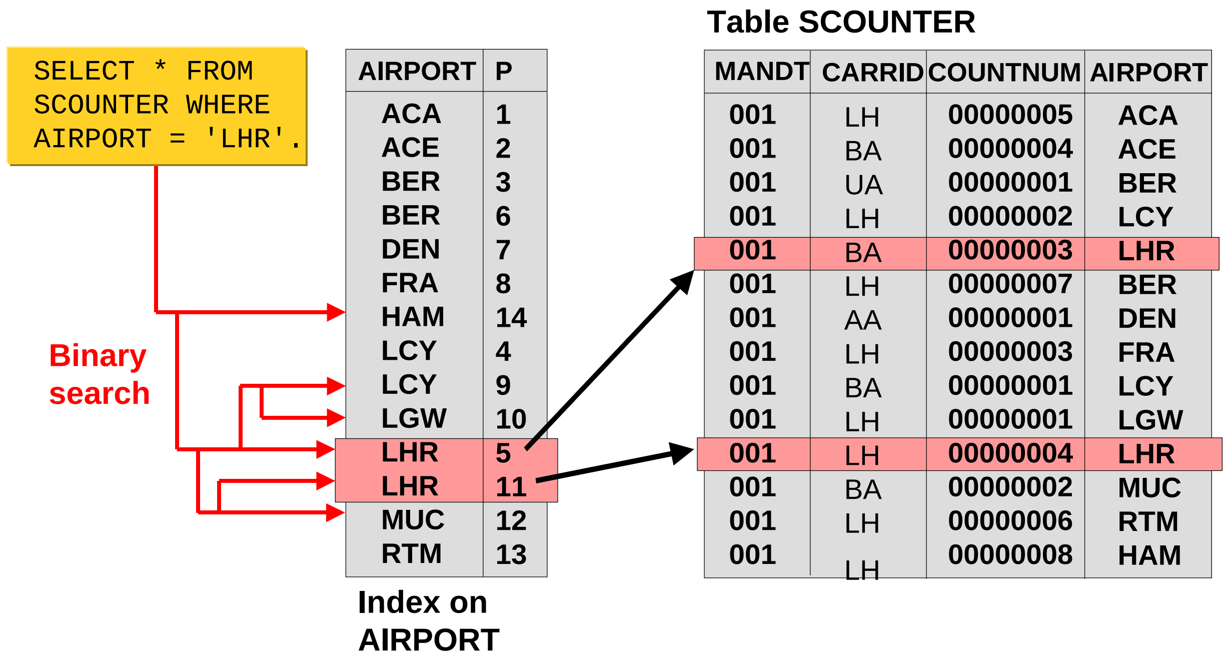
- **Indexes**
  - Primary index and secondary index
  - Structure of an index
  - Data access using an index
- **Table buffering**
  - Advantages of buffering
  - Local table buffers
  - Buffering types
  - Buffer synchronization
  - Which tables should be buffered?

Course Objectives

At the end of this unit you will be able to:

- Judge when table accesses can be speeded up by using indexes
- Create indexes in the ABAP Dictionary
- Explain the different buffering types
- Judge when it makes sense to buffer a table and which buffering type you should choose
- Buffer a table using the technical settings

Structure of an Index



```
SELECT * FROM
SCOUNTER WHERE
AIRPORT = 'LHR'.
```

AIRPORT	P
ACA	1
ACE	2
BER	3
BER	6
DEN	7
FRA	8
HAM	14
LCY	4
LCY	9
LGW	10
LHR	5
LHR	11
MUC	12
RTM	13

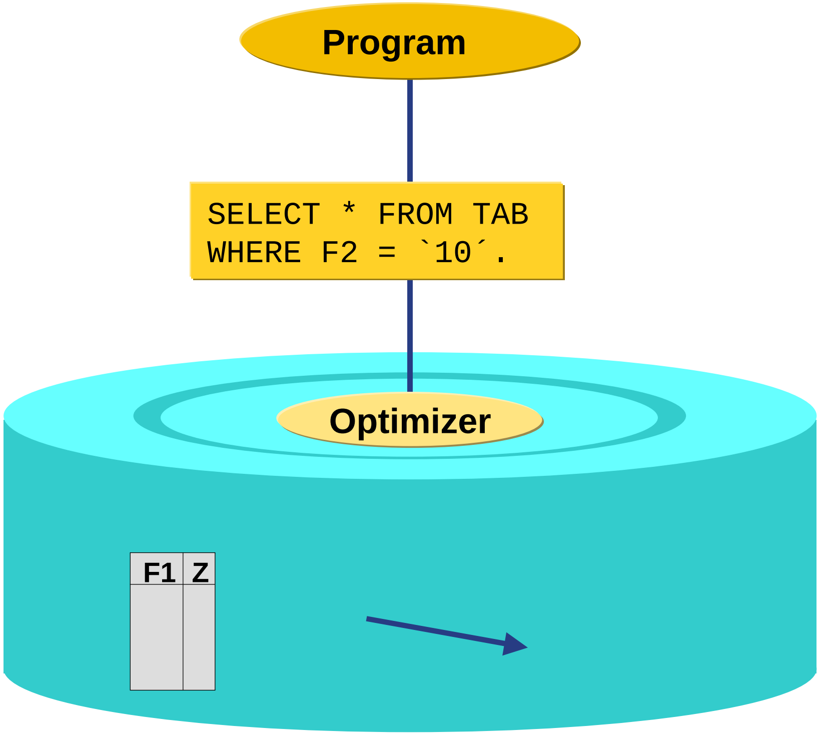
Index on AIRPORT

Table SCOUNTER

MANDT	CARRID	COUNTNUM	AIRPORT
001	LH	00000005	ACA
001	BA	00000004	ACE
001	UA	00000001	BER
001	LH	00000002	LCY
001	BA	00000003	LHR
001	LH	00000007	BER
001	AA	00000001	DEN
001	LH	00000003	FRA
001	BA	00000001	LCY
001	LH	00000001	LGW
001	LH	00000004	LHR
001	BA	00000002	MUC
001	LH	00000006	RTM
001	LH	00000008	HAM

- An index can be used to speed up the selection of data records from a table.
- An index can be considered to be a copy of a database table reduced to certain fields. The data is stored in sorted form in this copy. This sorting permits fast access to the records of the table (for example using a binary search). Not all of the fields of the table are contained in the index. The index also contains a pointer from the index entry to the corresponding table entry to permit all the field contents to be read.
- When creating indexes, please note that:
  - An index can only be used up to the last specified field in the selection! The fields which are specified in the WHERE clause for a large number of selections should be in the first position.
  - Only those fields whose values significantly restrict the amount of data are meaningful in an index.
  - When you change a data record of a table, you must adjust the index sorting. Tables whose contents are frequently changed therefore should not have too many indexes.
  - Make sure that the indexes on a table are as disjunct as possible.

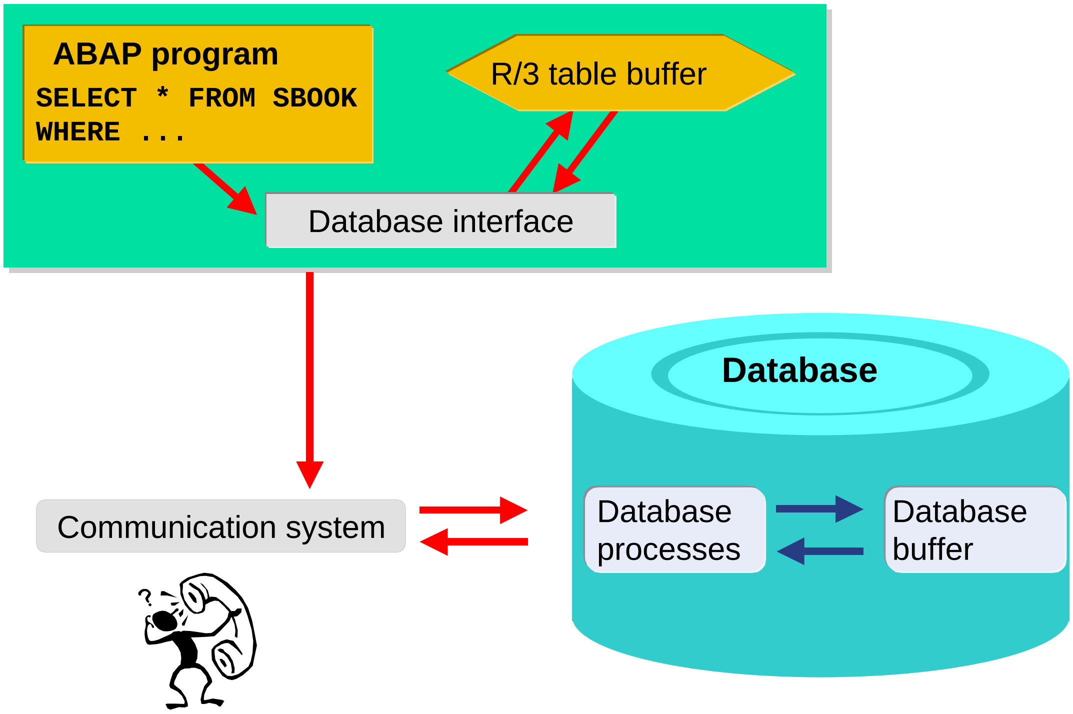
Access with Indexes



```
SELECT * FROM TAB
WHERE F2 = `10´.
```

- The database optimizer decides which index on the table should be used by the database to access data records.
- You must distinguish between the primary index and secondary indexes of a table. The primary index contains the key fields of the table. The **primary index** is automatically created in the database when the table is activated. If a large table is frequently accessed such that it is not possible to apply primary index sorting, you should create **secondary indexes** for the table.
- The indexes on a table have a three-character index ID. '0' is reserved for the primary index. Customers can create their own indexes on SAP tables; their IDs must begin with Y or Z.
- If the index fields have key function, i.e. they already uniquely identify each record of the table, an index can be called a *unique index*. This ensures that there are no duplicate index fields in the database.
- When you define a secondary index in the ABAP Dictionary, you can specify whether it should be created on the database when it is activated. Some indexes only result in a gain in performance for certain database systems. You can therefore specify a list of database systems when you define an index. The index is then only created on the specified database systems when activated.

Data Access using the Buffer



ABAP program

```
SELECT * FROM SBOOK
WHERE ...
```

R/3 table buffer

Database interface

Communication system

Database

Database processes

Database buffer

- Table buffering increases the performance when the records of the table are read.
- The records of a buffered table are read directly from the local buffer of the application server on which the accessing transaction is running when the table is accessed. This eliminates time-consuming database accesses. The access improves by a factor of 10 to 100. The increase in speed depends on the structure of the table and on the exact system configuration. Buffering therefore can greatly increase the system performance.
- If the storage requirements in the buffer increase due to further data, the data that has not been accessed for the longest time is displaced. This displacement takes place asynchronously at certain times which are defined dynamically based on the buffer accesses. Data is only displaced if the free space in the buffer is less than a predefined value or the quality of the access is not satisfactory at this time.
- Entering $TAB in the command field resets the table buffers on the corresponding application server. Only use this command if there are inconsistencies in the buffer. In large systems, it can take several hours to fill the buffers. The performance is considerably reduced during this time.

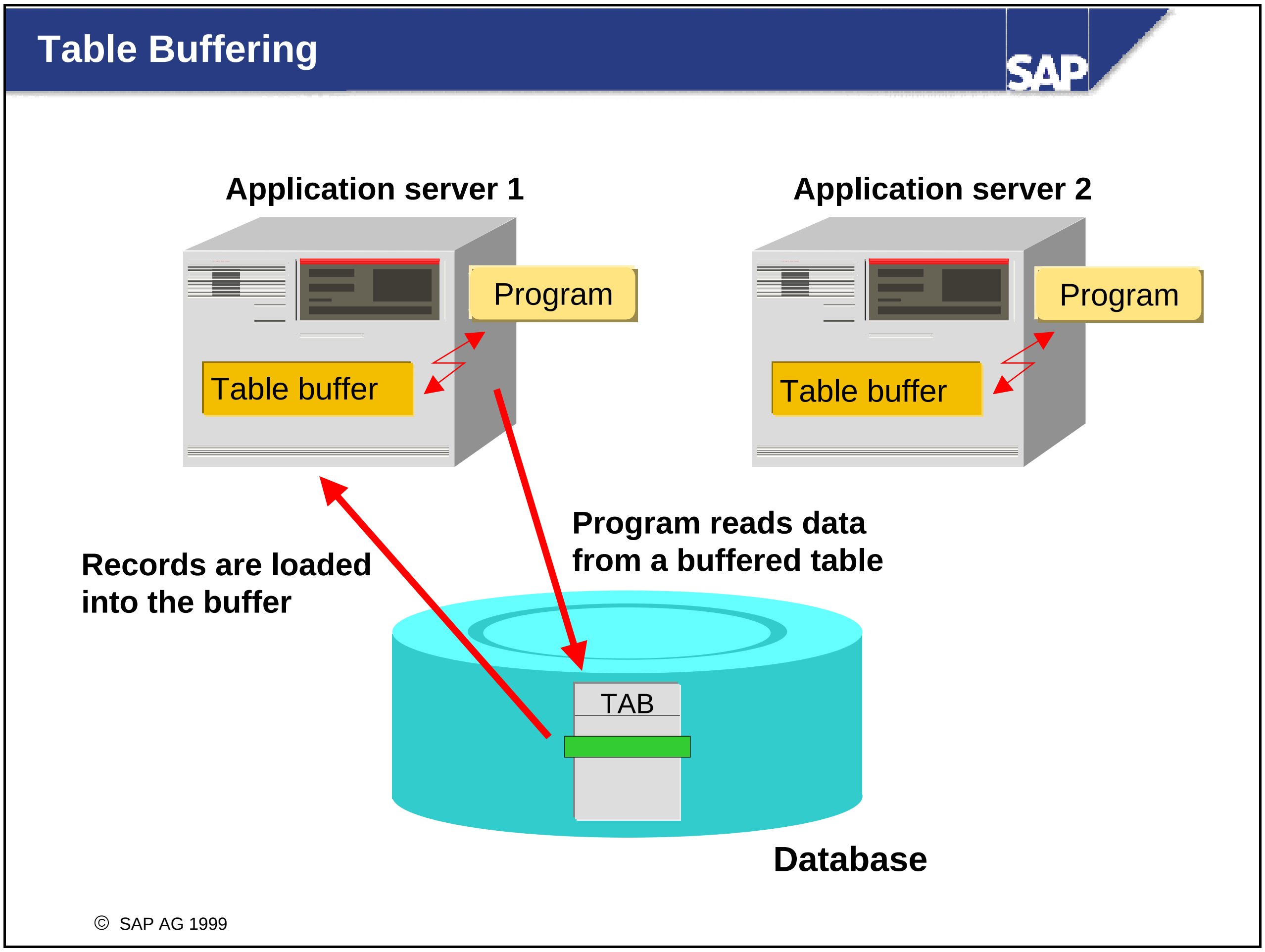


Table Buffering

Application server 1

Application server 2

Program

Program

Table buffer

Table buffer

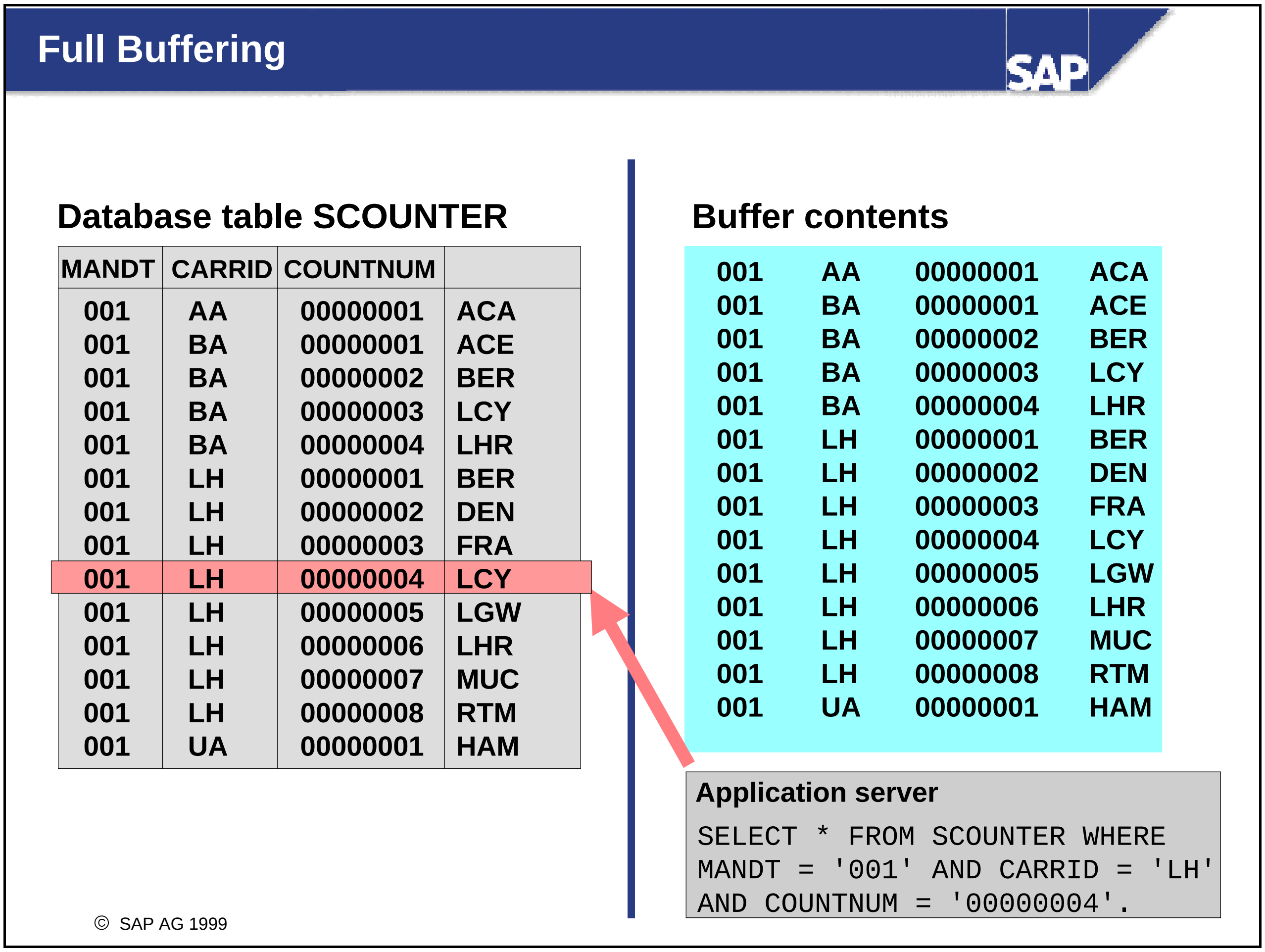
Records are loaded into the buffer

Program reads data from a buffered table

TAB

Database

- The R/3 System manages and synchronizes the buffers on the individual application servers. If an application program accesses data of a table, the database interfaces determines whether this data lies in the buffer of the application server. If this is the case, the data is read directly from the buffer. If the data is not in the buffer of the application server, it is read from the database and loaded into the buffer. The buffer can therefore satisfy the next access to this data.
- The buffering type determines which records of the table are loaded into the buffer of the application server when a record of the table is accessed. There are three different buffering types.
- With **full buffering**, all the table records are loaded into the buffer when one record of the table is accessed.
- With **generic buffering**, all the records whose left-justified part of the key is the same are loaded into the buffer when a table record is accessed.
- With **single-record buffering**, only the record that was accessed is loaded into the buffer.



Full Buffering

Database table SCOUNTER

MANDT	CARRID	COUNTNUM	
001	AA	00000001	ACA
001	BA	00000001	ACE
001	BA	00000002	BER
001	BA	00000003	LCY
001	BA	00000004	LHR
001	LH	00000001	BER
001	LH	00000002	DEN
001	LH	00000003	FRA
001	LH	00000004	LCY
001	LH	00000005	LGW
001	LH	00000006	LHR
001	LH	00000007	MUC
001	LH	00000008	RTM
001	UA	00000001	HAM

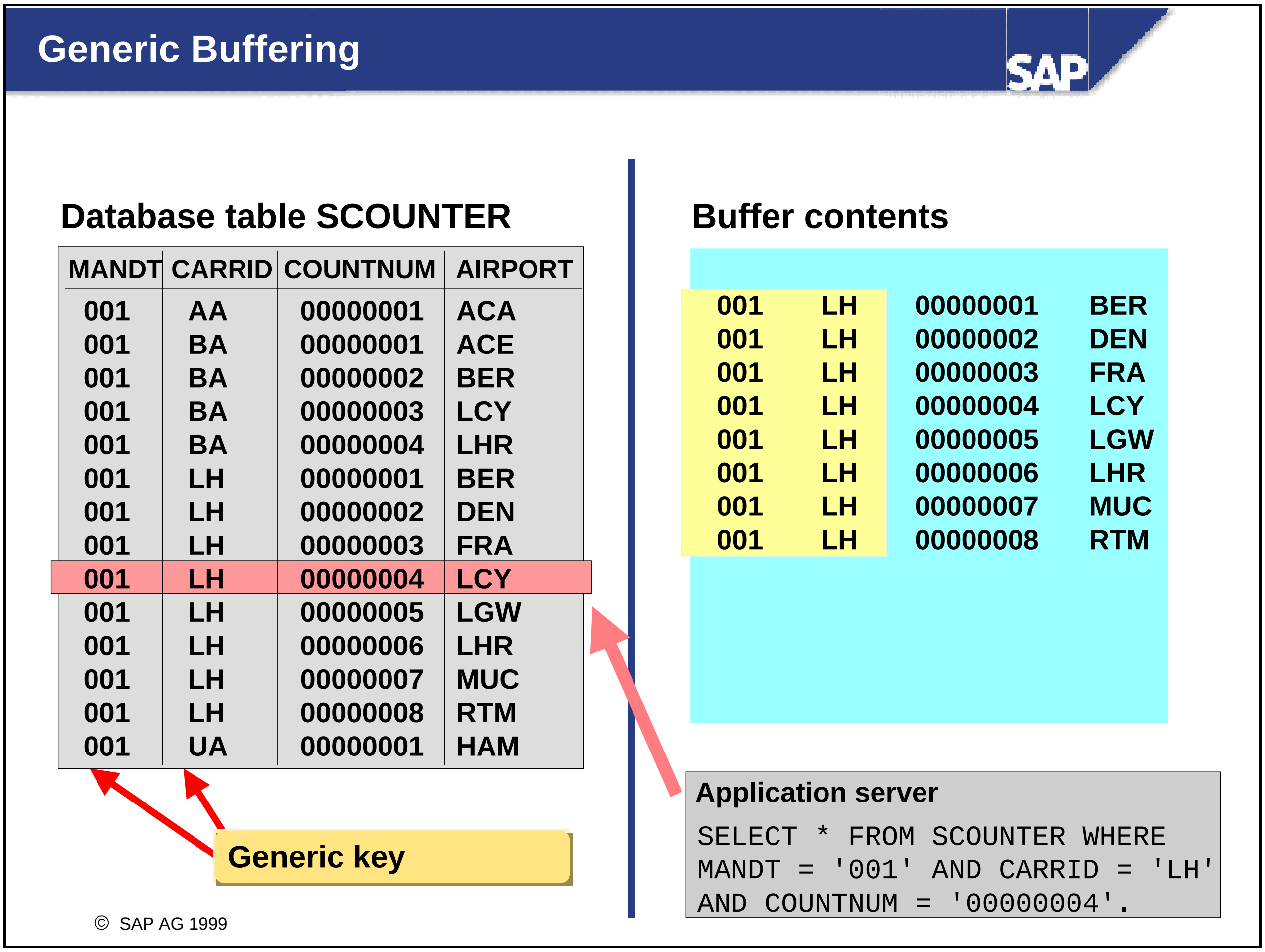
Buffer contents

001	AA	00000001	ACA
001	BA	00000001	ACE
001	BA	00000002	BER
001	BA	00000003	LCY
001	BA	00000004	LHR
001	LH	00000001	BER
001	LH	00000002	DEN
001	LH	00000003	FRA
001	LH	00000004	LCY
001	LH	00000005	LGW
001	LH	00000006	LHR
001	LH	00000007	MUC
001	LH	00000008	RTM
001	UA	00000001	HAM

Application server

```
SELECT * FROM SCOUNTER WHERE
MANDT = '001' AND CARRID = 'LH'
AND COUNTNUM = '00000004'.
```

- With full buffering, the table is either completely or not at all in the buffer. When a record of the table is accessed, all the records of the table are loaded into the buffer.
- When you decide whether a table should be fully buffered, you must take the table size, the number of read accesses and the number of write accesses into consideration. The smaller the table is, the more frequently it is read and the less frequently it is written, the better it is to fully buffer the table.
- Full buffering is also advisable for tables having frequent accesses to records that do not exist. Since all the records of the table reside in the buffer, it is already clear in the buffer whether or not a record exists.
- The data records are stored in the buffer sorted by table key. When you access the data with SELECT, only fields up to the last specified key field can be used for the access. The left-justified part of the key should therefore be as large as possible for such accesses. For example, if the first key field is not defined, the entire table is scanned in the buffer. Under these circumstances, a direct access to the database could be more efficient if there is a suitable secondary index there.



Generic Buffering

Database table SCOUNTER

MANDT	CARRID	COUNTNUM	AIRPORT
001	AA	00000001	ACA
001	BA	00000001	ACE
001	BA	00000002	BER
001	BA	00000003	LCY
001	BA	00000004	LHR
001	LH	00000001	BER
001	LH	00000002	DEN
001	LH	00000003	FRA
001	LH	00000004	LCY
001	LH	00000005	LGW
001	LH	00000006	LHR
001	LH	00000007	MUC
001	LH	00000008	RTM
001	UA	00000001	HAM

Generic key

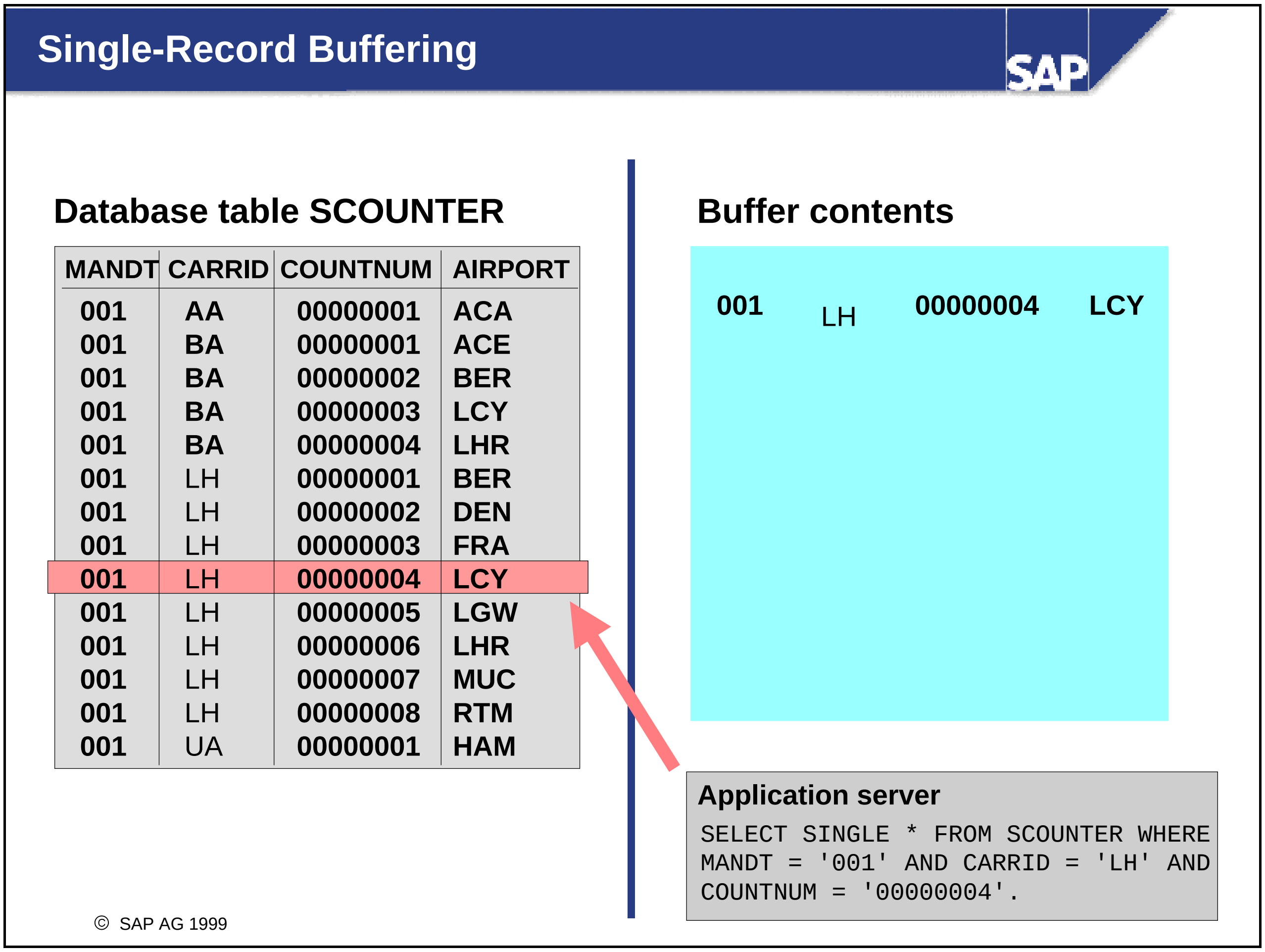
Buffer contents

001	LH	00000001	BER
001	LH	00000002	DEN
001	LH	00000003	FRA
001	LH	00000004	LCY
001	LH	00000005	LGW
001	LH	00000006	LHR
001	LH	00000007	MUC
001	LH	00000008	RTM

Application server

```
SELECT * FROM SCOUNTER WHERE
MANDT = '001' AND CARRID = 'LH'
AND COUNTNUM = '00000004'.
```

- With generic buffering, all the records whose generic key fields agree with this record are loaded into the buffer when one record of the table is accessed. The **generic key** is a left-justified part of the primary key of the table that must be defined when the buffering type is selected. The generic key should be selected so that the generic areas are not too small, which would result in too many generic areas. If there are only a few records for each generic area, full buffering is usually preferable for the table. If you choose too large a generic key, too much data will be invalidated if there are changes to table entries, which would have a negative effect on the performance.
- A table should be generically buffered if only certain generic areas of the table are usually needed for processing.
- Client-dependent, fully buffered tables are automatically generically buffered. The client field is the generic key. It is assumed that not all of the clients are being processed at the same time on one application server. Language-dependent tables are a further example of generic buffering. The generic key includes all the key fields up to and including the language field.
- The generic areas are managed in the buffer as independent objects. The generic areas are managed analogously to fully buffered tables. You should therefore also read the information about full buffering.



Single-Record Buffering

Database table SCOUNTER

MANDT	CARRID	COUNTNUM	AIRPORT
001	AA	00000001	ACA
001	BA	00000001	ACE
001	BA	00000002	BER
001	BA	00000003	LCY
001	BA	00000004	LHR
001	LH	00000001	BER
001	LH	00000002	DEN
001	LH	00000003	FRA
001	LH	00000004	LCY
001	LH	00000005	LGW
001	LH	00000006	LHR
001	LH	00000007	MUC
001	LH	00000008	RTM
001	UA	00000001	HAM

Buffer contents

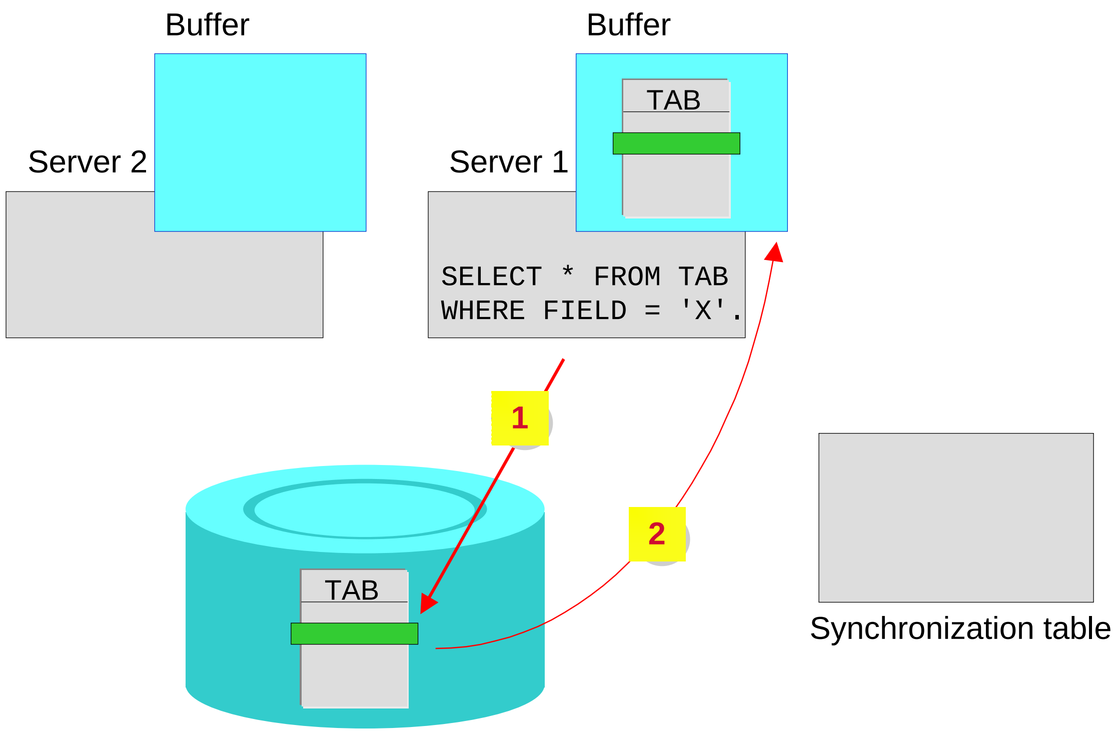
001 LH 00000004 LCY

Application server

```
SELECT SINGLE * FROM SCOUNTER WHERE
MANDT = '001' AND CARRID = 'LH' AND
COUNTNUM = '00000004'.
```

- Only those records that are actually accessed are loaded into the buffer. Single-record buffering saves storage space in the buffer compared to generic and full buffering. The overhead for buffer administration, however, is higher than for generic or full buffering. Considerably more database accesses are necessary to load the records than for the other buffering types.
- Single-record buffering is recommended particularly for large tables in which only a few records are accessed repeatedly with SELECT SINGLE. All the accesses to the table that do not use SELECT SINGLE bypass the buffer and directly access the database.
- If you access a record that was not yet buffered using SELECT SINGLE, there is a database access to load the record. If the table does not contain a record with the specified key, this record is recorded in the buffer as non-existent. This prevents a further database access if you make another access with the same key.
- You only need one database access to load a table with full buffering, but you need several database accesses with single-record buffering. Full buffering is therefore generally preferable for small tables that are frequently accessed.

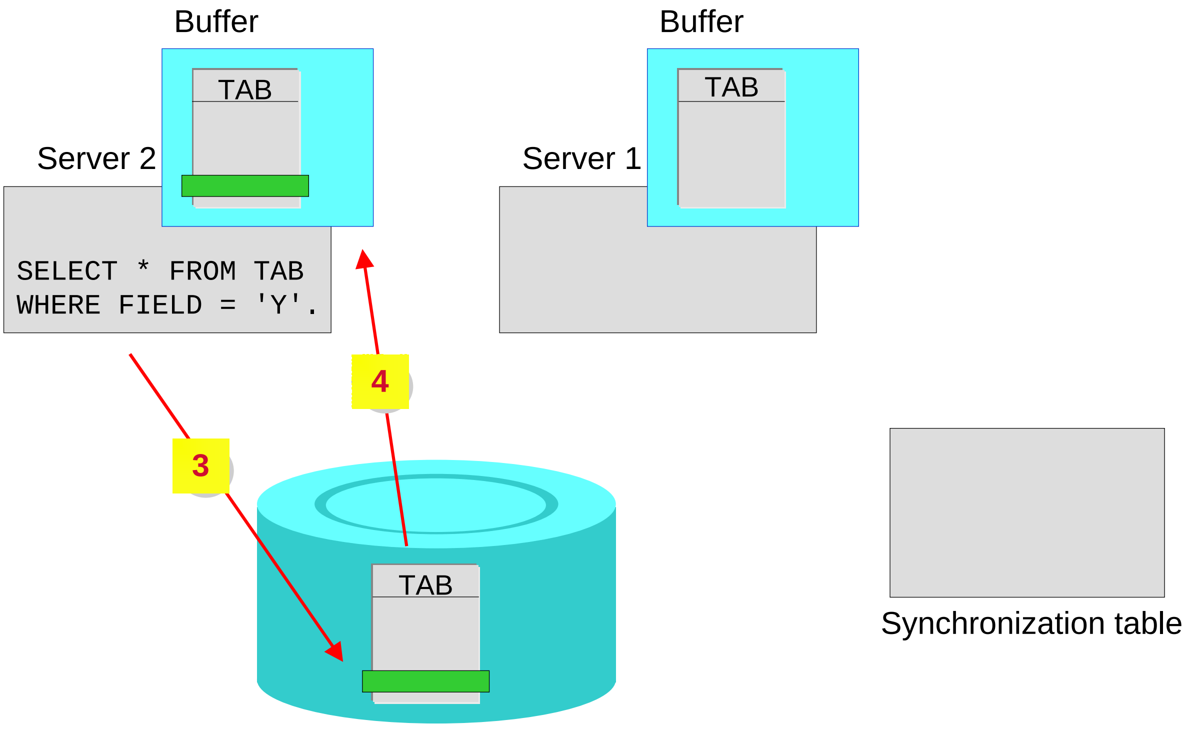
Buffer Synchronization 1



```
SELECT * FROM TAB
WHERE FIELD = 'X'.
```

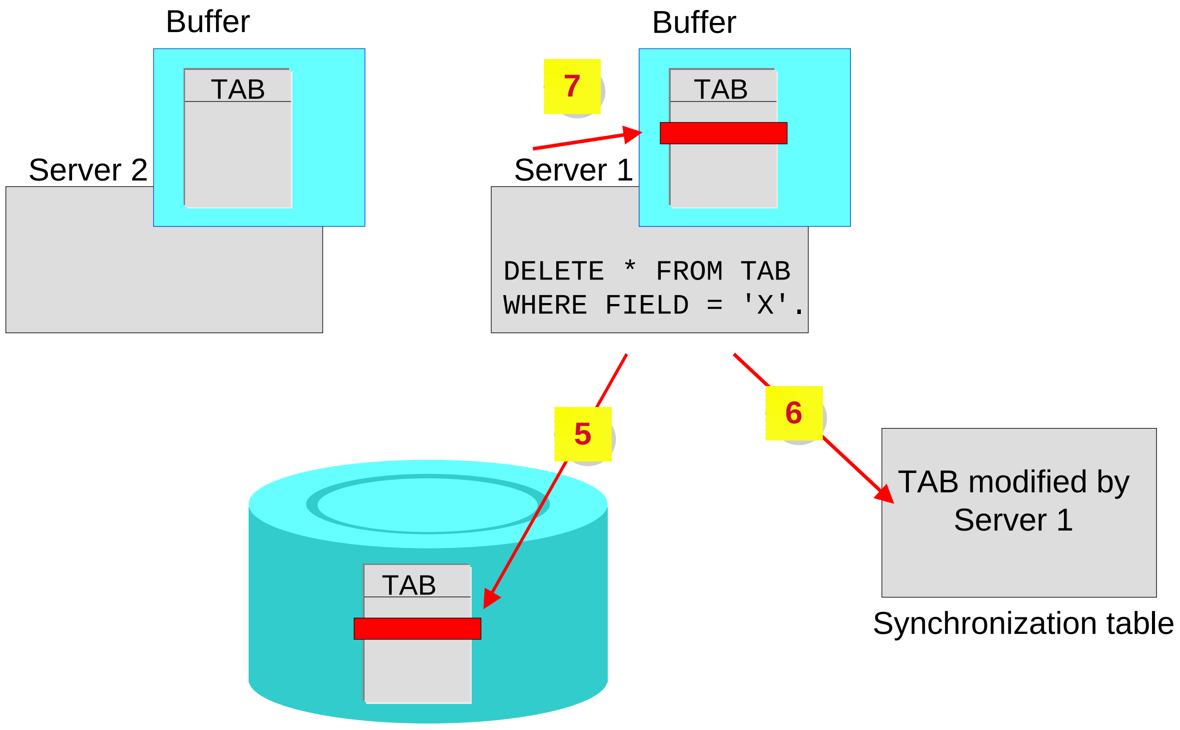
- Since the buffers reside locally on the application servers, they must be synchronized after data has been modified in a buffered table. Synchronization takes place at fixed time intervals that can be set in the system profile. The corresponding parameter is *rdisp/bufreftime* and defines the length of the interval in seconds. The value must lie between 60 and 3600. A value between 60 and 240 is recommended.
- The following example shows how the local buffers of the system are synchronized. A system with two application servers is assumed.
- **Starting situation:** Neither server has yet accessed records of the table TAB to be fully buffered. The table therefore does not yet reside in the local buffers of the two servers.
- **Timepoint 1:** Server 1 reads records from table TAB on the database.
- **Timepoint 2:** Table TAB is fully loaded into the local buffer of server 1. Accesses from server 1 to the data of table TAB now use the local buffer of this server.

Buffer Synchronization 2



- **Timepoint 3:** Server 2 accesses records of the table. Since the table does not yet reside in the local buffer of server 2, the records are read directly from the database.
- **Timepoint 4:** Table TAB is loaded into the local buffer of server 2. Server 2 therefore also uses its local buffer to access data of TAB the next time it reads.

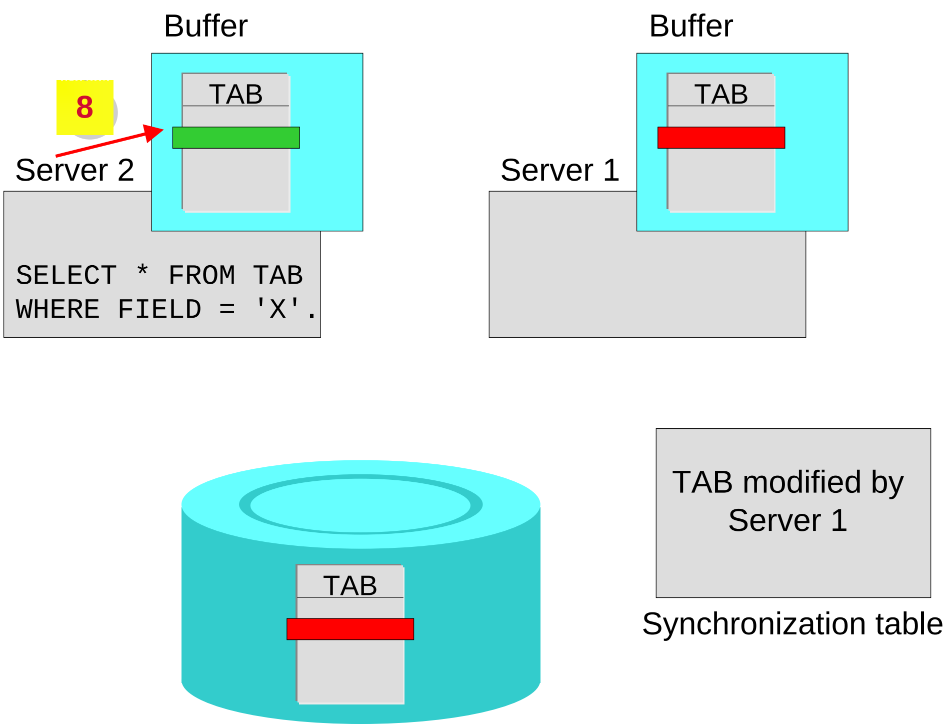
Buffer Synchronization 3



```
DELETE * FROM TAB
WHERE FIELD = 'X'.
```

- **Timepoint 5:** Server 1 deletes records from table TAB and updates the database.
- **Timepoint 6:** Server 1 writes an entry in the synchronization table.
- **Timepoint 7:** Server 1 updates its local buffer.

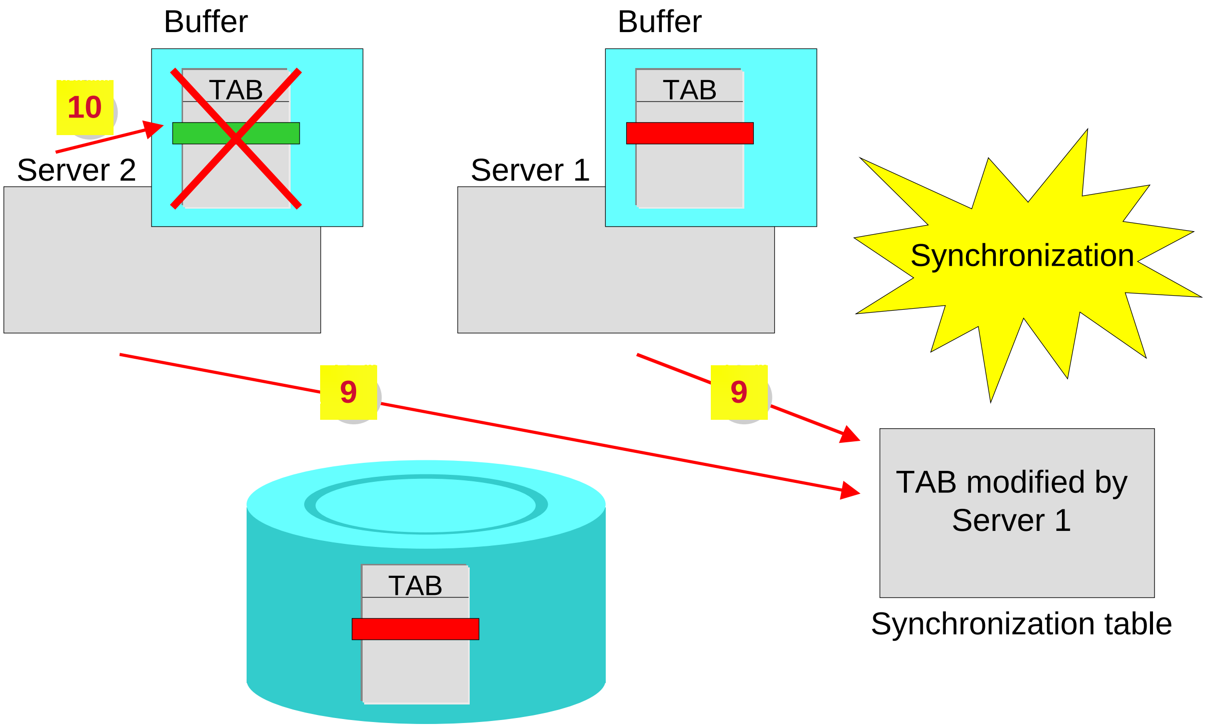
Buffer Synchronization 4



© SAP AG 1999

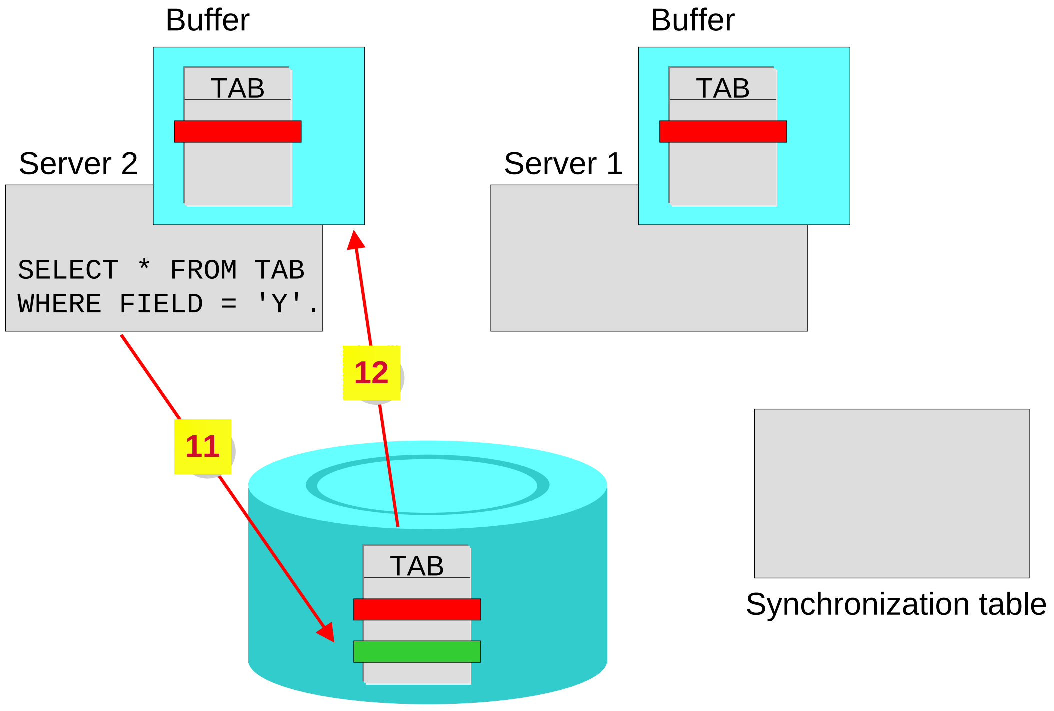
- **Timepoint 8:** Server 2 accesses the deleted data records. Since table TAB resides in its local buffer, the access uses this local buffer.
- Server 2 therefore finds the records although they no longer exist in the database table.
- If the same access were made from an application program to Server 1, this program would recognize that the records no longer exist. At this time the behavior of an application program therefore depends on the server on which it is running.

