

## **Summer Training Report (Sep-Dec 2022)**

**Place of Training: Diligent Global, Hyderabad, Telangana.**

**1 Sep 2022 – 31 Dec 2022**



Submitted to

**Department of Computer Science & Engineering**

Summer Training In-charge at TINJRIT: **Mr. Aaditya Maheshwari**

By

**VAIBHAV BHATNAGAR**

**(Batch 2019-2023)**

**Branch: Computer Science & Engineering**

**Roll No.: 19ETCCS076**

**Techno India NJR Institute of Technology**

Plot-T, Bhamashah (RIICO) Industrial Area, Kaladwas, Udaipur – 313003, Rajasthan

## Certificate I



**Diligent Tech India Pvt. Ltd.**

Date: 10.01.2023

### TO WHOMSOEVER IT MAY CONCERN

This is to certify that Mr. Vaibhav Bhatnagar has completed Technical Internship from 14<sup>th</sup> September 2022 to 10<sup>th</sup> January 2023. The project is undertaken by him in SAP ABAP.

We wish you all success in your future endeavors.

Yours faithfully,

A handwritten signature in blue ink that reads "Mansi Duggal" is written over a circular blue ink stamp. The stamp contains the text "DILIGENT TECH. INDIA PVT. LTD." around the perimeter and a small star in the center.

Mansi Duggal  
HR Manager

No. 204, 2nd Floor, Crystal Empire, Behind D-Mart, Baner Pune Maharashtra 411045

## **ACKNOWLEDGMENTS**

I take this opportunity to express my profound gratitude and deep regards to my guide Mansi Duggal (Manager Human Resources) for his exemplary guidance, monitoring and constant encouragement throughout the course of the training. The blessing, help and guidance given by him time to time shall carry me a long way in the journey of life on which I am about to embark.

I specially take the opportunity to thank our coordinator Mr. Yogendra Singh Solanki for their valuable information and guidance which helped me in completing this task through various stages. I also take this opportunity to express a deep sense of gratitude to all my teachers of Computer Science and Engineering Department for their coordinial support.

I am obliged to the staff members of the Diligent Global Farhan Sir, Vaibhav Sir, Nooruddin Sir, Shahrukh Sir, Vikas Sir, Kutubuddin Sir, Ridhima Ma'am, Beeru Sir for the valuable information provided by them in their respective fields. I am grateful for their corporation provided by them during my training period.

I am thankful to the almighty and my parents for their moral support and my friends with whom I shared my day-to-day experience and received lots of suggestions that improved my quality of work.

## Contents

Cerificate I .....	ii
Acknowledgements .....	iii
Contents .....	iv
List of Figures .....	v
Internship Training Overview .....	vi
<b>Unit 1: Introduction to the Dictionary .....</b>	<b>7</b>
Lesson: Overview of the Functions of the ABAP Dictionary .....	8
<b>Unit 2: Data objects in the ABAP Dictionary .....</b>	<b>13</b>
Lesson: Basic Data Types .....	14
Lesson: Tables in the ABAP Dictionary.....	25
Lesson: Special SAP Tables.....	36
<b>Unit 3: Performance When Accessing Tables .....</b>	<b>40</b>
Lesson: Performance During Table Access.....	41
<b>Unit 4: Input Checks .....</b>	<b>55</b>
Lesson: Consistency Through Input Checks .....	56
<b>Appendix .....</b>	<b>67</b>