Buffer Synchronization 5



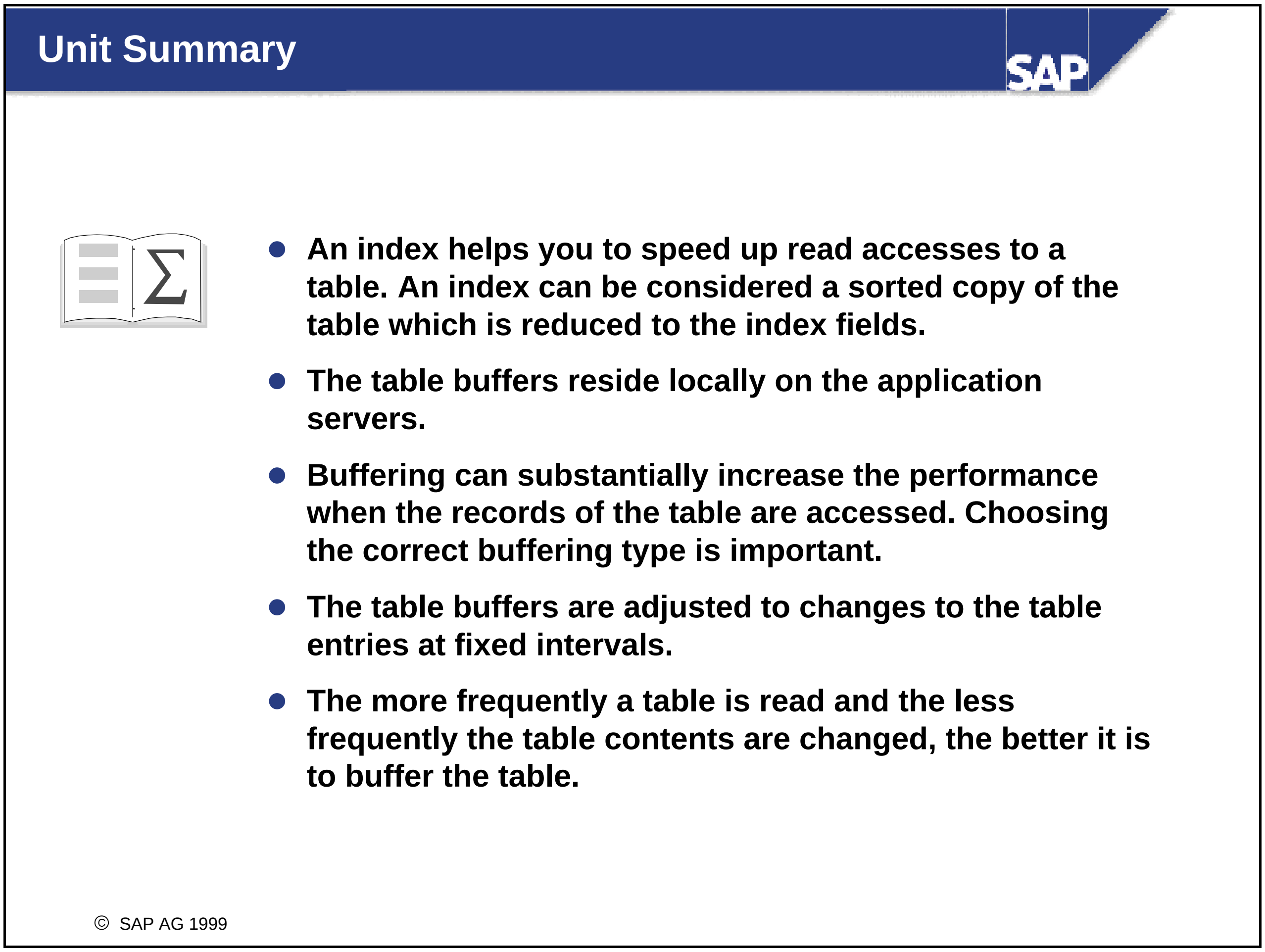
- **Timepoint 9:** The moment of synchronization has arrived. Both servers look in the synchronization table to see if another server has modified one of the tables in its local buffer in the meantime.
- **Timepoint 10:** Server 2 finds that table TAB has been modified by Server 1 in the meantime. Server 2 therefore invalidates the table in its local buffer. The next access from Server 2 to data of table TAB therefore uses the database. Server 1 does not have to invalidate the table in its buffer since it itself is the only one to modify table TAB. Server 1 therefore uses its local buffer again the next time to access records of table TAB.

Buffer Synchronization 6



```
SELECT * FROM TAB
WHERE FIELD = 'Y'.
```

- **Timepoint 11:** Server 2 again accesses records of table TAB. Since TAB is invalidated in the local buffer of Server 2, the access uses the database.
- **Timepoint 12:** The table is again loaded into the local buffer of Server 2. The information about table TAB is now consistent again in both servers and the database.
- **Advantages and disadvantages of this method of buffer synchronization**:
  - **Advantage**: The load on the network is kept to a minimum. If the buffers were to be synchronized immediately after each modification, each server would have to inform all other servers about each modification to a buffered table via the network. This would have a negative effect on the performance.
  - **Disadvantage**: The local buffers of the application servers can contain obsolete data between the moments of synchronization.
- This means that:
  - Only those tables which are written very infrequently (read mostly) or for which such temporary inconsistencies are of no importance may be buffered.
  - Tables whose entries change frequently should not be buffered. Otherwise there would be a constant invalidation and reload, which would have a negative effect on the performance.



Unit Summary

- An index helps you to speed up read accesses to a table. An index can be considered a sorted copy of the table which is reduced to the index fields.
- The table buffers reside locally on the application servers.
- Buffering can substantially increase the performance when the records of the table are accessed. Choosing the correct buffering type is important.
- The table buffers are adjusted to changes to the table entries at fixed intervals.
- The more frequently a table is read and the less frequently the table contents are changed, the better it is to buffer the table.

Exercises: Performance during Table Access

Unit: Performance during Table Access

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

- Create indexes
- Maintain the buffering attributes of a table

In their daily work, airline employees need fast access to the data in the employee administration tables. In this exercise, access to the data in these tables should be speeded up.

3-1 The combination of first and last names are often used to access the personnel data of an employee. The last name is more often known (i.e. specified in the access) than the first name.

Create an index that supports this access. Make sure that the index is created in the database.

3-2 To set up a flight crew, you have to assign employees (pilots and stewards) to flights. Create a table in which the employees involved and their functions can be entered for each flight.

A table with the corresponding structure already exists in the system. Copy this table SFLCREW to table ZFLCREWxx. Replace the existing data element for the employee number with your own data element.

Do not forget to activate table ZFLCREWxx.

3-3 Reconsider the settings you made for buffering tables ZDEPMENTxx and ZFLCREWxx. Keep the following information for using these tables in mind:

The carriers have between 10 and 30 departments. Only a few carriers (maximum 3) are administered in the tables. The data about the crews of completed flights are rolled out to an archive file every three months. Table ZFLCREWxx therefore has relatively few entries (at most 5,000 per carrier).

Tables ZDEPMENTxx and ZFLCREWxx are accessed very frequently. Data records are read repeatedly from these tables.

Administrative employees of only one airline work on one application server. The data about the flight crew is only of interest within the airline. Administrative employees of an airline, however, often have to access departmental data of other airlines, since the airlines share some services.

Start Program BC430_CHECK in Transaction SE38. This program checks whether your solutions are correct and fills the new table ZFLCREWxx with sample data needed for later exercises.

If you do the supplementary exercise, only start the program after completing the supplementary exercise.

3-4 **Supplementary Exercise:** Using an index on the areas might result in a gain in performance when accessing the employee data, for example if all pilots are frequently selected.

In performance measurements on different database systems, however, it was found there is only a gain in performance for the ADABAS and SQL Server database systems. Create an index and make sure that it is only created on the ADABAS and SQL Server database systems.

Solutions: Performance during Table Access

Unit: Performance during Table Access

3-1 The personnel data of the employees is managed in Table ZEMPLOYxx. Create an index for this table. It has to contain fields *Client*, *Lastname* and *Firstname*. Since the last name is usually specified and is much more selective than the first name, it must be in its index. The order of the fields is then *Client*, *Lastname* and then *Firstname*.

To create the index:

1) In display mode, go to the maintenance screen of table ZEMPLOYxx and choose *Indexes*.

2) In the next dialog box, confirm that you want to create an index.

3) In the following dialog box, enter a three-place index ID and choose *Continue*.

4) The maintenance screen for the index appears. Enter a *Short description*.

5) Choose *Table fields*. A list of all the fields in the table appears. Mark fields *Client*, *Lastname* and *Firstname* and choose *Copy*.

6) The fields are copied from the dialog box to the index in that order. If field *Firstname* is before field *Lastname*, you have to change the order of the fields. Do this by placing the cursor on the line with field *Firstname* and choosing *Cut*. Now place the cursor on the first free line after field *Lastname* and choose *Paste*.

7) The index is certainly not a unique index since there can be employees with the same first and last names. There is no reason to create the index only in certain database systems. You should therefore leave the standard settings *Non-unique index* and *Index in all database systems*.

8) Activate the index. The index is automatically created in the database.

3-2 To copy table SFLCREW:

1) In the initial screen of the ABAP Dictionary, enter SFLCREW in field *Database table*. Choose *Copy*.

2) In the next dialog box, enter the name ZFLCREWxx in field *to table* and choose *Continue*.

3) In change mode, go to the table maintenance screen and replace data element SEMP_NUM with the data element you created for the employee number.

4) Activate the table.

3-3 You can maintain the buffering settings for the specified tables in their technical settings. To do so, go to the maintenance screen for the table in display mode and choose *Technical settings*. The desired maintenance screen appears and you can switch to change mode here.

Since the contents of table ZDEPMENTxx are rarely changed but frequently read, it is not advisable to buffer the table. Mark *Buffering switched on*. Since there are no restrictions on the access and the table is small, you should select full buffering. Mark *Fully buffered*.

Activate the technical settings of table ZDEPMENTxx.

The data of table ZFLCREWxx are often read repeatedly. Accesses that change the contents are rare. You should therefore buffer the table. Mark *Buffering switched on*. Usually only the data of one airline is needed on an application server. You should therefore buffer the table generically with the generic key *Client* and *Carrier*. Mark *Generic buffering* and select 2 as the number of generic key fields.

Activate the technical settings of table ZFLCREWxx.

3-4 If you do as specified in 3.1, the system displays the index you created in a dialog box. In this dialog box choose *Create*. Include fields *Client, Carrier* and *Area* in the index. This is not a unique index either.

To create the index only in the Adabas and SQL Server database systems:

1) Mark *For selected database systems*.

2) Then press the arrow symbol in this line. Select *Selection list*. Using the F4 help, select the identifiers for the Adabas (ADA) and SQL Server (MSS) database systems in the list.

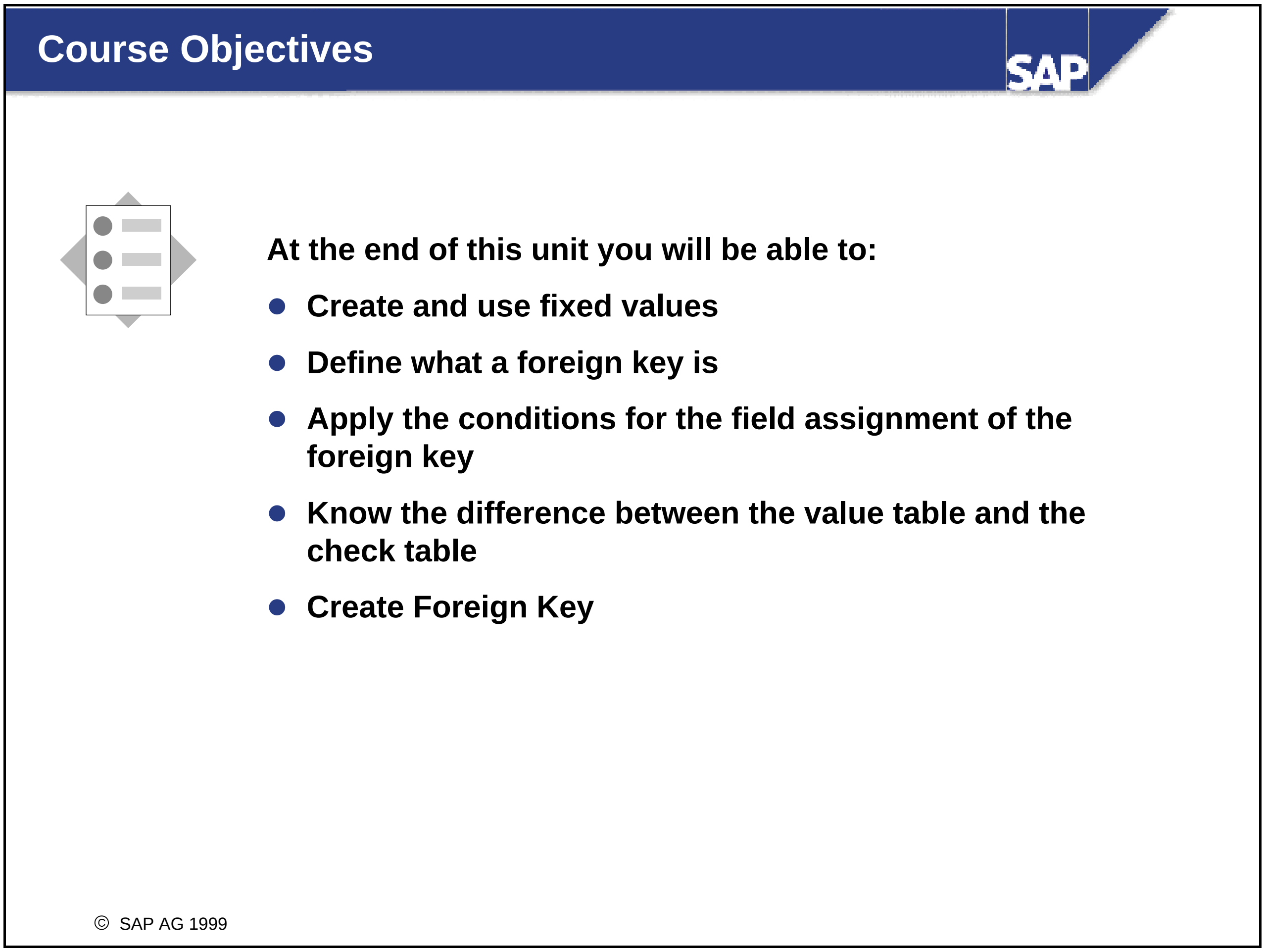
3) Choose *Continue*.

4) Activate the index.

The index is only created in the database if your training system is running on one of the selected database systems.

Consistency through Input Checks

- Fixed values
- Value table
- What is a foreign key?
- Field assignment using the check field
- Foreign key table / check table
- Semantic attributes of the foreign key
- Text table

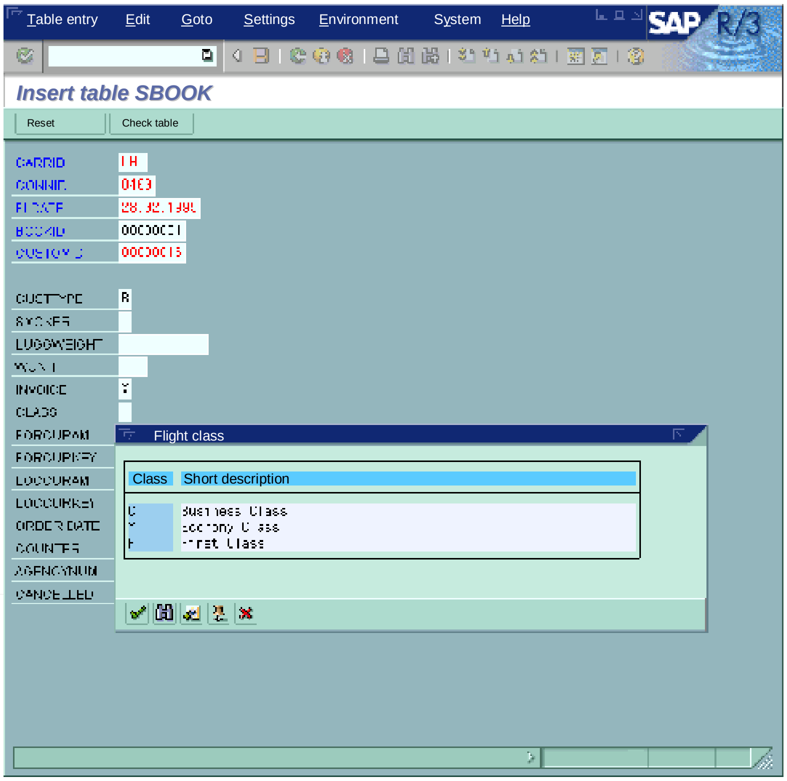


Course Objectives

At the end of this unit you will be able to:

- Create and use fixed values
- Define what a foreign key is
- Apply the conditions for the field assignment of the foreign key
- Know the difference between the value table and the check table
- Create Foreign Key

Fixed Values



Insert table SBOOK

CARRID	LH
CONNID	0400
FLDATE	28.02.1995
BOOKID	00000001
CUSTOMID	00000015
CUSTTYPE	B
SMOKER	
LUGGWEIGHT	
WUNIT	
INVOICE	X
CLASS	
FORCURAM	
FORCURKEY	
LOCCURAM	
LOCCURKEY	
ORDER DATE	
COUNTER	
AGENCYNUM	
CANCELLED	

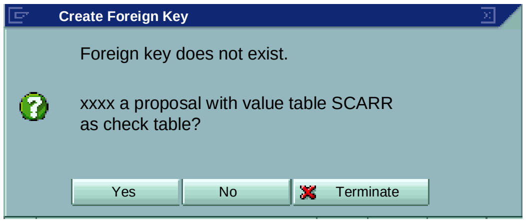
Flight class

Class	Short description
C	Business Class
Y	Economy Class
F	First Class

© SAP AG 1999

- The domain describes the value range of a field by specifying its data type and field length. If only a limited set of values is allowed, they can be defined as fixed values.
- Specifying fixed values causes the value range of the domain to be restricted by these values. Fixed values are immediately used as check values in screen entries. There is also an F4 help.
- Fixed values are only checked in screens. No check is made when data records are inserted in a table by an ABAP program.
- Fixed values can either be listed individually or defined as an interval.

Value Table



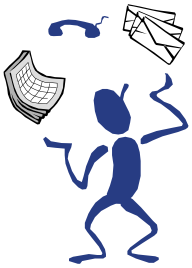
Create Foreign Key

Foreign key does not exist.

xxxx a proposal with value table SCARR as check table?

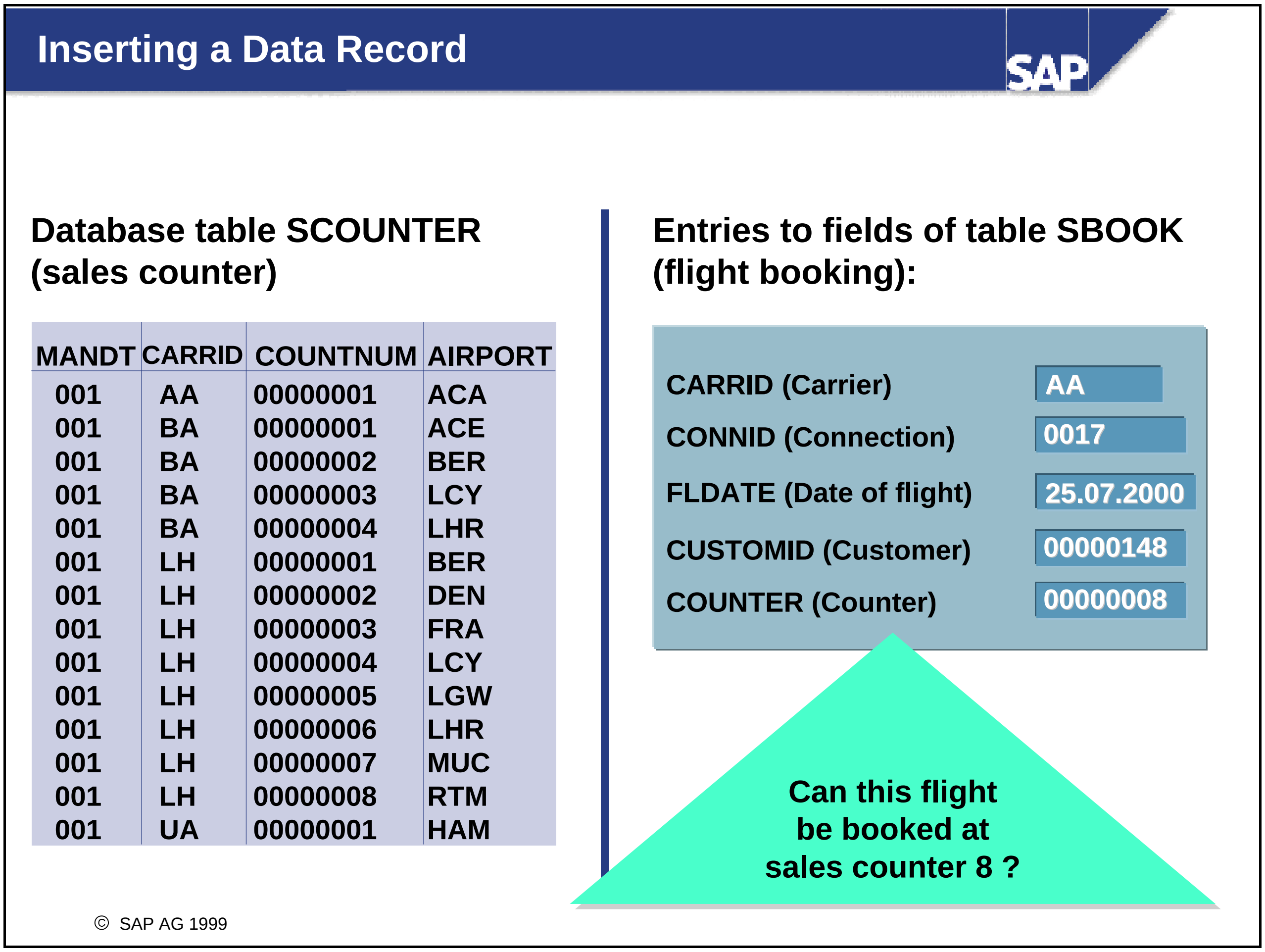
Table SCARR

MANDT	CARRID	CARRNAME	CURRCODE
401	AA	American Airlines	USD
401	BA	British Airways	GBP
401	LH	Lufthansa	DEM
410	UA	United Airlines	USD



DOMAIN S_CARR_ID
Value table SCARR

- The value range of a field can also be defined by specifying a value table in the domain.
- In contrast to fixed values, however, simply specifying a value table does not cause the input to be checked. There is no F4 help either.
- If you enter a value table, the system can make a proposal for the foreign key definition.
- A value table only becomes a check table when a foreign key is defined.
  If you refer to a domain with a value table in a field, but no foreign key was defined at field level, there is no check.



Inserting a Data Record

Database table SCOUNTER (sales counter)

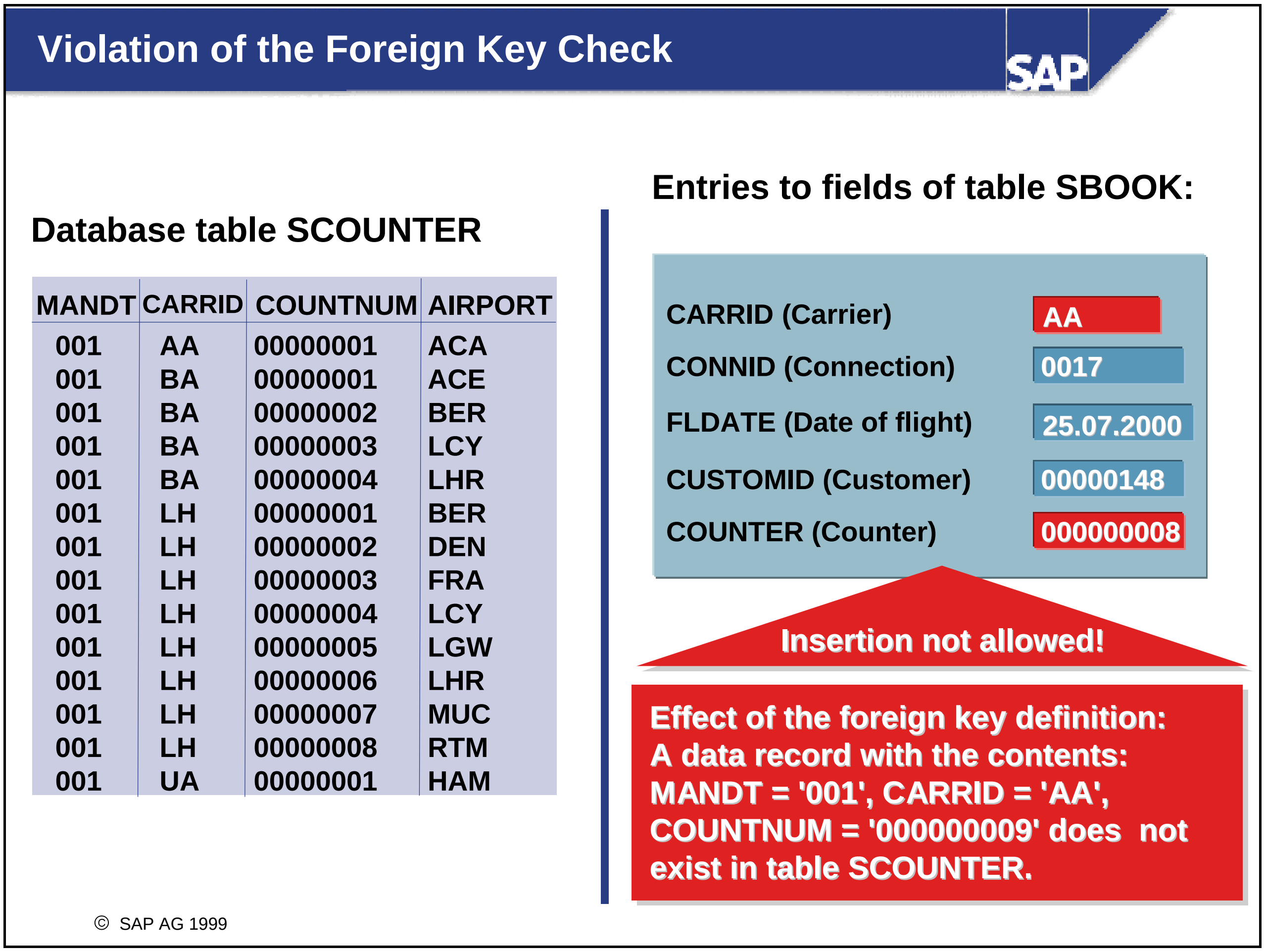
MANDT	CARRID	COUNTNUM	AIRPORT
001	AA	00000001	ACA
001	BA	00000001	ACE
001	BA	00000002	BER
001	BA	00000003	LCY
001	BA	00000004	LHR
001	LH	00000001	BER
001	LH	00000002	DEN
001	LH	00000003	FRA
001	LH	00000004	LCY
001	LH	00000005	LGW
001	LH	00000006	LHR
001	LH	00000007	MUC
001	LH	00000008	RTM
001	UA	00000001	HAM

Entries to fields of table SBOOK (flight booking):

CARRID (Carrier)	AA
CONNID (Connection)	0017
FLDATE (Date of flight)	25.07.2000
CUSTOMID (Customer)	00000148
COUNTER (Counter)	00000008

Can this flight be booked at sales counter 8 ?

- A customer wants to book a flight with American Airlines (AA). This flight with flight number 0017 is to be on November 22, 1997. The booking should be made at counter 8.
- Table SBOOK contains all the flight bookings of the carriers.
- Table SCOUNTER contains all the valid counters of the carriers.
- If an entry is made in field COUNTER of table SBOOK, you must make sure that only valid counters can be entered. This means that the counters must be stored in table SCOUNTER.
- **Question:**
  Are you allowed to insert the above data record in table SBOOK?



Violation of the Foreign Key Check

Database table SCOUNTER

MANDT	CARRID	COUNTNUM	AIRPORT
001	AA	00000001	ACA
001	BA	00000001	ACE
001	BA	00000002	BER
001	BA	00000003	LCY
001	BA	00000004	LHR
001	LH	00000001	BER
001	LH	00000002	DEN
001	LH	00000003	FRA
001	LH	00000004	LCY
001	LH	00000005	LGW
001	LH	00000006	LHR
001	LH	00000007	MUC
001	LH	00000008	RTM
001	UA	00000001	HAM

Entries to fields of table SBOOK:

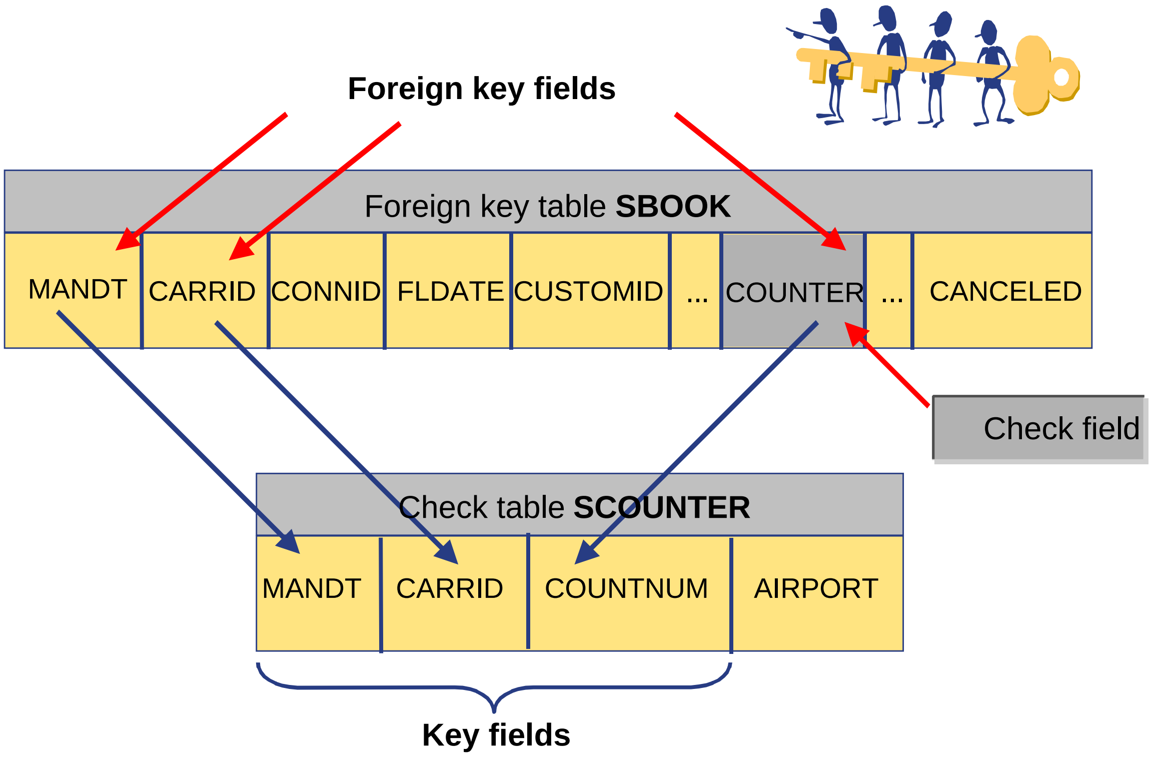
CARRID (Carrier)	AA
CONNID (Connection)	0017
FLDATE (Date of flight)	25.07.2000
CUSTOMID (Customer)	00000148
COUNTER (Counter)	000000008

Insertion not allowed!

Effect of the foreign key definition: A data record with the contents: MANDT = '001', CARRID = 'AA', COUNTNUM = '000000009' does not exist in table SCOUNTER.

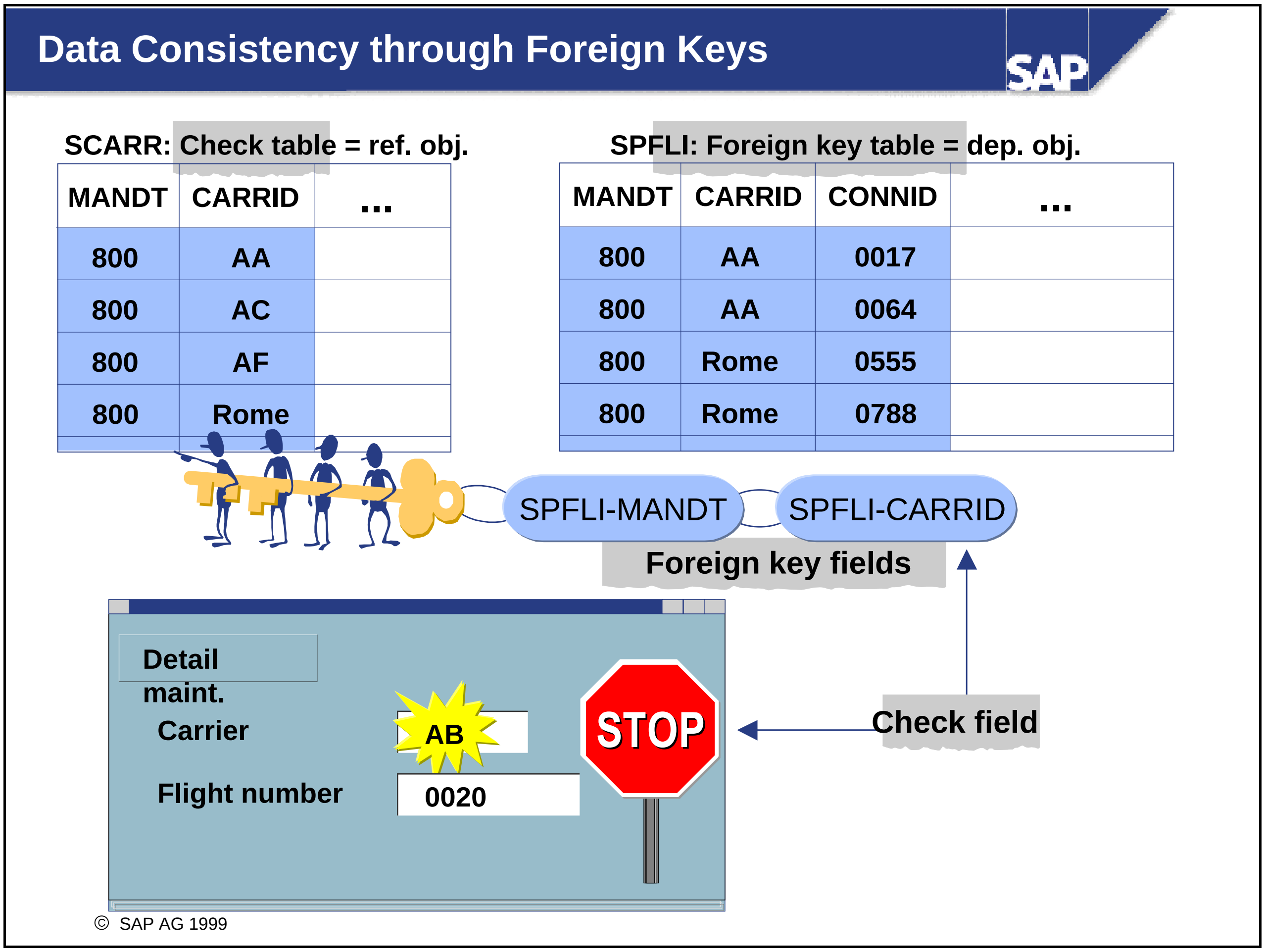
- The flight cannot be booked because American Airlines (AA) does not have a counter 8.
- No data record is selected in table SCOUNTER for the entries in the example. The entry for table SBOOK is rejected.
- In the ABAP Dictionary, such relationships between two tables are called **foreign keys** and they must be defined explicitly for the fields.
- Foreign keys are used to ensure that the data is consistent. Data that has been entered is checked against existing data to ensure that it is consistent.

Foreign Key Fields / Check Fields



- **EXAMPLE**: In this example, the **foreign key table** is table SBOOK. The purpose of the foreign key is to ensure that only valid counters of carriers can be assigned to a booking. **Check table** SCOUNTER contains exactly this information. Each counter is identified with three key fields in this table: MANDT, CARRID, and COUNTNUM.
- In order to define the foreign key, these three fields are assigned to fields of the foreign key table (foreign key fields) with which the input to be checked is entered on the screen. In table SBOOK these are the fields: MANDT, CARRID, COUNTER. The entry is accepted if it represents a valid counter; otherwise the system will reject it.
- The foreign key is defined for field SBOOK-COUNTER (check field), which means that the entry in this field is checked. Field COUNTER is therefore called the **check field** for this foreign key.
  A foreign key is defined for field COUNTER, table SBOOK, resulting in the following field assignment:

Check table	**Foreign key table**
SCOUNTER-MANDT	SBOOK-MANDT
SCOUNTER-CARRID	SBOOK-CARRID
SCOUNTER-COUNTNUM	SBOOK-COUNTER



Data Consistency through Foreign Keys

SCARR: Check table = ref. obj.

MANDT	CARRID	...
800	AA	
800	AC	
800	AF	
800	Rome	

SPFLI: Foreign key table = dep. obj.

MANDT	CARRID	CONNID	...
800	AA	0017	
800	AA	0064	
800	Rome	0555	
800	Rome	0788	

SPFLI-MANDT SPFLI-CARRID

Foreign key fields

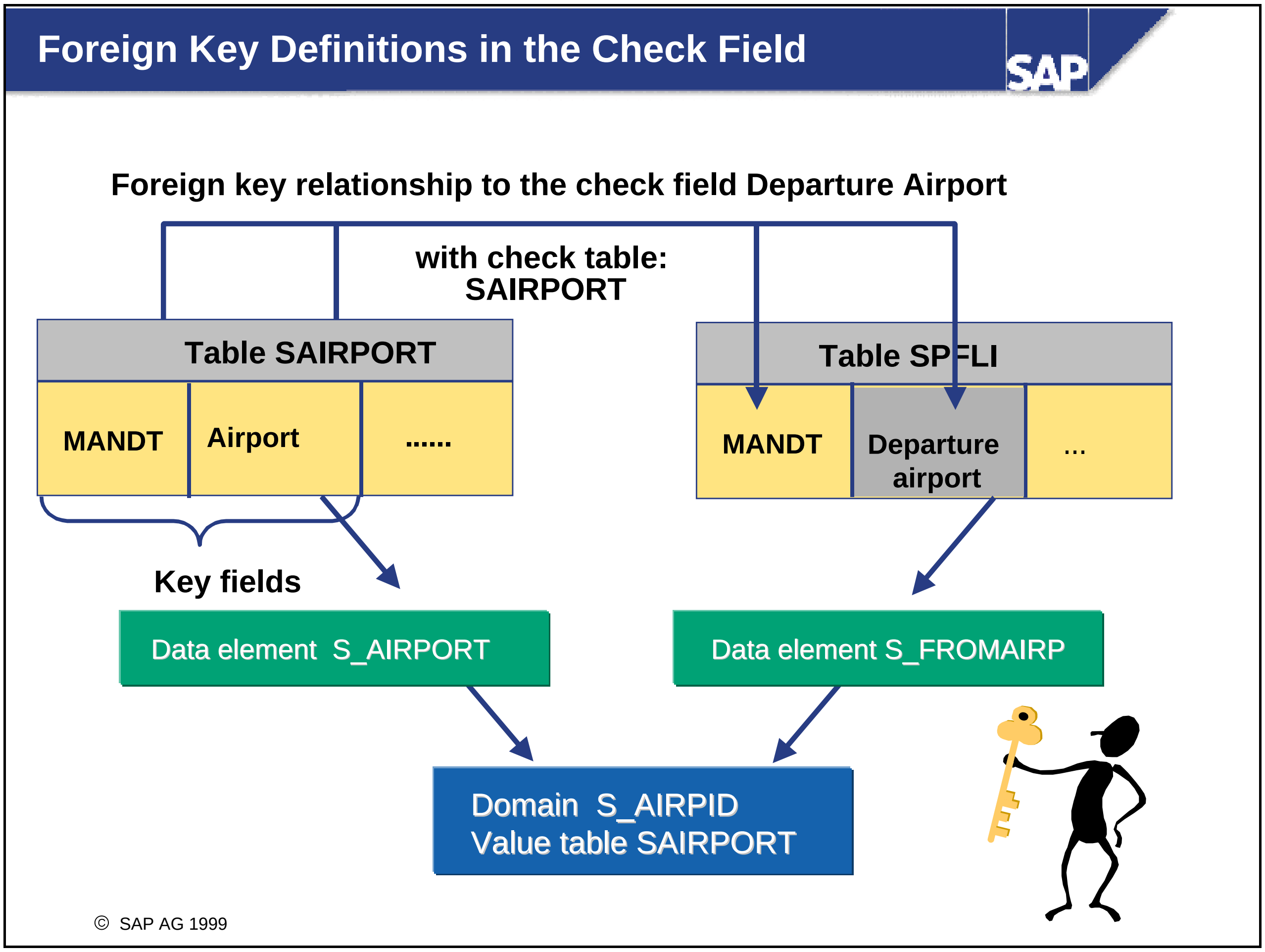
Detail maint.

Carrier AB

Flight number 0020

Check field

- A combination of fields of a table is called a foreign key if this field combination is the primary key of another table.
- A foreign key links two tables.
- The check table is the table whose key fields are checked. This table is also called the *referenced table*.
- An entry is to be written in the foreign key table. This entry must be consistent with the key fields of the check table.
- The field of the foreign key table to be checked is called the *check field*.
- Foreign keys can only be used in screens. Data records can be written to the table without being checked using an ABAP program.
- **Example:** A new entry is to be written in table SPFLI (flight schedule). There is a check whether the airline carrier entered is stored in table SCARR (carrier) for field SPFLI-CARRID. The record is only copied to table SPFLI (foreign key table) if this is the case. A foreign key is defined for field SPFLI-CARRID (check field), i.e. the checks are on this field. The corresponding check table is table SCARR with the primary key fields MANDT and CARRID.



Foreign Key Definitions in the Check Field

Foreign key relationship to the check field Departure Airport

with check table: SAIRPORT

Table SAIRPORT

MANDT

Airport

......

Key fields

Table SPFLI

MANDT

Departure airport

...

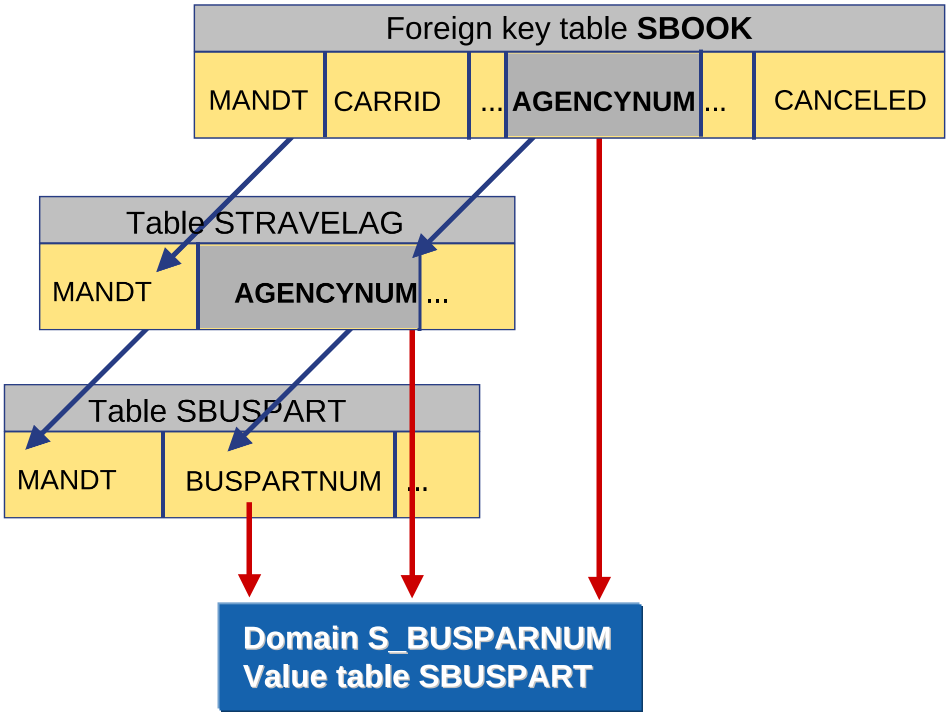
Data element S_AIRPORT

Data element S_FROMAIRP

Domain S_AIRPID
Value table SAIRPORT

- In the ABAP Dictionary, the same domain is required for the check field and referenced key field of the check table so that you do not compare fields with different data types and field lengths. **Domain equality is essential. Different data elements can be used, but they must refer to the same domain.**
- The requirement for domain equality is only valid for the check field. For all other foreign key fields, it is sufficient if the data type and the field length are equal. You nevertheless should strive for domain equality. In this case the foreign key will remain consistent if the field length is changed because the corresponding fields are both changed. If the domains are different, the foreign key will be inconsistent if for example the field length is changed.
- If the domain of the check field has a value table, you can have the system make a proposal with the value table as check table. In this case a proposal is created for the field assignment in the foreign key.
- **CAUTION!**
  The constellation that a domain that itself has table SAIRPORT as value table is used following field SAIRPORT-Airport is correct !! However, a foreign key is never defined on this field (avoiding a loop).

Check Table not Equal to Value Table



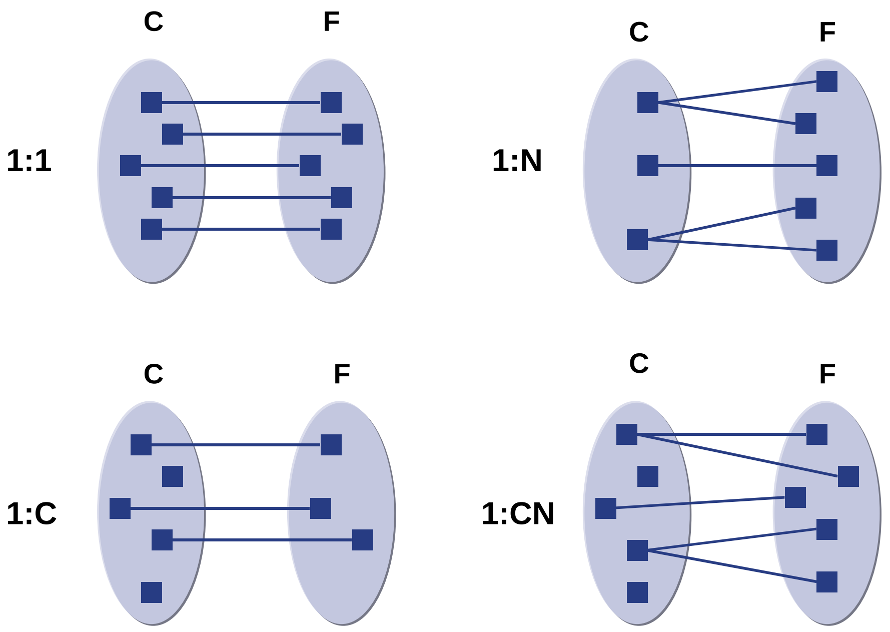
- In the above example for the foreign key definition for field SBOOK-AGENCYNUM, the system proposal is as follows based on the value table in the domain:
  Check table: SBUSPART
  Field assignment:

Check table	**Foreign key table**
SBUSPART-MANDT	SBOOK-MANDT
SBUSPART-BUSPARTNUM	SBOOK-AGENCYNUM

- **This proposal does not do what we want it to do:**
  Table SBUSPART contains all the business partners of airline carriers. However, only agencies are allowed for field SBOOK-AGENCYNUM. Table SBUSPART therefore contains invalid data for this field. The system proposal is therefore incorrect! The right check table is table STRAVELAG. It is a subset of table SBUSPART due to its foreign key definition on field AGENCYNUM.
  You must overwrite the system proposal with table STRAVELAG. If you do not know the correct check table, the system can help you by listing all the tables in question. This includes all the tables that have a key field with domain S_ BUSPARNUM.

Semantic Attributes

Cardinality



- The cardinality describes the foreign key relationship with regard to how many records of the check table are assigned to records of the foreign key table. The cardinality is always defined from the point of view of the check table.
- The type of the foreign key field defines whether or not the foreign key field identifies a table entry. This means that the foreign key fields are either key fields or they are not key fields or they are a special case, namely the key fields of a text table.
- There are the following kinds of foreign key fields:
  - **not specified:** No information about the kind of foreign key field can be given
  - **no key fields/candidates:** The foreign key fields are neither primary key fields of the foreign key table nor do they uniquely identify a record of the foreign key table (key candidates). The foreign key fields therefore do not (partially) identify the foreign key table.
  - **Key fields/candidates:** The foreign key fields are either primary key fields of the foreign key table or they uniquely identify a record of the foreign key table (key candidates). The foreign key fields therefore (partially) identify the foreign key table.
  - **Key fields of a text table***:* The foreign key table is a text table of the check table, i.e. the key of the foreign key table only differs from the key of the check table in an additional language key field. This is a special case of the category *Key fields / candidates*.

Text Table

Foreign Key Relationship with Check Table SMEAL
Type of foreign key fields: key fields of a text table

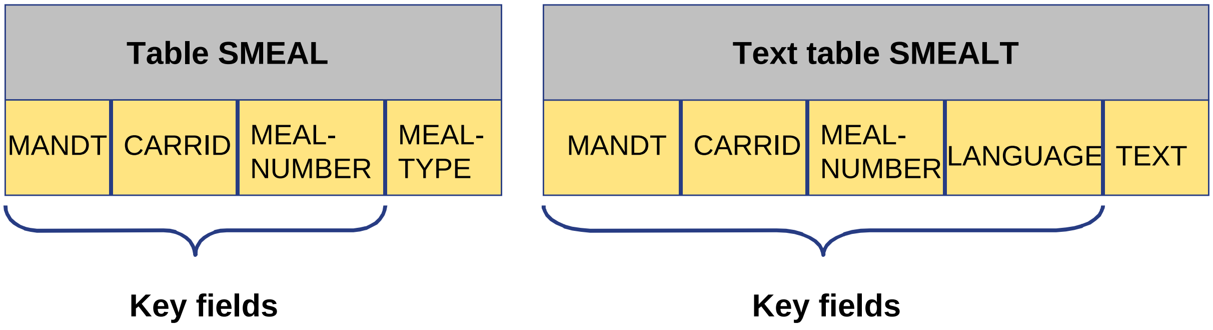


Table SMEAL			
MANDT	CARRID	MEAL-NUMBER	MEAL-TYPE

Text table SMEALT				
MANDT	CARRID	MEAL-NUMBER	LANGUAGE	TEXT

- Table SMEAL contains the meals served to the passengers during a flight. The meal names are maintained in table SMEALT.
- Table SMEALT is the text table for table SMEAL since the key of SMEALT consists of the key of SMEAL and an additional language key field (field with data type LANG).
- Table SMEALT can contain explanatory text in several languages for each key entry of SMEAL.
- To link the key entries with the text, the text table SMEALT must be linked with table SMEAL using a foreign key. *Key fields of a text table* must be chosen for the type of the foreign key fields.
- **The foreign key relationship is defined from SMEALT to SMEAL.**
- Only one text table can be created for a table.

Summary

- Two ways to check the data consistency were introduced in this unit:
  - Fixed values (static)
  - Foreign keys (dynamic)
- The fixed values are stored in the domain.
- The foreign key permits you to check the values you enter against existing data.
- The foreign key relationship is maintained for the check field (field of the foreign key table).
- The check field must be assigned to a check table key field having the same domain.

Exercises: Consistency through Input Checks

Unit: Consistency through Input Checks

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

- Create fixed values
- Set value tables to the correct context
- Define foreign keys
- Use the above mechanism to ensure that the data is consistent

When you enter or change the employee master data, only consistent data, i.e. valid airline carriers, departments, areas, etc., should be allowed. This will be implemented in the next exercise.

4-1 The employees of the airlines are divided into administration personnel (A), flight personnel (F) and service personnel (S). They are assigned to activity areas A, F or S accordingly. Make sure that only valid activity areas can be entered in table ZEMPLOYxx.

4-2 Define suitable foreign keys for tables ZEMPLOYxx and ZDEPMENTxx. Use the tables of the flight model or tables T000 (client) and SCURX (currency code) as well as your tables to define the foreign keys. Define a foreign key check for each of the following fields:

ZEMPLOYxx-Client

ZEMPLOYxx-Carrier

ZEMPLOYxx-Department code

ZEMPLOYxx-Currency

and

ZDEPMENTxx-Client

ZDEPMENTxx-Carrier

Maintain the data for table ZEMPLOYxx and test the effect of your foreign key relationships.

4-3 Some employees of carriers work in travel agencies in order to sell flights for their companies there. Enhance table ZEMPLOYxx with a field that documents for each employee the travel agency in which he or she works.

Enhance table ZEMPLOYxx accordingly and define the foreign key relationship.

The table of all travel agencies is called **STRAVELAG.**

4-4 Create a text table ZDEPMENTTxx for table ZDEPMENTxx to explain the department code for the employees of the carriers in all countries.

Create the corresponding table and use data elements SPRAS and S_TEXT for the field definition.

Define the required foreign key relationships.

Start Program BC430_CHECK in Transaction SE38. This program checks whether your solutions are correct and fills the new table ZDEPMENTTxx with sample data needed for later exercises.

Solutions: Consistency through Input Checks

Unit: Consistency through Input Checks

4-1 Call the domain maintenance screen for field ZEMPLOYxx-Area. You can do this by navigating from the table maintenance screen to the corresponding data element and from here to the domain (by double-clicking). Click on tab page *Value range* and enter the following fixed values:

Fixed value	Short description
A	Administration personnel
F	Flight personnel
S	Service personnel

Activate your domain.

4-2 To maintain the individual foreign keys, call the maintenance routine for the particular tables. Click on tab page *Fields*.

Create foreign key ZEMPLOYxx-Client as follows:

1) Place the cursor on field ZEMPLOYxx-Client. Press *Foreign key* or choose *Goto* → *Foreign key*.

2) Since you are using domain MANDT for field ZEMPLOY-Client, the system proposes value table T000 as check table.

3) Have the system make a proposal for the foreign key definition. Check the proposal. The following fields must be assigned:

Check table	CkTabFld	For. key table	For. key fld
T000	*MANDT*	ZEMPLOYxx	*Client*

4) Enter a short text and define the semantic attributes as follows:

- Type of foreign key fields: *key fields/candidates*
- Cardinality: 1:CN

5) Save your foreign key.

Create foreign key ZEMPLOYxx-Carrier as follows:

1) Place the cursor on field ZEMPLOYxx-Carrier. Press *Foreign key* or choose *Goto* → *Foreign key*.

2) Since you are using domain S_CARR_ID for field ZEMPLOY-Carrier, you can use value table SCARR for the foreign key definition.

3) Have the system make a proposal for the foreign key definition. Check the proposal. The following fields must be assigned:

Check table	ChkTablFld	For. key table	Foreign key field
SCARR	*MANDT*	ZEMPLOYxx	*Client*

SCARR	*CARRID*	ZEMPLOYxx	Carrier

4) Enter a short text and define the semantic attributes as follows:

- Type of foreign key fields: *key fields/candidates*
- Cardinality: 1:CN

5) Save your foreign key.

Create foreign key ZEMPLOYxx-Department code as follows:

1) To get a proposal for the foreign key definition, you must change the domain for field ZEMPLOYxx-Department code. This is not absolutely necessary for later foreign key definitions, but makes the definition easier. Enter value table ZDEPMENTxx in the domain for this field and activate the domain.

2) Place the cursor on field ZEMPLOYxx-Department code. Press *Foreign key* or choose *Goto* → *Foreign key*.

3) Since you are using the domain of field ZDEPMENTxx-Department code for field ZEMPLOY-Department code, you can use value table ZDEPMENTxx for the foreign key definition.

4) Have the system make a proposal for the foreign key definition. Check the proposal. The following fields must be assigned:

Check table	ChkTablFld	For. key table	Foreign key field
ZDEPMENTxx	*Client*	ZEMPLOYxx	*Client*
ZDEPMENTxx	Carrier	ZEMPLOYxx	Carrier
ZDEPMENTxx	Department code	ZEMPLOYxx	Department code

5) Enter a short text and define the semantic attributes as follows:

- Type of foreign key fields: *non-key-fields/candidates*
- Cardinality: 1:CN

6) Save your foreign key.

Create foreign key ZEMPLOYxx-Currency as follows:

1) Place the cursor on field ZEMPLOYxx-Currency. Press *Foreign key* or choose *Goto* → *Foreign key*.

2) Since you are using domain S_CURR for field ZEMPLOYxx-Currency, you can use value table SCURX for the foreign key definition.

3) Have the system make a proposal for the foreign key definition. Check the proposal. The following fields must be assigned:

Check table	ChkTablFld	For. key table	For. key fld
SCURX	*CURRKEY*	ZEMPLOYxx	*Currency*

4) Enter a short text and define the semantic attributes as follows:

- Type of foreign key fields: *non-key-fields/candidates*
- Cardinality: 1:CN

5) Save your foreign key.

Create foreign key ZDEPMENTxx-Client as follows:

See foreign key ZEMPLOYxx-Client.

Create foreign key ZDEPMENTxx-Carrier as follows:

See foreign key ZEMPLOYxx-Carrier.

In the maintenance screen for table ZEMPLOYxx choose *Utilities* → *Table contents* → *Create entries*.

Enter data and check whether your foreign key functions correctly using the F4 help.

4-3 Create a new field *Agency* in your table ZEMPLOYxx as follows:

1) Navigate to the field maintenance screen for table ZEMPLOYxx. Insert a new field *Agency* in the field list (press *New rows*). In the maintenance screen for table STRAVELAG you can see that the suitable data element is called S_AGNCYNUM. Assign this data element to your new field ZEMPLOYxx-Agency.

2) Position the cursor on the field *Agency* and have the system make a proposal for the foreign key definition.

3) Check the proposal. The check table is SBUSPART. This check table is not correct since it contains all the business partners of carriers and not only agencies.

4) The correct check table is STRAVELAG. It contains the agencies the carriers work with.

5) You can improve understanding by looking at the definition of table STRAVELAG in a second session. Field AGENCYNUM has a foreign key with check table SBUSPART. This means that table STRAVELAG is a subset of table SBUSPART.
In the foreign key definition for ZEMPLOYxx-Agency, overwrite entry SBUSBART in the input field *Check table* with STRAVELAG.

6) Press *Copy*. The system recognizes the change in the check table and suggests that it create a proposal. Accept this offer and check the proposal. The following fields must be assigned:

Check table	ChkTablFld	For. key table	For. key fld
STRAVELAG	*MANDT*	ZEMPLOYxx	*Client*
STRAVELAG	AGENCYNUM	ZEMPLOYxx	*Agency*

7) Enter a short description. Enter cardinality 1:CN and mark that the foreign key fields are not key fields/candidates. Choose *Copy*.

8) Activate the table.

9) In the maintenance screen for table ZEMPLOYxx choose *Utilities* → *Table contents* → *Create entries*.
Verify your foreign key using the F4 help.

4-4 Create your text table ZDEPMENTTxx as follows:

1) Copy table ZDEPMENTxx to table ZDEPMENTTxx.

2) Navigate to the field maintenance screen for table ZDEPMENTTxx. Delete all the fields that are not key fields. Create the following new fields:

Field name	Data element	Data type, Length
Language	*SPRAS*	*LANG*
Description	*S_TEXT*	*CHAR40*

Field ZDEPMENTxx-Language must be a key field.

3) The foreign keys for fields ZDEPMENTTxx-Client and ZDEPMENTTxx-Carrier were already correctly defined by copying. You still have to define the foreign keys of fields ZDEPMENTTxx-Department code and ZDEPMENTTxx-Language.

4) Place the cursor on field ZDEPMENTTxx-Department code. Press *Foreign key* or choose *Goto* → *Foreign key*.

5) Since you are using the domain of field ZDEPMENTxx-Department code for field ZDEPMENTTxx-Department code, you can use value table ZDEPMENTxx for the foreign key definition.

6) Have the system make a proposal for the foreign key definition. Check the proposal. The following fields must be assigned:

Check table	ChkTablFld	For. key table	For. key fld
ZDEPMENTxx	*Client*	ZDEPMENTTxx	*Client*
ZDEPMENTxx	Carrier	ZDEPMENTTxx	Carrier
ZDEPMENTxx	Department code	ZDEPMENTTxx	Department code

7) Enter a short text and define the semantic attributes as follows:

- Type of foreign key fields: **key fields of a text table**
- Cardinality: 1:CN

8) Save your foreign keys.

9) Place the cursor on field ZDEPMENTTxx-Language. Press *Foreign key* or choose *Goto* → *Foreign key*.

10) Since you are using domain SPRAS for field ZDEPMENTTxx-Language, you can use value table T002 for the foreign key definition.

11) Have the system make a proposal for the foreign key definition. Check the proposal. The following fields must be assigned:

Check table	ChkTablFld	For. key table	Foreign key field
T002	*SPRAS*	ZDEPMENTTxx	*Language*

12) Enter a short text and define the semantic attributes as follows:

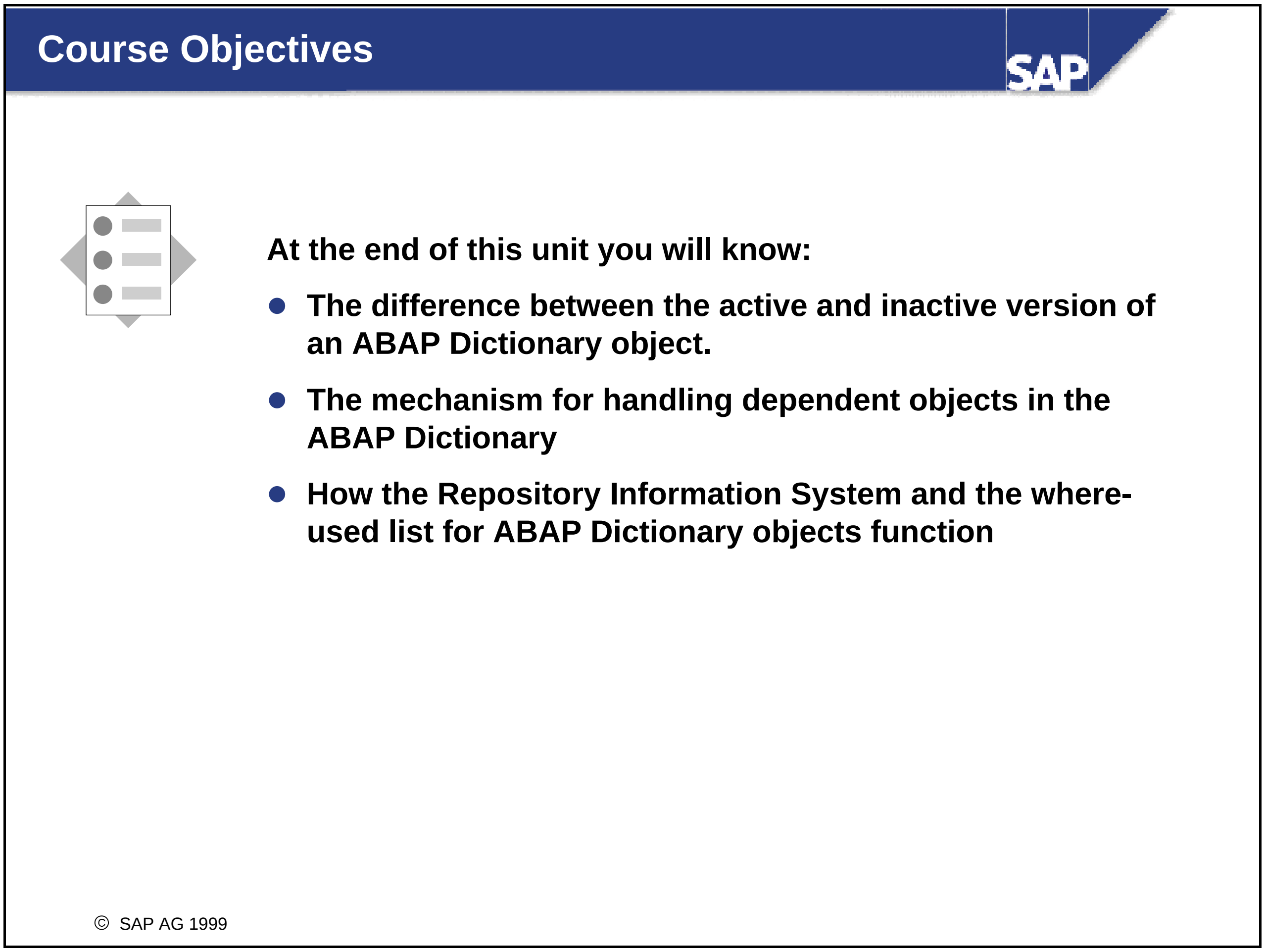
- Type of foreign key fields: *key fields/candidates*
- Cardinality: 1:CN

13) Save your foreign key.

14) Activate the table.

Dependencies of ABAP Dictionary Objects

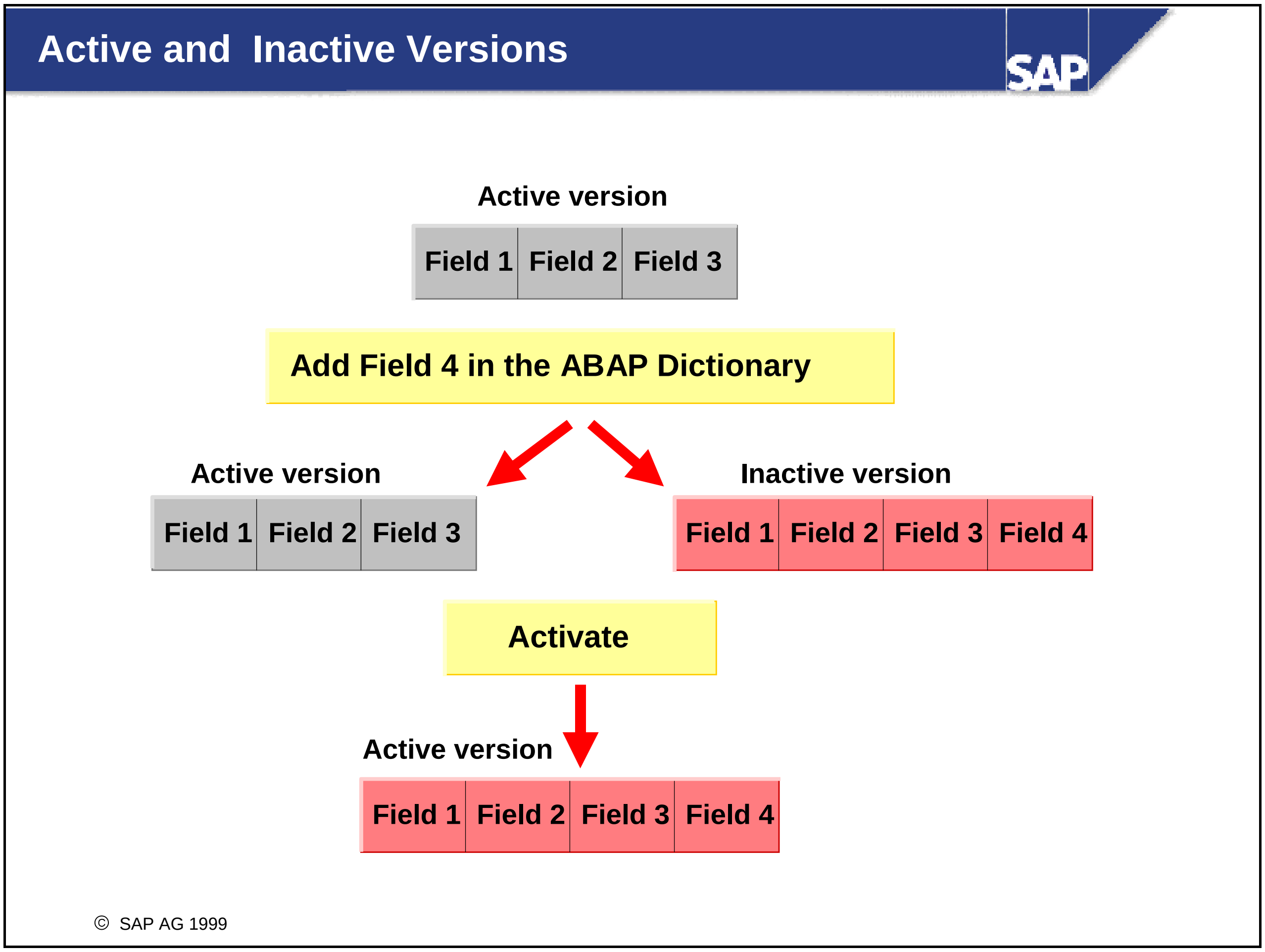
- Activation of ABAP Dictionary objects
- Handling of dependent objects
- Where-used list and R/3 Repository Information System as seen by the ABAP Dictionary



Course Objectives

At the end of this unit you will know:

- The difference between the active and inactive version of an ABAP Dictionary object.
- The mechanism for handling dependent objects in the ABAP Dictionary
- How the Repository Information System and the where-used list for ABAP Dictionary objects function



Active and Inactive Versions

Active version

Field 1 | Field 2 | Field 3

Add Field 4 in the ABAP Dictionary

Active version

Field 1 | Field 2 | Field 3

Inactive version

Field 1 | Field 2 | Field 3 | Field 4

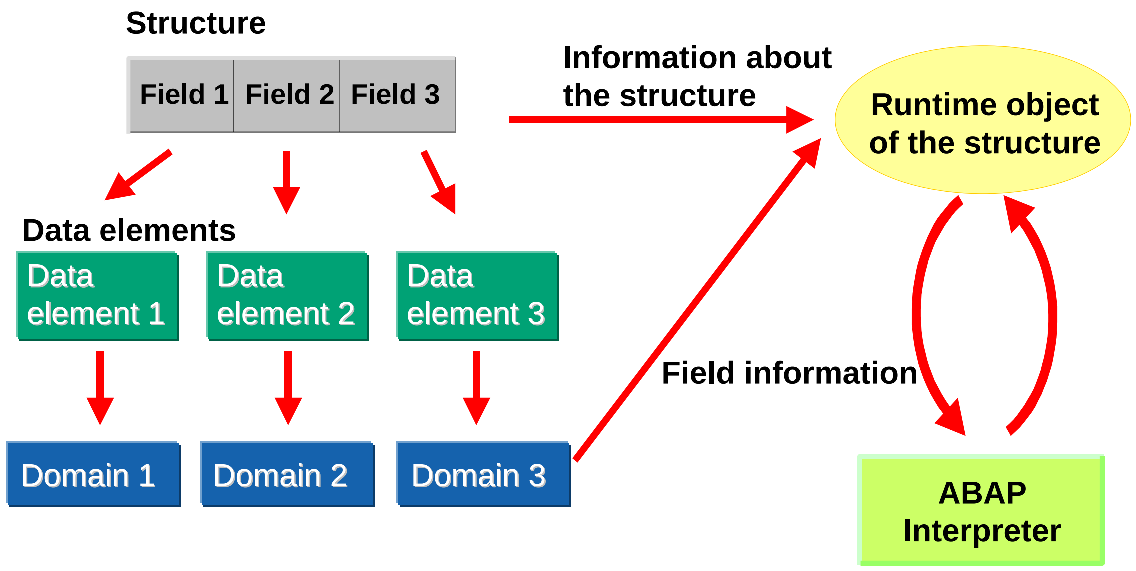
Activate

Active version

Field 1 | Field 2 | Field 3 | Field 4

- During development, you sometimes need to change an (active) object already used by the system. Such changes are supported in the ABAP Dictionary by separating the active and inactive versions.
- The **active** version of an ABAP Dictionary object is the version that the components of the runtime environment (for example ABAP processor, database interface) access. This version is not initially changed.
- An **inactive** version is created when an active object is changed. The inactive version can be saved without checking. It has no effect on the runtime system.
- At the end of the development process, the inactive version can be made the active version. This is done by **activation**. The inactive version of the object is first checked for consistency. If it is consistent, the inactive version replaces the active one. From now on, the runtime system uses the new active version.
- The above example shows how the object status changes. An active structure contains three fields. A field is added to this structure in the ABAP Dictionary. After this action, there is an active version with three fields and an inactive version with four fields. During activation, the active version is overwritten with the inactive version. The inactive version thus becomes the active version. After this action there is only the active version with four fields.

Runtime Objects



© SAP AG 1999

- The information about a structure (or table) is distributed in the ABAP Dictionary in domains, data elements, and the structure definition. The runtime object (nametab) combines this information into a structure in a form that is optimized for access from ABAP programs. The runtime object is created when the structure is activated.
- The runtime objects of the structures are buffered so that the ABAP runtime system can quickly access this information.
- The runtime object contains information about the overall structure (e.g. number of fields) and the individual structure fields (field name, position of the field in the structure, data type, length, number of decimal places, reference field, reference table, check table, conversion routine, etc.).
- The runtime object of a table contains further information needed by the database interface for accessing the table data (client dependence, buffering, key fields, etc.).
- Runtime objects are created for all ABAP Dictionary objects that can be used as types in ABAP programs. These are data elements, table types and views, as well as structures and tables.

Handling of Dependent Objects

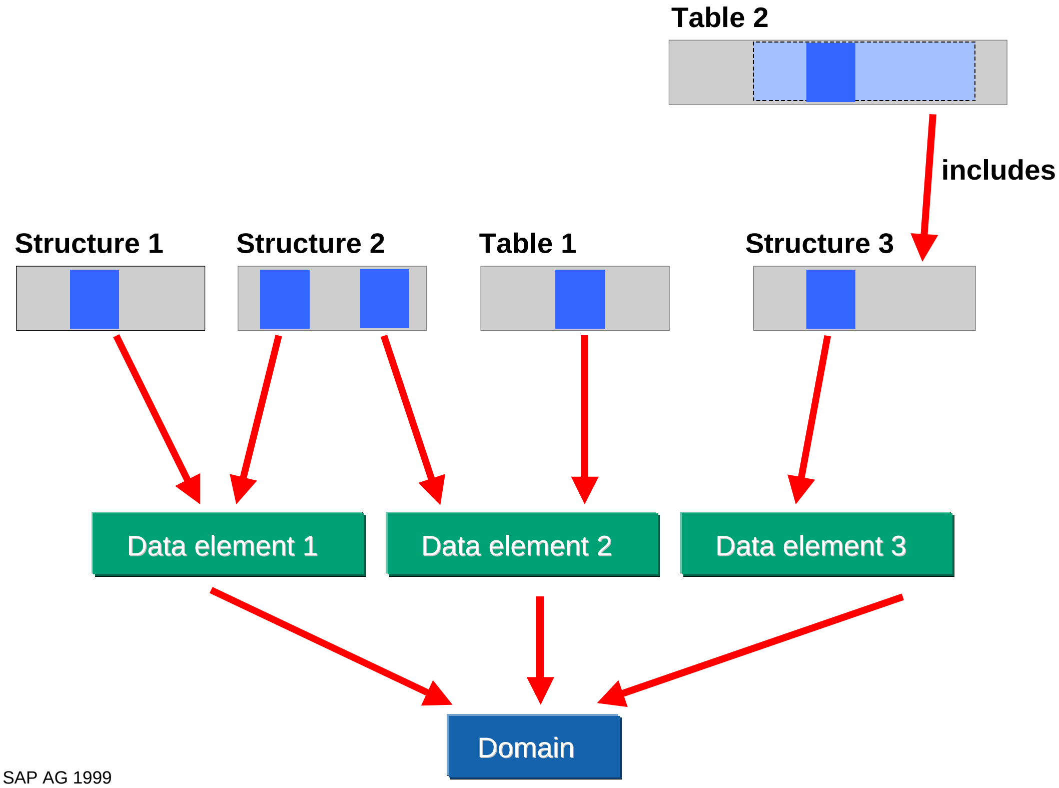


Table 2

includes

Structure 1

Structure 2

Table 1

Structure 3

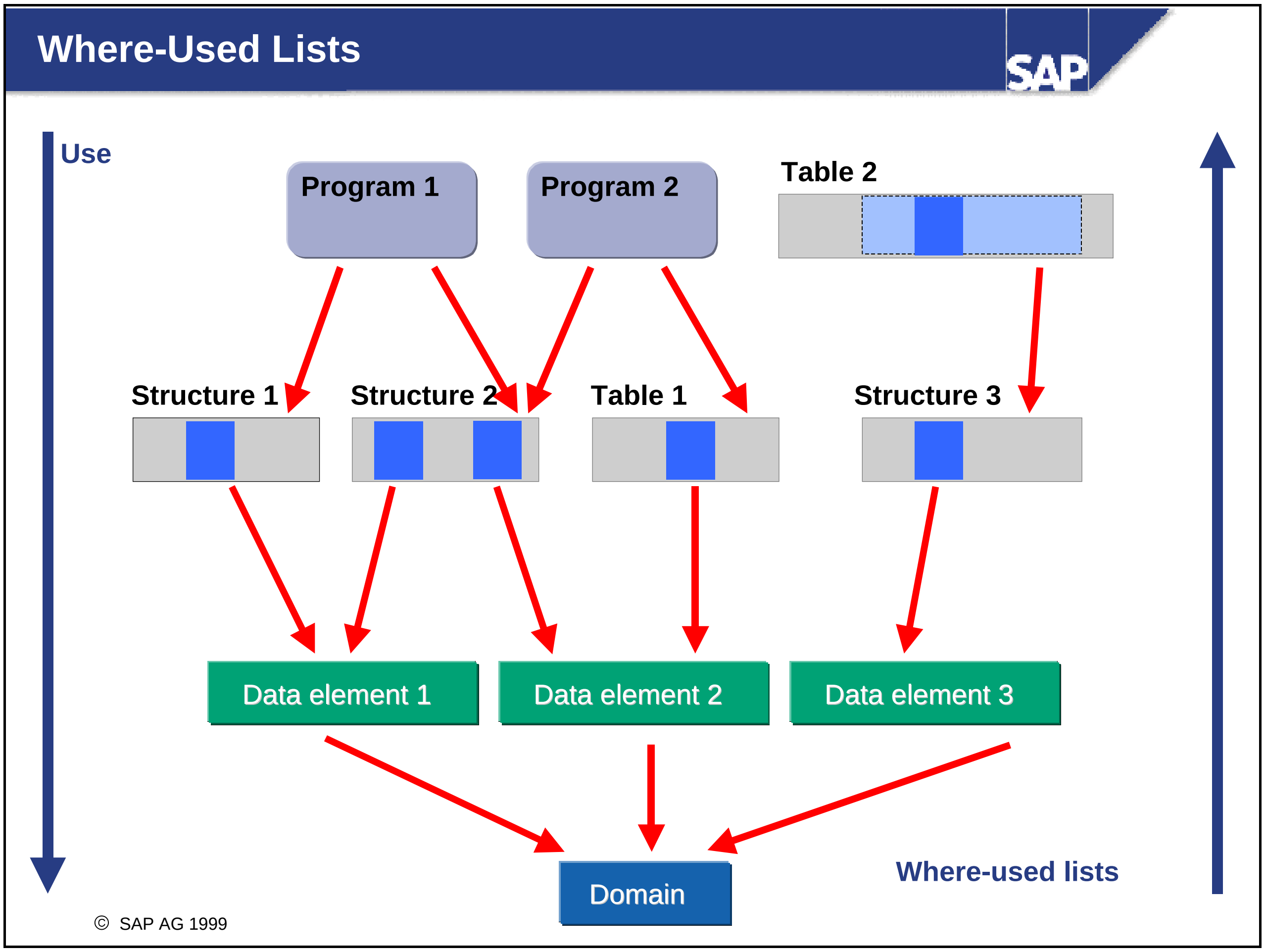
Data element 1

Data element 2

Data element 3

Domain

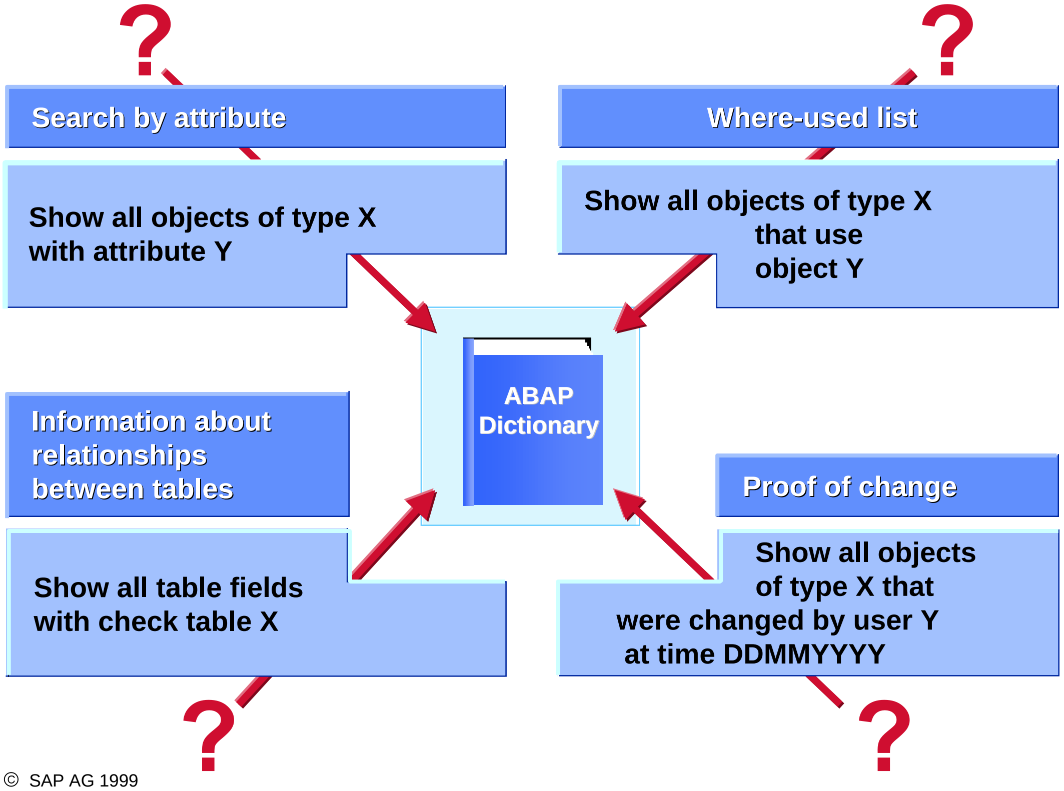
- If an object that is already active is modified, this can affect other objects that use it (directly or indirectly). These objects using another object are called **dependent** objects. On the one hand, it might be necessary to adjust the runtime objects of these dependent objects to the changes. On the other hand, a change might sometimes make a dependent object inconsistent.
- For this reason, the dependent objects are determined and activated (if necessary) when an active object is activated. The active versions of the dependent objects are activated again. In particular, new and inactive versions of objects using the changed object are not changed.
- **Example:** When you change a domain, for example its data type, all the data elements, structures and tables referring to this domain must be activated again. This activation is automatically triggered when the domain is activated. This ensures that all affected runtime objects are adjusted to the changed type information.
- If an ABAP Dictionary object has a table as dependent object, its database object as well as its runtime object might have to be adjusted when the dependent object is activated. The method used here will be discussed in the next unit.



Where-Used Lists

- Changing an ABAP Dictionary object might also affect its dependent objects. Before making a critical change (such as changing the data type or deleting a field) you should therefore define the set of objects affected in order to estimate the implications of the planned action.
- There is a **where-used list** for each ABAP Dictionary object with which you can find all the objects that refer to this object. You can call the where-used list from the maintenance transaction of the object.
- You can find direct and indirect usages of an ABAP Dictionary object with the where-used list. You also have to define which usage object types should be included in the search (e.g. all structures and tables using a data element). You can also search for usages that are not ABAP Dictionary objects (e.g. all programs using a table). The search can also be limited by development class or user namespace.
- If an object is probably used by several objects, you should perform the search in the background.

The Repository Information System ABAP Dictionary



Search by attribute

Show all objects of type X with attribute Y

Where-used list

Show all objects of type X that use object Y

ABAP Dictionary

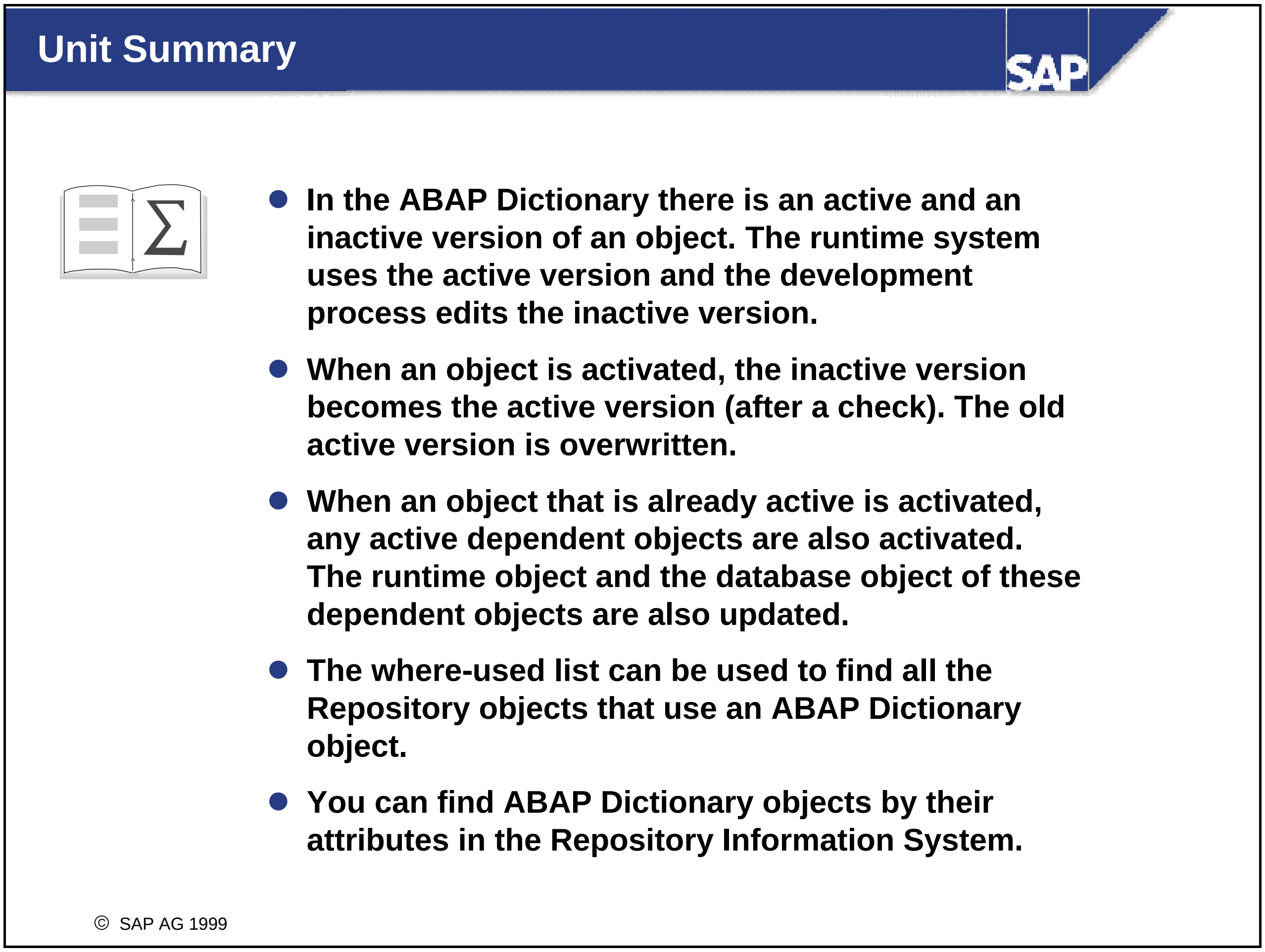
Information about relationships between tables

Show all table fields with check table X

Proof of change

Show all objects of type X that were changed by user Y at time DDMMYYYY

- The Repository Information System ABAP Dictionary is part of the general Repository Information System. It helps you search for ABAP Dictionary objects and their users.
- The where-used list for Repository objects can be called from the information system. The information system also enables you to search for objects by their attributes.
- In addition to the object-specific search criteria (e.g. buffering type for tables), you can search for all objects by development class, short description or author and date of last change.
- The object lists created by the Repository Information System are entirely integrated in the ABAP Workbench. They permit you to navigate directly to the maintenance transactions of the objects found.



Unit Summary

- In the ABAP Dictionary there is an active and an inactive version of an object. The runtime system uses the active version and the development process edits the inactive version.
- When an object is activated, the inactive version becomes the active version (after a check). The old active version is overwritten.
- When an object that is already active is activated, any active dependent objects are also activated. The runtime object and the database object of these dependent objects are also updated.
- The where-used list can be used to find all the Repository objects that use an ABAP Dictionary object.
- You can find ABAP Dictionary objects by their attributes in the Repository Information System.

Exercises: Dependencies of ABAP Dictionary Objects

Unit: Dependencies of ABAP Dictionary Objects

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

- Enhance tables and structures with fields
- Use the R/3 Repository Information System and the where-used list for ABAP Dictionary objects

Information about the head of the department should be stored in the employee management system. The change log should also be made more detailed. This exercise makes the appropriate enhancements to the tables and structures.

5-1 Each department of an airline has a head of department. The assignment between the department and the head of the department should be mapped in the flight model. Enhance table ZDEPMENTxx with field *Dephead* Define a suitable foreign key for this field.

Use the two-step domain concept.

5-2 The change log for tables ZEMPLOYxx and ZDEPMENTxx is not precise enough. In addition to the person who made the last change and the date of this change, you also want to record the time of the last change. Have a suitable field inserted in both tables as easily as possible. Use data element S_TIME.

Make sure that the field is inserted in both tables. Check the activation log of the tables and structures involved.

Start Program BC430_CHECK in Transaction SE38. It checks whether your solutions are correct.

5-3 Create a list of the following ABAP Dictionary objects:

5-3-1 All domains with fixed values whose names begin with Z.

5-3-2 All table fields that use data element S_FNAME.

5-3-3 All tables of the flight model (development class BC_DATAMODEL) that have delivery class A.

5-4 Determine all the programs that use table SFLIGHT.

5-5 What are the data elements created by your neighbors called?

Unit: Dependencies of ABAP Dictionary Objects

5-1 In our model, people are identified by their personnel number. Therefore the new field to be added to table ZDEPMENTxx must contain personnel numbers (and not names). The field should therefore refer to the domain for personnel numbers that you created in the first exercise. Since the person to be managed in this case has a special role, you should create a new data element and not use the one already created for the personnel number. You can either first create the data element and then maintain the table, or create the data element from the table maintenance screen with forward navigation. The second way is described below:

1) Go to change mode in the maintenance screen for table ZDEPMENTxx. Click on tab page *Fields*.

2) Choose *New rows*.

3) Enter the new field directly following the existing fields by entering a suitable field name in the first column and entering a name for the data element to be created in column *Field type*.

4) Save the table definition.

5) Double click on the name of the new data element to be created. Confirm that you want to create a data element.

6) Enter a short text for the data element. Enter the domain name that you already created for the personnel number in field *Domain*.

7) Click on tab page *Field label* and enter the corresponding text there.

8) Activate the data element. Go back to the maintenance screen for table ZDEPMENTxx by choosing *Back.*

9) Create the foreign key for the new field in the usual manner. The check table is table ZEMPLOYxx. If you stored this table as a value table for the domain for the personnel number, the system will make this proposal. If not you have to enter it yourself. You can copy the system proposal in the field allocation of the foreign key. The cardinality is 1:CN and the foreign key fields are not foreign key fields/candidates.

10) Activate the table.

5-2 The fields for the change log can be found in the include structure ZCHANGExx. The new field should therefore be inserted in this structure. The field is automatically inserted in tables ZEMPLOYxx and ZDEPMENTxx using the include mechanism. Proceed as follows:

1) In the initial screen of the ABAP Dictionary, select *Data type* and enter ZCHANGExx in the corresponding field. Choose *Change*.

2) Click on tab page *Components*. Enter the name for the new field in the first free row of the component list and enter S_TIME in column *Component type*.

3) Activate the structure.

4) With *Utilities* → *Activation log* you can find the activation log of the structure. You can see here that tables ZEMPLOYxx and ZDEPMENTxx are activated as dependent objects and were enhanced with the new field.

5) Go to display mode in the maintenance screen for table ZEMPLOYxx (or ZDEPMENTxx). Choose *Utilities* → *Table contents* → *Create entries.* You can see here that the table was really enhanced with the corresponding field.

5-3 All the exercises can be solved with the Repository Information System. You can do this from the initial screen of the ABAP Dictionary with *Environment →?Repository Information System*. Expand the nodes for the ABAP Dictionary.

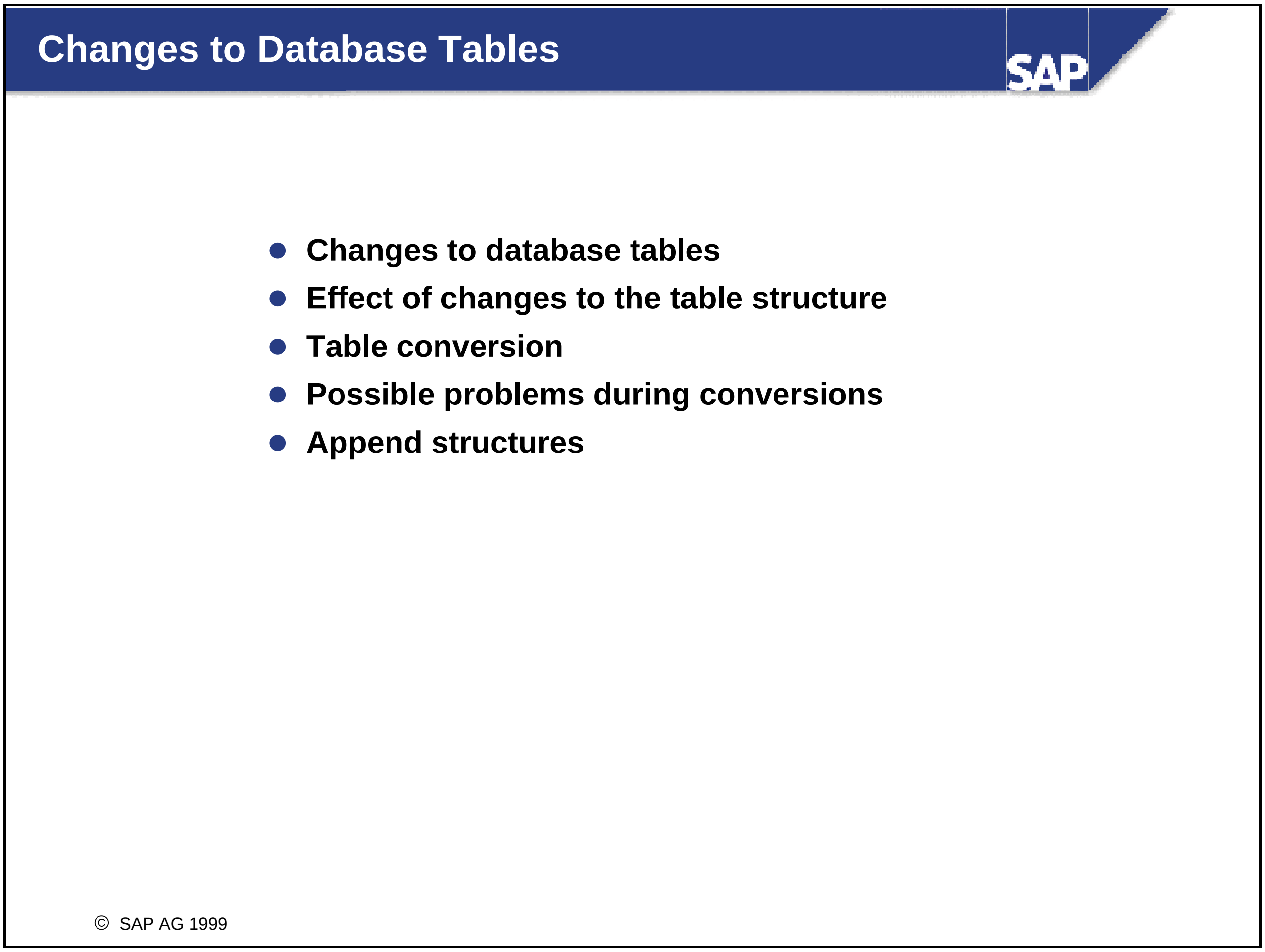
1) Expand the node *Basic objects*. Double-click on *Domains*. In the selection screen, enter Z* in the first field. Choose *All selections*. In the enhanced selection screen, mark *Only domains with fixed values*. You can create the desired list with *Execute*.

2) Choose *Back* twice to return to the initial screen of the Repository Information System. Expand the *Fields* node. Double-click on *Table fields*. Choose *All selections* and enter S_FNAME in field *Data element*. You can create the desired list with *Execute*.

3) Choose *Back* twice to return to the initial screen of the Repository Information System. Node *Basic objects* is still expanded. Double-click on *Database tables*. Enter development class BC_DATAMODEL and (after choosing *All selections*) delivery class A in the selection screen. You can create the desired list with *Execute*.

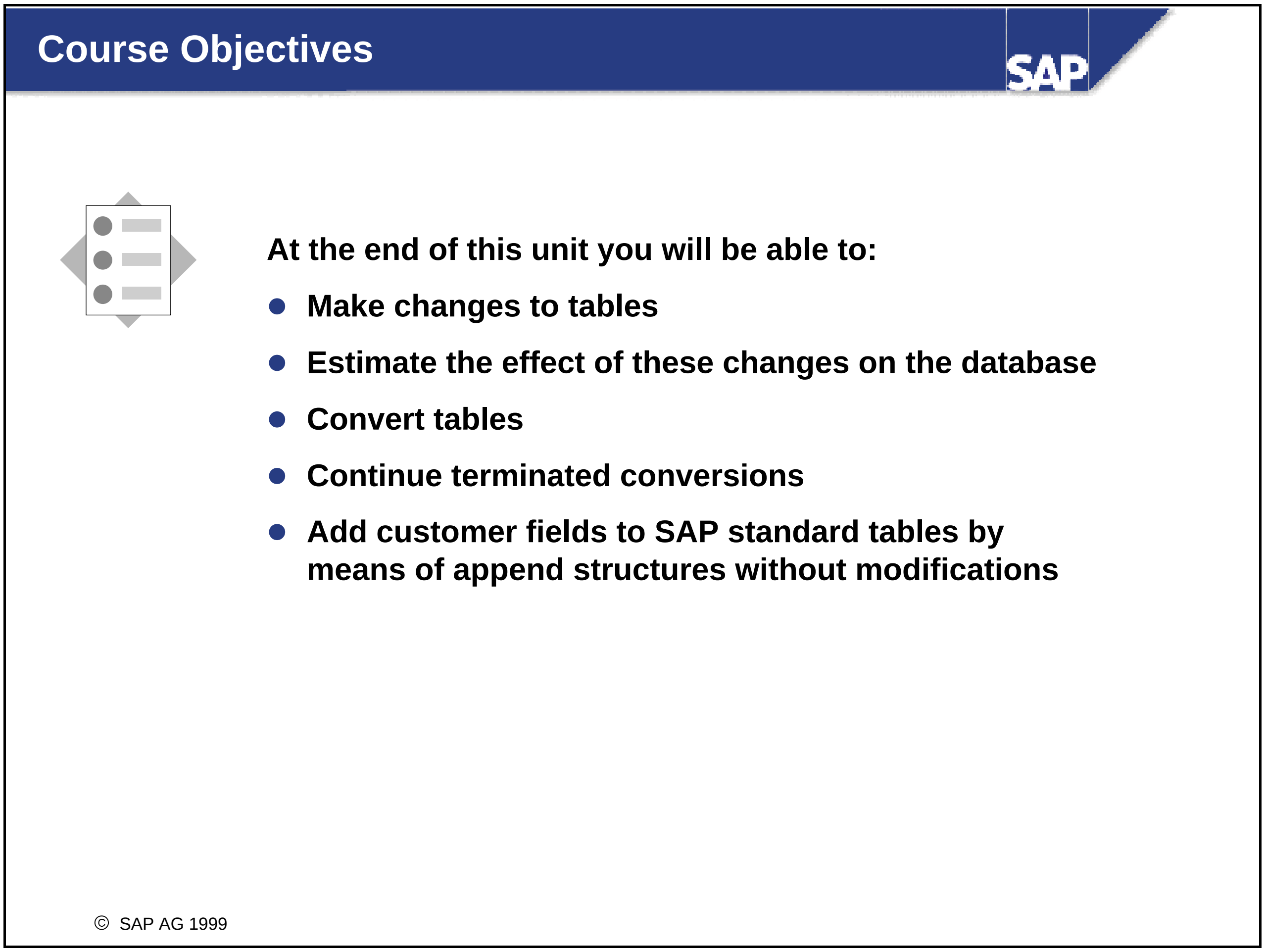
5-4 Go to the initial screen of the ABAP Dictionary. Choose *Database table a*nd enter SFLIGHT in the corresponding field. Choose *Where-used list*. The usage in programs is already marked (alone) in the next dialog box. You can create the desired list with *Execute*.

5-5 You can create the desired list again in the Repository Information System ABAP Dictionary. Expand the node *Basic objects* and double-click on *Data elements*. You can define your neighbor's data elements with either a pattern search for the name (if your neighbors adhered to the given naming convention) or with *Last changed by* (after choosing *All selections*). If you have two groups of neighbors, you have to use *Multiple selection*. You can restrict the selection with the date of the last change (the last change should be no earlier than the beginning of the course) at least in the first case (naming convention).