## **List of Figures**

<b>Fig No.</b>	<b>Title</b>	<b>Page</b>
<b>Fig 1</b>	Function of the ABAP Dictionary .....	8
<b>Fig 2</b>	Database objects in the ABAP Dictionary.....	9
<b>Fig 3</b>	Type Definitions in the ABAP Dictionary .....	10
<b>Fig 4</b>	Services of the ABAP Dictionary.....	11
<b>Fig 5</b>	Linking to the development and runtime environment.....	12
<b>Fig 6</b>	Data Dictionary Initial Screen (SE11).....	15
<b>Fig 7</b>	Data Types in the ABAP Dictionary.....	16
<b>Fig 8</b>	Domain.....	17
<b>Fig 9</b>	Data Element.....	19
<b>Fig 10</b>	Structures.....	26
<b>Fig 11</b>	Using Simple Structures in ABAP.....	28
<b>Fig 12</b>	Nested Structure.....	29
<b>Fig 13</b>	Internal Tables.....	30
<b>Fig 14</b>	Deep structure .....	35

## **Internship Training Overview**

### **Training Prerequisites**

- Good programming experience in any different programming language

### **Training Goals**

This training consisted of:

- Work with the ABAP Workbench tools
- Write your own simple ABAP programs
- Carry out database read accesses
- Develop simple reuse components (subroutines, function modules and methods) and use them in programs
- Program dynamic screen processing
- Program user dialogs using the different screen elements in the SAP System
- Describe the function of the ABAP Dictionary in the SAP system
- Define database objects and use them
- Create and administer user-defined data types
- Use the services in the ABAP Dictionary
- Understand how the ABAP Dictionary is linked to the development and runtime environments

## Unit 1: Introduction to the Dictionary

### **Unit Overview**

The unit overview lists the individual lessons that make up this unit.

### **Unit Objectives**

After completing this unit, you will be able to:

- Name the function of the ABAP Dictionary in the R/3 system
- Describe the possible ways of defining data objects and data types
- Describe the services provided by the ABAP Dictionary
- Explain how the ABAP Dictionary is linked to the development and runtime

## Lesson: Overview of the Functions of the ABAP Dictionary

### Lesson Overview

In this lesson, you will gain an overview of the functional scope of the ABAP Dictionary



### Lesson Objectives

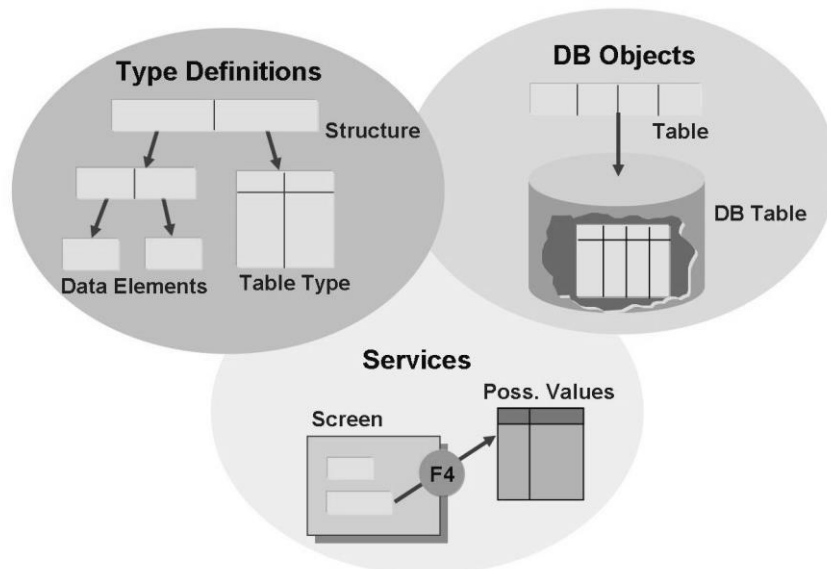
After completing this lesson, you will be able to:

- Name the function of the ABAP Dictionary in the R/3 system
- Describe the possible ways of defining data objects and data types
- Describe the services provided by the ABAP Dictionary
- Explain how the ABAP Dictionary is linked to the development and runtime environments

### Business Example

You should explain the main possibilities offered by the ABAP Dictionary to a colleague