Changes to Database Tables

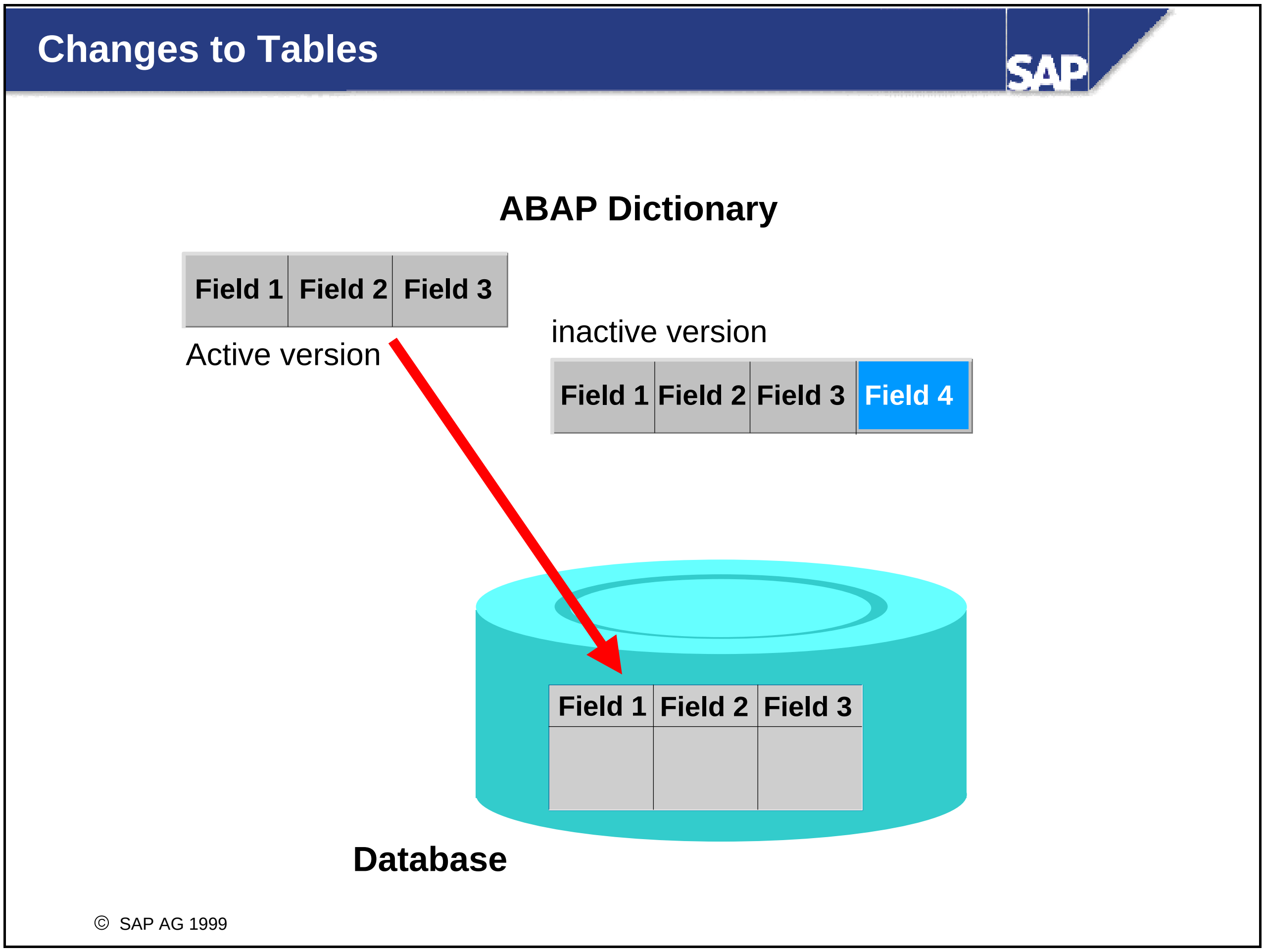
- Changes to database tables
- Effect of changes to the table structure
- Table conversion
- Possible problems during conversions
- Append structures



Course Objectives

At the end of this unit you will be able to:

- Make changes to tables
- Estimate the effect of these changes on the database
- Convert tables
- Continue terminated conversions
- Add customer fields to SAP standard tables by means of append structures without modifications



Changes to Tables

ABAP Dictionary

Field 1 Field 2 Field 3

Active version

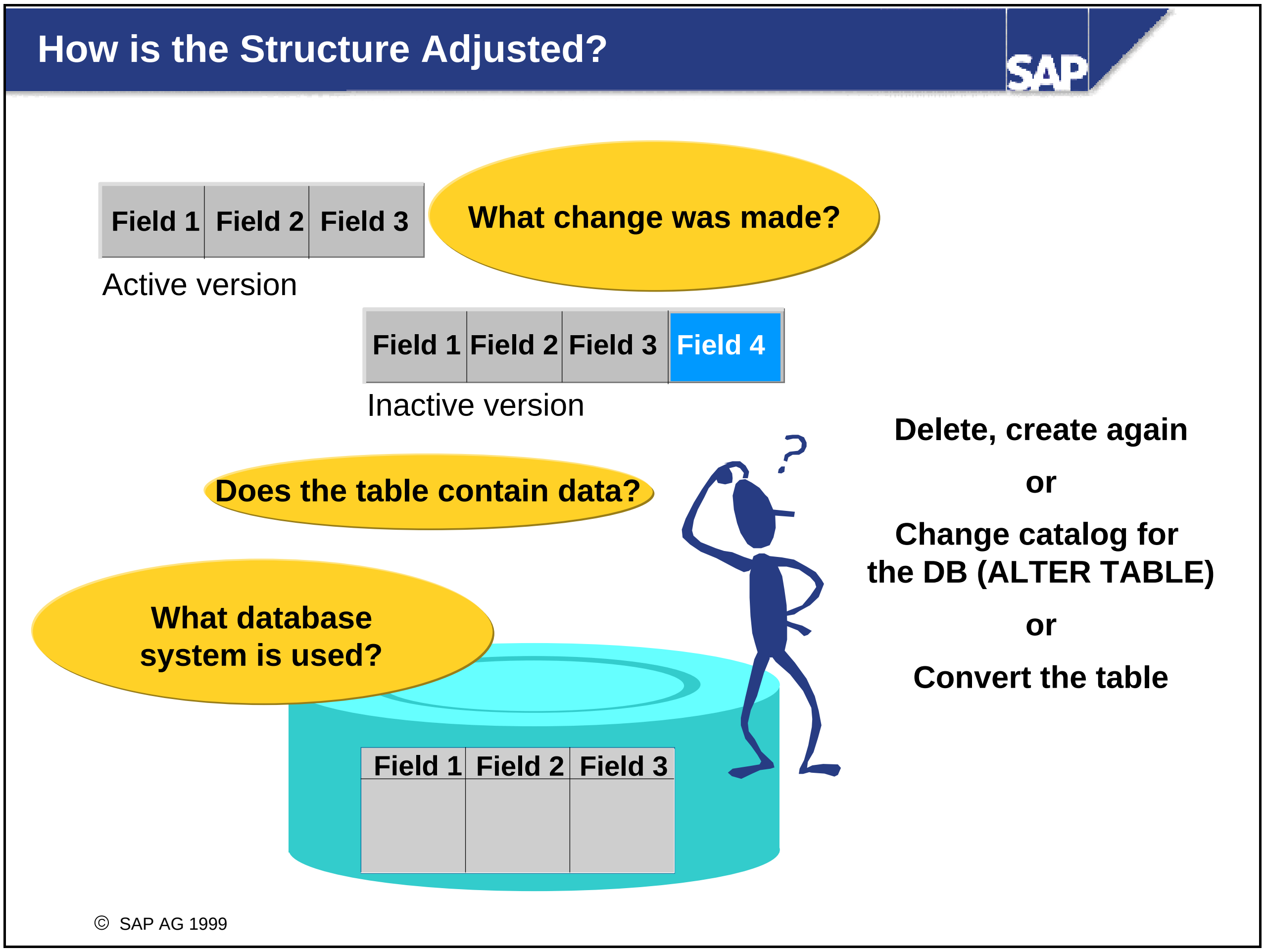
inactive version

Field 1 Field 2 Field 3 Field 4

Field 1 Field 2 Field 3

Database

- Correct access by ABAP programs to a database table is only possible if the runtime object of the table is consistent with the structure of the table in the database. Each time the table is changed in the ABAP Dictionary, you must check if the database structure of the table must be adjusted to the changed ABAP Dictionary definition of the table when it is activated (when the runtime object is rewritten).
- The database structure does not have to be altered for certain changes to the ABAP Dictionary. For example, you do not have to change the database structure when the order of the fields in the ABAP Dictionary is changed (other than for key fields). In this case the changed structure is simply activated in the ABAP Dictionary and the database structure remains unchanged.



How is the Structure Adjusted?

Field 1	Field 2	Field 3

Active version

What change was made?

Field 1	Field 2	Field 3	Field 4

Inactive version

Does the table contain data?

What database system is used?

Field 1	Field 2	Field 3

Delete, create again

or

Change catalog for the DB (ALTER TABLE)

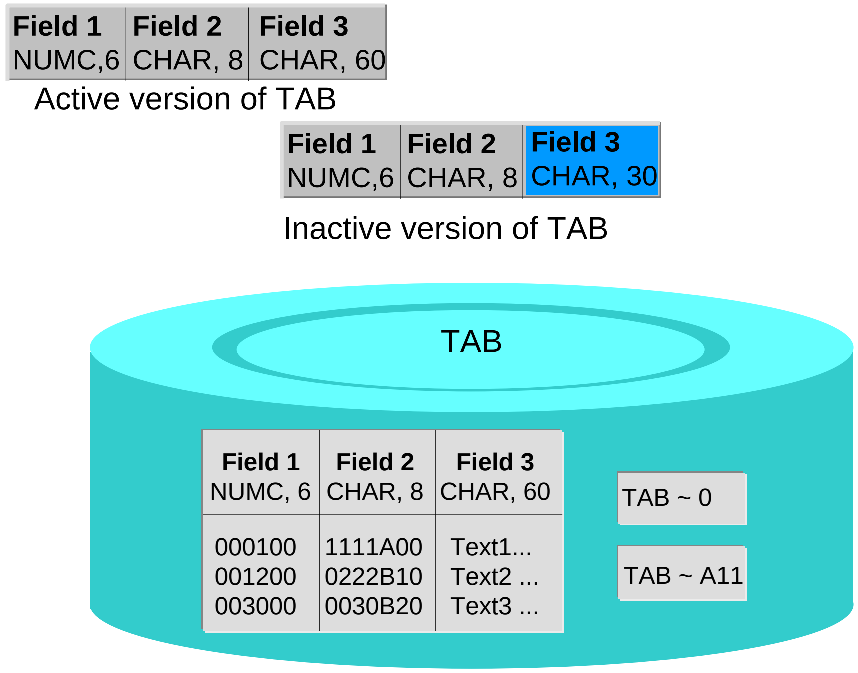
or

Convert the table

© SAP AG 1999

- The database table can be adjusted to the changed definition in the ABAP Dictionary in three different ways:
  - By deleting the database table and creating it again. The table on the database is deleted, the inactive table is activated in the ABAP Dictionary, and the table is created again on the database. Data existing in the table is lost.
  - By changing the database catalog (ALTER TABLE). The definition of the table on the database is simply changed. Existing data is retained. However, indexes on the table might have to be built again.
  - By converting the table. This is the most time-consuming way to adjust a structure.
- If the table does not contain any data, it is deleted in the database and created again with its new structure. If data exists in the table, there is an attempt to adjust the structure with ALTER TABLE. If the database system used is not able to do so, the structure is adjusted by converting the table.

Conversion Process 1



Field 1	Field 2	Field 3
NUMC,6	CHAR, 8	CHAR, 60

Active version of TAB

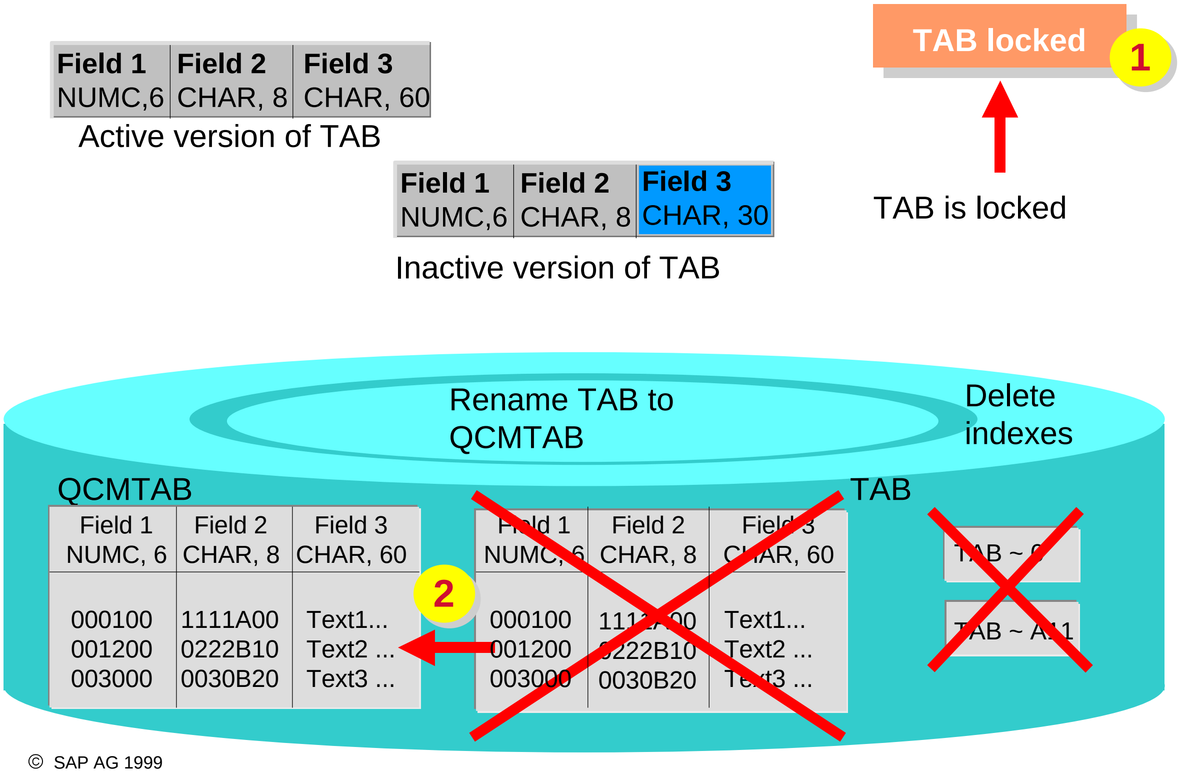
Field 1	Field 2	Field 3
NUMC,6	CHAR, 8	CHAR, 30

Inactive version of TAB

Field 1 NUMC, 6	Field 2 CHAR, 8	Field 3 CHAR, 60
000100	1111A00	Text1...
001200	0222B10	Text2 ...
003000	0030B20	Text3 ...

- The following example shows the steps necessary during conversion.
- **Starting situation:** Table TAB was changed in the ABAP Dictionary. The length of field 3 was reduced from 60 to 30 places.
- The ABAP Dictionary therefore has an active (field 3 has a length of 60 places) and an inactive (field 3 still has 30 places) version of the table.
- The active version of the table was created in the database, which means that field 3 currently has 60 places in the database. A secondary index with the ID A11, which was also created in the database, is defined for the table in the ABAP Dictionary.
- The table already contains data.

Conversion Process 2



TAB locked 1

TAB is locked

Field 1	Field 2	Field 3
NUMC,6	CHAR, 8	CHAR, 60

Active version of TAB

Field 1	Field 2	Field 3
NUMC,6	CHAR, 8	CHAR, 30

Inactive version of TAB

Rename TAB to QCMTAB

Delete indexes

QCMTAB

Field 1 NUMC, 6	Field 2 CHAR, 8	Field 3 CHAR, 60
000100 001200 003000	1111A00 0222B10 0030B20	Text1... Text2 ... Text3 ...

2

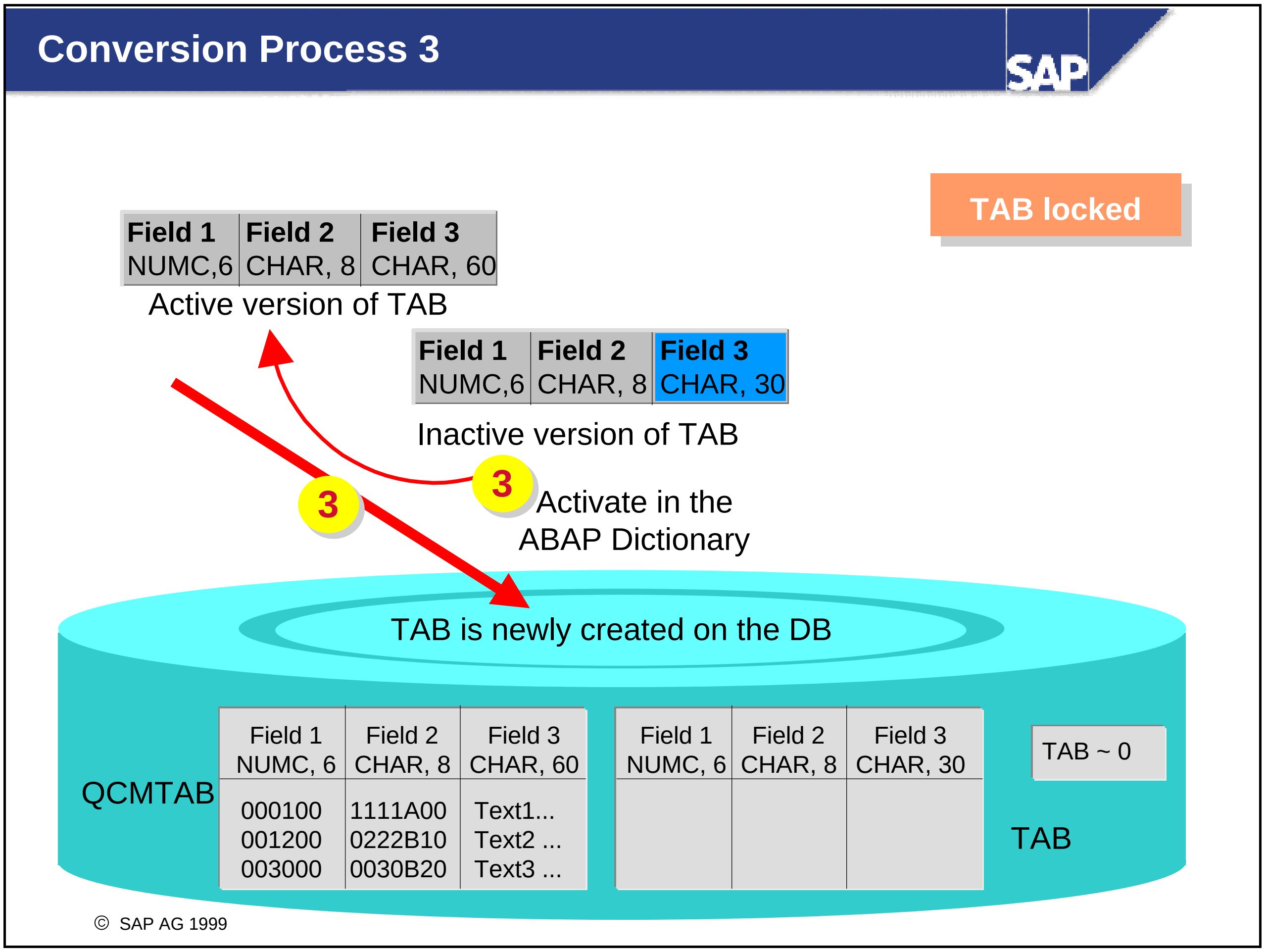
TAB

Field 1 NUMC, 6	Field 2 CHAR, 8	Field 3 CHAR, 60
000100 001200 003000	1111A00 0222B10 0030B20	Text1... Text2 ... Text3 ...

TAB ~ 0

TAB ~ A11

- **Step 1:** The table is locked against further structure changes. If the conversion terminates due to an error, the table remains locked. This lock mechanism prevents further structure changes from being made before the conversion has been completed correctly. Data could be lost in such a case.
- **Step 2:** The table in the database is renamed. All the indexes on the table are deleted. The name of the new (temporary) table is defined by the prefix QCM and the table name. The name of the temporary table for table TAB is therefore QCMTAB.



Conversion Process 3

TAB locked

Field 1	Field 2	Field 3
NUMC,6	CHAR, 8	CHAR, 60

Active version of TAB

Field 1	Field 2	Field 3
NUMC,6	CHAR, 8	CHAR, 30

Inactive version of TAB

3 Activate in the ABAP Dictionary

3

TAB is newly created on the DB

QCMTAB

Field 1 NUMC, 6	Field 2 CHAR, 8	Field 3 CHAR, 60
000100	1111A00	Text1...
001200	0222B10	Text2 ...
003000	0030B20	Text3 ...

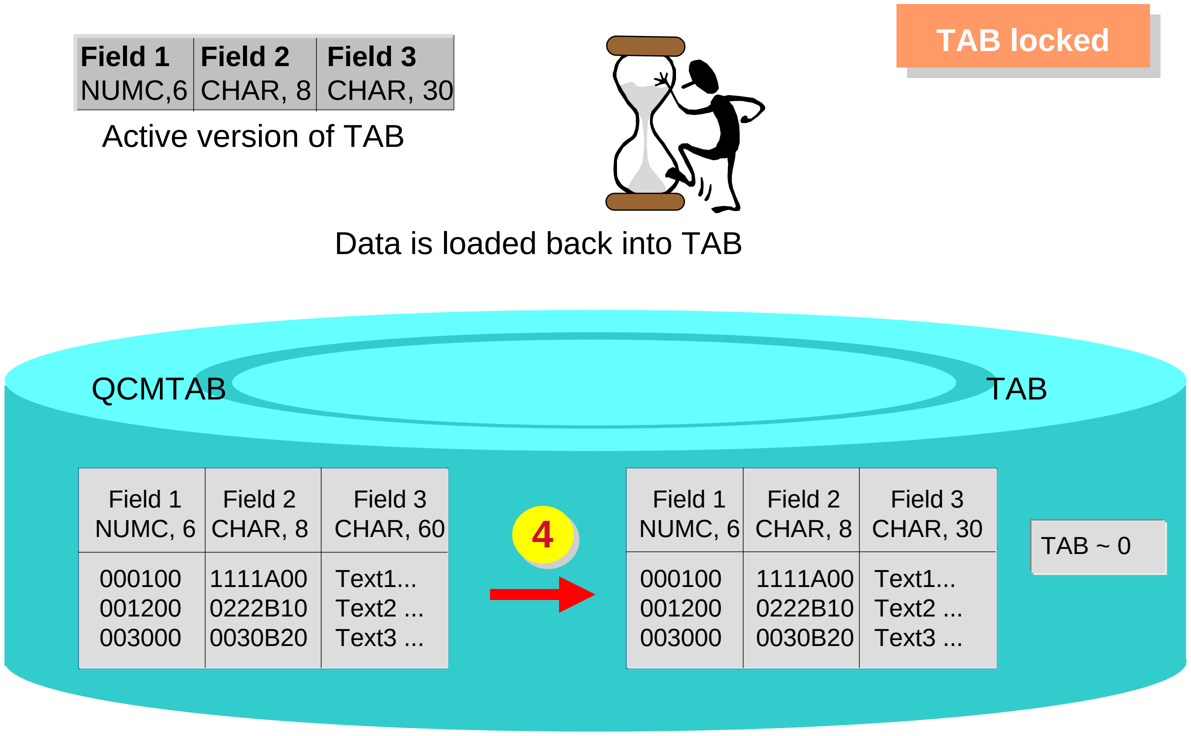
Field 1 NUMC, 6	Field 2 CHAR, 8	Field 3 CHAR, 30

TAB ~ 0

TAB

- **Step 3:** The inactive version of the table is activated in the ABAP Dictionary. The table is created on the database with its new structure and with the primary index. The structure of the database table is the same as the structure in the ABAP Dictinary after this step. The database table, however, does not contain any data.
- The system also tries to set a database lock for the table being converted. If the lock is set, application programs cannot write to the table during the conversion.
- The conversion is continued, however, even if the database lock cannot be set. In such a case application programs can write to the table. Since in such a case not all of the data might have been loaded back into the table, the table data might be inconsistent.
- **You should therefore always make sure that no applications access the table being converted during the conversion process.**

Conversion Process 4



TAB locked

Field 1	Field 2	Field 3
NUMC,6	CHAR, 8	CHAR, 30

Active version of TAB

Data is loaded back into TAB

QCMTAB

Field 1 NUMC, 6	Field 2 CHAR, 8	Field 3 CHAR, 60
000100 001200 003000	1111A00 0222B10 0030B20	Text1... Text2 ... Text3 ...

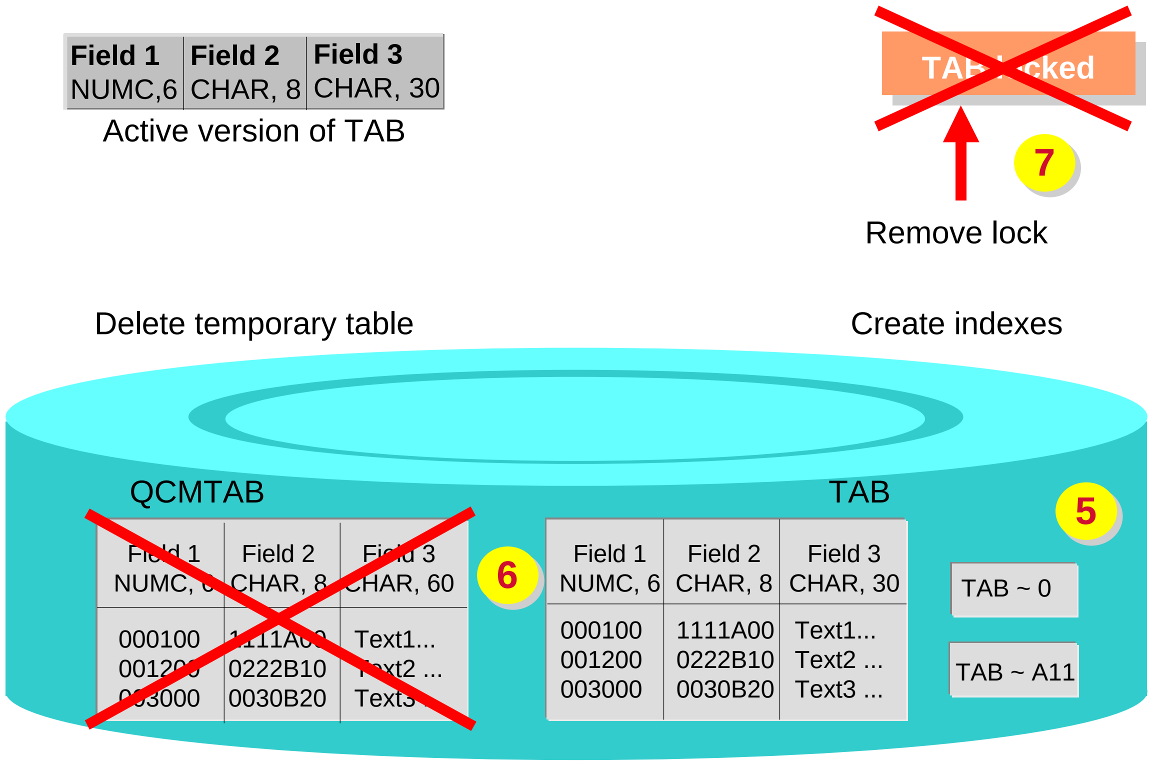
TAB

Field 1 NUMC, 6	Field 2 CHAR, 8	Field 3 CHAR, 30
000100 001200 003000	1111A00 0222B10 0030B20	Text1... Text2 ... Text3 ...

TAB ~ 0

- **Step 4:** The data is loaded back from the temporary table (QCM table) to the new table (with MOVE-CORRESPONDING). The data exists in the database table and in the temporary table after this step. When you reduce the size of fields, for example, the extra places are truncated when you reload the data.
- Since the data exists in both the original table and temporary table during the conversion, the storage requirements increase during the process. You should therefore verify that sufficient space is available in the corresponding tablespace before converting large tables.
- There is a database commit after 16 MB when you copy the data from the QCM table to the original table. A conversion process therefore needs 16 MB resources in the rollback segment. The existing database lock is released with the Commit and then requested again before the next data area to be converted is edited.
- When you reduce the size of keys, only one record can be reloaded if there are several records whose key cannot be distinguished. It is not possible to say which record this will be. In such a case you should clean up the data of the table before converting.

Conversion Process 5



Field 1	Field 2	Field 3
NUMC,6	CHAR, 8	CHAR, 30

Active version of TAB

~~TAB locked~~

Remove lock

Delete temporary table

Create indexes

QCMTAB

~~Field 1~~	~~Field 2~~	~~Field 3~~
~~NUMC, 6~~	~~CHAR, 8~~	~~CHAR, 60~~
~~000100~~ ~~001200~~ ~~003000~~	~~1111A00~~ ~~0222B10~~ ~~0030B20~~	~~Text1...~~ ~~Text2 ...~~ ~~Text3 ...~~

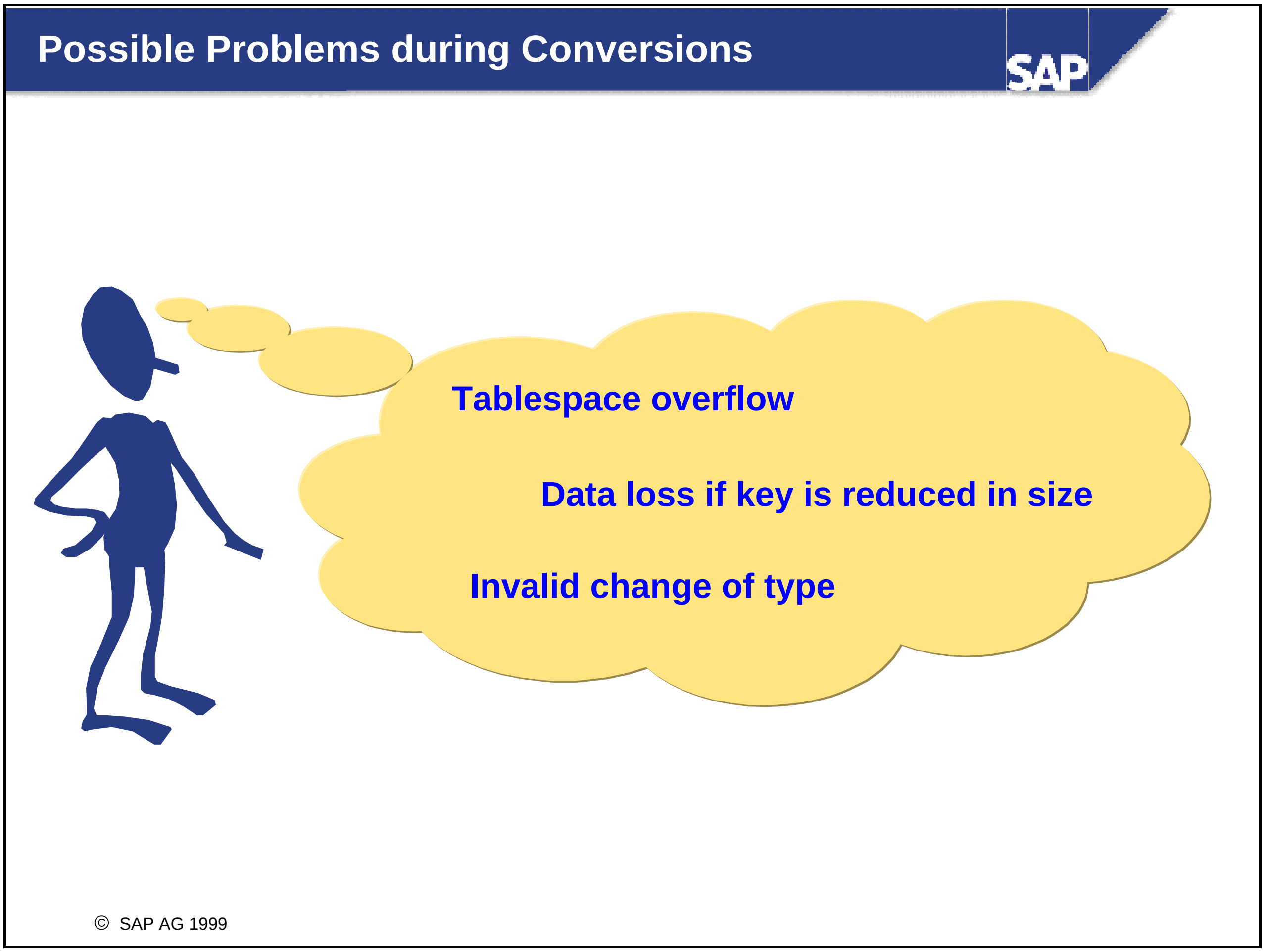
TAB

Field 1	Field 2	Field 3
NUMC, 6	CHAR, 8	CHAR, 30
000100 001200 003000	1111A00 0222B10 0030B20	Text1... Text2 ... Text3 ...

TAB ~ 0

TAB ~ A11

- **Step 5:** The secondary indexes defined in the ABAP Dictionary for the table are created again.
- **Step 6:** The temporary table (QCM table) is deleted.
- **Step 7:** The lock set at the beginning of the conversion is deleted.
- If the conversion terminates, the table remains locked and a restart log is written.
- **Caution:** The data of a table is not consistent during conversion. Programs therefore should not access the table during conversion. Otherwise a program could for example use incorrect data when reading the table since not all the records were copied back from the temporary table. **Conversions therefore should not run during production!** You must at least deactivate all the applications that use tables to be converted.
- **You must clean up terminated conversions.** Programs that access the table might otherwise run incorrectly. In this case you must find out why the conversion terminated (for example overflow of the corresponding tablespace) and correct it. Then continue the terminated conversion.



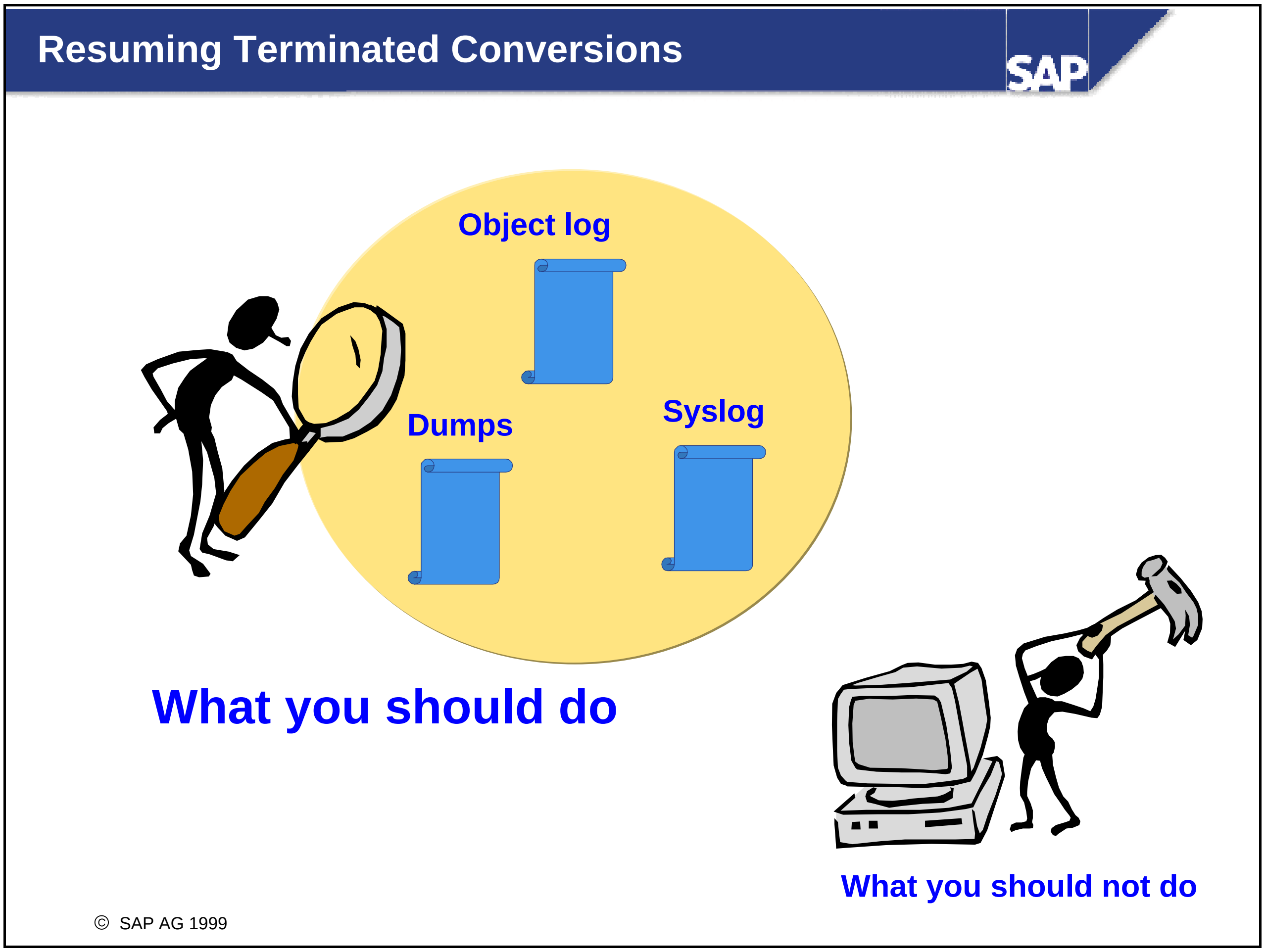
Possible Problems during Conversions

Tablespace overflow

Data loss if key is reduced in size

Invalid change of type

- Since the data exists in both the original table and temporary table during conversion, the storage requirements increase during conversion. If the tablespace overflows when you reload the data from the temporary table, the conversion will terminate. In this case you must extend the tablespace and start the conversion in the database utility again.
- If you shorten the key of a table (for example when you remove or shorten the field length of key fields), you cannot distinguish between the new keys of existing records of the table. When you reload the data from the temporary table, only one of these records can be loaded back into the table. It is not possible to say which record this will be. If you want to copy certain records, you have to clean up the table **before** the conversion.
- During a conversion, the data is copied back to the database table from the temporary table with the ABAP statement MOVE-CORRESPONDING. Therefore only those type changes that can be executed with MOVE-CORRESPONDING are allowed. All other type changes cause the conversion to be terminated when the data is loaded back into the original table. In this case you have to recreate the old state prior to conversion. Using database tools, you have to delete the table, rename the QCM table to its old name, reconstruct the runtime object (in the database utility), set the table structure in the Dictionary back to its old state and then activate the table.



Resuming Terminated Conversions

Object log

Dumps

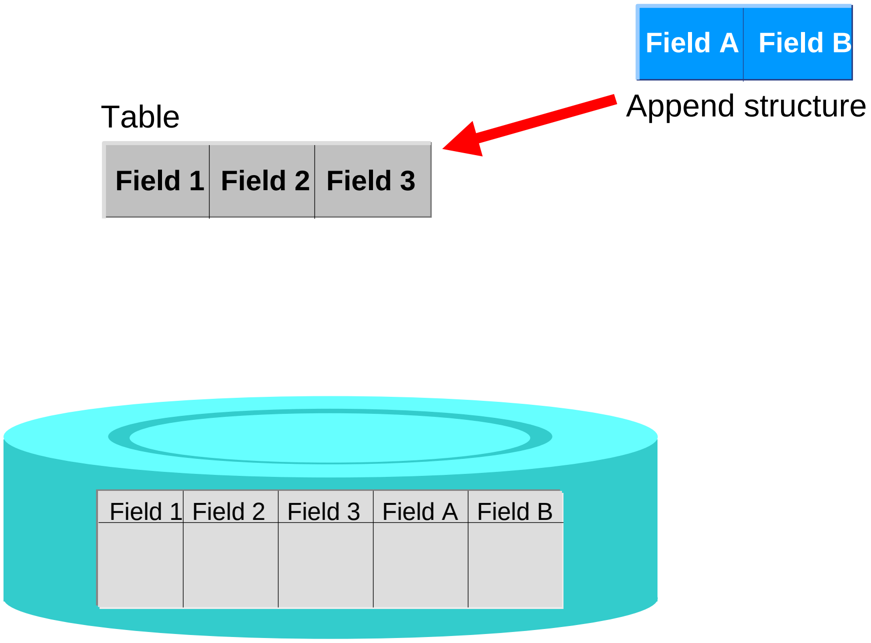
Syslog

What you should do

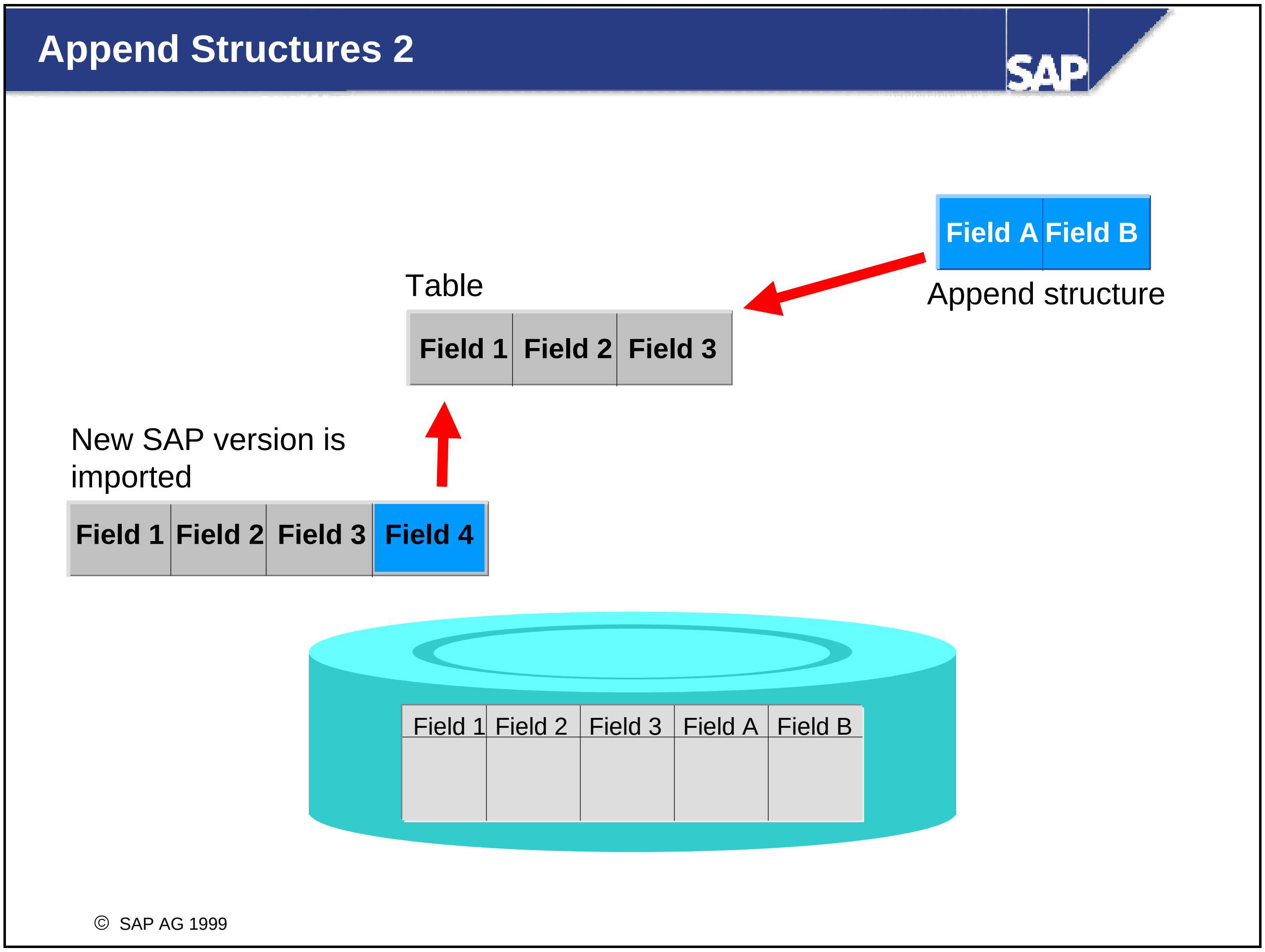
What you should not do

- If a conversion terminates, the lock entry for the table set in the first step is retained. The table can no longer be edited with the maintenance tools of the ABAP Dictionary (Transaction SE11).
- A terminated conversion can be analyzed with the database utility (Transaction SE14) and then resumed. The database utility provides an *analysis tool* with which you can find the cause of the error and the current state of all the tables involved in the conversion.
- You can usually find the precise reason for termination in the *object log*. If the object log does not provide any information about the cause of the error, you have to analyze the *syslog* or the *short dumps*.
- If there is a terminated conversion, two options are displayed as pushbuttons in the database utility:
  - After correcting the error, you can resume the conversion where it terminated with the *Continue adjustment* option.
  - There is also the *Unlock table* option. This option only deletes the existing lock entry for the table . You should **never** choose *Unlock table* for a terminated conversion if the data only exists in the temporary table, i.e. if the conversion terminated in steps 3 or 4.

Append Structures 1



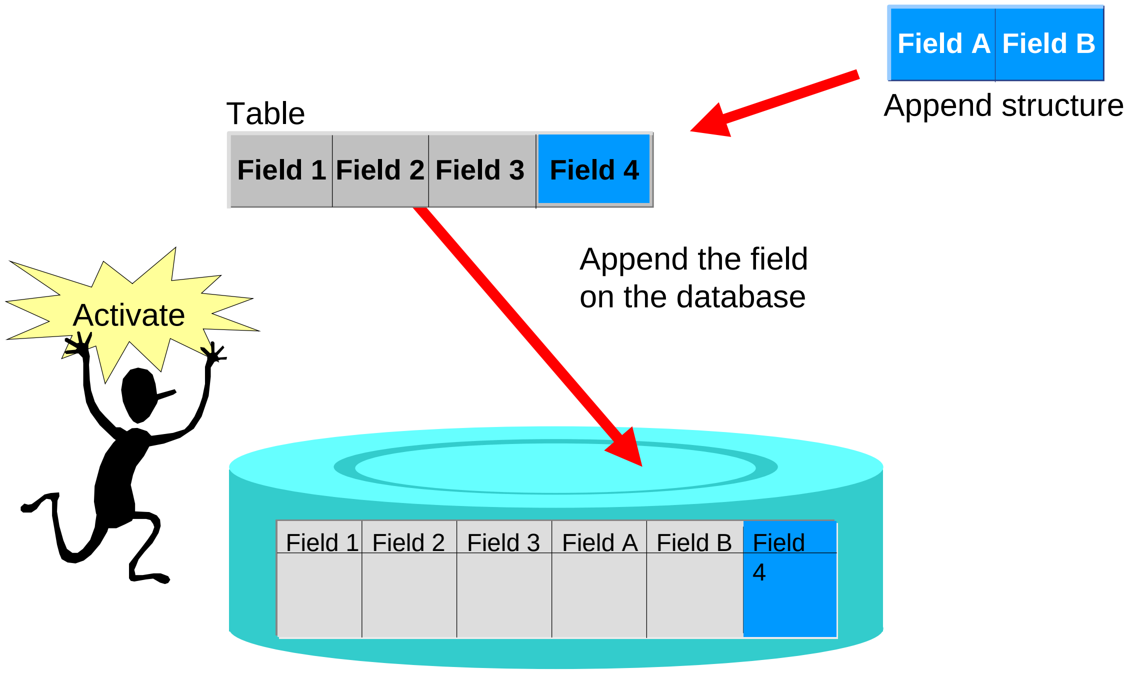
- Append structures permit you to append customer fields to a SAP standard table without having to modify the table definition.
- An append structure is a structure which is assigned to exactly one table. There can be several append structures for a table.
- When a table is activated, all the active append structures for the table are found and their fields are appended to the table. If an append structure is created or changed, the table to which it is assigned is also activated and the changes also take effect there when it is activated.
- Like all structures, an append structure defines a type that can be used in ABAP programs.



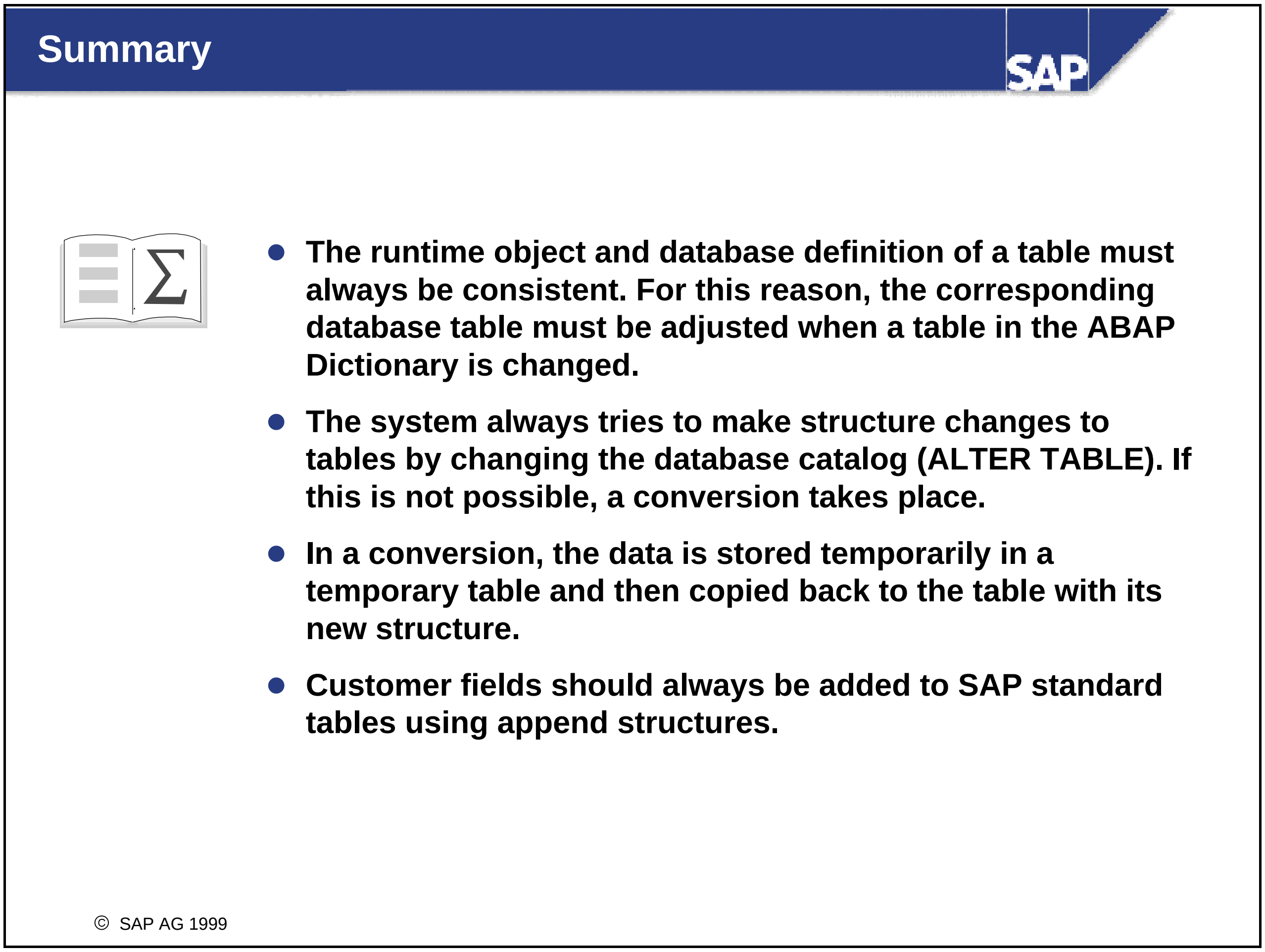
Append Structures 2

- Customers create append structures in their namespace. The append structures are thus protected against overwriting during an upgrade.
- The new versions of the standard tables are imported during the upgrade. When the standard tables are activated, the fields contained in the active append structures are appended to the new standard tables. When append structures are added to a table, you therefore do not have to manually adjust the customer modifications to the new SAP version of the table (Transaction SPDD) during the upgrade.
- The order of the fields in the ABAP Dictionary can differ from the order of the fields in the database. You therefore do not have to convert the table when you add an append structure or insert fields in an existing append structure. The new fields are simply appended to the table in the database. You can always adjust the structure by adjusting the database catalog (ALTER TABLE).

Append Structures 3



- The new version of the SAP standard table is activated and the new field is appended to the database table.
- Please note the following points about append structures:
  - No append structures may be created for pooled and cluster tables.
  - If a long field (data type LCHR or LRAW) occurs in a table, it cannot be extended with append structures. This is because such long fields must always be in the last position of the field list, i.e. they must be the last field of the table.
  - If you as a customer add an append structure to an SAP table, the fields in this append structure should be in the customer namespace for fields, that is they should begin with YY or ZZ. This prevents name collisions with new fields inserted in the standard table by SAP.
  - If you as a partner have your own reserved namespace for your developments, the fields you select in append structures should always lie in this namespace.



Summary

- The runtime object and database definition of a table must always be consistent. For this reason, the corresponding database table must be adjusted when a table in the ABAP Dictionary is changed.
- The system always tries to make structure changes to tables by changing the database catalog (ALTER TABLE). If this is not possible, a conversion takes place.
- In a conversion, the data is stored temporarily in a temporary table and then copied back to the table with its new structure.
- Customer fields should always be added to SAP standard tables using append structures.

Exercises: Changes to Database Tables

Unit: Changes to Database Tables

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

- Make changes to existing objects
- Convert tables
- Enhance standard tables with append structures without modifying them

The original design of employee management is no longer appropriate after some organizational changes at the airlines. It is therefore necessary to make small changes to objects already created in previous exercises. These changes will be made in this exercise.

6-1 The design of table ZFLCREWxx is no longer appropriate. The field for the role of the employee during the flight is too long. Correct this error by reducing the field length to 15 places.

Create a new data element ZROLExx and replace the existing data element with the new one. When you define data element ZROLExx, do not use a domain; instead enter the data type and length directly when you define the data element. Activate the table.

6-2 The employees with management or maintenance functions have their workplace at an airport. They can be reached at this airport using a telephone number. Also record where the administrative employees work (office). Include this information in table ZEMPLOYxx.

Create an append structure for table ZEMPLOYxx containing the following information:

Field	Data element
Airport	*S_AIRPORT*
Office	S_BUREAUNO
Telephone	S_TELNO

Note that the field names in an append structure must lie in the customer namespace for fields. The field names should therefore begin with ZZ or YY.

6-3 Create a suitable foreign key for field *Airport* of the append structure.

The table of all airports is called SAIRPORT.

For a complete definition of the foreign key you must also define a field of the appending table (ZEMPLOYxx).

Start Program BC430_CHECK with Transaction SE38 after this exercise. The program checks whether your solutions are correct.

Solutions: Changes to Database Tables

Unit: Changes to Database Tables

6-1 Go to change mode in the maintenance screen for table ZFLCREWxx. Take the following steps.

1) Overwrite data element SEMP_ROLE in column *Field type* with the name of your data element ZROLExx. Save the change.
2) Double-click on the name ZROLExx. In the next dialog box, confirm that you want to create the data element.
3) The maintenance screen for data elements appears. Enter a short text.
4) Click on tab page *Definition*. Mark *Built-in type*. You can now enter values for fields *Datatype*, *Length* and *Decplaces*. Enter CHAR as data type and 15 as length.
5) On tab page *Field label*, maintain the text for the data element.
6) Activate the data element.
7) You go to the activation log. You are reminded that shortening the field requires a conversion of table ZFLCREWxx.
8) Go back to the table maintenance screen.
9) Navigate to the database utility with *Utilities* → *Database utility*.
10) Choose *Activate and adjust database*. Confirm the next security query. The system converts the table.

6-2 To define the append structure for table ZEMPLOYxx:

1) Go to display mode in the maintenance screen for table ZEMPLOYxx.
2) Choose *Goto* ? *Append structure*.
3) In the next dialog box, enter the required name for the append structure. It must satisfy the usual naming conventions. Choose *Continue*.
4) The maintenance screen for the append structure is displayed. Maintenance of the append structure is analogous to that for a structure.
5) Enter a short text and insert fields *Airport*, *Office* and *Telephone* with the specified data elements.
6) Note that all the fields of an append structure must lie in the customer namespace. The field names therefore must begin with ZZ or YY.
7) Activate the append structure. Display the activation log with *Goto* ? *Activation log*.

Table ZEMPLOYxx is automatically adjusted when the append structure is activated. The new fields are appended to the existing fields in the database.

6-3 The foreign key must be defined in the append structure maintenance routine. You can go there either by entering its name as data type in the initial screen of the ABAP Dictionary and choosing *Change* or with *Goto* → *Append structure* in the maintenance screen for table ZEMPLOYxx. Take the following steps:

1) Place the cursor on field *Airport*. Choose the key icon. Copy the system proposal.
2) In a dialog box you are informed that the foreign key is not completely specified. Choose *Continue*.

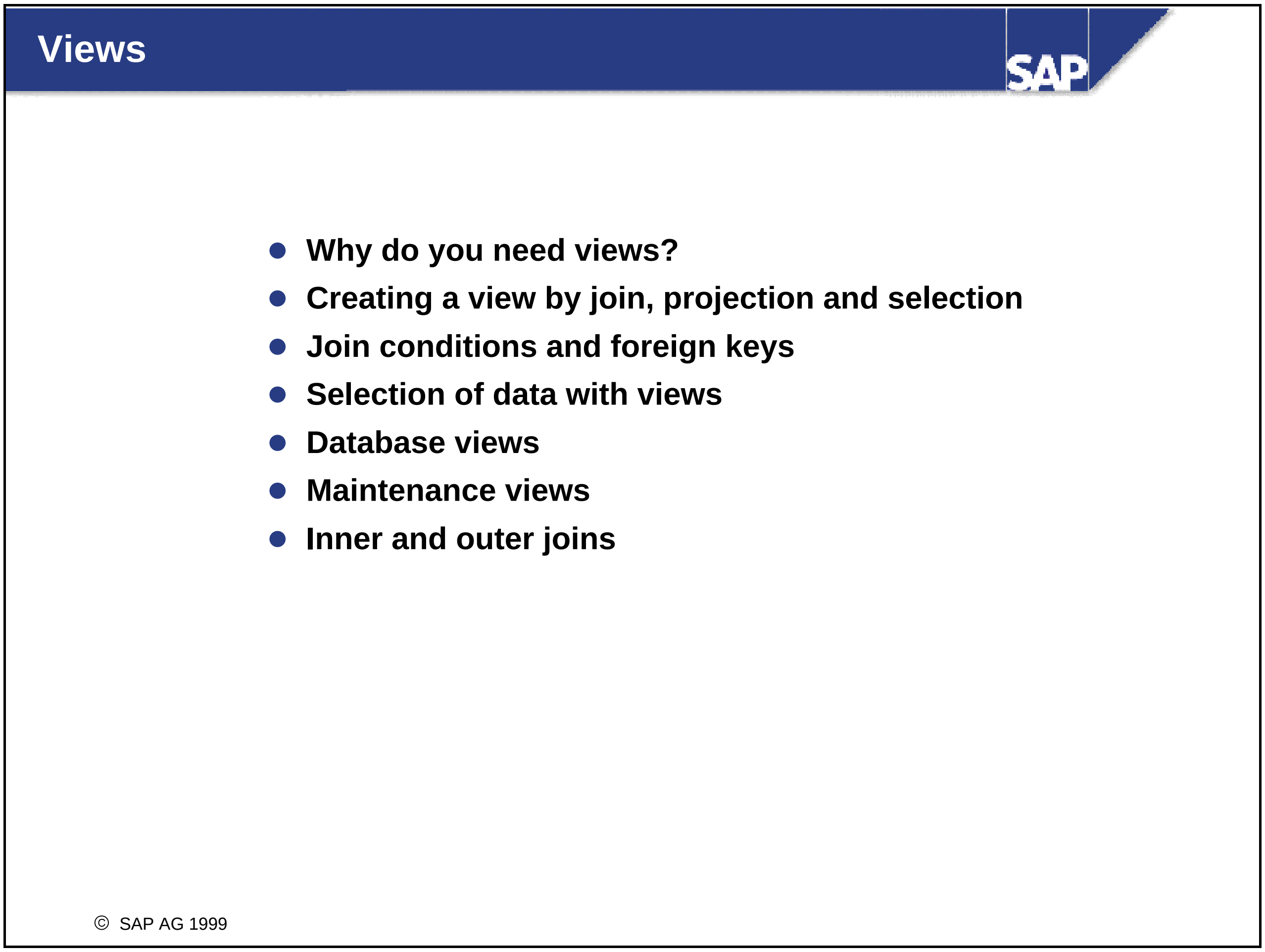
   The foreign key cannot be fully specified in the append structure since the key of check table SAIRPORT contains the client field and the field for the airport code, but the append structure does not contain the client field.

3) The maintenance screen for the foreign key appears. The check table and the field assignment are already filled due to the system proposal. Enter a suitable *Short description*.

4) You now have to complete the field assignment. Remove the marking for generic mapping. Then enter ZEMPLOYxx in column *For. key table* and the name of the client field of this table in column *For. key field*.

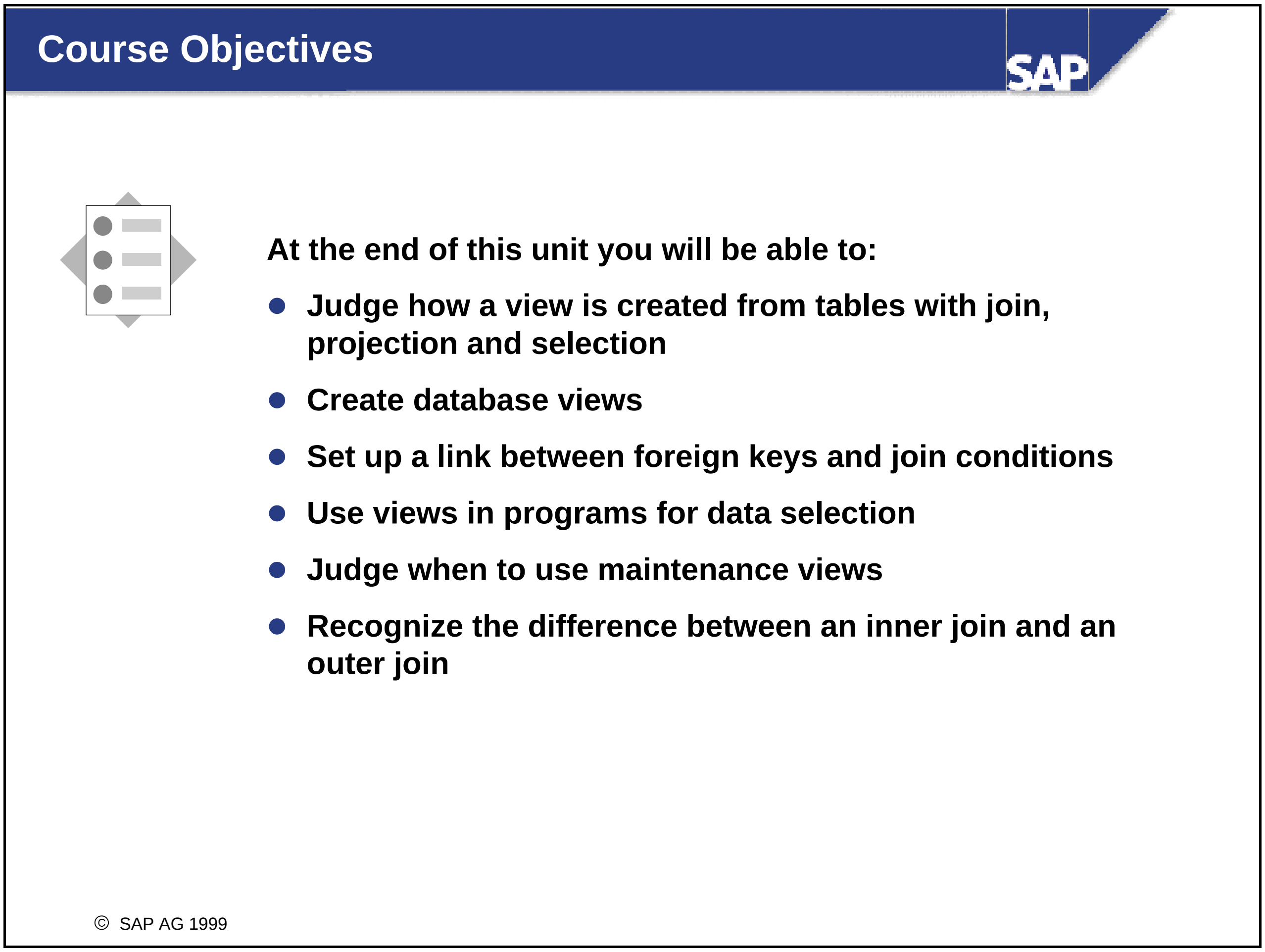
5) Choose *Non-key-fields/candidates* as the type of foreign key fields (since the Airport field is not a key field of table ZEMPLOYxx) and as cardinality *C* (since not every employee is assigned to an airport) to *CN* (since several employees can be assigned to the same airport).

6) Choose *Copy* and activate the append structure.



Views

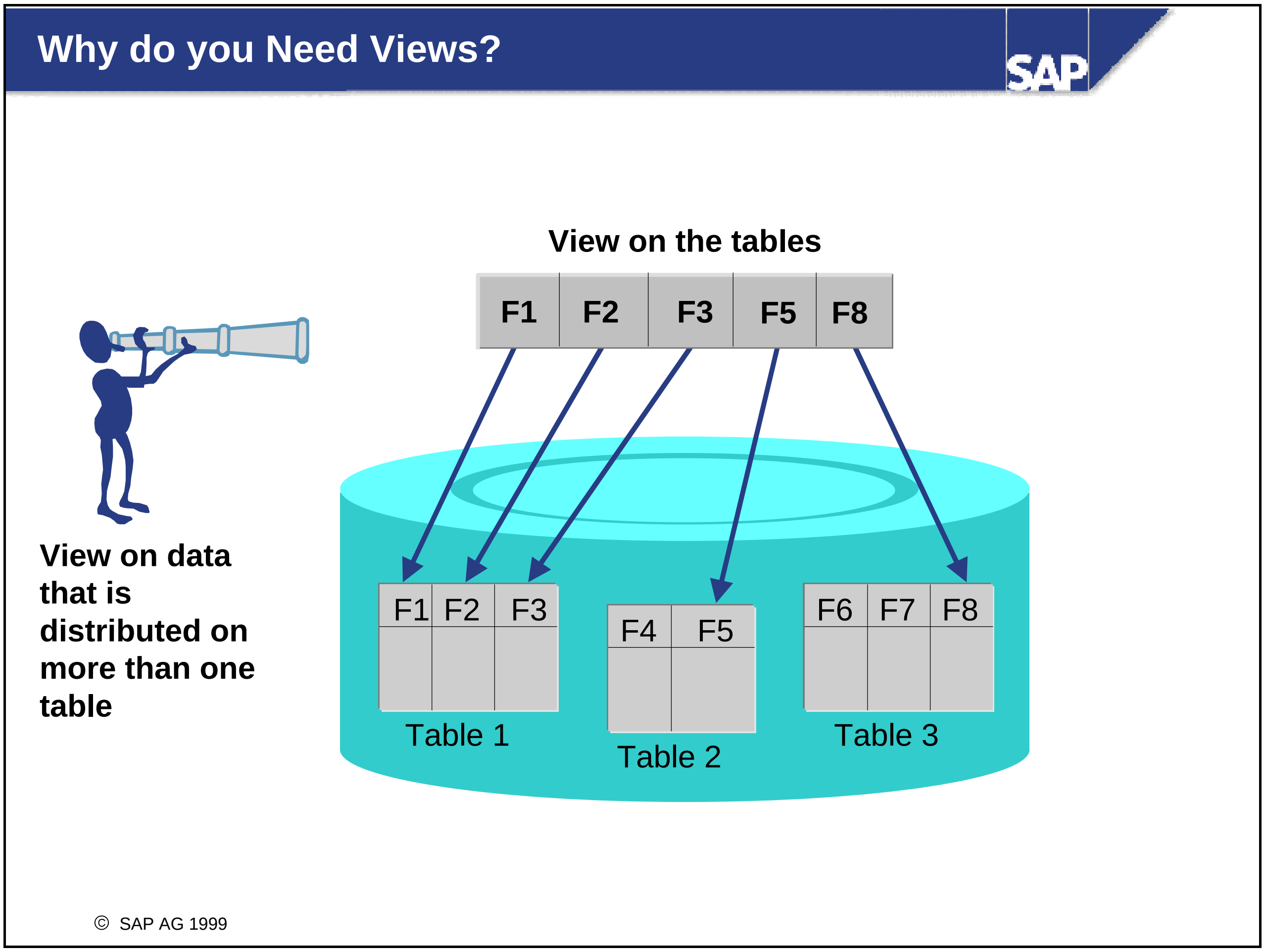
- Why do you need views?
- Creating a view by join, projection and selection
- Join conditions and foreign keys
- Selection of data with views
- Database views
- Maintenance views
- Inner and outer joins



Course Objectives

At the end of this unit you will be able to:

- Judge how a view is created from tables with join, projection and selection
- Create database views
- Set up a link between foreign keys and join conditions
- Use views in programs for data selection
- Judge when to use maintenance views
- Recognize the difference between an inner join and an outer join

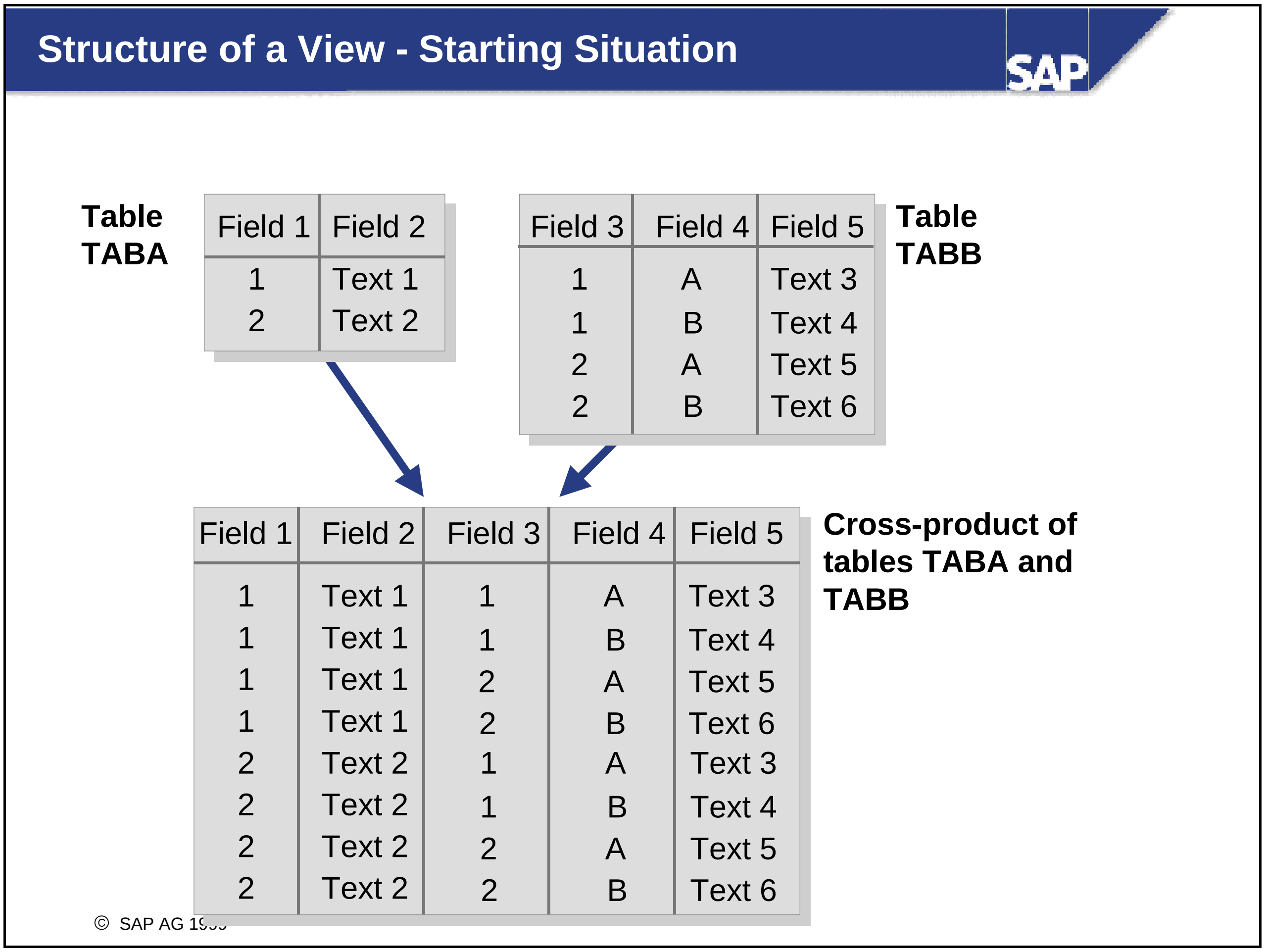


Why do you Need Views?

View on data that is distributed on more than one table

© SAP AG 1999

- Data for an application object is often distributed on several database tables. Database systems therefore provide you with a way of defining application-specific views on data in several tables. These are called views.
- Data from several tables can be combined in a meaningful way using a view (join). You can also hide information that is of no interest to you (projection) or only display those data records that satisfy certain conditions (selection).
- The data of a view can be displayed exactly like the data of a table in the extended table maintenance.



Structure of a View - Starting Situation

Table TABA

Field 1	Field 2
1	Text 1
2	Text 2

Table TABB

Field 3	Field 4	Field 5
1	A	Text 3
1	B	Text 4
2	A	Text 5
2	B	Text 6

Cross-product of tables TABA and TABB

Field 1	Field 2	Field 3	Field 4	Field 5
1	Text 1	1	A	Text 3
1	Text 1	1	B	Text 4
1	Text 1	2	A	Text 5
1	Text 1	2	B	Text 6
2	Text 2	1	A	Text 3
2	Text 2	1	B	Text 4
2	Text 2	2	A	Text 5
2	Text 2	2	B	Text 6

- The structure of a view and selection of the data using this view will be shown with an example.
- Given two tables TABA and TABB. Table TABA contains 2 entries and table TABB 4 entries.
- The tables are first appended to one another. This results in the cross-product of the two tables, in which each record of TABA is combined with each record of TABB.

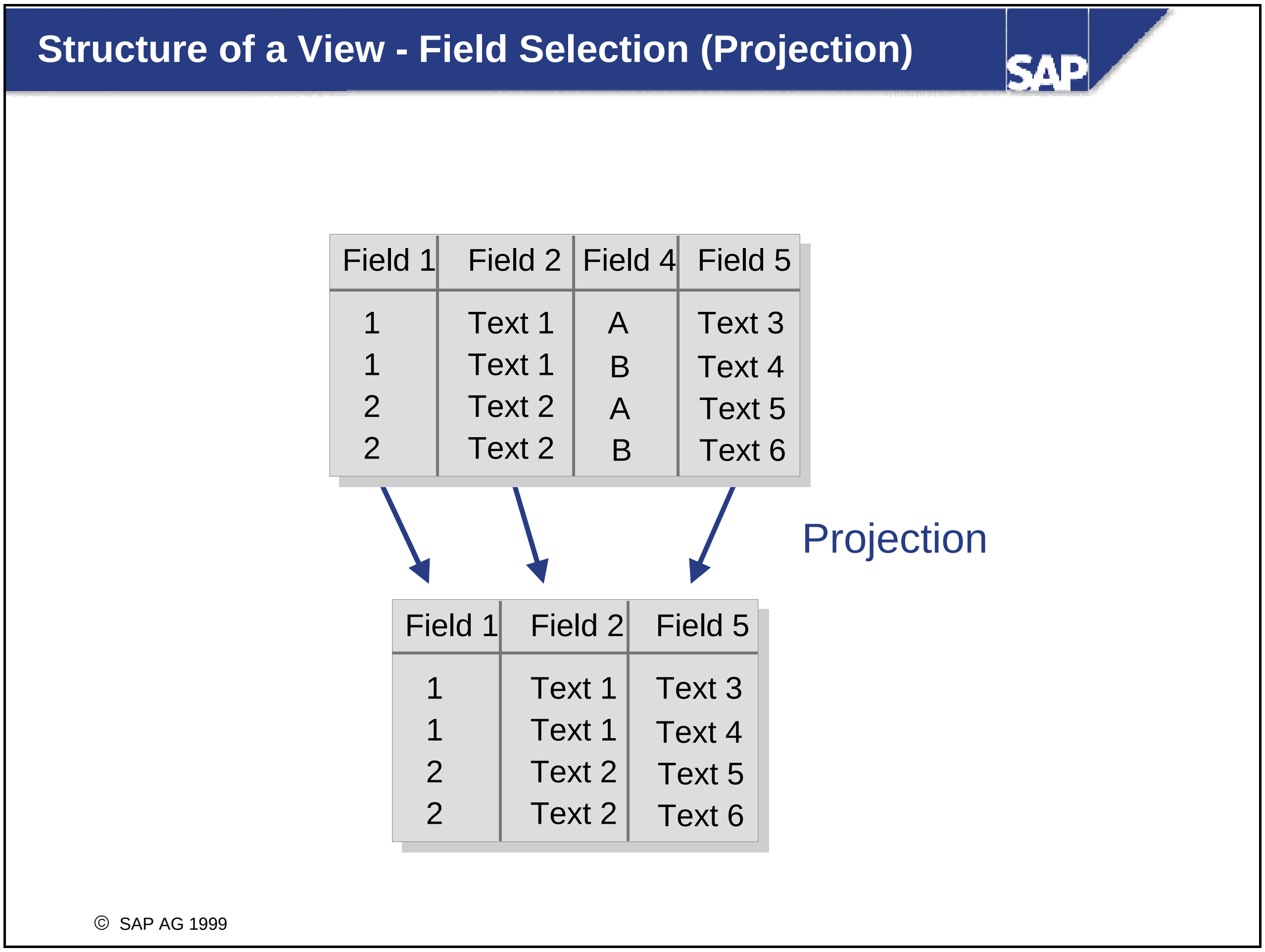
Structure of a View - Join Condition

Join condition: TABA - Field 1 = TABB - Field 3

Reduction of the cross-product

Field 1	Field 2	~~Field 3~~	Field 4	Field 5
1	Text 1	~~1~~	A	Text 3
1	Text 1	~~1~~	B	Text 4
~~1~~	~~Text 1~~	~~2~~	~~A~~	~~Text 5~~
~~1~~	~~Text 1~~	~~2~~	~~B~~	~~Text 6~~
~~2~~	~~Text 2~~	~~1~~	~~A~~	~~Text 3~~
~~2~~	~~Text 2~~	~~1~~	~~B~~	~~Text 4~~
2	Text 2	~~2~~	A	Text 5
2	Text 2	~~2~~	B	Text 6

- Usually the entire cross-product is not a meaningful selection. You should therefore limit the cross-product with a join condition. The join condition describes how the records of the two tables are related.
- In our example, Field 3 of TABB identifies Field 1 of TABA. The join condition is then:
  TABA - Field 1 = TABB - Field 3
- With this join condition, all the records whose entry in Field 1 is not identical to the entry in Field 3 are removed from the cross product. The column for Field 3 in the view is therefore unnecessary.



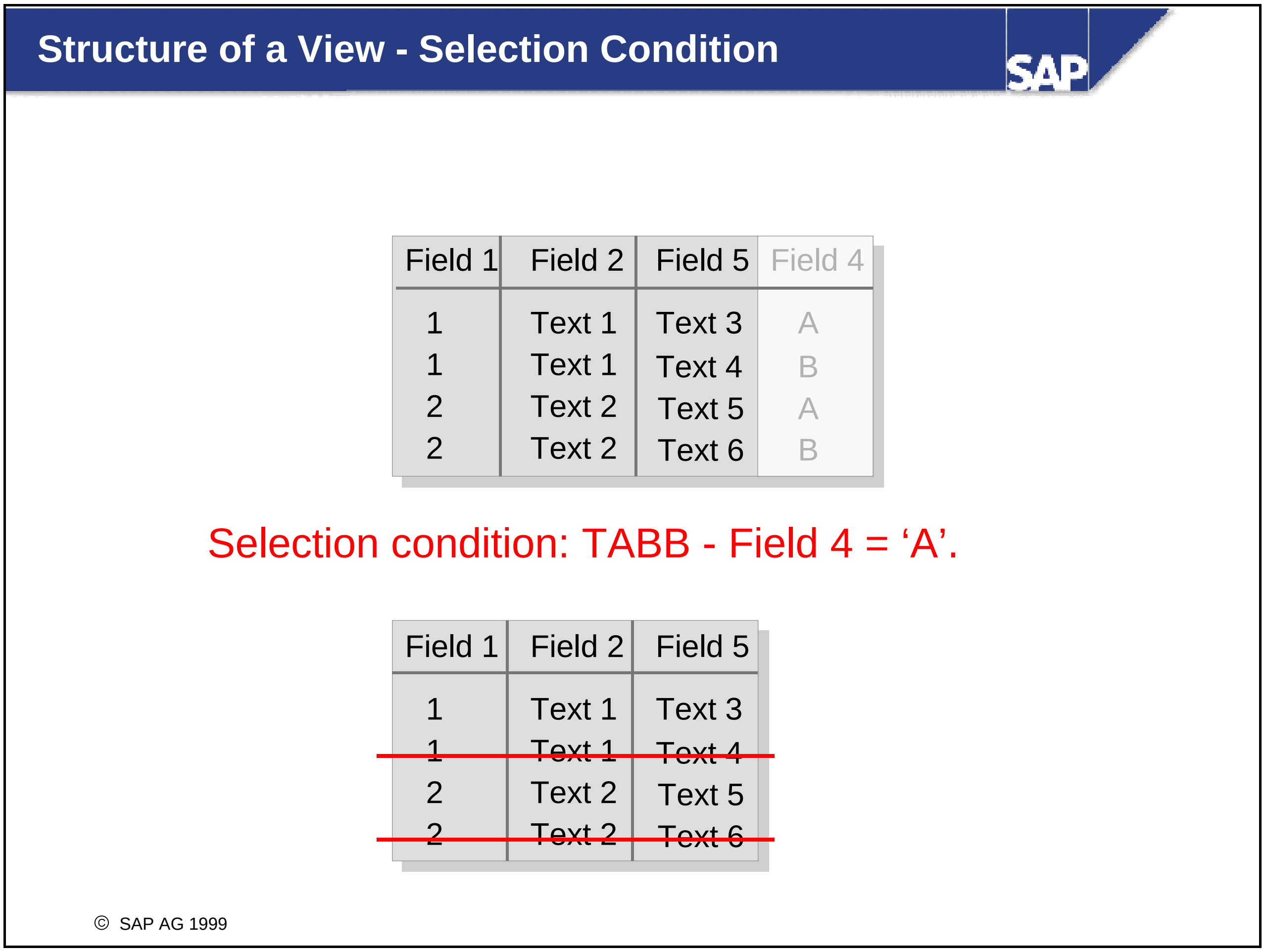
Structure of a View - Field Selection (Projection)

Field 1	Field 2	Field 4	Field 5
1	Text 1	A	Text 3
1	Text 1	B	Text 4
2	Text 2	A	Text 5
2	Text 2	B	Text 6

Projection

Field 1	Field 2	Field 5
1	Text 1	Text 3
1	Text 1	Text 4
2	Text 2	Text 5
2	Text 2	Text 6

- Often some of the fields of the tables involved in a view are of no interest. You can explicitly define the set of fields to be included in the view (projection).
- In our example, Field 4 is of no interest and can therefore be hidden.



Structure of a View - Selection Condition

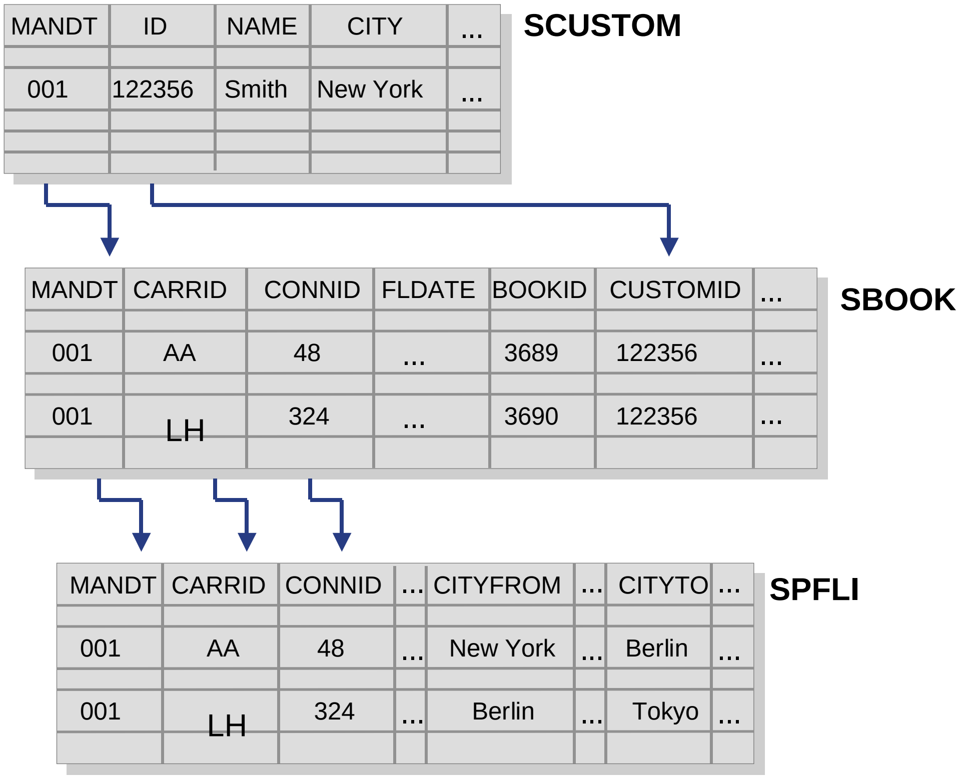
Field 1	Field 2	Field 5	Field 4
1	Text 1	Text 3	A
1	Text 1	Text 4	B
2	Text 2	Text 5	A
2	Text 2	Text 6	B

Selection condition: TABB - Field 4 = ‘A’.

Field 1	Field 2	Field 5
1	Text 1	Text 3
~~1~~	~~Text 1~~	~~Text 4~~
2	Text 2	Text 5
~~2~~	~~Text 2~~	~~Text 6~~

- The set of records that can be displayed with the view can be further restricted with a selection condition.
- In our example, only those records with value 'A' in Field 4 should be displayed with the view.
- A selection condition therefore can also be formulated with a field that is not contained in the view.

How are Tables Linked to Views?



SCUSTOM

MANDT	ID	NAME	CITY	...
001	122356	Smith	New York	...

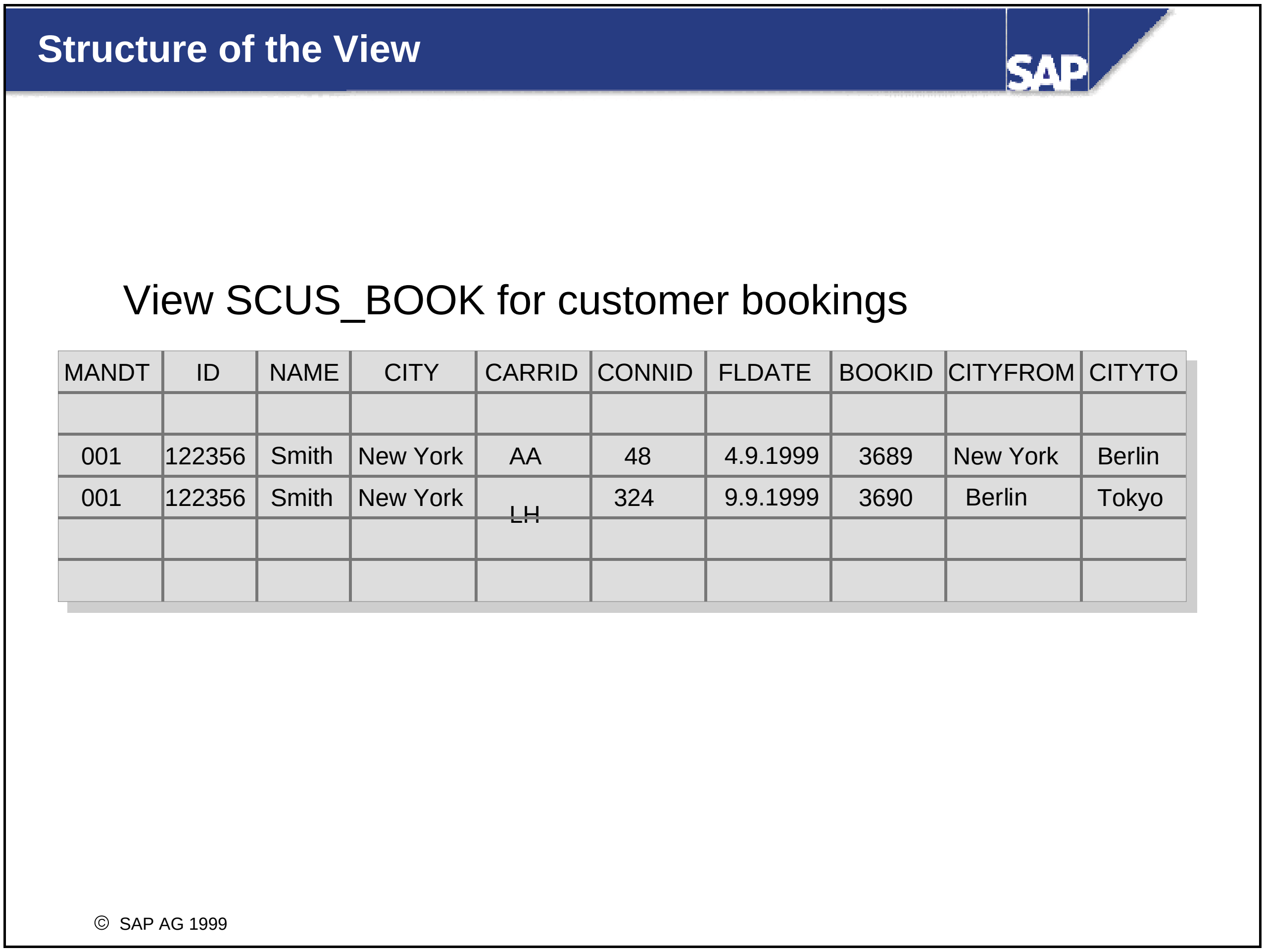
SBOOK

MANDT	CARRID	CONNID	FLDATE	BOOKID	CUSTOMID	...
001	AA	48	...	3689	122356	...
001	LH	324	...	3690	122356	...

SPFLI

MANDT	CARRID	CONNID	...	CITYFROM	...	CITYTO	...
001	AA	48	...	New York	...	Berlin	...
001	LH	324	...	Berlin	...	Tokyo	...

- **Example:** Travel agencies sometimes have to check which customer is booked on which flights. The corresponding data is distributed on several tables:
  - SCUSTOM: Customer data, such as the customer number, name and address
  - SBOOK: Booking data, such as the carrier, flight number and passenger (customer number)
  - SPFLI: Flight data, such as the city of departure and city of arrival
- You have to create a view on tables SCUSTOM, SBOOK and SPFLI to obtain the booking data.
- In this case the join conditions are:
  - SBOOK-MANDT = SCUSTOM-MANDT
  - SBOOK-CUSTOMID = SCUSTOM-ID
  - SPFLI-MANDT = SBOOK-MANDT
  - SPFLI-CARRID = SBOOK-CARRID
  - SPFLI-CONNID = SBOOK-CONNID



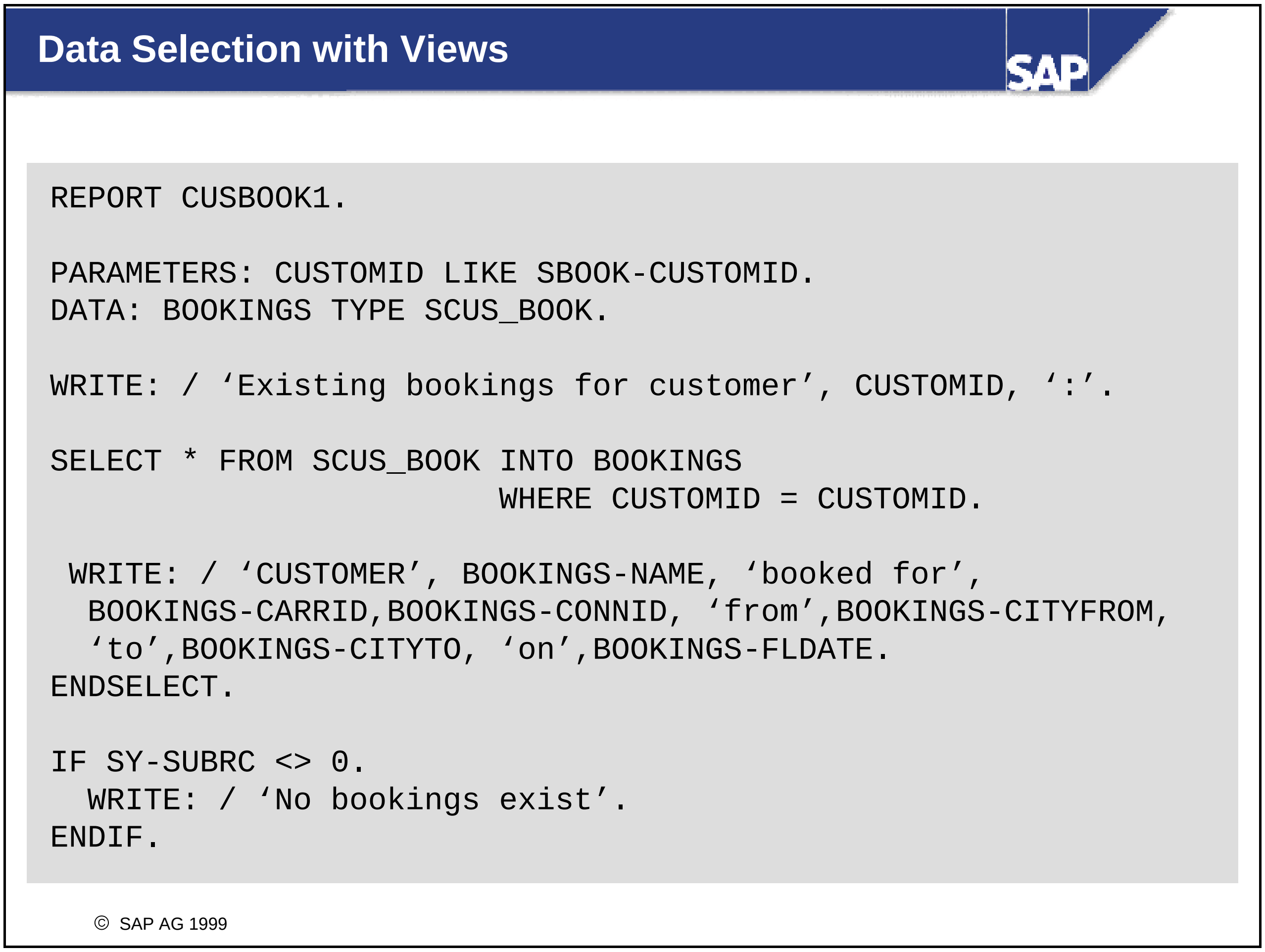
Structure of the View

View SCUS_BOOK for customer bookings

MANDT	ID	NAME	CITY	CARRID	CONNID	FLDATE	BOOKID	CITYFROM	CITYTO
001	122356	Smith	New York	AA	48	4.9.1999	3689	New York	Berlin
001	122356	Smith	New York	LH	324	9.9.1999	3690	Berlin	Tokyo

- You can get the bookings for a particular customer by selecting the corresponding records for keys MANDT and CUSTOMID in table SBOOK.
- You can get the flight data from table SPFLI for each booking in table SBOOK by selecting the corresponding record for the keys MANDT, CARRID and CONNID from table SPFLI.
- You can display only the customer bookings which were not canceled using the view with the following selection condition:

  SBOOK-CANCELED <> 'X'
- The join conditions can also be derived from the existing foreign key relationships. Copying the join conditions from the existing foreign keys is supported in the maintenance transaction.
- The field names of the underlying table fields are normally used as field names in the view. However, you can also choose a different field name. This is necessary for instance if two fields with the same name are to be copied to the view from different tables. In this case you must choose a different name for one of the two fields in the view.



Data Selection with Views

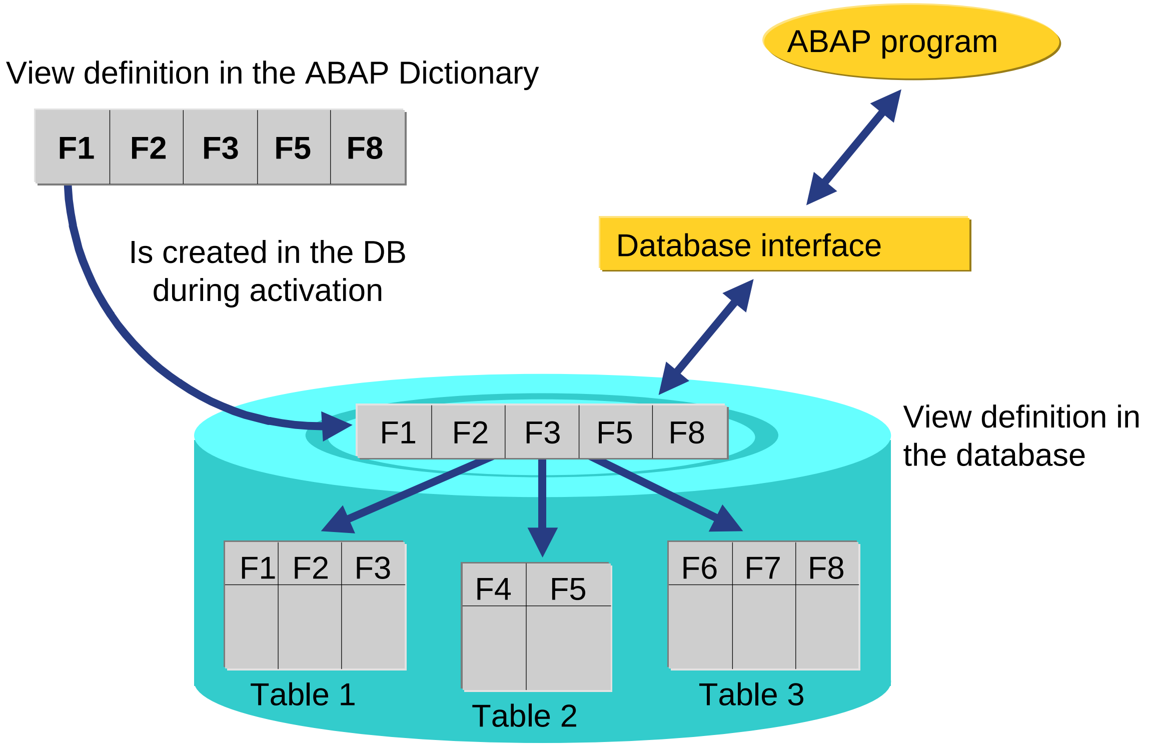
```
REPORT CUSBOOK1.

PARAMETERS: CUSTOMID LIKE SBOOK-CUSTOMID.
DATA: BOOKINGS TYPE SCUS_BOOK.

WRITE: / ‘Existing bookings for customer’, CUSTOMID, ‘:’.

SELECT * FROM SCUS_BOOK INTO BOOKINGS
                        WHERE CUSTOMID = CUSTOMID.

 WRITE: / ‘CUSTOMER’, BOOKINGS-NAME, ‘booked for’,
  BOOKINGS-CARRID,BOOKINGS-CONNID, ‘from’,BOOKINGS-CITYFROM,
  ‘to’,BOOKINGS-CITYTO, ‘on’,BOOKINGS-FLDATE.
ENDSELECT.

IF SY-SUBRC <> 0.
  WRITE: / ‘No bookings exist’.
ENDIF.
```

- You would get the same results using nested SELECT statements:

```
SELECT * FROM SCUSTOM WHERE ID = CUSTOMID.
  SELECT * FROM SBOOK WHERE CUSTOMID = SCUSTOM-ID.
    SELECT * FROM SPFLI WHERE CARRID = SBOOK-CARRID AND
                    CONNID = SBOOK-CONNID
    WRITE: / ‘Customer’, SCUSTOM-NAME, ‘booked on’, SPFLI-CARRID, SPFLI-CONNID, ‘from’,
    SPFLI-CITYFROM, ‘to’, SPFLI-CITYTO, ‘on’, SBOOK-FLDATE.
    ENDSELECT.
  ENDSELECT.
ENDSELECT.
```

- Selection with a database view, however, is usually more efficient than selection with a nested SELECT statement.
- As of Release 4.0 you can formulate the join condition directly in OPEN SQL.
- A view has type character and can be accessed in programs like all other types and can be used to define data objects.

Database Views



ABAP program

View definition in the ABAP Dictionary

F1	F2	F3	F5	F8

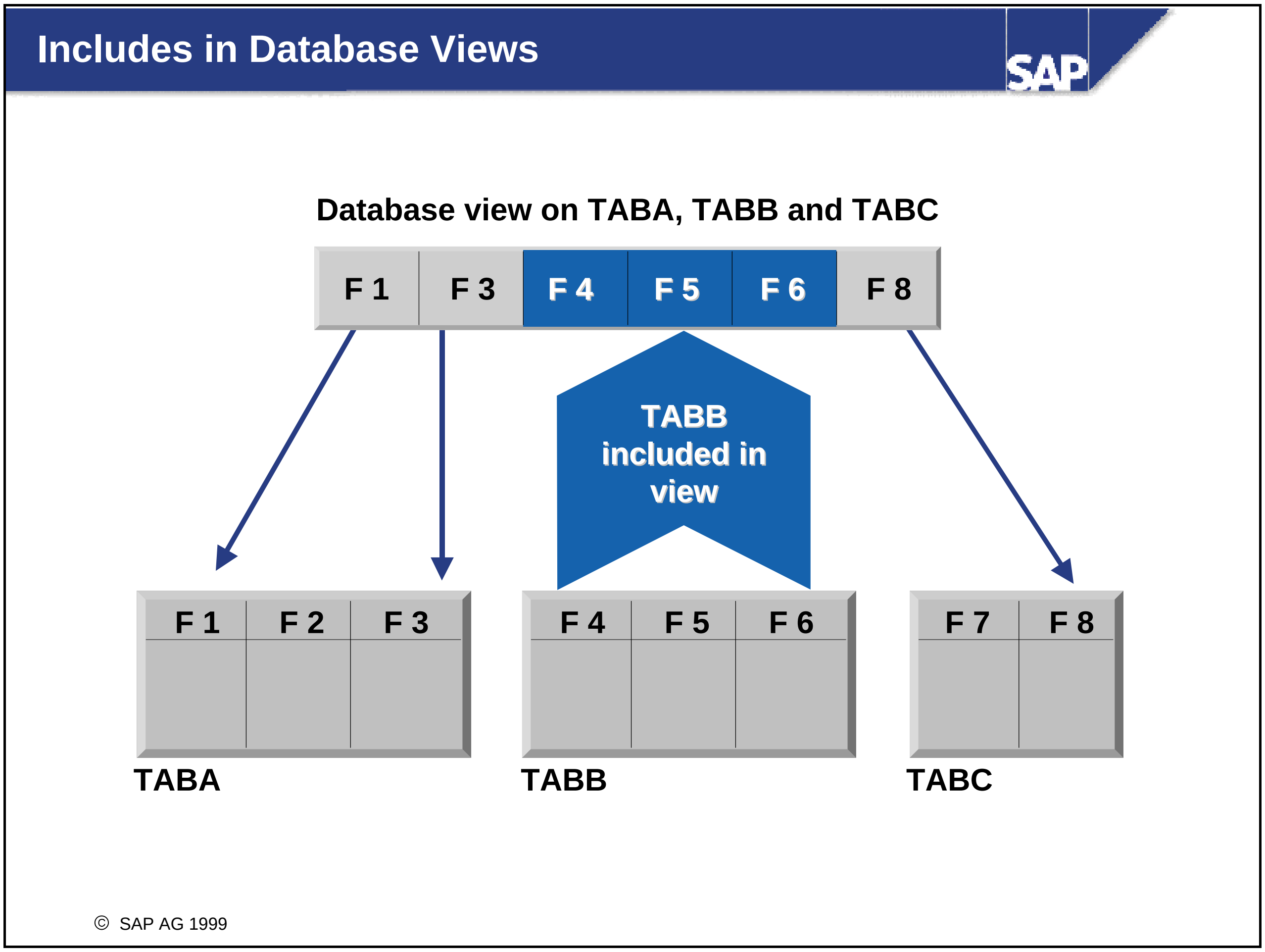
Is created in the DB during activation

Database interface

F1	F2	F3	F5	F8

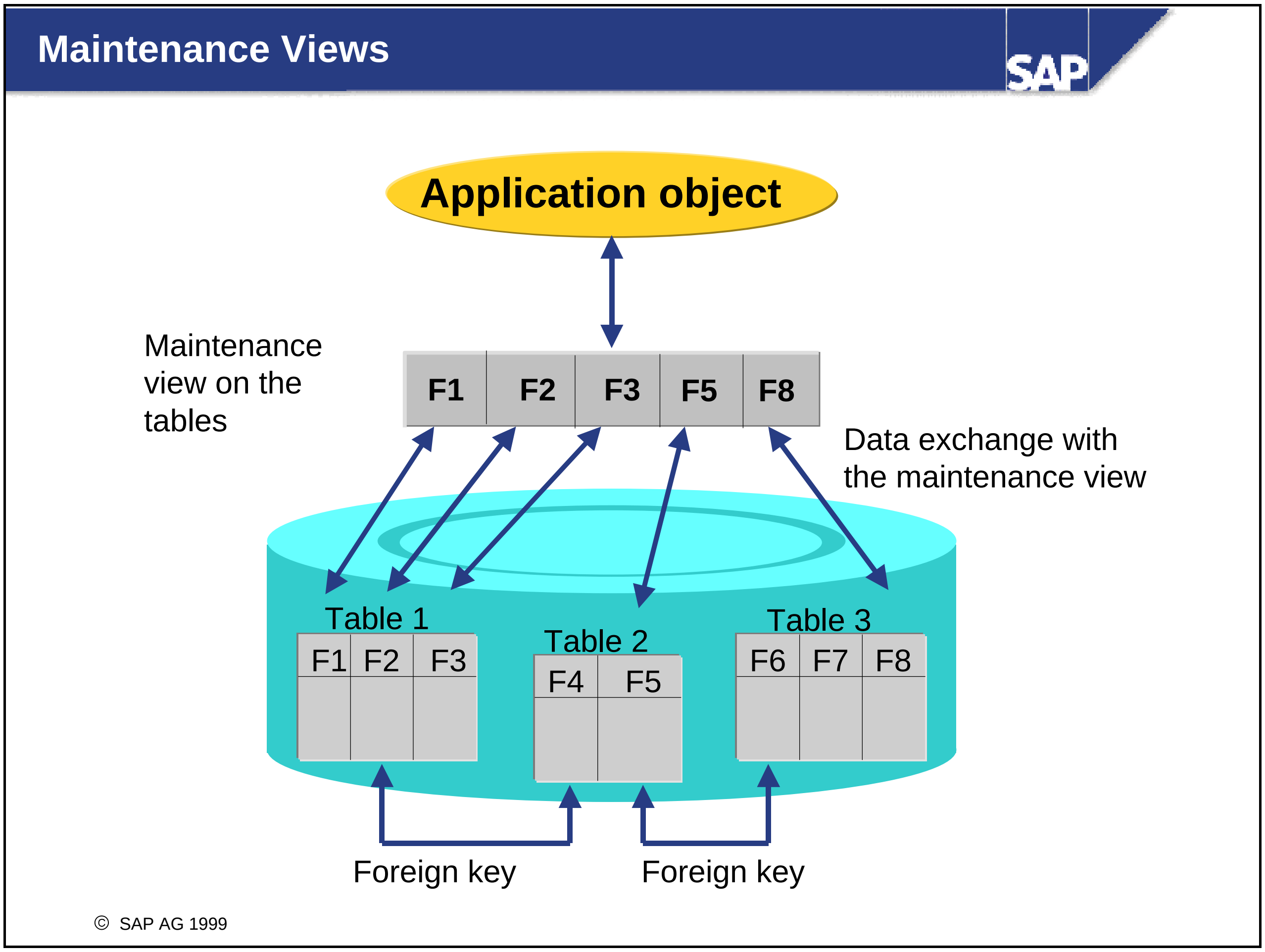
View definition in the database

- A database view is defined in the ABAP Dictionary and automatically created on the database during activation. Accesses to a database view are passed directly to the database from the database interface. The database software performs the data selection.
- If the definition of a database view is changed in the ABAP Dictionary, the view created on the database must be adjusted to this change. Since a view does not contain any data, this adjustment is made by deleting the old view definition and creating the view again in the ABAP Dictionary with its new definition.
- The maintenance status defines whether you can only read with the view or whether you can also write with it. If a database view was defined with more than one table, this view must be read only.
- The data read with a database view can be buffered. View data is buffered analogously to tables. The technical settings of a database view control whether the view data may be buffered and how this should be done. The same settings (buffering types) can be used here as for table buffering. The buffered view data is invalidated when the data in one of the base tables of the view changes.



Includes in Database Views

- You can include entire tables in database views. In this case all the fields of the included table become fields of the view (whereby you can explicitly exclude certain fields). If new fields are included in the table or existing fields are deleted, the view is automatically adjusted to this change. A new or deleted field is therefore automatically included in the view or deleted from it.
- If an append structure is added to a table included in a view, the fields added with the append structure are automatically included in the view.
- To include a table in a view, you must enter the character '*' in field *View field* in the view maintenance, the name of the table to be included in the field *Table* and the character '*' again in the field *Field name*.
- If you do not want to include a field of the included table in the view, proceed as follows:
  - Enter a '-' in the field *View field*.
  - Enter the name of the included table in the field *Table*.
  - Enter the name of the field in the field *Field name*.



Maintenance Views

Application object

Maintenance view on the tables

F1 F2 F3 F5 F8

Data exchange with the maintenance view

Table 1

F1 F2 F3

Table 2

F4 F5

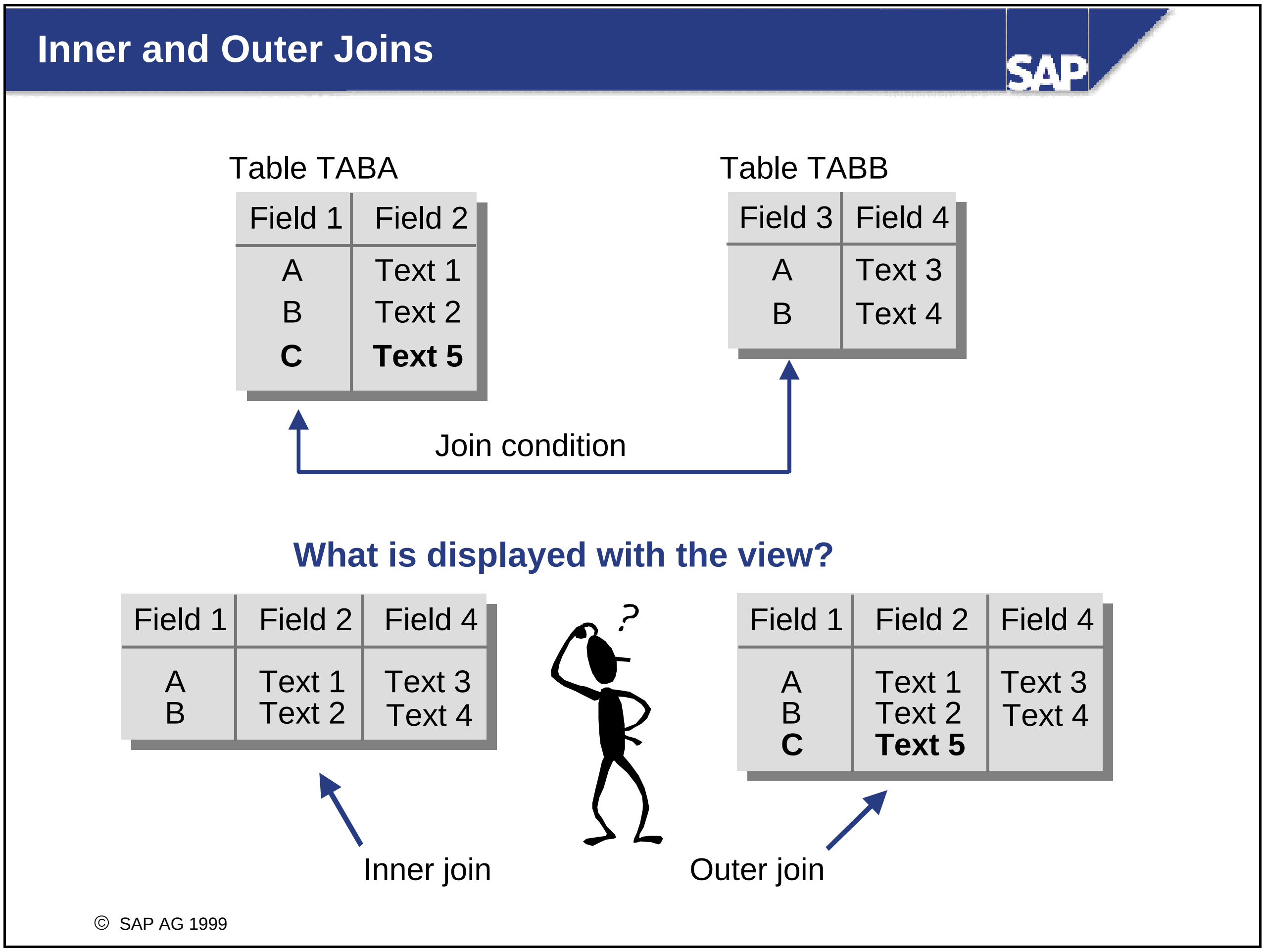
Table 3

F6 F7 F8

Foreign key

Foreign key

- ■ Data that is distributed on more than one table often forms a logical unit, called an application object. You should be able to display, change and create the data of such an application object together. Users usually are not interested in the technical implementation of the application object, such as the distribution of the data on several tables.
- ■ You can maintain complex application objects in a simple way using a maintenance view. The data is automatically distributed on the underlying database tables.
- ■ All the tables used in a maintenance view must be linked with a foreign key. This means that the join conditions are always derived from the foreign key in the maintenance view. You cannot enter the join conditions directly as in a database view.
- ■ A maintenance interface with which the data of the view can be displayed, changed and created must be generated from the definition of a maintenance view in the ABAP Dictionary.
- ■ When the maintenance interface is created, function modules that distribute the data maintained with the view on the underlying tables are automatically generated.
- ■ The maintenance interface is generated with the Transaction *Generate Table View* (Transaction SE54) or from the view maintenance screen with *Environment* **->** *Tab.maint.generator*.



Inner and Outer Joins

Table TABA

Field 1	Field 2
A	Text 1
B	Text 2
C	**Text 5**

Table TABB

Field 3	Field 4
A	Text 3
B	Text 4

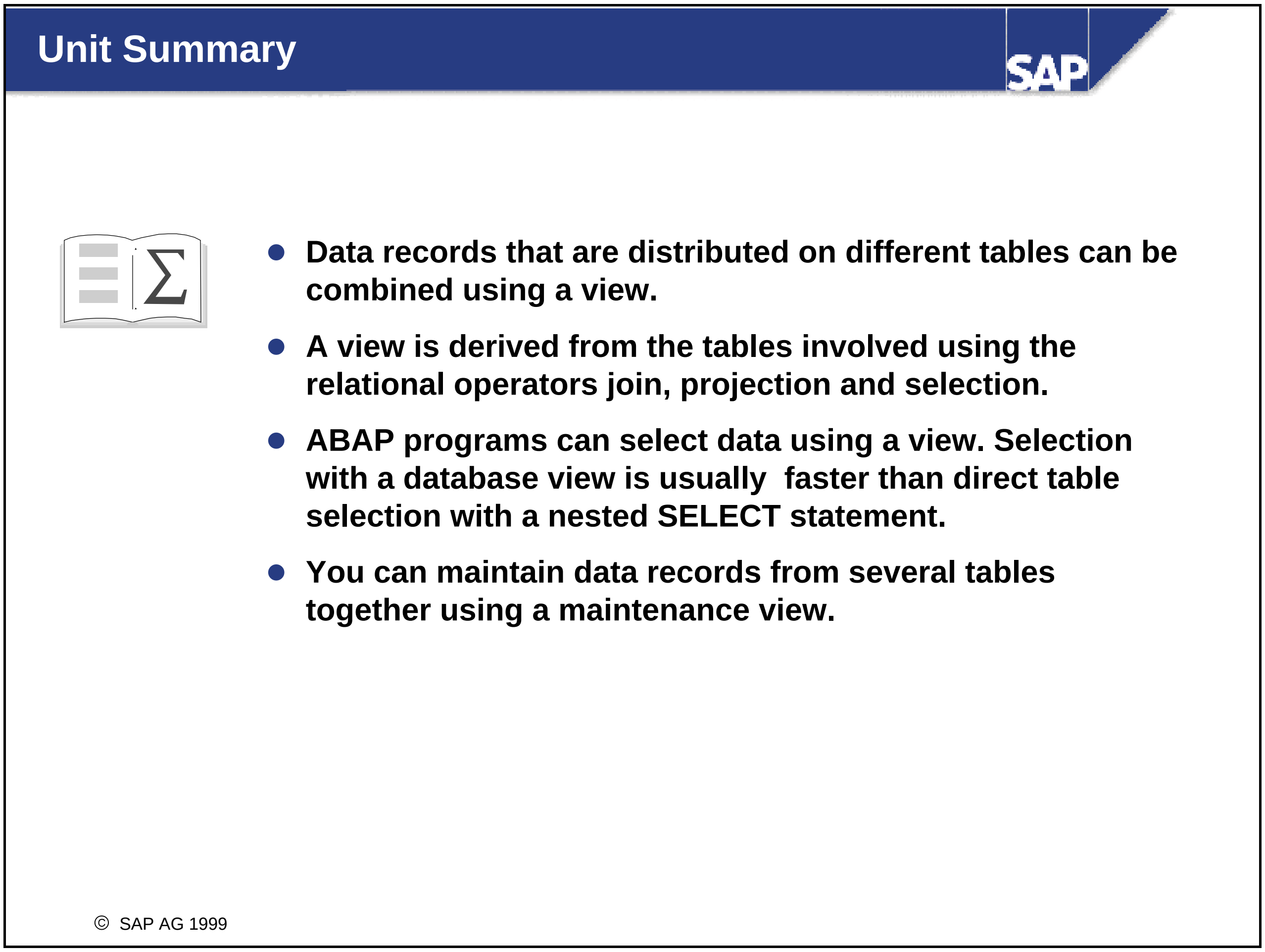
What is displayed with the view?

Field 1	Field 2	Field 4
A	Text 1	Text 3
B	Text 2	Text 4

Field 1	Field 2	Field 4
A	Text 1	Text 3
B	Text 2	Text 4
C	**Text 5**	

© SAP AG 1999

- The set of data that can be selected with a view greatly depends on whether the view implements an inner join or an outer join.
- With an inner join, you only get those records which have an entry in all the tables included in the view. With an outer join, on the other hand, those records that do not have a corresponding entry in some of the tables included in the view are also selected.
- The hit list found with an inner join can therefore be a subset of the hit list found with an outer join.
- Database views implement an inner join. You only get those records which have an entry in all the tables included in the view.
- Maintenance views implement an outer join.



Unit Summary

- Data records that are distributed on different tables can be combined using a view.
- A view is derived from the tables involved using the relational operators join, projection and selection.
- ABAP programs can select data using a view. Selection with a database view is usually faster than direct table selection with a nested SELECT statement.
- You can maintain data records from several tables together using a maintenance view.

Exercises: Views

Unit: Views

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

- Create views
- Define join conditions
- Define selection conditions
- Buffer database views

The data existing for an employee is distributed on several tables (corresponding to the relational data model). For some exercises, however, a complete view on this data is needed. In this exercise the corresponding views are implemented by creating views.

7-1 The flight personnel (all pilots and stewards) must be selected when a flight crew is set up. Not all the data in table ZEMPLOYxx may be displayed when this data is accessed. For example, the employee setting up the teams may not see the salary of the crew members. The telephone number of the employee's department should be output in case of questions.

7-1-1 Create a suitable database view ZEMPFLYxx that satisfies these requirements. The following information about an employee should be displayed:

- Client
- Carrier
- Personnel number
- First name
- Last name
- Telephone number of the department
- Department code

7-1-2 Make sure that only flight personnel can be selected with the view.

7-1-3 You probably will have to access the data using the view frequently. The selected data should therefore be buffered in order to increase performance. Choose *full buffering* as buffering type.

7-2 To set up the flight crews, you have to select the existing employee assignments for flights. Additional information about the flight, such as the city of departure and city of arrival, is of particular interest.

7-2-1 Create a database view ZCREWSxx to display which employees are assigned to what flights.

The following data should be displayed:

- Client
- Carrier
- Flight connection

- Date of flight
- Personnel number
- First and last names of the employee
- Role of the employee on the flight (pilot, copilot, steward)
- Type of airplane
- City of departure of the flight
- City of arrival of the flight

Use the join conditions from the foreign key between tables ZEMPLOYxx and ZFLCREWxx. Create these foreign keys before defining the view.

Table SFLIGHT contains the information about the type of airplane. The cities of departure and arrival can be found in table SPFLI.

7-2-2 Enter yourself as an employee of carrier AA (American Airlines) and assign yourself to a flight as pilot. Then display your cities of departure and arrival with the view.

7-3 **Supplementary Exercise:** Write an ABAP program which outputs the crew assigned to a flight. Select the data with view ZCREWSxx. The following data should be output:

- Carrier
- Flight
- Date of flight
- City of departure
- City of arrival
- Pilot, copilot
- Steward

If you already attended an ABAP programming course, you can now edit the output. A clear form for the output would be for example:

<*Carrier*> <*Flight*> on <*Date_of_flight*> from <*City_of_departure*> to <*City_of_arrival*>:

Pilot: *xx*

Copilot: *xx*

Stewards: *xx, xx, xx*

xx is here the first name, last name and personnel number of the corresponding person.

7-4 **Supplementary Exercise:** Create a maintenance view with the name ZPARTNERxx, with which you can easily maintain new business partners. The business partners are entered in table SBUSPART. A business partner can be either a flight customer or a travel agency. If it is a travel agency, there will be a corresponding entry in table STRAVELAG.

The view should also permit you to maintain tables SBUSPART and STRAVELAG at one time. Include all the necessary fields of the tables in the view.

Generate the maintenance interface. Use the following parameters:

Function group: ZZBC430xx

Authorization group: SUNI

Maintenance type: one-step

Overview screen: 100

Maintain the data of a new travel agency using the enhanced table maintenance (*System → Services → Table maintenance → Ext. table maint.).*

Solutions: Views

Unit: Views

7-1 The view should permit a view on data in tables ZEMPLOYxx and ZDEPMENTxx. To create the view:

1) In the initial screen of the ABAP Dictionary, mark object type *View*, enter the object name ZEMPFLYxx and choose *Create*.

2) A dialog box appears in which you should select the view type. Mark *Database view* and press *Choose*.

3) Enter a short text in the next screen.

The view should display data about employees (from table ZEMPLOYxx) and departments (from table ZDEPMENTxx).

4) First enter table ZEMPLOYxx in field *Tables*.

5) Choose *Relationships*. All the foreign key relationships of table ZEMPLOYxx to other tables are listed. Mark the relationship to table ZDEPMENTxx and choose *Copy*.

6) The join conditions are copied from the foreign key. In a different mode, display the foreign key between the two tables and notice the relationship between the foreign key and the join conditions.

7) You now have to copy fields from the tables to the view. Click on tab page *View fields*.

8) Choose *Table fields*. In the next dialog box, mark table ZEMPLOYxx and choose *Choose*.

9) All the fields of table ZEMPLOYxx are listed. Mark fields *Client*, *Carrier*, *Personnel number*, *First name*, and *Last name*. Choose *Copy*. The fields are now inserted in the view.

10) Again choose *Table fields*. In the dialog box, choose table ZDEPMENTxx and insert fields *Department telephone* and *Department code* in the view as described above.

Only flight personnel should be selected with the view. You can define this restriction with a selection condition.

11) Click on tab page *Selection conditions*.

12) The restriction whether an employee belongs to the flight personnel is contained in field ZEMPLOYxx-Area. Enter it in columns *Table* and *Field name*.

13) Flight personnel is identified by the value 'F' in field *Area*. Enter EQ in the column *Operator* and 'F' (including the apostrophes) in column *Compar. value*.

You now have to buffer the view.

14) Choose *Goto* ? *Technical settings*. The maintenance screen for the technical settings of the view appears. With the exception of some attributes that are meaningless for views and which are therefore not displayed, the screen is analogous to the corresponding maintenance screen for tables.

15) Mark *Buffering switched on* and *Fully buffered*.

16) Save the technical settings and return to the view maintenance screen.

17) Activate the view.

7-2 The view should permit a common view on data in tables ZFLCREWxx, ZEMPLOYxx, SFLIGHT and SPFLI. To create the view:

1) First create the foreign key. To do this, go to the maintenance screen for table ZFLCREWxx. Define a foreign key with check table ZEMPLOYxx for field EMP_NUM.

Use the field assignment proposed by the system. The cardinality of the relationship is 1:CN and the foreign key fields are key fields.

2) Select object type *View* in the initial screen of the ABAP Dictionary, enter the object name ZCREWSxx and choose *Create*.

3) A dialog box appears in which you should select the view type. Mark *Database view* and press *Choose*.

4) Enter a short text in the next screen.

The view should display data about the assignment of employees to flights (in table ZFLCREWxx), about employee data (in table ZEMPLOYxx) and about flight data (in table SFLIGHT). The information about the cities of departure and arrival are in the flight schedule (table SPFLI) and not directly in table SFLIGHT.

5) First enter table ZFLCREWxx in field *Tables*.

6) Now include table ZEMPLOYxx in the view and link it with table ZFLCREWxx. You can create the join conditions from the foreign key between the tables.

7) Position the cursor on ZFLCREWxx and choose *Relationships*.

8) A dialog box appears listing all existing foreign key relationships of table ZFLCREWxx to other tables. Mark the relationship to table ZEMPLOYxx and choose *Copy*.

9) Table ZEMPLOYxx is entered in field *Tables* and the join conditions are created from the foreign key between the two tables.

Create the following join conditions:

ZEMPLOYxx-Client = ZFLCREWxx-CLIENT

ZEMPLOYxx-Carrier = ZFLCREWxx-CARRID

ZEMPLOYxx-Personnel number = ZFLCREWxx-EMP_NUM

10) Include table SFLIGHT in the view in the same way. Create the join conditions from the foreign key between tables ZFLCREWxx and SFLIGHT. The join condition should have the following form:

SFLIGHT-MANDT = ZFLCREWxx-CLIENT

SFLIGHT-CARRID = ZFLCREWxx-CARRID

SFLIGHT-CONNID = ZFLCREWxx-CONNID

SFLIGHT-FLDATE = ZFLCREWxx-FLDATE

11) Include table SPFLI in the view (as described above). The join conditions can be created from the foreign key between tables SFLIGHT and SPFLI. The join condition should have the following form:

SPFLI-MANDT = SFLIGHT-MANDT

SPFLI-CARRID = SFLIGHT-CARRID

SPFLI-CONNID = SFLIGHT-CONNID

12) Now include the following fields in the view (see the solution to Exercise 1):

From table ZFLCREWxx fields MANDT, CARRID, CONNID, FLDATE, EMP_NUM and ROLE.

From table ZEMPLOYxx fields *Last name* and *First name*.

From table SFLIGHT field PLANETYPE.

From table SPFLI fields CITYFROM and CITYTO.

13) Activate the view.

To enter yourself as an employee, call *Utilities* → *Table contents* → *Create entries* from the maintenance screen of table ZEMPLOYxx. You can maintain the data record here. Enter yourself in table ZFLCREWxx for a flight in the same way. You can then search for the information from the maintenance screen for view ZFLCREWxx with *Utilities* → *Content*.

7-3 Look at program SVIEW_CREW as a sample solution.

7-4 Proceed as follows:

1) In the initial screen of the ABAP Dictionary, mark object type *View*, enter the object name ZPARTNERxx and choose *Create*.

2) A dialog box appears in which you should select the view type. Mark *Maintenance view* and choose *Choose*.

3) Enter a short text in the next screen.

You want to maintain the data in tables SBUSPART and STRAVELAG together in the maintenance view. If you wanted to enter a new partner directly, you would first have to enter it in table SBUSPART. Only then could you enter the corresponding data in table STRAVELAG (because of the existing foreign key check between SBUSPART and STRAVELAG). You therefore first have to include table SBUSPART in the definition of the maintenance view.

4) Enter table SBUSPART in the field *Tables*. The key fields of this table are automatically included in the view as fields.

5) Place the cursor in field *Tables* on entry SBUSPART. Choose *Relationships*.

6) A dialog box appears listing all existing foreign key relationships of table SBUSPART to other tables. Mark the foreign key relationship to table STRAVELAG and choose *Copy*.

7) The join conditions are created from the foreign key. The join conditions have the following form:

   SBUSPART-MANDANT = STRAVELAG-MANDT

   SBUSPART-BUSPARTNUM = STRAVELAG-AGENCYNUM

8) You now have to include the fields of both tables in the view. Go to tab page *View fields*. Position the cursor on table SBUSPART and choose *Table fields*. A list of all the fields of the table appears. Choose *Select all* and then press *Copy*.

9) Include all the fields of table STRAVELAG with the exception of fields MANDT and AGENCYNUM in the view in the same way. These fields are linked to the corresponding fields of table SBUSPART with the join conditions and therefore should not appear in the view.

10) Activate the view.

Now generate a maintenance interface for the view.

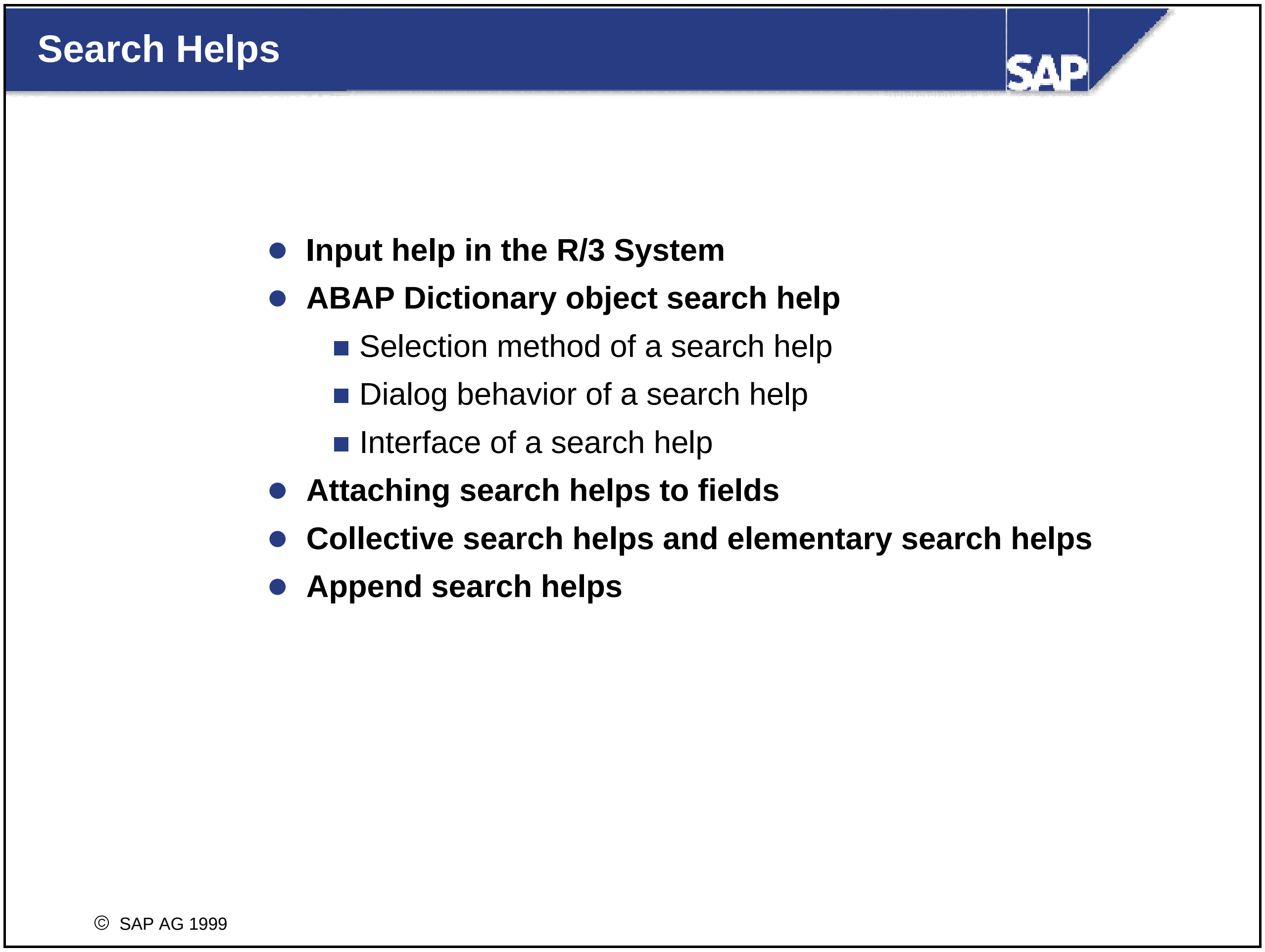
11) Choose *Utilities* → *Table maintenance generator*.

12) Enter authorization group SUNI and function group ZZBC430xx in the next screen.

13) Mark maintenance type *one-step*. Select number *0100* as maintenance screen number of the overview screen.

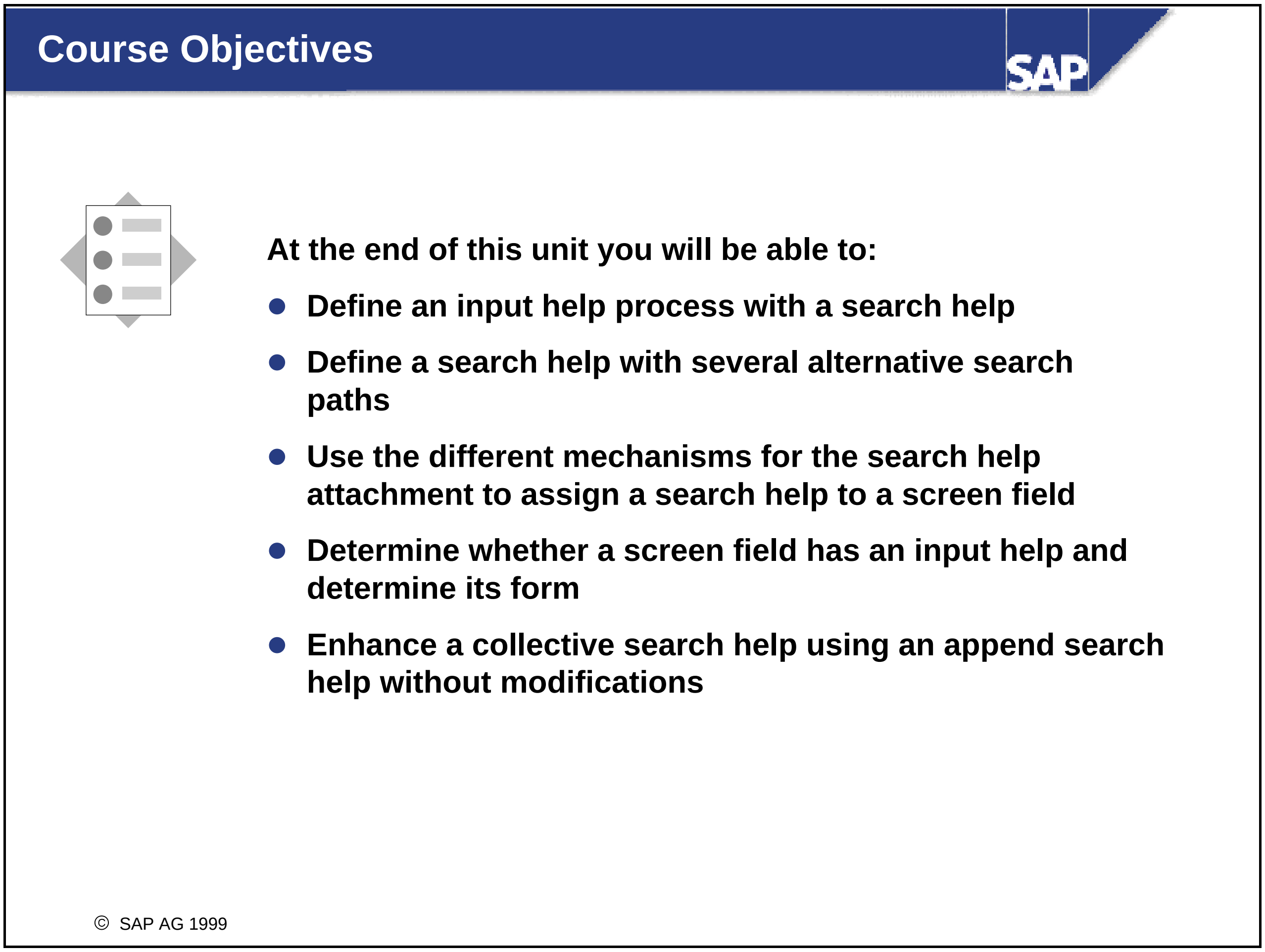
14) Choose *Create*. The development class of the function group and the generated maintenance objects are prompted. In both cases choose *Local object*.

15) Call the extended table maintenance with the given menu path and enter the data of a new travel agency. With the Data Browser (in the menu environment of the initial screen of the ABAP Dictionary), verify that the data of the new travel agency was written in tables SBUSPART and STRAVELAG.



Search Helps

- Input help in the R/3 System
- ABAP Dictionary object search help
  - Selection method of a search help
  - Dialog behavior of a search help
  - Interface of a search help
- Attaching search helps to fields
- Collective search helps and elementary search helps
- Append search helps

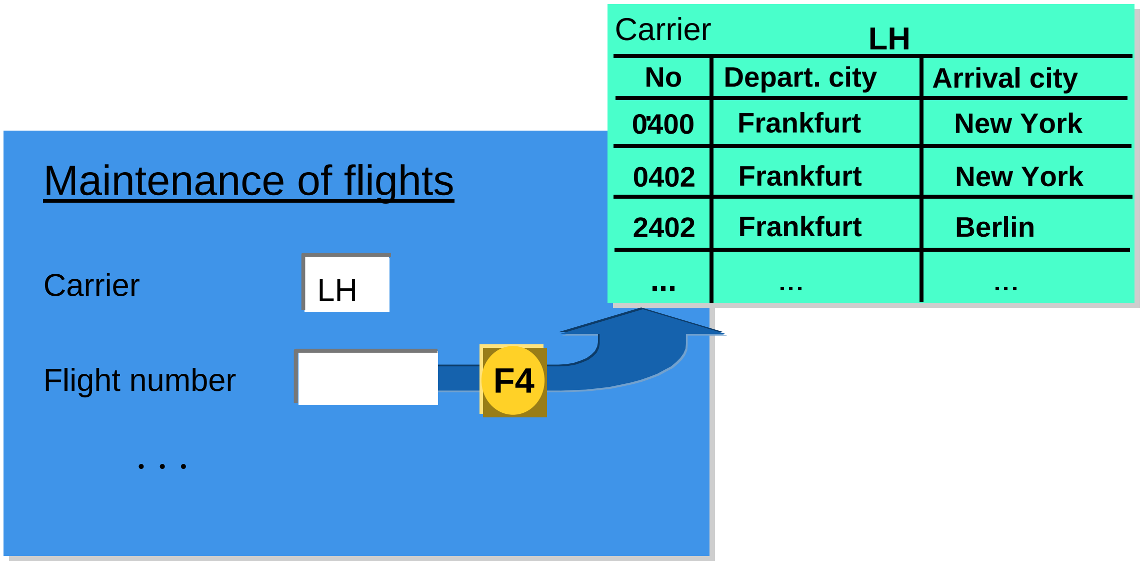


Course Objectives

At the end of this unit you will be able to:

- Define an input help process with a search help
- Define a search help with several alternative search paths
- Use the different mechanisms for the search help attachment to assign a search help to a screen field
- Determine whether a screen field has an input help and determine its form
- Enhance a collective search help using an append search help without modifications

R/3 Standard Function: Input Help



Maintenance of flights

Carrier LH

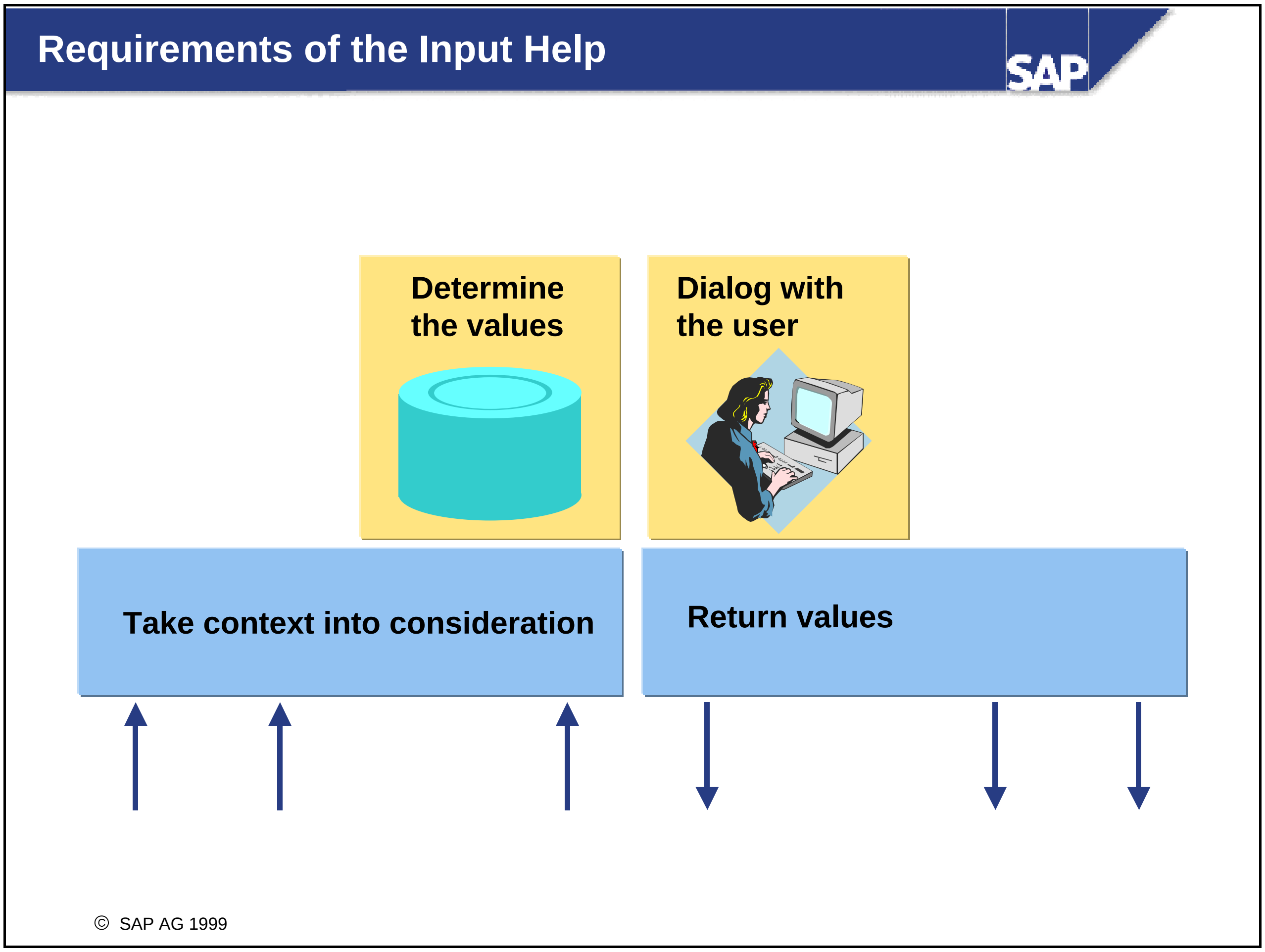
Flight number

. . .

Carrier LH

No	Depart. city	Arrival city
0400	Frankfurt	New York
0402	Frankfurt	New York
2402	Frankfurt	Berlin
...	...	...

- The input help (F4 help) is a standard function of the R/3 System. It permits the user to display a list of possible values for a screen field. A value can be directly copied to an input field by list selection.
- The fields having an input help are shown in the R/3 System by the input help key to the right of the field. This key appears as soon as the cursor is positioned on the corresponding screen field. The help can be started either by clicking on this screen element or with function key F4.
- If the number of possible entries for a field is very large, you can limit the set of displayed values by entering further restrictions.
- The display of the possible entries is enhanced with further useful information about the displayed values. This feature is especially useful if the field requires the entry of a formal key.
- Since the input help is a standard function, it should look and behave the same throughout the entire R/3 System. The development environment therefore provides tools for assigning a standardized input help to a screen field.
- The precise description of the input help for a field is usually defined by its semantics. For this reason, the input help for a field is normally defined in the ABAP Dictionary.



Requirements of the Input Help

Determine the values

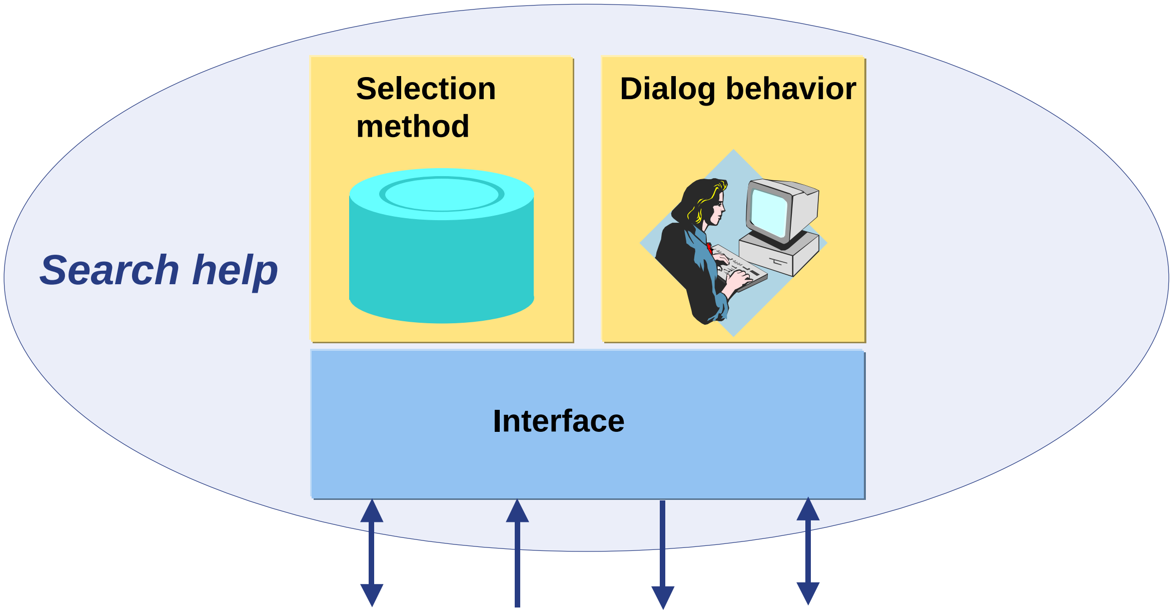
Dialog with the user

Take context into consideration

Return values

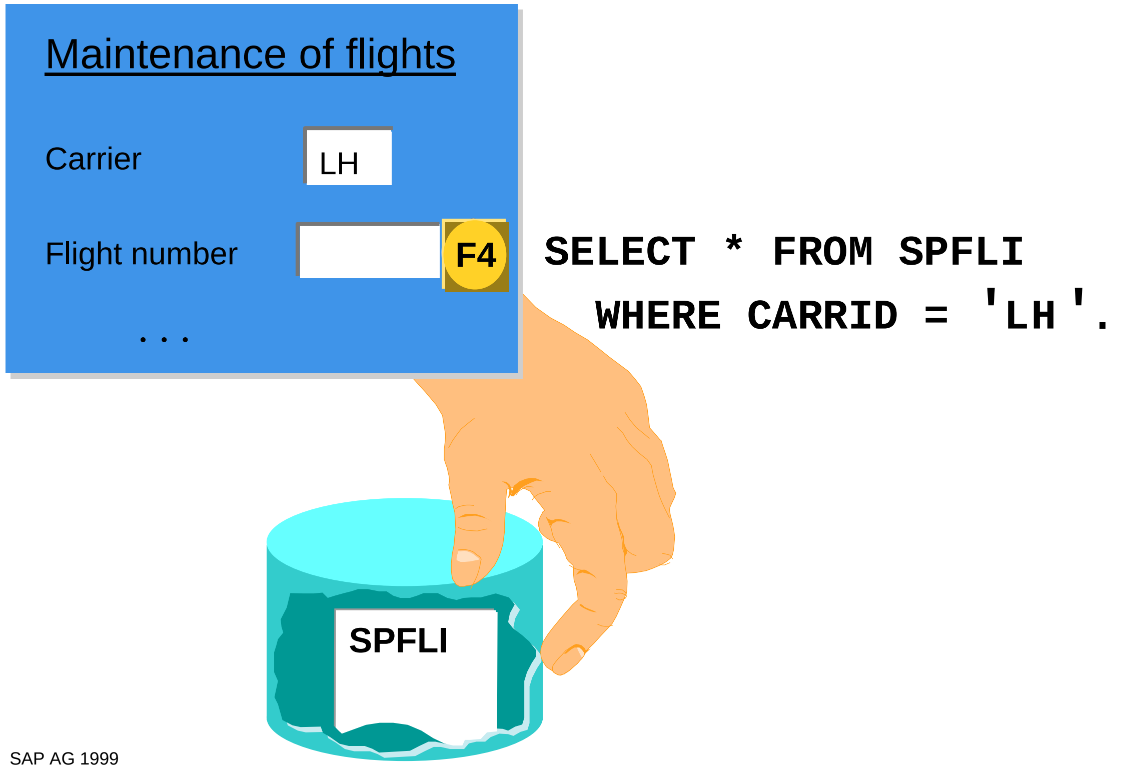
- A number of requirements must be met for the input help of a screen field (**search field**):
- Information (about the context) known to the system must be taken into consideration in the input help. This includes entries the user already made in the current input template as well as information obtained in previous dialog steps. Normally the input help uses the context to limit the set of possible values.
- The input help must determine the values that can be offered to the user for selection. The data to be displayed as supplementary information in the list of possible values must also be determined. When the possible values are determined, the restrictions resulting from the context and from further search conditions specified by the user must also be taken into consideration.
- The input help must hold a dialog with the user. This dialog always contains the presentation of the possible values (with supplementary information) in list form and the possibility to select a value from this list. A search template in which the user can define conditions for the values to be displayed is also sometimes required .
- If the user selects a value, the input help must return it to the search field. The input template often contains more fields (often only display fields) containing further explanatory information about the search field. The input help should also update the contents of these fields in this case.

ABAP Dictionary Object Search Help



- The ABAP Dictionary object **search help** is used to describe an input help. The definition of a search help contains the information the system needs to satisfy the described requirements.
- The **interface** of the search help controls the data transfer from the input template to the F4 help and back. The interface defines the context data to be used and the data to be returned to the input template when a value is selected.
- The **internal behavior** of the search help describes the F4 process itself. This includes the **selection method** with which the values to be displayed should be determined as well as the **dialog behavior** describing the interaction with the user.
- As with a function module, search helps distinguish between the interface with which it exchanges data with other software components and the internal behavior (for function modules, the latter is defined by the source text).
- It only makes sense to define a search help if there is a mechanism available with which the search help can be accessed from a screen. This mechanism is called the **search help attachment** and will be described later.
- Like the editor for function modules, the editor for search helps also enables you to test an object. You can thus test the behavior of a search help without assigning it to a screen field.

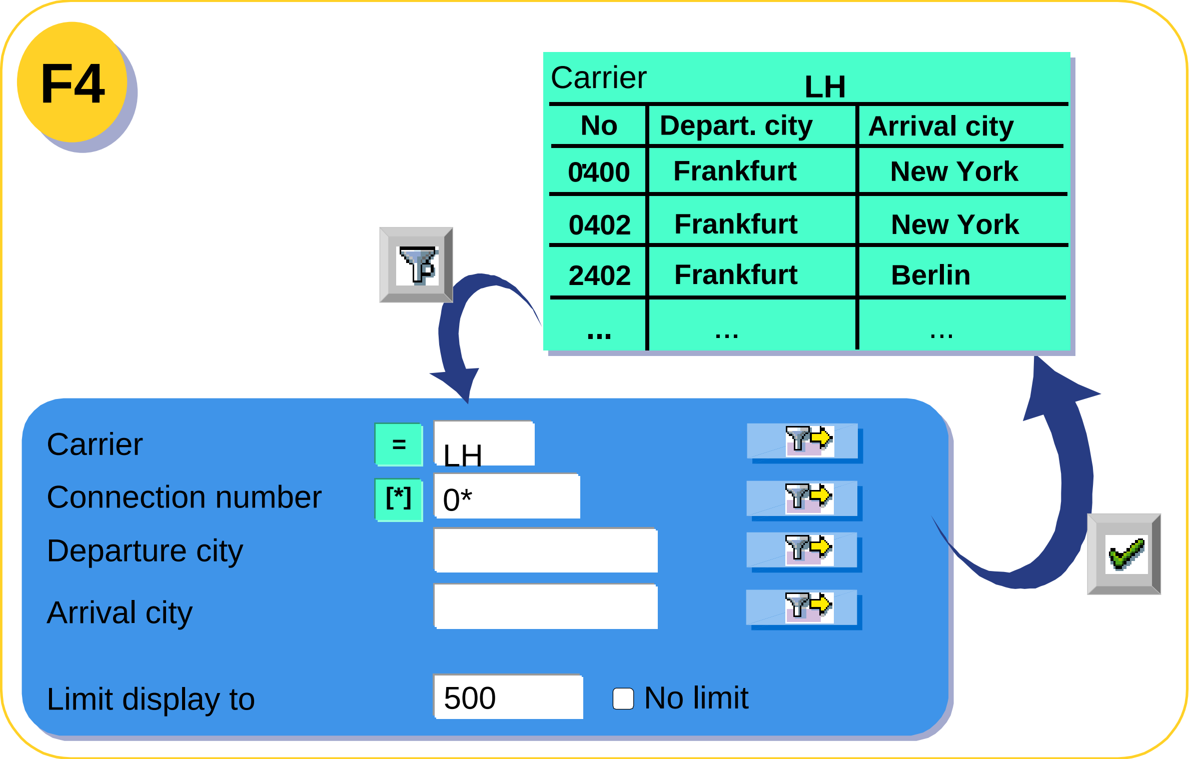
Selection Method of a Search Help



```
SELECT * FROM SPFLI
  WHERE CARRID = 'LH'.
```

- The possible values displayed for a field by the input help are determined at runtime by a selection from the database. When a search help is defined, you must define the database object from which the data should be selected by specifying a table or a view as the **selection method**.
- It makes sense to use a view as selection method if the data about the possible values that is relevant for the input help is distributed on several tables. If this data is all in one table or in the corresponding text table, you can use the table as a selection method. The system automatically ensures that the text of the text table is used in the user's logon language.
- If there is not yet a view that combines the data that is relevant for an input help, you must first create it in the ABAP Dictionary.
- Maintenance views may not be used as the selection method for search helps. Normally a database view is used. However, you should note that database views (in the R/3 System) are always created with an inner join. As a result, only those values having an entry in each of the tables involved are offered in the input help. Sometimes the values should be determined with an outer join. In this case you should choose a help view as the selection method. You can find more information about help views in the appendix.
- If the selection method of a search help is client-dependent, the possible values are only selected in the user's logon client.

Description of the Dialog Behavior



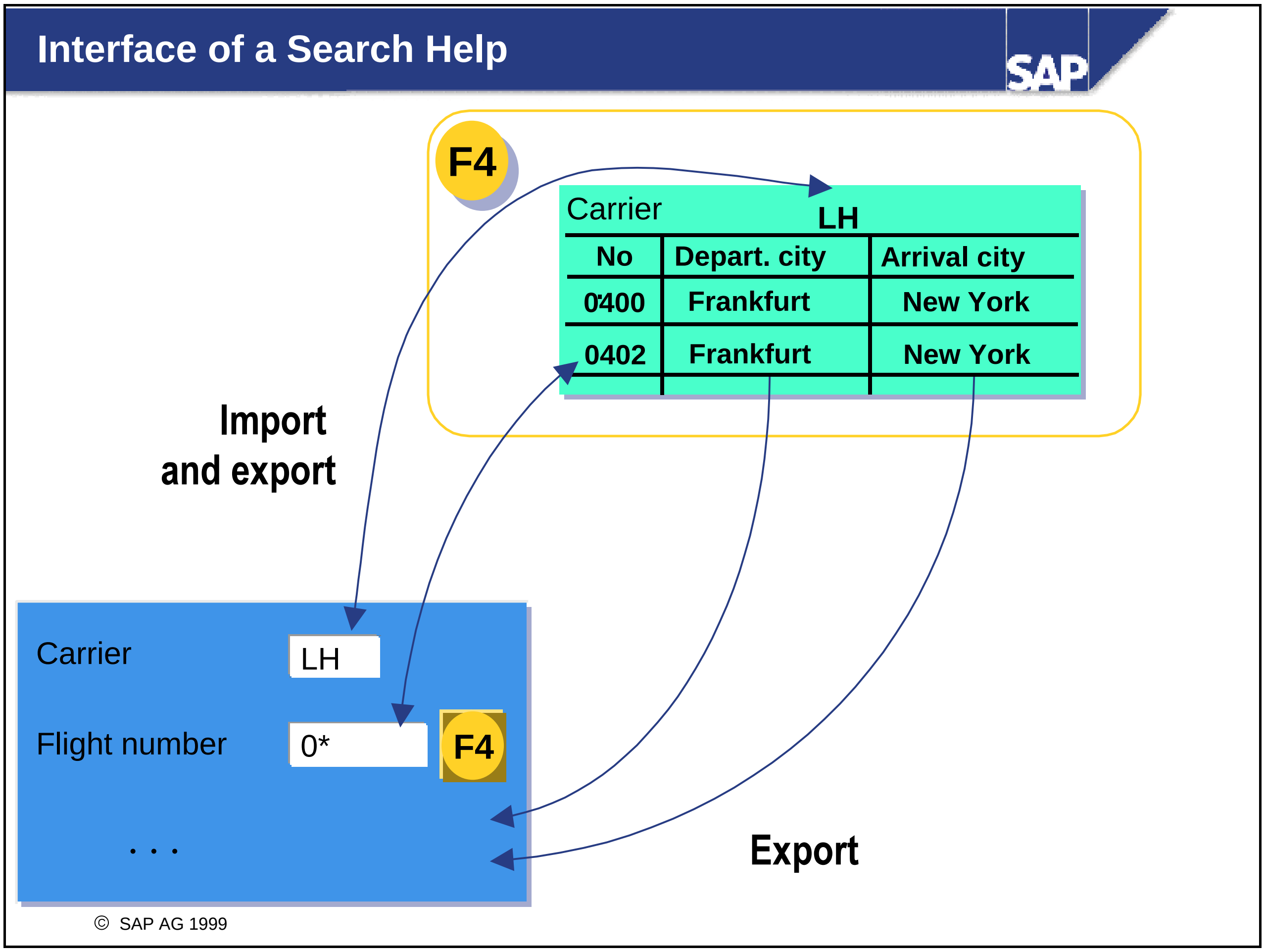
Carrier LH

No	Depart. city	Arrival city
0400	Frankfurt	New York
0402	Frankfurt	New York
2402	Frankfurt	Berlin
...	...	...

Carrier	=	LH
Connection number	[*]	0*
Departure city		
Arrival city		
Limit display to		500

□ No limit

- The possible values are presented in the **dialog box for displaying the hit list** and the user can select values from here. If the possible values are formal keys, further information should also be displayed.
- If the hit list is very large, the user should be able to define further restrictions for the attributes of the entry. Restricting the set of data in this way both increases the clarity of the list and reduces the system load. Additional conditions can be entered in a further dialog window, the **dialog box for restricting values**.
- The dialog type of a search help defines whether the dialog box for restricting values should be displayed before determining the hit list.
- You must define the characteristics to appear on either (or both) of the dialog boxes as **parameters** in the search help. You can use all the fields of the selection method (with the exception of the client field) and the non-key fields of your text table as parameters.
- You define which parameter should appear in which dialog box (in what order) by assigning the parameters positions in the two dialog boxes. You can thus use different parameters (or different orders) in the two dialog boxes.
- Types must be defined for search help parameters with data elements. These define the display in the two dialog boxes. If nothing else is defined, a parameter uses the data element of the corresponding field of the selection method.



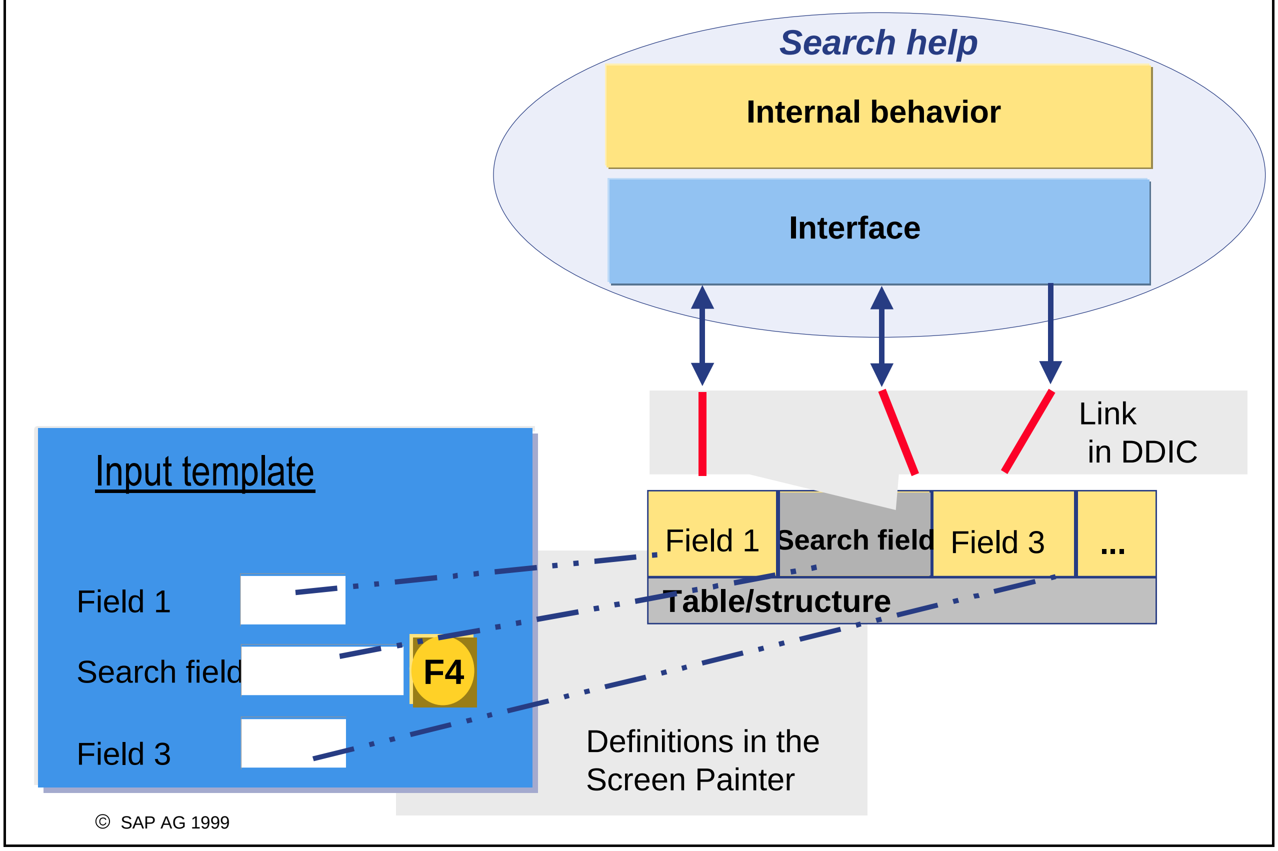
Interface of a Search Help

Carrier LH

No	Depart. city	Arrival city
0400	Frankfurt	New York
0402	Frankfurt	New York

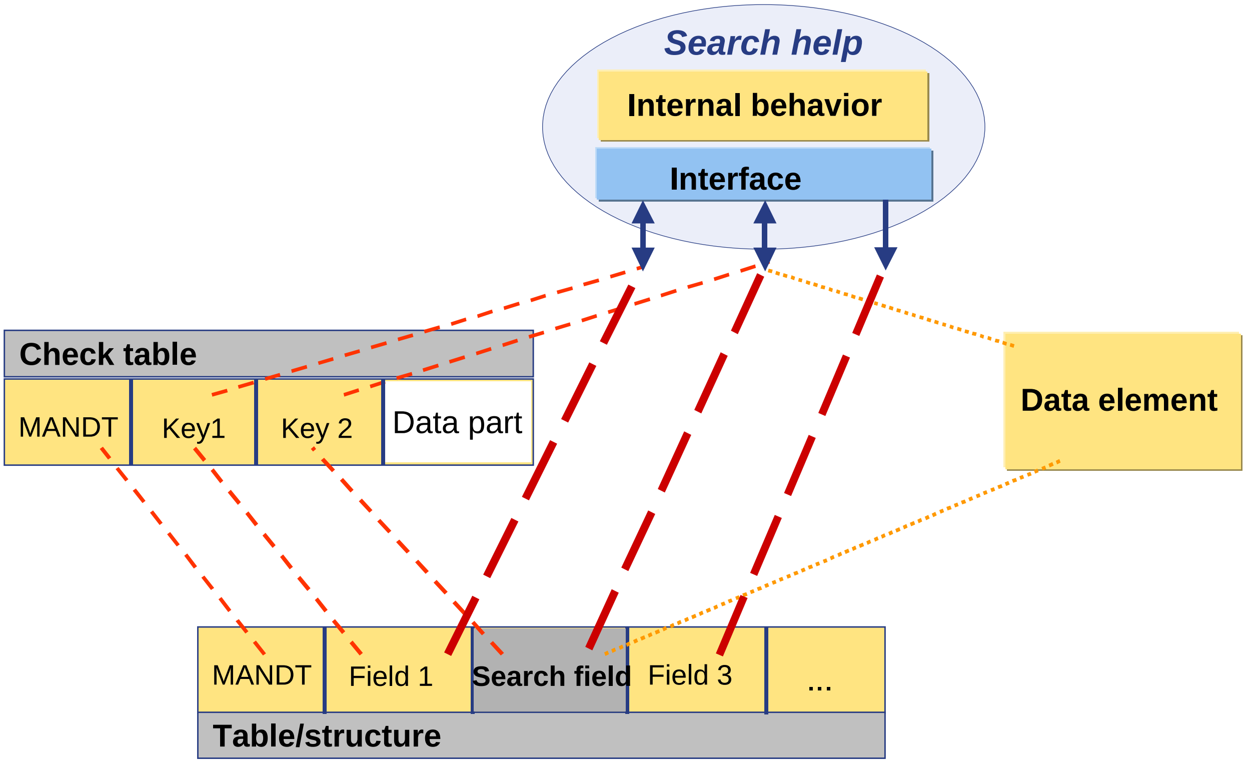
- When you define a parameter of a search help, you must also define whether it should be used to copy data to the input help (IMPORT parameter) or whether to return data from the input help (EXPORT parameter).
- The IMPORT and EXPORT parameters of a search help together make up your interface. (This is also analogous to function modules.)
- You can also define interface parameters that do not appear in either the dialog box for displaying the hit list or the dialog box for restricting values. This is useful for example when screen fields that do not appear on either of the two dialog boxes are to be updated when you select a value.
- The location from which the IMPORT parameters of a search help get their values and the screen fields in which the contents of the EXPORT parameters of the search help are returned are defined in the search help attachment.
- The search field is a special case. Its contents are only used in the input help if it is a search string (that is, if it contains a ´*´ or a ´+´) and the parameter linked with the search field is an IMPORT parameter.
- Parameters that only contain additional information about the search field should not be defined as IMPORT parameters since the user must otherwise empty the corresponding screen fields each time before he can define a new value with the input help.

How do you Use Search Helps?



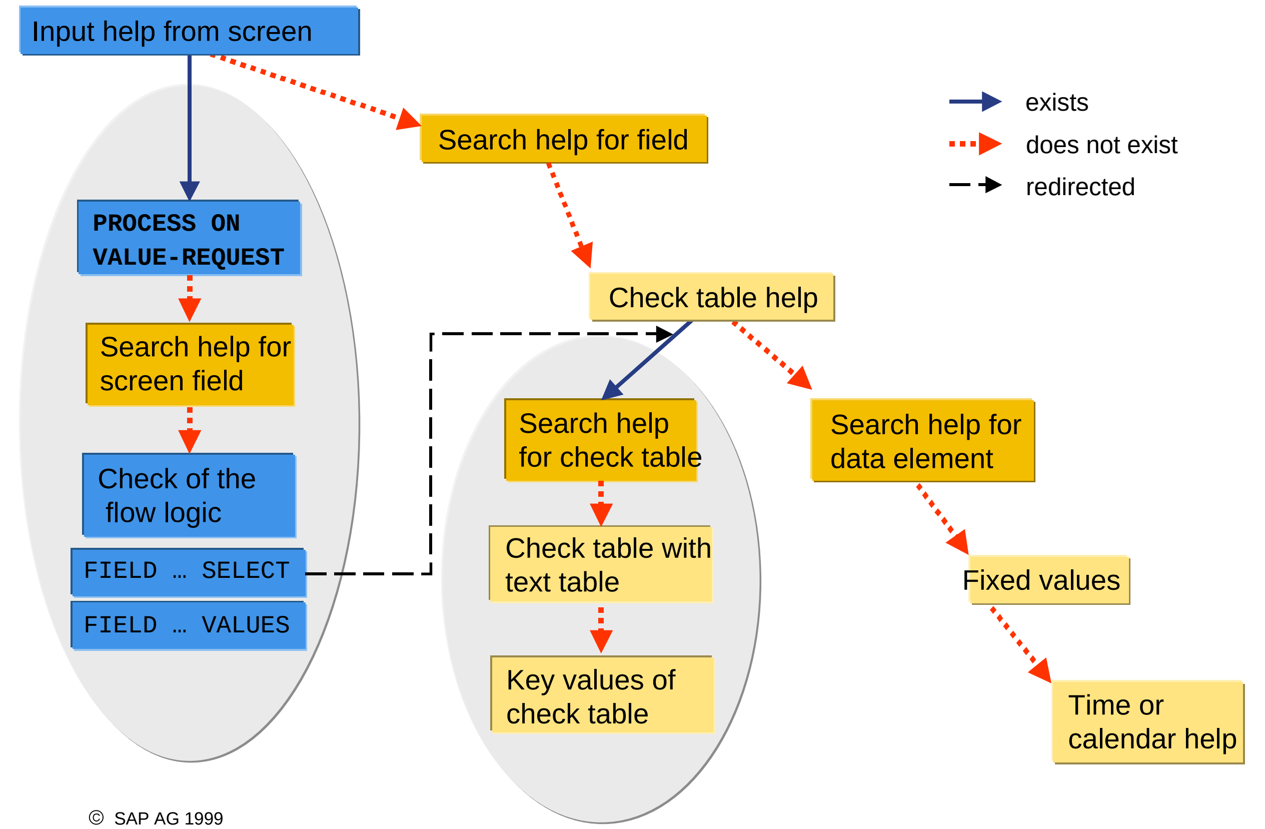
- A search help describes the flow of an input help. The search help can only take effect using a mechanism that assigns the search help to this field. This mechanism is called the **search help attachment** to the field.
- Attaching a search help to a field has an effect on the field's behavior. It is therefore considered to be part of the field definition.
- The semantic and technical attributes of a screen field (type, length, F1 help, ...) are not normally defined directly when the input template is defined. On the contrary, only a reference to an ABAP Dictionary field (usually with the same name) is specified in the Screen Painter. The screen field takes on the attributes of this field from the ABAP Dictionary.
  The same principle is also used to define the input help of a screen field. The search help is thus attached to the ABAP Dictionary search field and not to the screen field.
- In the search help attachment, the interface parameters of the search help and the screen fields providing data for the input help or getting data from the input help are assigned to one another. The search field must be assigned to an EXPORT parameter of the search help at this time. This parameter should also be an IMPORT parameter so that the user can take advantage of search patterns that are already entered.
- Fields that do not have a search help attachment can also have an input help since further mechanisms (e.g. domain fixed values) are also used for the F4 help.

Search Help Attachment in the ABAP Dictionary

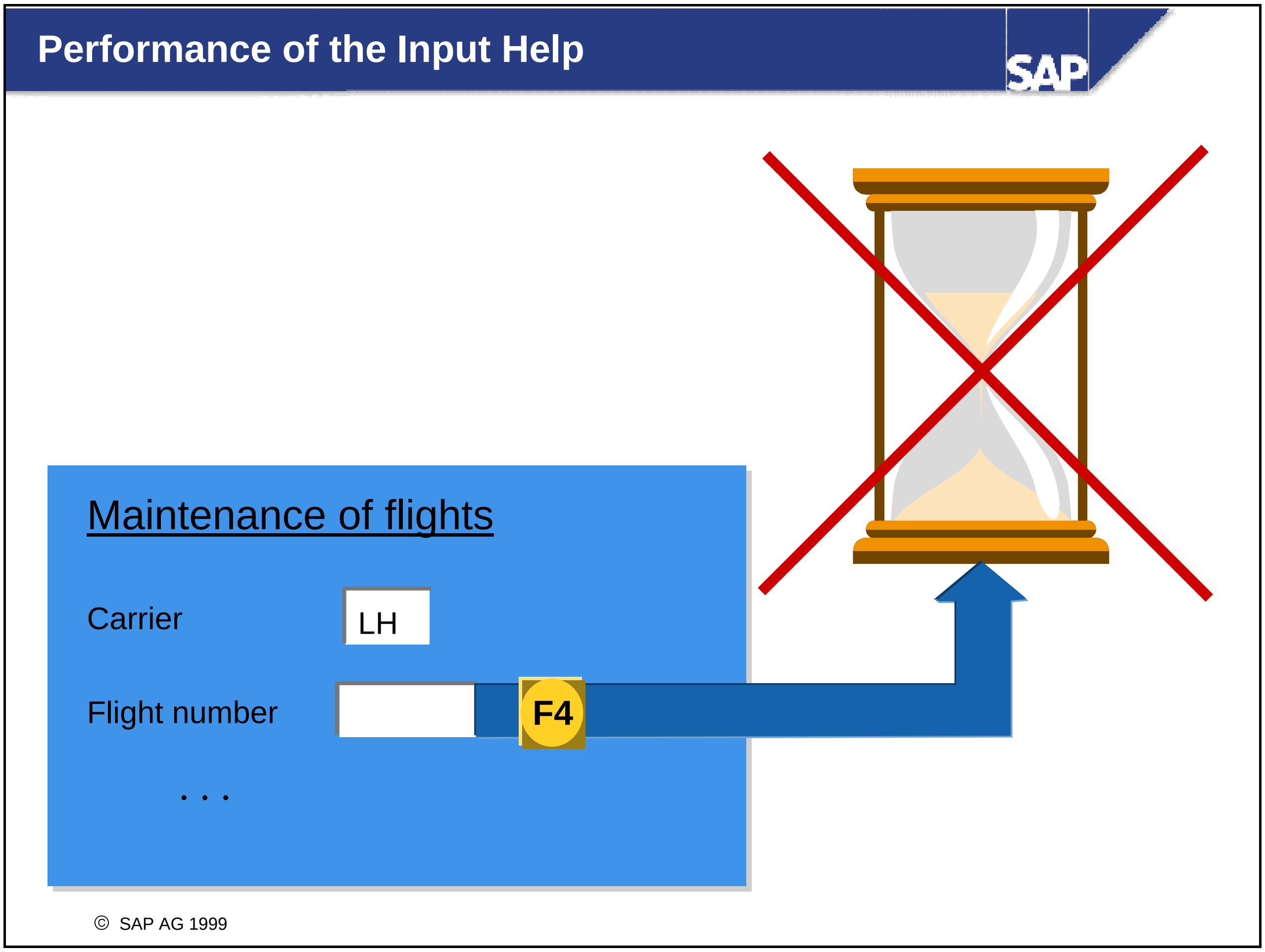


- There are three mechanisms for attaching a search help to a field of the ABAP Dictionary.
- A search help can be attached directly to a field of a structure or table. The definition of this attachment is analogous to that of a foreign key. You have to define an assignment (between the interface parameters of the search help and the fields of the structure) for which the system makes a proposal.
- If a field has a check table, its contents are automatically offered as possible values in the input help. The key fields of the check table are displayed. If a check table has a text table, its first character-like non-key field is displayed.
  If you are not satisfied with the described standard display of the data of the check table, you can attach a search help to the check table. This search help is used for all the fields that have this table as check table. You have to define an assignment between the interface of the search help and the key of the check table when you define the attachment.
- The semantics of a field and its possible values are defined by its data element. You can therefore attach a search help to a data element. The search help is then available for all the fields that refer to this data element. In the attachment you must define an EXPORT parameter of the search help for the data transfer.
- Attaching a search help to a check table (or a data element) can result in a high degree of reusability. However, there are restrictions on passing further values via the interface of the search help.

Overview: Mechanisms for the Input Help



- In order to be able to offer a meaningful input help for as many screen fields as possible, the R/3 System uses a number of mechanisms. If there is more than one such mechanism available for a field, the one that is furthest left or at the top of the above hierarchy is used.
- In addition to the options described above for defining the input help of a field in the ABAP Dictionary, you can also define it in the screen field. The disadvantage, however, is that there is no automatic reuse.
- With the screen event POV you can program the input help of a field by yourself. You can adjust the design of the help to the standard help using the function modules F4IF_FIELD_VALUE_REQUEST or F4IF_INT_TABLE_VALUE_REQUEST.
  However, you should check to see if the part of the input help that you programmed yourself should be implemented as a search help exit instead (see appendix).
- You can also attach a search help to a screen field in the Screen Painter. There are some functional restrictions on this kind of attachment as compared with attachment in the Dictionary.
- You should no longer use the input checks defined directly in the flow logic of the screen, from which it is also possible to derive input helps.
- The function *Technical info* is offered in the hit list in the menu of the right mouse key. It can be used to find out which of the specified mechanisms is being used.



Performance of the Input Help

Maintenance of flights

Carrier LH

Flight number

. . .

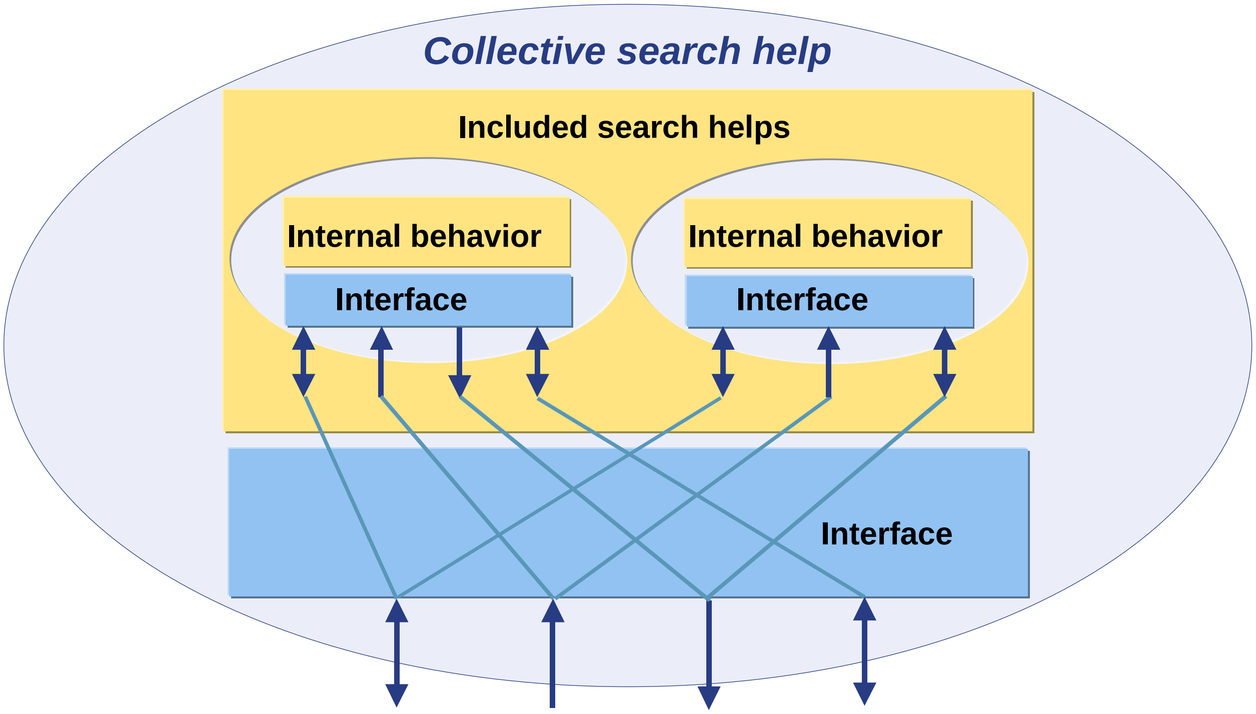
- You sometimes have to search a large amount of data with an input help. This means that you might have to wait a long time for the possible entries to be displayed, and can also result in a significant increase in the load on the system.
- When you define a search help, you should therefore check whether you should take measures to optimize the accessing behavior for the selection method. This is especially true if the selection uses a view and thus more than one physical table.
- If the number of entries in the selection method is very large, you should restrict the hit list with further conditions. This also increases the clarity of the hit list. The additional conditions can directly result from the context, or can be entered in the dialog box for restricting values by the user. The performance of the input help can frequently be significantly improved by creating an index on the fields used to formulate the restrictions.
- If the number of entries in the selection method is relatively small, you should always check whether the selection method can be buffered.

Alternative Search Paths

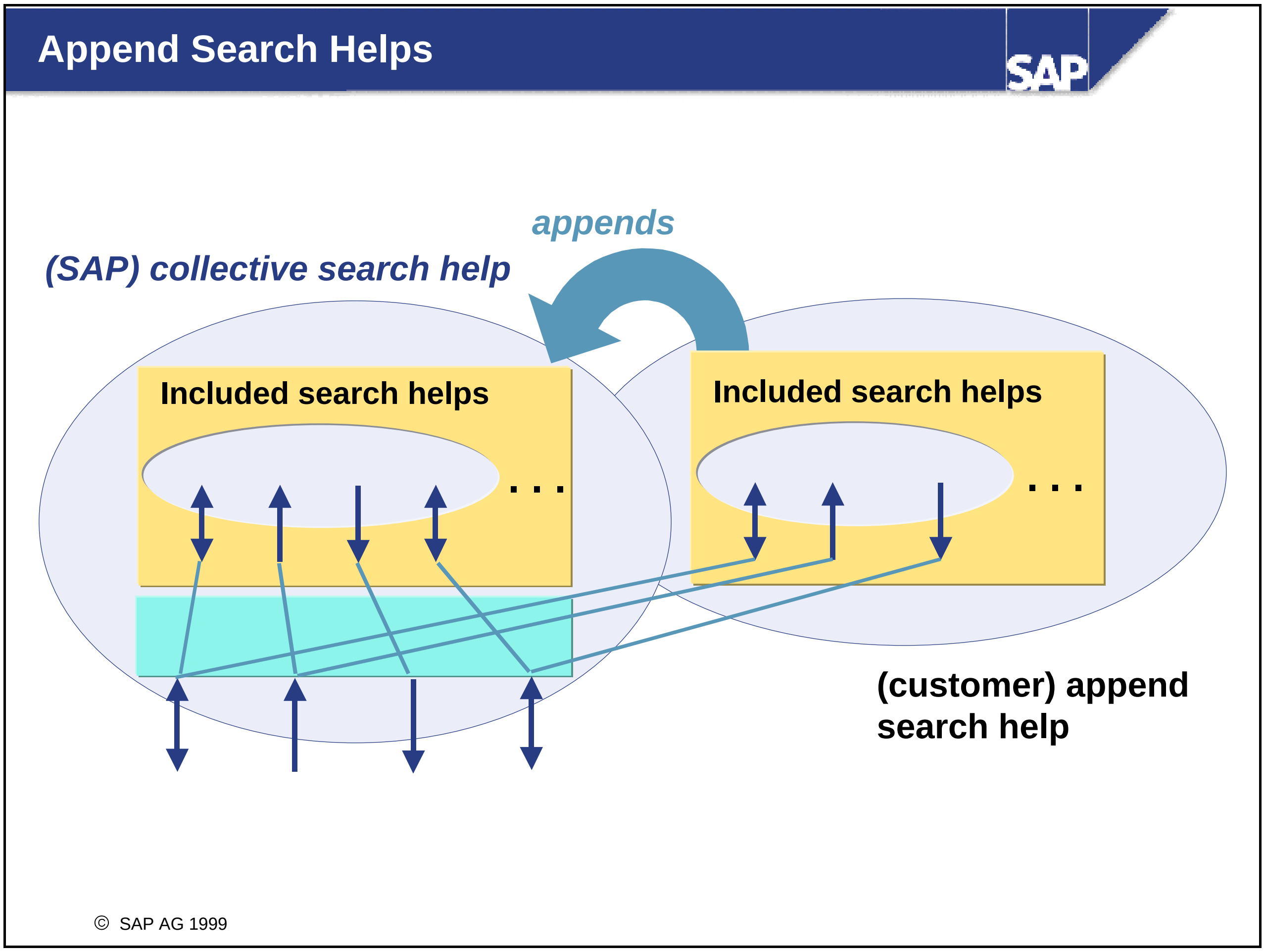
What was the booking number for my flight to New York?
What bookings were made in our travel agency?

- In the relational data model, entities are usually represented by formal keys. In real life, however, these entities are often identified by one or more of their attributes. For example, the key for a person is the personnel number. A person will generally describe another with his name and possibly his address.
- The attributes used to identify an entity can differ from one user to the next and from situation to situation. A user wants to use these attributes in an input help to define a value for a field that requires that a formal key be entered.
- We therefore need **search paths** permitting access to the data using non-key fields. Several different search paths should be possible for one field.
- A search path for a field can be implemented with a search help having the form described above. To describe an input help with more than one alternative search path, a set of search helps can be combined into a new object in the R/3 System. Since this object is the description of the input help for a field, it is also called a search help.
- In contrast to the **elementary search helps** described above, the search helps that combine several search paths are called **collective search helps**.
- Collective search helps are sometimes used to map the distribution of the possible entries for a field into several (disjunct) datasets.

Collective Search Helps and Elementary Search Helps

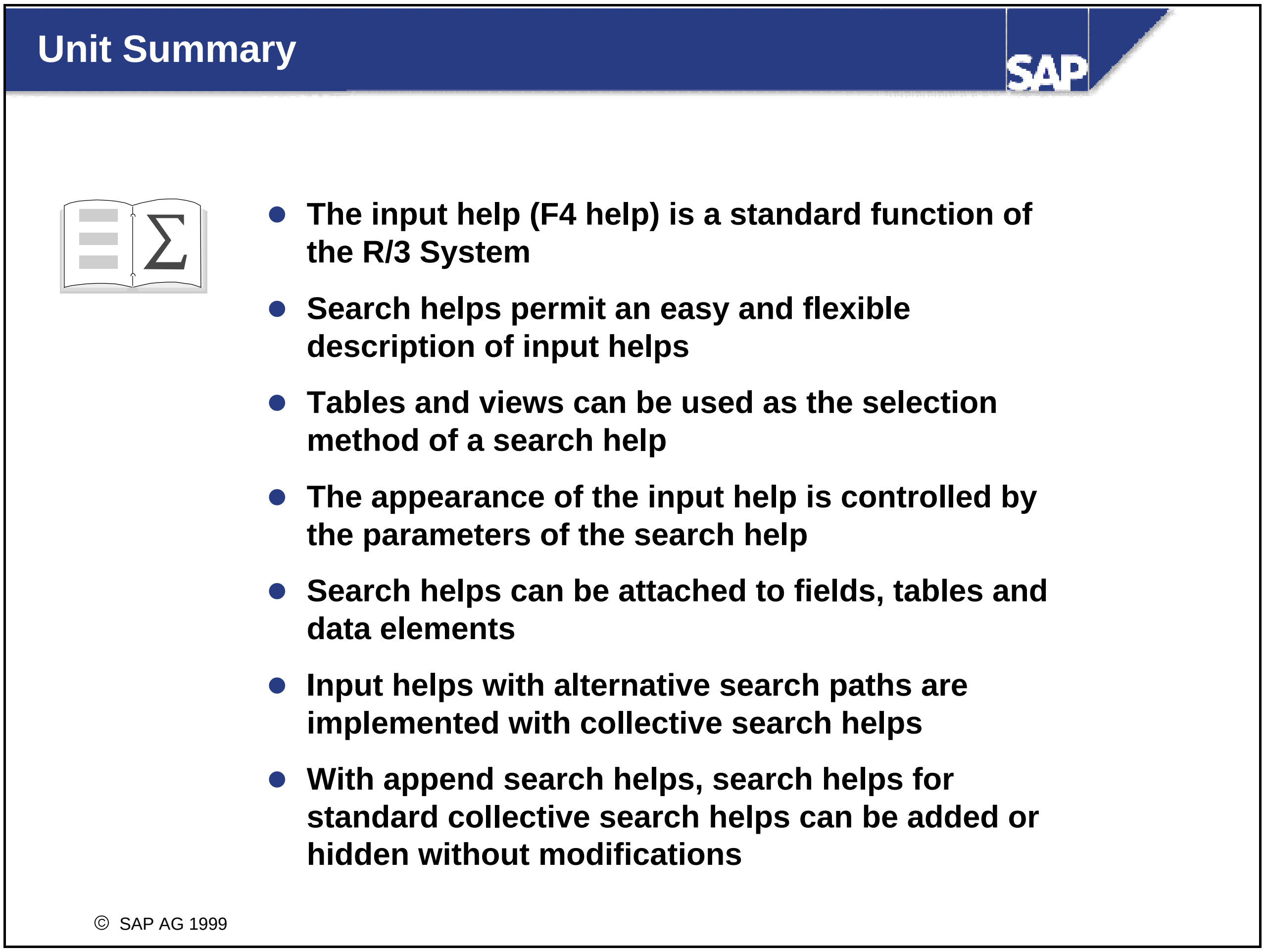


- Like an elementary search help, a collective search help has an interface of IMPORT and EXPORT parameters with which it exchanges data. Using this interface, the collective search help can be attached to fields, tables and data elements exactly like an elementary search help.
  Only one search help can be attached to a field, table or data element. Several search paths are therefore attached with a collective search help.
- You can omit the components for describing the dialog behavior and data selection when you define a collective search help. The included search helps are listed here. You must assign the parameters of the collective search help to the interface parameters of the included search help for each inclusion.
- A search help can also be included in several collective search helps and at the same time itself be attached to fields, tables and data elements. A collective search help can also be included in another collective search help.
- When you use a collective search help, you are offered the elementary search helps contained in the collective search help as parallel tab pages. If you repeatedly use a collective search help, the tab page that was last used is automatically active. This is because most users always use the same search path.



Append Search Helps

- The set of search paths that are meaningful for an object greatly depends on the particular circumstances of the SAP customer. The customer often would like to enhance the standard SAP collective search helps with his own elementary search helps. Release 4.6 provides an append technique that permits the enhancement of collective search helps without modifications.
- An **append search help** is a collective search help that is assigned to another collective search help (its appending object) and that enhances it with the search helps it includes. The append search help uses the interface of its appending objects.
- The append search help lies in the customer namespace. Normally the search helps included in the append search help are also created by the customer and lie in the customer's namespace. However, the required elementary search help might already be provided by SAP, in which case the customer only has to add it to his own append search help.
- Append search helps are used with SAP to improve component separation. Some SAP collective search helps therefore already have one or more append search helps in the standard search help. Customer enhancements should always be made by creating a separate append search help.
- SAP collective search helps often contain elementary search helps that are not required by all customers. The search helps you do not need can be hidden using an append search help. To do this, the corresponding search help must be included in the append search help and the *hidden* flag must be set.



Unit Summary

- The input help (F4 help) is a standard function of the R/3 System
- Search helps permit an easy and flexible description of input helps
- Tables and views can be used as the selection method of a search help
- The appearance of the input help is controlled by the parameters of the search help
- Search helps can be attached to fields, tables and data elements
- Input helps with alternative search paths are implemented with collective search helps
- With append search helps, search helps for standard collective search helps can be added or hidden without modifications

Exercises: Search Helps

Unit: Search Helps

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

- Implement input helps with elementary search helps
- Apply the different features of search help attachments in the ABAP Dictionary
- Define input helps with more than one search path using collective search helps
- Add or remove search paths for collective search helps without modifications

Many management tasks require that you search for employee data. Suitable search options are needed to do this. Such search options will be implemented in this exercise.

8-1 Go to the display screen for table ZDEPMENTxx and call *Utilities* → *Table contents* →*?Create entries*. An input template appears in which you can create new entries for table ZDEPMENTxx (i.e. new departments). The head of the new department should also be defined here. Make this entry in field *Department head*. Maintenance of this field should be supported with an input help that displays the (personnel number of the) employee.

Verify that the field already has an input help. Find out which input help mechanism is used here.

The objective is to make the input help for check table ZEMPLOYxx more user-friendly. Check your success by calling the input help again.

To do this, create an elementary search help ZEMPLOYxx.

The following attributes should appear in the specified order in the hit list:

- Carrier
- First name
- Last name
- Personnel number
- Department code

Because of the large number of employees, you should restrict the displayed values by specifying the first and/or last names of the person wanted before displaying the hit list. Keep in mind that the last name is used more frequently as a restriction than the first name.

If a carrier was already specified before the input help was called, only its employees are offered. Otherwise an input field on the input template for the carrier will be filled when the employee is selected.

Make sure that the search help defined for the check table help of table ZEMPLOYxx is used and check your success as described above.

8-2 Verify that the input help for field ZFLCREWxx-EMP_NUM is defined with the search help you just created by calling *Create entries* for table ZFLCREWxx. Understand the underlying mechanism.

The input help for field ZFLCREWxx-EMP_NUM shows all employees. However, you only want to look at flight personnel for the given field. The objective is to correct this.

You should therefore create a search help ZEMPLOY_FLYxx that only displays flight personnel. The display attributes of the search help should be identical to those of search help ZEMPLOYxx. However, since the group of flight personnel is not too large, the hit list can be displayed immediately in this case. The user should be able to limit the employees with their first and last names from this list.

Using a suitable search help attachment, make sure that the search help is used for the given field and check your success as described.

Consider whether you can use work from previous exercises here. Is it necessary or sensible to create your own view for this exercise?

8-3 **Supplementary Exercise:** With *Create entries,* verify that the fields containing the personnel number of the last person to make the change in tables ZEMPLOYxx and ZDEPMENTxx do not have their own input help.

The objective here is to define a suitable input help for these two fields. The two search helps defined so far cannot be used because table changes can only be made by administrative employees. The input help to be defined should therefore display only these.

Define search help ZEMPLOY_ADMxx that displays the administrative employees of the airlines. The search help should have the same display attributes as ZEMPLOY_FLYxx. However, it is not easy to estimate the number of administrative employees. Make sure that the search help directly displays the values found if there are no more than 100. Otherwise you should first offer a search template in which you can define the employee's airline as well as the first and last names.

Attach the search help to the data element. Verify that the search help is now used both in ZEMPLOYxx and in ZDEPMENTxx for the input help of field *Lastchangedby*.

Check whether the requirements for field *Carrier* specified in Exercise 1 are satisfied. How do you explain this effect? Can you get better results by using a different type of attachment?

8-4 You might want to offer further search paths for finding employees. To do this, take the following steps:

- Copy search help ZEMPLOYxx to search help ZEMPLOY_SIMPLExx.
- Convert search help ZEMPLOYxx to a collective search help.
- Include search help ZEMPLOY_SIMPLExx in search help ZEMPLOYxx.

Check if the input help of field ZDEPMENTxx-Supervisor changed due to these operations.

8-5 You really have to enhance the input help for the employees with another search path: This search path should give you an overview of all the employees involved in a flight.

To do this, create another elementary search help ZEMPLOY_FLIGHTxx.
When you use this search help, the user can limit the search to the flight personnel for certain flights before displaying the possible values. The flight should be identified by its cities of arrival and departure and by its flight date.
The carrier should be defined as in Exercise 1.
The following information should appear in the hit list for the search help:

- Carrier
- First name
- Last name

- Personnel number
- Flight
- Date of flight

Make sure that the search help you created is available as an alternative search path for finding employees and verify your results.

Use your solutions from previous units.

8-6 Your system has special requirements when searching for employees:

8-6.1.1 An additional search path that only offers flight personnel should be offered.

8-6.1.2 The search path with which employees can be found by their flights is not required.

Change the input help for field ZDEPMENTxx-Department head accordingly without modifying search help ZEMPLOYxx (or a table that is involved).

8-7 **Supplementary Exercise:** Call the function *Create entries* for table ZEMPLOYxx. Verify that the check table help for table ZDEPMENTxx is used for field *Department code*. Find out where the displayed text field comes from. Check the behavior of the input help for field *Carrier* on the input template.

Now enhance the check table help just tested so that the telephone number of the department appears in the hit list in addition to the information already displayed. Take the necessary steps and check your success as usual.

8-8 **Supplementary Exercise:** This exercise demonstrates the use of help views (see appendix).

Enhance the search help defined in Exercise 7 so that the last name of the head of the department also appears in the hit list. Make sure that those departments that have no description in the user's logon language or for which the *Department head* field is empty are also displayed.

Make sure that the column header with the last names of the department heads is called 'Department head' and not 'Last name' in the hit list. Use data element S_HEAD.

Check your success in the usual manner.

Solutions: Search Helps

Unit: Search Helps

8-1 Starting with the maintenance transaction for table ZDEPMENTxx, call the F4 help as described. Choose *Techn. info* (with the right mouse button) in the hit list. In *Search help* you can find out that the input help is the check table help for table ZEMPLOYxx and that it is a pure check table help (without a search help and without a text table).

To create search help ZEMPLOYxx:

1) Choose *Search help* in the initial screen of the ABAP Dictionary and enter ZEMPLOYxx in the corresponding field.

2) Choose *Create*. In the next dialog box, confirm that you want to create an elementary search help.

3) Enter a short text for your search help.

4) The search help should support the search for employees. These are managed in table ZEMPLOYxx. You therefore have to select this table (or a view on this table) as selection method. The table is sufficient for this exercise. Enter it in field *Selection method*.

5) To obtain the required behavior, choose dialog type *Complex dialog with value restriction*.

6) Choose the search help parameters using the F4 help. You should retain the hit list with the possible search help parameters by selecting *Hold list*, since you don't have to call the help again in this case. Select fields *Carrier*, *First name*, *Last name*, *Personnel number* and *Department code* as parameters.

7) Mark all parameters as EXPORT parameters (column *EXP*). Mark the attribute to be searched for (i.e. *Personnel number*) and the hierarchically higher *Carrier* as IMPORT parameters (Column *IMP*). This ensures that a corresponding entry in the input template is taken into consideration (as described in the exercise).

8) You can define the hit list by assigning the corresponding position numbers (e.g. 1, 2, 3, 4, 5) in column *LPos*.

9) You can define the dialog box for restricting values by assigning position numbers in column *SPos*. You should therefore enter positive numbers in these columns for parameters *First name* and *Last name*, where the value of *Last name* should be smaller than that of *First name*.

10) Activate your search help. The search help is not yet effective for field ZDEPMENTxx-Department head. However, you can try out the search help immediately with the *Test* function.

The search help you just created can only improve the check table help of table ZEMPLOYxx (and thus the input help of field ZDEPMENTxx-Department head) if it was attached to table ZEMPLOYxx. You can do this as follows:

1) Go to change mode in the maintenance screen for this table. Choose *Goto → Search help → For table*. In the next dialog box, enter the name of search help ZEMPLOYxx. Choose *Continue*.

2) The proposal created by the system for assigning the search help parameters to the key fields of the table is probably correct. Check this and copy the definition. Activate table ZEMPLOYxx.

3) Call the *Create entries* function for table ZDEPMENTxx again. The input help of field *Department head* should now behave as desired. If you call *Techn. info* again, you can confirm that the search help you just defined is in effect.

8-2 Call the input help as described. With *Techn. info* you can verify that search help ZEMPLOYxx is really in effect and that this is because table ZEMPLOYxx is also check table of field ZFLCREWxx-EMP_NUM.

The search help to be created for the flight personnel should be very similar to the search help for all employees. It would therefore make sense to copy search help ZEMPLOYxx to search help ZEMPLOY_FLYxx and then modify it. Alternatively. search help ZEMPLOY_FLYxx can be created analogously to the method described above, also making the following changes:

- The short text for search help ZEMPLOY_FLYxx should be adjusted to suit its meaning.
- Change the dialog type to *Immediate value display (Dropdown)*.

The two changes, however, do not solve the main problem in this exercise, namely that the search help should display only flight personnel. You can do this by selecting a view that only contains flight personnel as selection method.
You already defined such a view in Exercise 7-1. Entering this view as selection method and activating the search help will solve the problem. However, this assumes that you named the view fields the same as the underlying fields of table ZEMPLOYxx. Otherwise you have to adjust the names of the search help parameters to the names of the view fields.

The solution just described, however, has one disadvantage. View ZEMPFLYxx is defined with a join on tables ZEMPLOYxx and ZDEPMENTxx. However, only information from table ZEMPLOYxx is needed for the search help. An unnecessarily complex database query is therefore created when you use search help ZEMPLOY_FLYxx. This can have a negative effect on the performance of the input help.

You should therefore create a new view having only base table ZEMPLOYxx. This view can be obtained for example by copying it from view ZEMPFLYxx. You then have to remove base table ZDEPMENTxx from this copy. Join conditions and view fields referring to this table are also deleted.
This view should now be entered as selection method for search help ZEMPLOY_FLYxx.

The search help only becomes effective for field ZFLCREWxx-EMP_NUM when it has been attached. Go to change mode in the maintenance screen for table ZFLCREWxx. Position the cursor on field EMP_NUM. Choose *Goto →?Search help →?For field*.
In the next dialog box, enter the name of search help ZEMPLOY_FLYxx. Choose *Continue*.
The proposal created by the system for assigning the search help parameters to the fields of the table is probably correct. Check this and copy the definition.
Activate table ZFLCREWxx.

Check your success as described.

Note: Of course you are not recommended to attach search help ZEMPLOY_FLYxx to table ZEMPLOYxx. This would have the desired effect for field ZFLCREWxx-EMP_NUM. However, only flight personnel would be offered for all other fields to be checked against table ZEMPLOYxx as well. This, however, is a senseless restriction for example for field ZDEPMENTxx-Department head

Note: Using the default values for search help parameters described in the appendix, you can also define the required search help ZEMPLOY_FLYxx without using a view at all. Keep selection method ZEMPLOYxx. Include *Area* as an additional parameter in the search help. Leave columns *IMP*, *EXP*, *LPos* and *SPos* empty for this parameter. Enter the value 'F' (including the apostrophes) in column *Default value*. The search help thus defined also does what you desire.

There is also a way to modify search help ZEMPLOYxx so that it can be used for the desired function without detracting from the results of Exercise 1. Parameter *Area* must be added to

search help ZEMPLOYxx here. It must be marked as an IMPORT parameter (mark column *IMP*, leave all other columns empty). You can now attach search help ZEMPLOYxx to field ZFLCREWxx-EMP_NUM. When you assign the fields to the search help parameters, you have to assign the constant 'F' (including the apostrophes) to parameter *Area*.
After these actions, the input help of field ZFLCREWxx-EMP_NUM will function as desired, whereas the input help of field ZDEPMENTxx-Department head is not affected by these changes.
This solution, however, would not result in a search help for the flight personnel. You would have to do this again when you solve Exercise 8-6.

8-3 Check if an input help exists for the fields as described. If it does not exist, it is possible that no foreign key was defined for these fields.

Search help ZEMPLOY_ADMxx to be defined should be very similar to search help ZEMPLOY_FLYxx. You should therefore create it by copying and then make the following changes:

- The short text for search help ZEMPLOY_ADMxx should be adjusted to suit its meaning.
- Change the dialog type to *Dialog depends on set of values*.
- Since the airline should also appear in the dialog box for restricting values, there must be an entry in column *SPos* for the corresponding parameters. This parameter must be lower than both existing parameters. If necessary, increment them.

You also have to make sure that the search help only displays administrative employees. It is best if you copy the view created in the previous exercise and replace the value 'F' in the copy with 'A' in the selection condition. After activating this view you can enter it as selection method for search help ZEMPLOY_ADMxx.
Activate the search help.

The search help will be effective for the two fields if you go in change mode to the maintenance transaction of the data element you created in Exercise 2-3 for *Last changed by*. In *Search help,* enter ZEMPLOY_ADMxx in field *Name.* In field *Parameter,* enter the name of the field for the personnel number (can be selecting with the F4 help).
Activate the data element.

Check your success in the usual manner. Copying the airline does not function correctly in both directions in this case. To check this, enter a value in field *Carrier* before calling the input help. It is not used in the input help. Vice versa, selecting a value for *Last changed by* does not update the airline.
This effect can be explained in that it is not possible to take further parameters into consideration when attaching a search help to a data element. In the present case, attaching the search help to field ZCHANGExx-Changer would not have corrected this error since field *Carrier* is not contained in structure ZCHANGExx. This field therefore could not have been taken into consideration in the attachment.

8-4 Proceed as follows:

1) Copy search help ZEMPLOYxx to search help ZEMPLOY_SIMPLExx and activate the new search help.
2) In change mode, go to the maintenance screen for search help ZEMPLOYxx. Choose *Edit → Change search help type* and confirm it in the next dialog box.
3) Click on tab page *Included search helps*. Enter search help ZEMPLOY_SIMPLExx.
4) Position the cursor on the search help just entered. Choose *Parameter assignment*. Have the system make a proposal for the assignment.
5) The proposal is probably correct. To be on the safe side, check it and then copy it.
6) Activate search help ZEMPLOYxx.

By calling the input help for field ZDEPMENTxx-Department head you can see that the input help is still functioning. With *Techn. info* you can verify that a collective search help is now in effect.

8-5 You already defined a suitable selection method (view ZCREWSxx) for the new elementary search help ZEMPLOY_FLIGHTxx in Exercise 7-2. You can now proceed as follows:

1) In the initial screen of the ABAP Dictionary select *Search help*. Enter the name ZEMPLOY_FLIGHTxx in the corresponding field and choose *Create*. In the next dialog box, confirm that you want to create an elementary search help.

2) Enter a short text. Choose *Complex dialog with value restriction* as dialog type.

3) Enter ZCREWSxx as selection method.

4) Choose the following search help parameters using the F4 help: *Carrier*, *First name*, *Last name*, *Personnel number*, *Flight number*, *Flight date*, *Departure city* and *Arrival city*.

5) Mark all parameters as EXPORT parameters (column *EXP*). Mark *Carrier* and *Personnel number* as IMPORT parameters (Column *IMP*).

6) Assign position numbers for the parameters in column *LPos*. Leave this column empty for parameters *Departure city* and *Arrival city*.

7) Assign position numbers for parameters *Departure city*, *Arrival city* and *Flight date* in column *SPos*.

8) Activate the search help ZEMPLOY_FLIGHTxx.

You now have to include search help ZEMPLOY_FLIGHTxx in collective search help ZEMPLOYxx. Proceed as follows:

1) In change mode, go to the maintenance screen for search help ZEMPLOYxx. Click on tab page *Included search helps*.

2) Enter search help ZEMPLOY_FLIGHTxx directly below search help ZEMPLOY_SIMPLExx in the list of search helps.

3) Position the cursor on the search help just entered. Choose *Parameter assignment*. In the next dialog box, confirm that you want to create a proposal for the parameter assignment.

4) The parameter assignment proposed by the system is probably correct. Check this and copy the assignment.

5) Activate search help ZEMPLOYxx.

You can check your success as usual by calling the input help for field ZDEPMENTxx-Department head.

8-6 Since you want to make the changes without modifying existing objects, you have to create an append search help for collective search help ZEMPLOYxx. Proceed as follows:

1) In display mode, go to the maintenance screen for search help ZEMPLOYxx. Choose *Goto → Append search helps*.

2) A name for the append search help is proposed in the next dialog box. You can copy this name.

3) Enter a short description for the append search help.

4) Click on tab page *Included search helps*.

5) Enter ZEMPLOY_FLYxx and ZEMPLOY_FLIGHTxx in the list of included search helps. Mark column *Hidden* for the second entry.

6) Position the cursor on the name of search help ZEMPLOY_FLYxx. Choose *Parameter assignment*. In the next dialog box, confirm that you want to create a proposal for the parameter assignment.

7) The parameter assignment proposed by the system is probably correct. Check this and copy the assignment.

8) Activate your append search help.

You can check your success as usual by calling the input help for field ZDEPMENTxx-Department head.

8-7 Call the input help for field ZEMPLOYxx-Department code as described. With *Techn. info* you can see that the input help is determined with check table ZDEPMENTxx of this field. You can also see that there is a text table for the check table.

In this case too, entries that already exist in field *Carrier* are taken into consideration in the F4 help. Similarly, field *Carrier* is updated when a value is selected from the hit list.

To make the required enhancement, you must create an elementary search help and attach it to table ZDEPMENTxx. Since all the data to be used in the input help are contained in table ZDEPMENTxx and its text table ZDEPMENTTxx, table ZDEPMENTxx can be used as the selection method of this search help.
Proceed as follows:

1) In the initial screen of the ABAP Dictionary select *Search help*. Enter a name for the search help to be created in the corresponding field.
2) Choose *Create* and confirm that you want to create an elementary search help in the next dialog box.
3) Enter a short text for your search help.
4) Enter ZDEPMENTxx as selection method.
5) With the input help, select search help parameters *Carrier*, *Department code, Description* and *Telephone*.
6) Mark all parameters as EXPORT parameters (column *EXP*). Mark parameters *Carrier* and *Department code* as IMPORT parameters (Column *IMP*).
7) Assign position numbers in the hit list for the parameters in column *LPos*.
8) The check table help can provide upon request a dialog box for restricting values having the fields *Carrier* and *Department code*. You can retain this behavior by assigning position number for these two parameters in column *SPos*.
9) Activate the search help.
10) Go to change mode in the maintenance screen for table ZDEPMENTxx. Choose *Goto* → *Search help* → *For table*.
11) In the next dialog box enter the name of the search help you just created and choose *Continue*.
12) The proposal created for assigning the search help parameters to the key fields of table ZDEPMENTxx is probably correct. Check this and copy the assignment.
13) Activate table ZDEPMENTxx.

Check your success in the usual manner.

8-8 You can find the last names of the department heads in table ZEMPLOYxx. The data of the search help must be selected with the three tables ZDEPMENTxx, ZDEPMENTTxx and ZEMPLOYxx.
You must therefore select a view as selection method of the search help. The exercise states that this view must implement an outer join. You must therefore choose a help view.
To define the help view:

1) In the initial screen of the ABAP Dictionary select *View*. In the corresponding field, enter a name beginning with the prefix H_Z for the help view.
2) Choose *Create* and confirm that you want to create a help view in the next dialog box.
3) Enter a short text for the help view.
4) Enter ZDEPMENTxx in the only input field in the area *Tables.*
5) Position the cursor on the table names just entered and choose *Relationships*. In the next dialog box mark the relationship to table ZEMPLOYxx under *Referenced tables* and the relationship to table ZDEPMENTTxx under *Dependent tables*. Copy this selection.
6) Click on tab page *View fields*. Some fields that you should not change are already entered here. You have to include the following fields in the view using the *Table fields* function: ZDEPMENTxx-Telephone, ZDEPMENTTxx-Description and ZEMPLOYxx-Last name.
7) Activate the help view.

You can now adjust the search help to the additional requirements. Proceed as follows:

1) Go to the maintenance screen for the search help created in the previous exercise. Replace ZDEPMENTxx with the help view you just created in field *Selection method*.

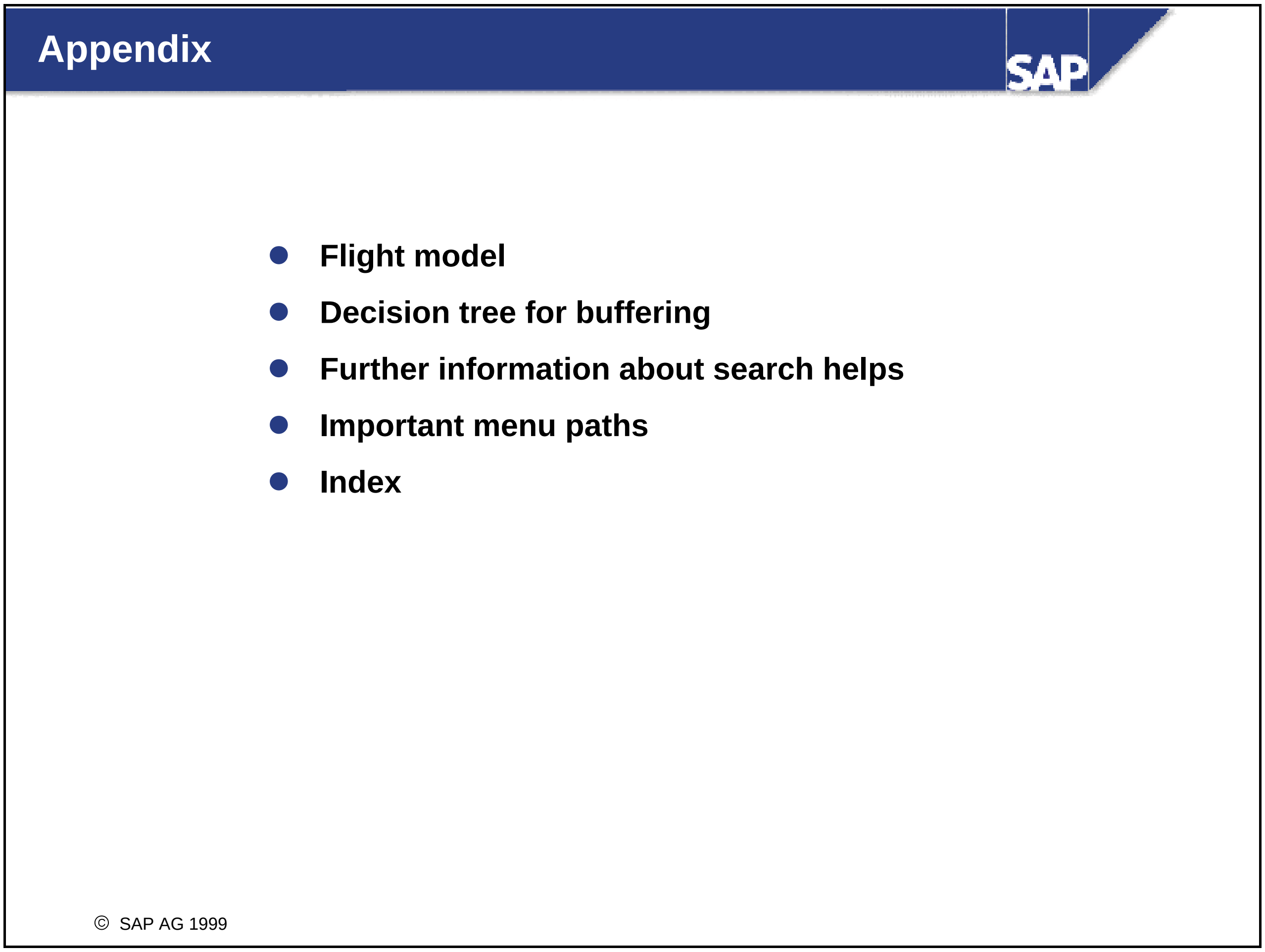
2) Choose the additional search help parameter *Last name* using the input help. Assign it a position in the hit list in column *LPos*. Note that this column may not contain a duplicate (positive) number. You might therefore have to adjust the position numbers of the other parameters.

3) To assign the desired title in the hit list of the column containing the last names of the department heads, mark column *Modified* for parameter *Last name* (it is to the right of column *Data element*). You can now enter values for the data element of this parameter. Replace the entered data element S_LNAME with S_HEAD.

4) Activate the search help.

Check your success in the usual manner.

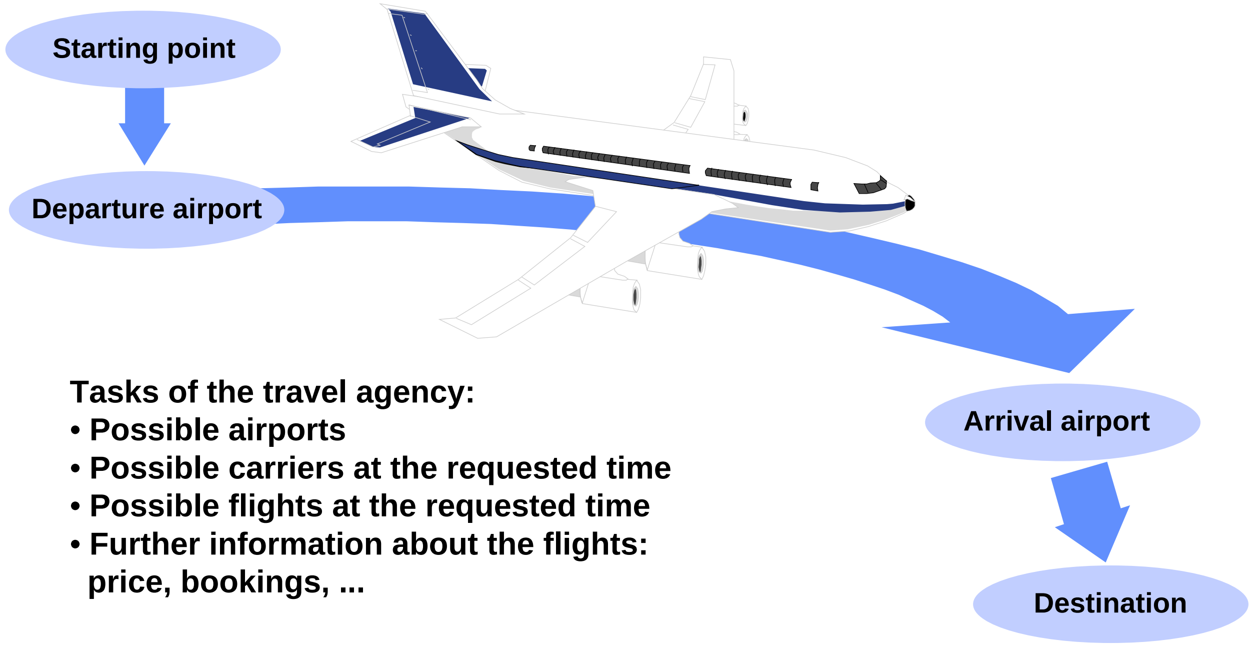
Note: You can already specify the alternative data element for column *Last name* when you define the help view. To do so you must mark column *Mod* for field *Last name* in the maintenance screen for the view fields of the help view. You can then replace data element S_LNAME with S_HEAD. In this case you can leave out step 4 in the above description.



Appendix

- Flight model
- Decision tree for buffering
- Further information about search helps
- Important menu paths
- Index

Flight Data Model for ABAP Training Courses



Starting point

Departure airport

Arrival airport

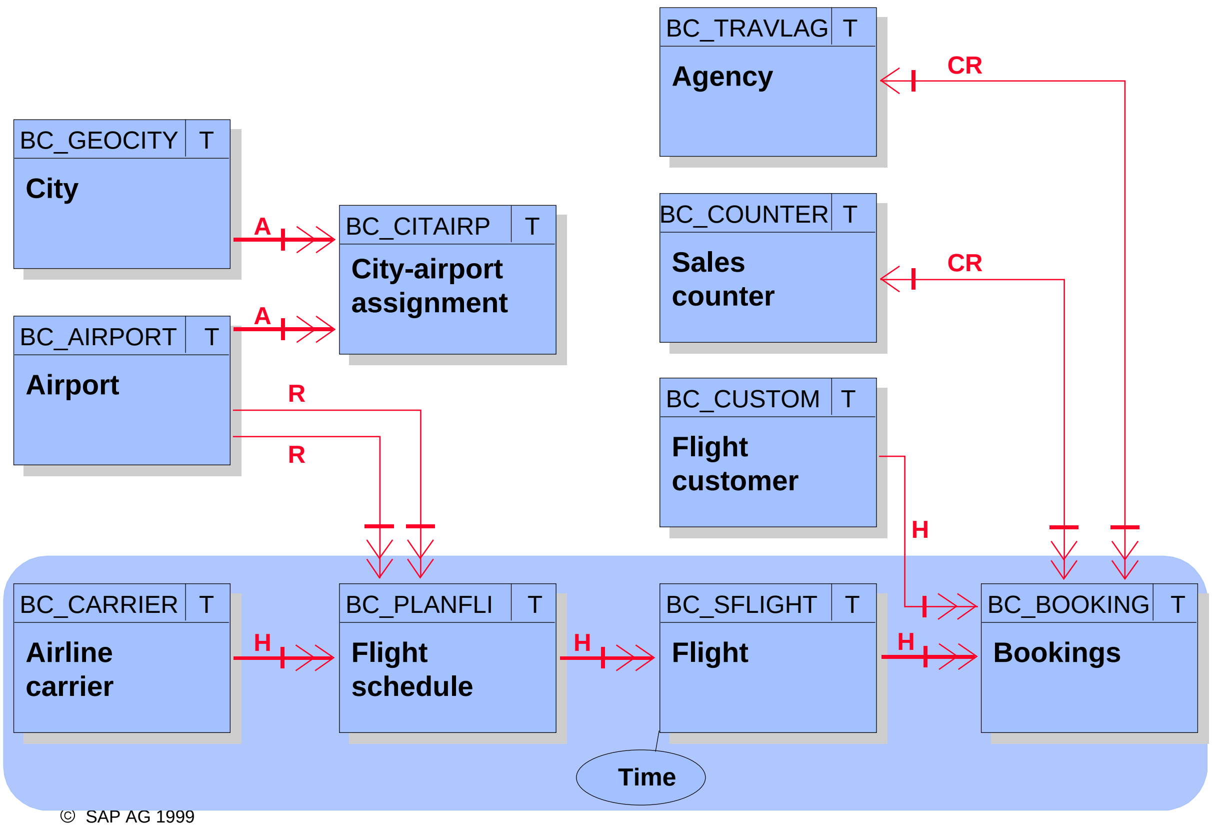
Destination

Tasks of the travel agency:

- **Possible airports**
- **Possible carriers at the requested time**
- **Possible flights at the requested time**
- **Further information about the flights: price, bookings, ...**

- One common flight data model is used in the ABAP training courses. Only a simple overview of this data model is shown here; details can be given if required.
- If you as a customer want to get from one place to another, you want to find out the following from your travel agency ...
- Which are the best airports for the given trip
- At what times are there flights on the given day
- Depending on your individual optimization conditions, what is the optimal solution, e.g. the best flight, the fastest connection, the connection that is nearest to the required arrival time, ...
- This view is different from that of a travel agency: The data is stored in the data model in tables in a central database according to technical criteria.. The customer is not normally interested in all the required data (for example you have to enter which customer booked which flights, when the bookings were made, how much the customer paid ...). The data must be collected with programs depending on the user requirements.

Data Model



BC_TRAVLAG | T
Agency

BC_GEOCITY | T
City

BC_CITAIRP | T
City-airport assignment

BC_COUNTER | T
Sales counter

BC_AIRPORT | T
Airport

BC_CUSTOM | T
Flight customer

BC_CARRIER | T
Airline carrier

BC_PLANFLI | T
Flight schedule

BC_SFLIGHT | T
Flight

BC_BOOKING | T
Bookings

Time

- Entities can be defined for all logically related information: The flight data model therefore contains an entity each for:
  - all cities,
  - all airports,
  - all carriers,
  - all flights,
  - all connections.
- The entities have a certain relationship to one another:
  - Flights start and end at an airport.
  - A connection is uniquely defined by its carrier, departure airport, arrival airport, and flight time
  - Flights can be offered on more than one day for each connection, but a flight can only be for a certain connection.
  - All nearby airports must be assigned to the cities.
- This data model manages all the required data without unnecessary redundancies and gives the agency the data it requires.

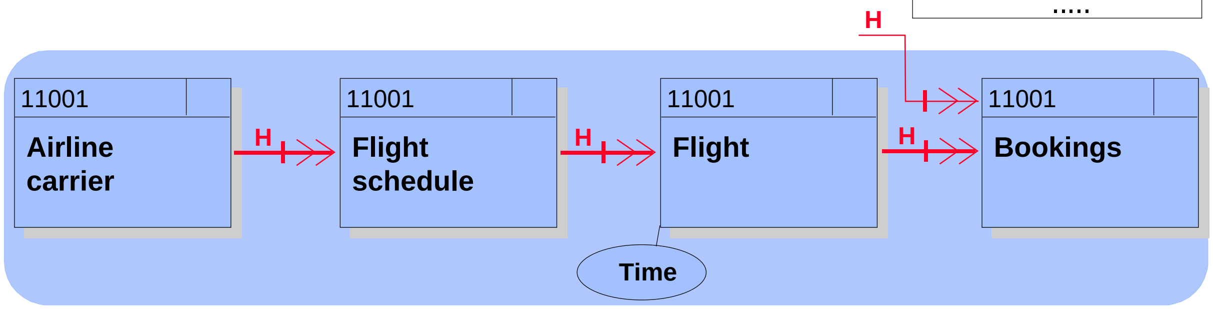
Implementation in the ABAP Dictionary

SCARR
Carrier
🔑 MANDT
🔑 CARRID
CARRNAME
.....

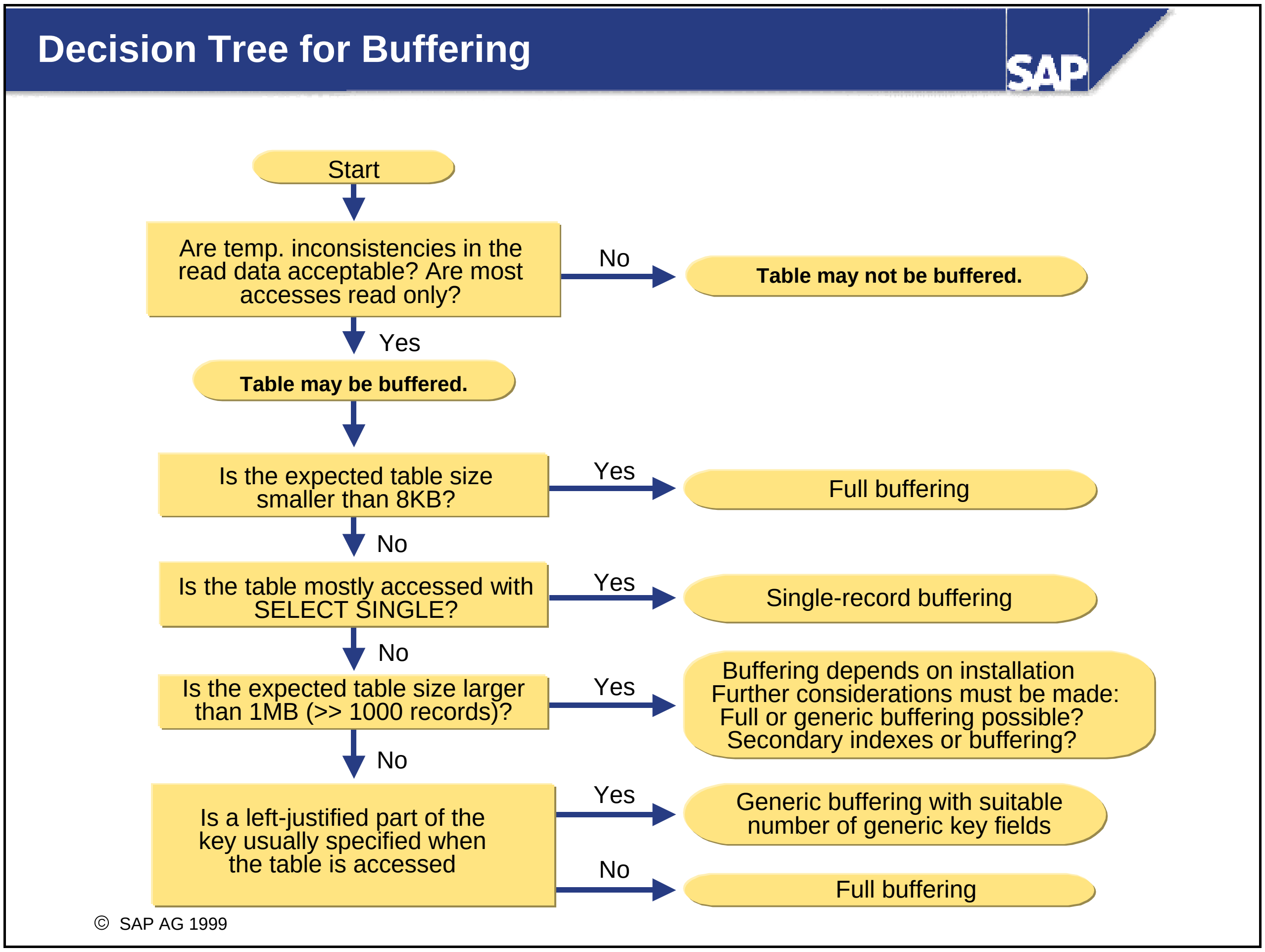
SPFLI
Flight schedule
🔑 MANDT
🔑 CARRID
🔑 CONNID
AIRPFROM
AIRPTO
DEPTIME
.....

SFLIGHT
Flight
🔑 MANDT
🔑 CARRID
🔑 CONNID
🔑 FLDATE
SEATSMAX
SEATSOCC
.....

SBOOK
Flight booking
🔑 MANDT
🔑 CARRID
🔑 CONNID
🔑 FLDATE
🔑 BOOKID
🔑 CUSTOMID
COUNTER
AGENCYNUM
.....



- The examples and exercises in the ABAP training courses and the ABAP documentation refer to the SAP flight data model. The Repository objects for the flight data model can be found in development class SAPBC_DATAMODEL.
- The following tables of the flight data model are mainly used in the ABAP training courses:
  - SPFLI: Table of flight connections
  - SFLIGHT Table of flights
  - SBOOK: Table of flight bookings
  - SCARR: Table of airlines
  - SBUSPART: Table of airline partners
  - STRAVELAG: Table of travel agencies
  - SCUSTOM: Table of flight customers
  - SCOUNTER: Table of sales desks



Decision Tree for Buffering

Start

Are temp. inconsistencies in the read data acceptable? Are most accesses read only?

No → Table may not be buffered.

Yes ↓

Table may be buffered.

Is the expected table size smaller than 8KB?

Yes → Full buffering

No ↓

Is the table mostly accessed with SELECT SINGLE?

Yes → Single-record buffering

No ↓

Is the expected table size larger than 1MB (>> 1000 records)?

Yes → Buffering depends on installation
Further considerations must be made:
Full or generic buffering possible?
Secondary indexes or buffering?

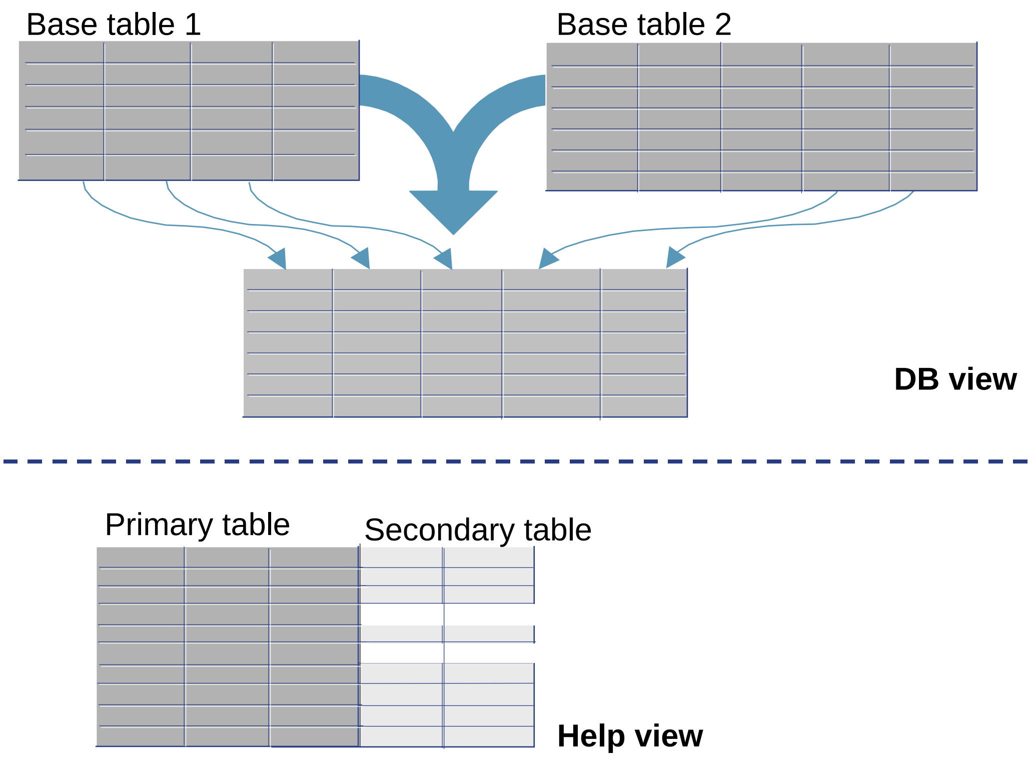
No ↓

Is a left-justified part of the key usually specified when the table is accessed

Yes → Generic buffering with suitable number of generic key fields

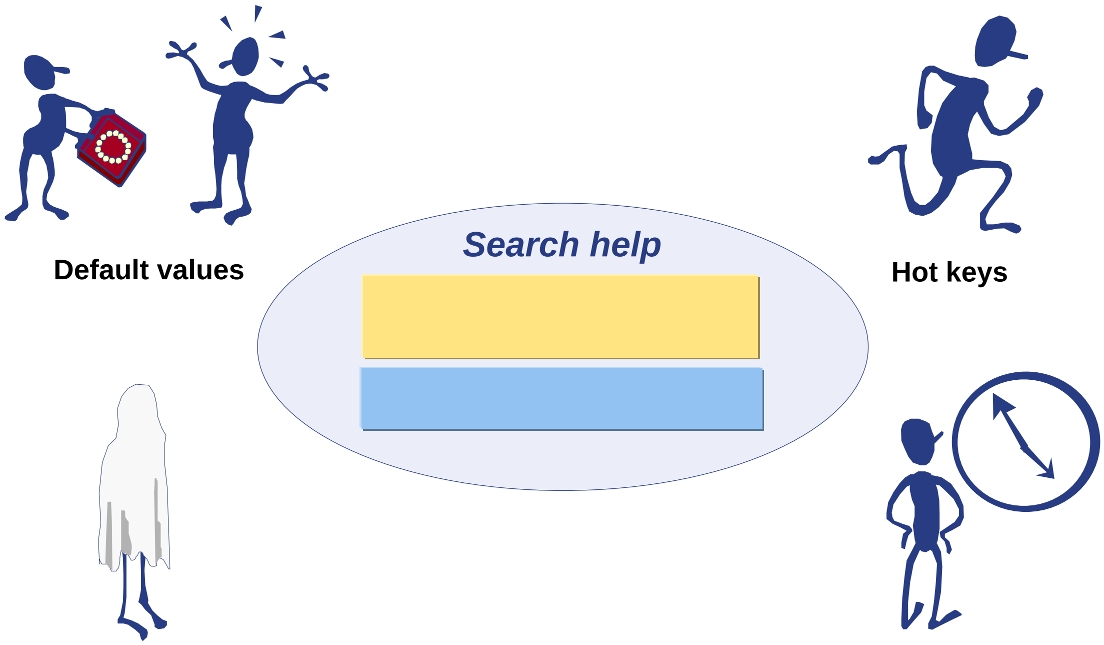
No → Full buffering

View as Selection Method of a Search Help



- If the selection method of a search help is a database view, the input help will only display the records for which there are entries in all the tables involved in the view (inner join). The set of possible entries is sometimes described by the entries in a primary table for which additional optional information can be added from further secondary tables. This view on the data can be implemented with a help view. The same outer join logic is used for help views as for maintenance views.
- A help view is defined analogously to a maintenance view. Help views can only be used as selection methods in search helps. Since the R/3 System cannot pass the selection directly to the database using a help view, it must generate its own access routines. The database view should therefore be used as selection method in preference to the help view.
- Selection using a table and text table corresponds to selection using a virtual help view. For this reason a help view should not be created in this case.
  Exception: The table contains a field having the same name as a non-key field of the text table. If this field of the text table is needed in the search help, you have to create a help view on the two tables because the field in the search help cannot be accessed directly.
- It is customary to start the name of help views with the prefix `H_`. Views beginning with the prefixes `H_Y` or `H_Z` therefore lie in the customer namespace.

Further Options for Search Helps



Default values

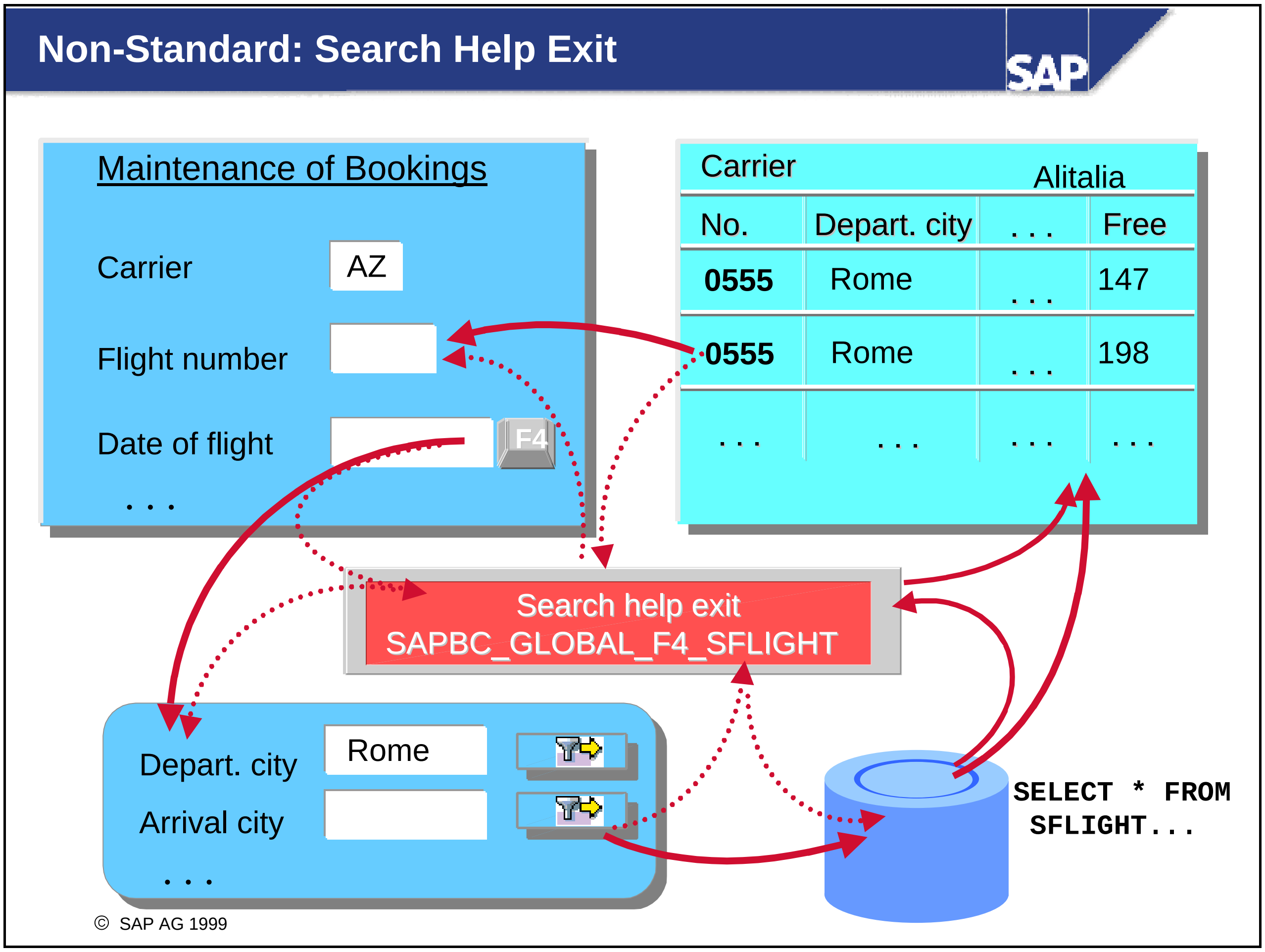
Search help

Hot keys

Hide included search helps

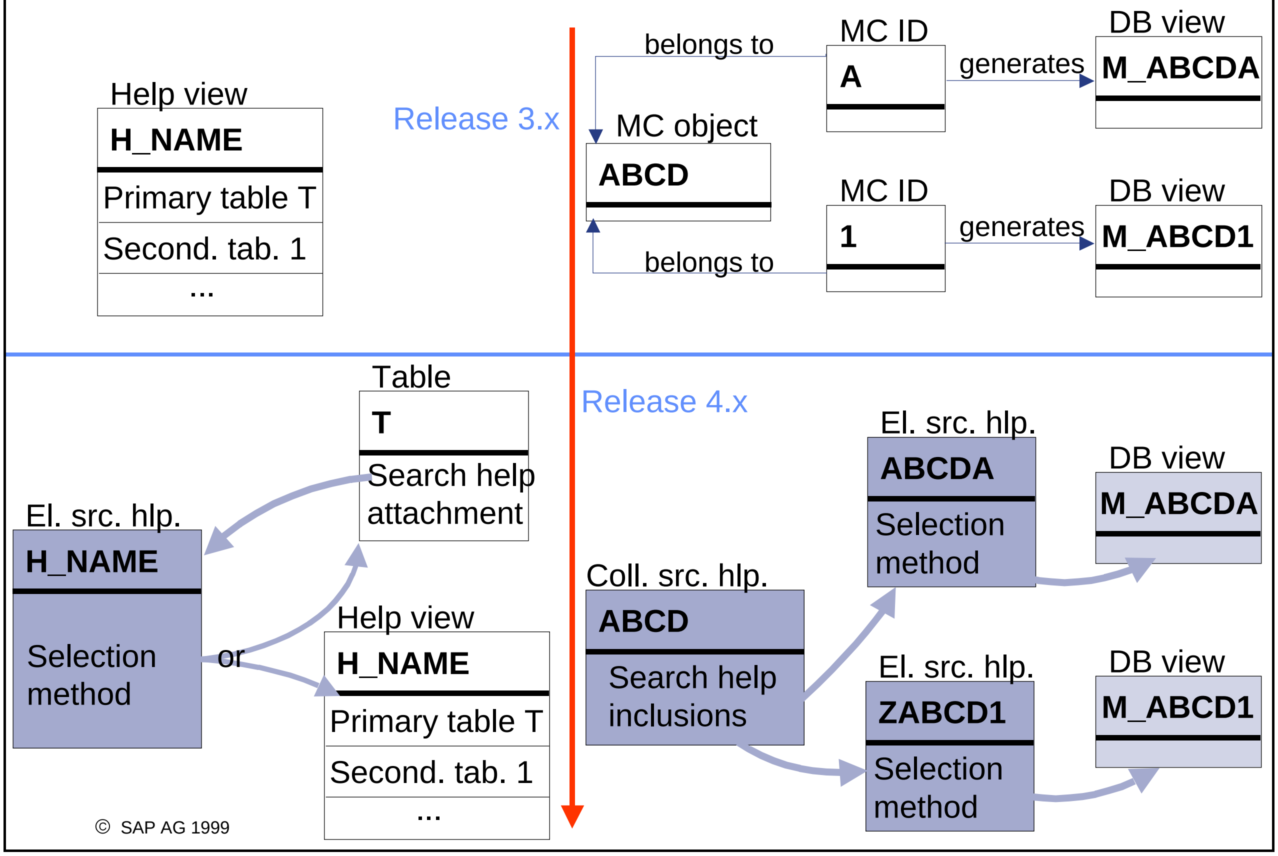
Display fields in the display box for restricting values

- A parameter can be pre-assigned a value by allocating a default value. The parameter is always given this value unless it is an IMPORT parameter that is linked with a field of the screen, its module pool or with a parameter of the including collective search help.
  The following can be default values: literals, system fields and GET parameters.
  You can use a default value to formulate a simple selection condition for a field of the selection method.
- A single letter or a digit can be assigned as hot key to an elementary search help. If this elementary search help is used in a screen field for the input help and if this field has the short cut =<hot key>.<SEL1>.<SEL2>... when the input help is called, this elementary search help is processed. <SEL1>, <SEL2>... is used as the field contents for the dialog box for restricting values (with an '*' added at the end) and the hit list is then displayed immediately.
- Individual search help inclusions can be hidden in a collective search help. You can thus deactivate individual search paths that are not wanted in a system. They should normally be hidden in an append search help as this can be done without modification.
- Parameters of an elementary search help can be marked as pure display fields in the dialog box for restricting values. In general, IMPORT parameters that are assigned unchangeable fields of the screen are displayed in this dialog box as unchangeable.



- A search help is an object that describes an input help within the system-wide standard. In some cases, the special semantics of a field requires that you deviate from this standard in some details. This can be implemented with a search help exit.
- A search help exit is a function module having a standardized interface. The function module F4IF_SHLP_EXIT_EXAMPLE can be used as template. If a search help has such a search help exit, the search help exit is called prior to each single step of the process. The administrative data of the help processor is passed via the interface. The search help exit can manipulate this data.
- The administration data also includes the information about the next step to be executed. The search help exit can now execute either preparatory actions for this step or the step itself (for example a data selection which cannot be implemented with a SELECT on a table or view). In the second case the search help exit also changes the information about the next step to be executed.
- Some function modules that can be used as search help exits or that can be used to manipulate the administration data in search help exits are already defined with the prefix F4UT_.
- Search help exits should only be used for exceptions. Using search help exits encourages non-standard solutions and make it more difficult to maintain the input help.

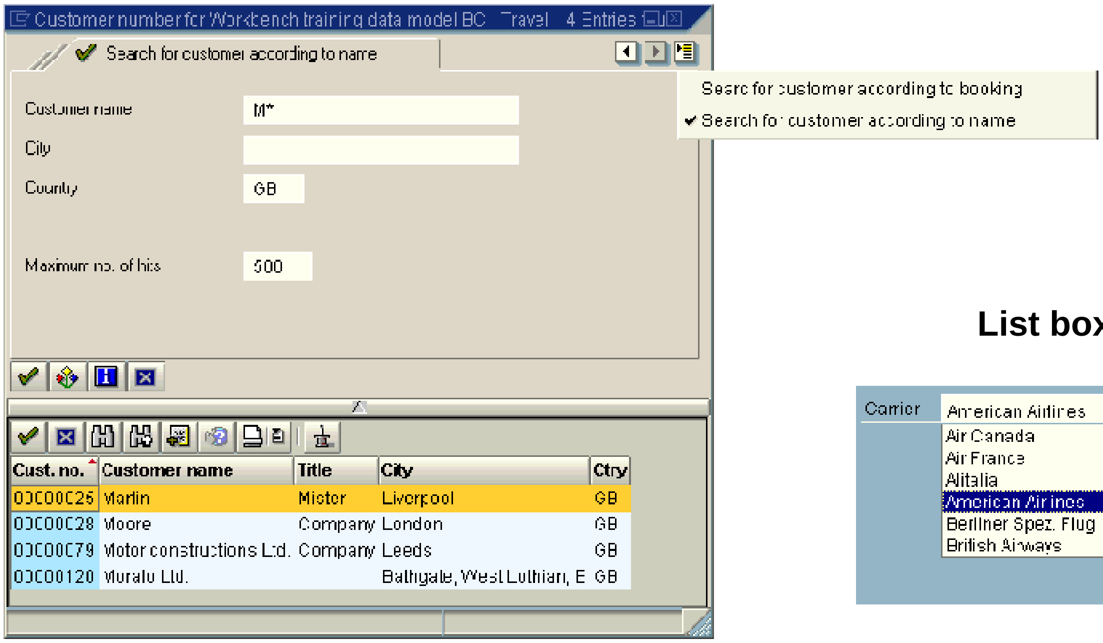
Migration of Existing Input Helps



- Search helps were introduced in Release 4.0. Previously, input helps were implemented by creating matchcodes and help views, which, however, had less functionality. When you upgrade to 4.x, search helps with the same name are created from these objects (if necessary, the name is preceded with a Y or Z). The original objects will initially remain in the system, but be meaningless.
- Prior to Release 4.0, a help view was a complete description of an input help that was automatically attached to its primary table (and only to it). An elementary search help is created from each help view. The primary table of the help view can be entered as the selection method for many cases, and the help view will be used for the others. The search help that is created is attached to the primary table of the help view.
- An elementary search help is created from a matchcode ID. The selection method assigned is the generated database view (for a transparent ID) or the generated pooled table (for a non-transparent ID) of the matchcode ID. In the first case, the generated view is administered as an independent object in the ABAP Dictionary. In the second case, the pooled table is still appended to its matchcode ID since the matchcode technique is used to update the data in this table.
- A collective search help is created from a matchcode object. Matchcodes were attached to input fields in the screen. These attachments are converted to attachments of the created collective search helps to the corresponding screen fields..
- The above slide illustrates the migration for help views and for transparent matchcodes.

Alternative Displays of the Input Help

Input Help Control



Customer number for Workbench training data model BC Travel 4 Entries

✔ Search for customer according to name

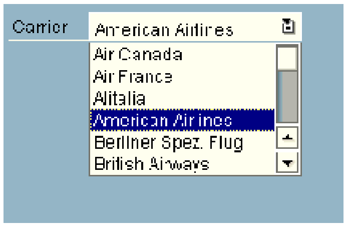
Search for customer according to booking
✔ Search for customer according to name

Customer name: M*
City:
Country: GB

Maximum no. of hits: 500

Cust. no.	Customer name	Title	City	Ctry
00000026	Martin	Mister	Liverpool	GB
00000028	Moore	Company	London	GB
00000079	Motor constructions Ltd.	Company	Leeds	GB
00000120	Muralu Ltd.		Bathgate, West Lothian, E	GB

List box



Carrier: American Airlines

- Air Canada
- Air France
- Alitalia
- American Airlines
- Berliner Spez. Flug
- British Airways

- The R/3 System recognizes three presentation forms for the input help:
  - List box
  - Control (modeless)
  - R/3 dialog (modal)
- The list box does not provide for any further selection conditions and no further columns are displayed in the hit list. However, the list box is the most user-friendly input help for clear one-column lists. The application developer decides whether to offer a field as list box and stores this information in the Screen Painter for the corresponding field. When the user calls the list box, the data to be displayed is obtained with the input help mechanism stored in the ABAP Dictionary or Screen Painter for the field. You can obtain more information on using the list box in the course BC410 *Programming User Dialogs*.
- Alternatively you can display fields that are not offered as a list box with a modeless control or with a modal dialog implemented with R/3 screen technology. With *Help* ® *Settings* each user can define the variant he prefers. This presentation form is then used for all input helps for this user. The system administrator can define the default value.
- The control is particularly useful if more than one field is to be filled sequentially with the same input help (for example in a table control). The *Hold list* function starts the control from the modal help.

Important Menu Paths and Transactions

ABAP Dictionary Maintenance/Display
Tools ? ABAP Workbench ? Development ? Dictionary
- Dictionary maintenance (SE11) - Dictionary display (SE12)

R/3 Repository Information System ABAP Dictionary

1. *Tools ? ABAP Workbench ? Development ? Dictionary (Dictionary object; Enter object name and choose operation)*
   Environment ? R/3 Repository Information System or
2. *Tools ? ABAP Workbench ? Overview ? R/3 Information System*
   - R/3 Repository Information System (SE84) / Repository Information System ABAP Dictionary (SE15)

Data Display/Maintenance (Data Browser)

1. *Tools ? ABAP Workbench ? Development ? Dictionary (Dictionary object:Table, Enter table name and choose operation)*
   Utilities ? Table contents ? Display or *Table contents ? Create entries*
   or
2. *Tools ? ABAP Workbench ? Overview ? Data Browser*
   Data display/maintenance (Data Browser) (SE16)

Index
Tools ? ABAP Workbench ? Development ? ABAP Dictionary (Dictionary object: table; Enter table name and choose operation)
Goto ? Indexes or Pushbutton *Indexes...*

Technical Settings
Tools ? ABAP Workbench ? Development ? ABAP Dictionary (Dictionary object: Table, enter table name and choose operation)
Goto ? Technical settings or pushbutton *Technical settings (SE13)*

Database Utility
Tools ? ABAP Workbench ? Development ? ABAP Dictionary (Dictionary object: Enter object name and choose operation)
Utilities ? Database utility
- Database utility (SE14)

Storage Parameters
Tools ? ABAP Workbench ? Development ? ABAP Dictionary (Dictionary object: Enter object name and select operation)
Utilities ? Database utility
Goto ? Storage parameters* or pushbutton *Storage parameters
- Database utility (SE14) ? *Goto ? Storage parameters*

Fixed Values
Tools ? ABAP Workbench ? Development ? ABAP Dictionary (Dictionary object: domain; Enter domain name and choose operation)
Tab page Value range

Maintenance View
Tools ? ABAP Workbench ? Development ? Other tools ? Table maint. view
- Table view generation (SE54)

Enhanced Data Display
System? Utilities ? Table maintenance ? Ext. table maint.
- Extended table maintenance (SM30)

Index

Dependent object 5-5
Activation 5-3
ALTER TABLE 6-4, 6-13
Analysis tool 6-11
Continue adjustment 6-11
Append structure 6-12, 6-13, 6-14, 7-12
Append search help 8-15
Restart log 6-9
Cluster table 6-14
Data class 2-8, 2-9
    customer 2-9
Database 1-4, 2-3, 3-4
Database object 1-3, 1-7
Database system 3-4, 6-4, 7-11
Database table 1-3, 2-6
Database interface 1-7, 3-5
Database view 1-3, 1-4, 7-11, 7-14
Data element 1-3, 1-5, 2-4
Documentation 1-6
Domain 2-10, 4-3, 4-4
    equality 4-9
Domain concept
    two-level 2-5
Screen 1-3
Input help 1-3, 1-6, 8-3, 8-4, 11-10
Input check 1-6
Development environment 1-7
Extent 2-10
Field help 1-3, 1-6
Field assignment 4-7
Fixed value 4-3
Foreign key 1-6, 4-6, 4-7, 4-8, 4-10, 7-9, 7-13
Foreign key field
    type of 4-11
Foreign key relationship 4-11
Foreign key table 4-7, 4-8
Generation of table view 7-13
Size category 2-8, 2-10
Help view 11-6, 11-9
Include
    structure 2-7, 5-5
    in database views 7-12
Index 1-4, 3-3, 3-4, 8-12
Index entry 3-3

Index ID 3-4
Information system 5-7
Join 7-3
- inner join 7-14
- outer join 7-14

Join condition 7-5, 7-9, 7-10, 7-13
Catalog change 6-4
Cardinality 4-11
Component 1-5
Cross-product 7-4, 7-5
Customer field 6-12
Customer modification 6-13, 8-15
Runtime object 5-4, 6-3
Runtime environment 1-7
Matchcode 11-9
Object log 6-11
Optimizer 3-4
Pooled table 6-14
Performance 1-6, 3-5, 3-8, 8-12
Maintenance status 7-11
Help view 7-13, 7-14
Primary index 3-4
Primary key 2-3, 3-8, 4-8
Projection 7-3, 7-6
Log 1-6, 2-8, 2-11
Check field 4-7, 4-8, 4-9
Check table 4-4, 4-7, 4-8
Buffer 3-5, 3-6, 3-7, 3-8, 3-9
Buffering 1-6, 2-8
- full 3-6, 3-7, 3-8, 3-9
- generic 3-6, 3-8
- single-record 3-6, 3-9
- of database view 7-11

Buffering type 3-6, 3-8, 7-11
Buffer synchronization 3-10, 3-11, 3-12, 3-13, 3-14, 3-15
Release upgrade 6-13
rdisp/bufreftime 3-10
Semantic attribute 4-11
Key 3-7
- generic 3-8
- reduce length of 6-8

Key field 3-4, 4-8
- generic 3-8

Secondary index 3-4, 3-7
Selection 7-3, 7-4
Selection condition 7-7, 7-9
SE54 7-13

SPDD 6-13
Storage area 2-10
Lock object 1-6
Structure 1-3, 1-5, 2-3
Structure adjustment 6-4
Search field 8-4, 8-9
Search help 1-6, 8-5, 11-7
    search help attachment 8-9, 8-10, 8-11, 8-14
    elementary 8-14
    collective search help 8-14, 8-15
    append search help 8-15
    dialog behavior of a search help 8-5, 8-7
    search help parameter 8-7, 8-8
    interface of a search help 8-5, 8-8, 8-9, 8-10, 8-14
    selection method of a search help 8-5, 8-6
    search help exit 11-8
Synchronization table 3-10, 3-11, 3-12, 3-13, 3-14, 3-15
Synchronization time 3-14, 3-15
Table 1-3, 1-6, 2-3
    transparent 2-6
    text table 4-12
    unlock 6-11
Table buffer 3-5, 3-6
Table type 1-3, 1-5
Table maintenance generator 7-13
Tablespace 6-9, 6-10
Technical settings 2-8
Type
    structured 2-6
    user-defined 1-3, 1-5
Type definition 1-3, 1-5, 1-7
Conversion 6-4
    terminated 6-9, 6-10
Conversion process 6-5, 6-6, 6-7, 6-8, 6-9
Unique index 3-4
Upgrade 6-13
Displacement 3-5
Version
    active 5-3, 6-3, 6-5, 6-6, 6-7, 6-8, 6-9
    inactive 6-3, 6-3, 6-5, 6-6, 6-7
Where-used list 5-6
View 1-4, 1-6, 7-3, 7-4, 7-5, 7-6, 7-7, 7-8, 7-9, 7-10, 7-11, 7-12, 7-13, 7-14, 11-6
Value table 4-4, 4-9, 4-10
$TAB 3-5