### Overview of the functions



**Figure 1: Function of the ABAP Dictionary**

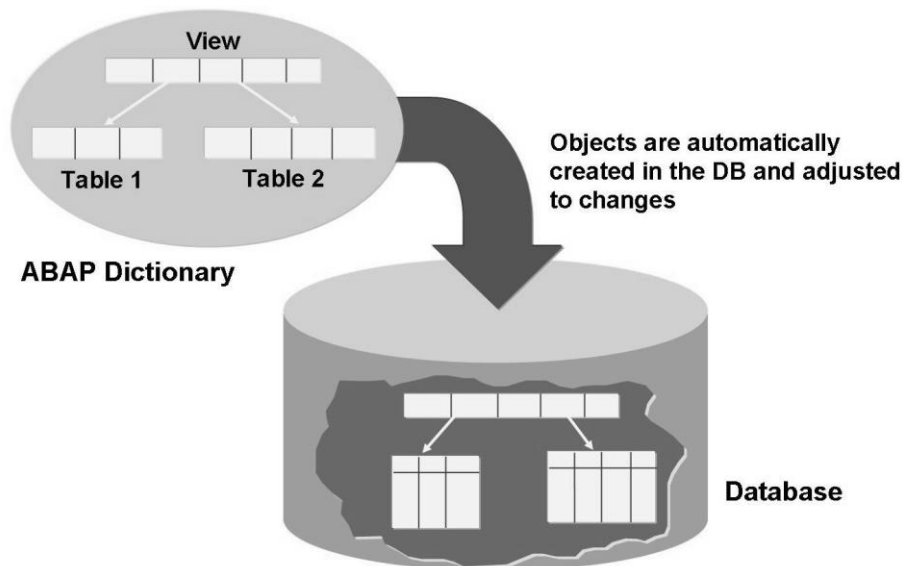
The ABAP Dictionary permits central management of all the type definitions used in the R/3 System.



## Lesson: Basic Data Types

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, object methods, etc. Database objects such as tables, indexes and 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.



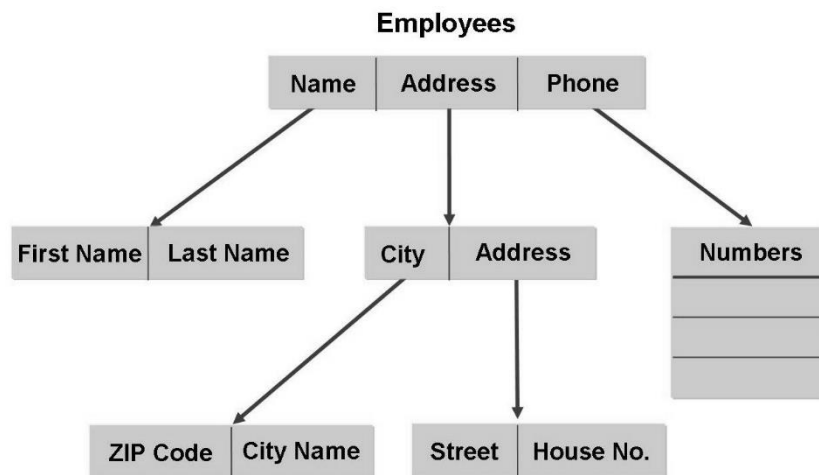
**Figure 2: Database objects in the ABAP Dictionary**

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.

## Lesson: Basic Data Types

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.



**Figure 3: Type Definitions in the ABAP Dictionary**

Three different type categories exist in the ABAP Dictionary:

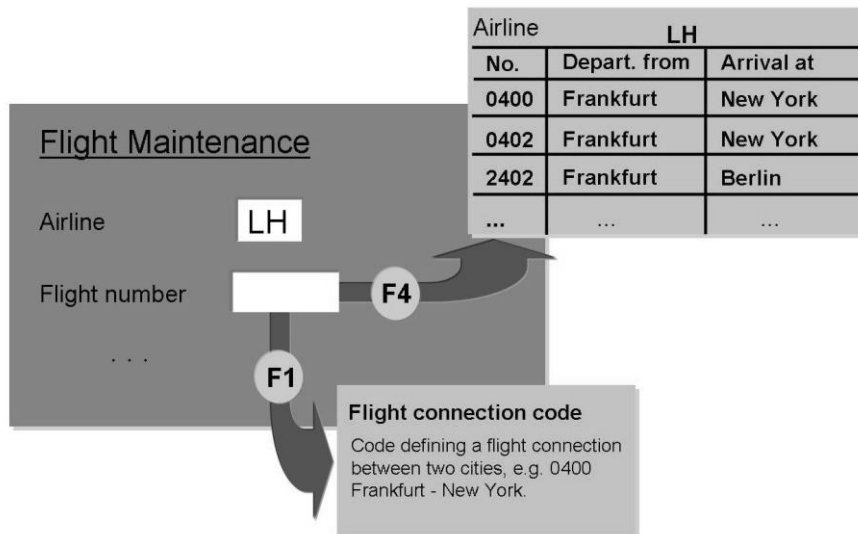
- **Data elements:** Describe an elementary type by defining the data type, length, and possibly decimal places.
- **Structures:** Consist of components that can have any type.
- **Table types:** Describe the structure of an internal table.

Any complex user-defined type can be built from these basic types.

**Example:** The data of an a

Types are used for example in ABAP programs or to define the types of interface parameters of function modules.

## Lesson: Basic Data Types

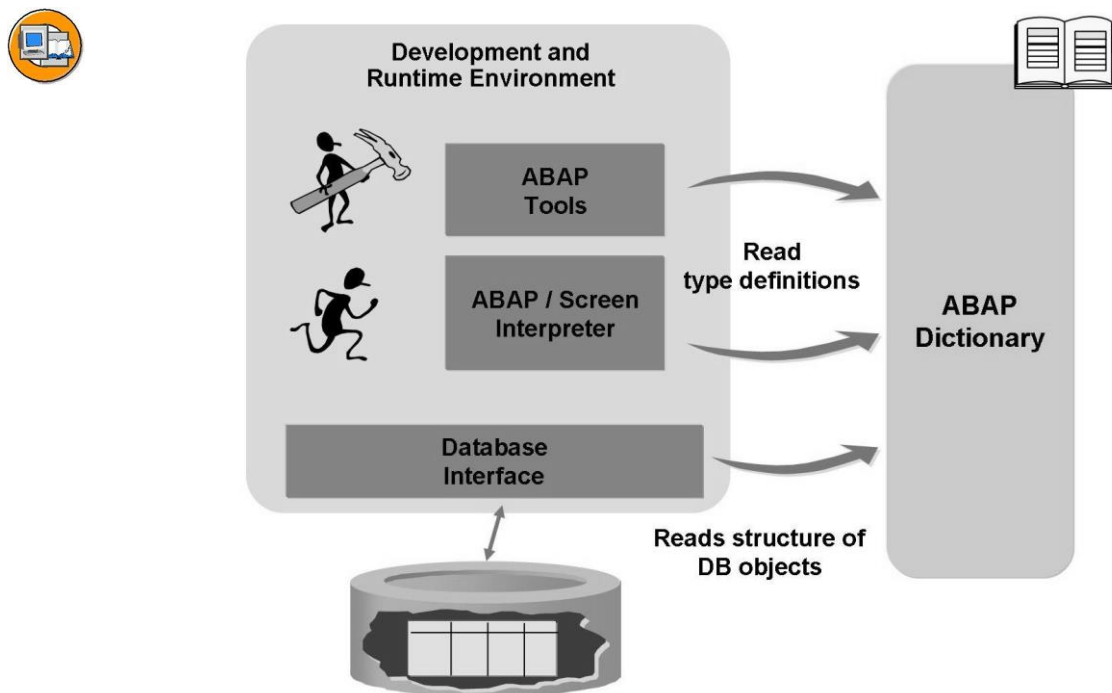


**Figure 4: Services of the ABAP Dictionary**

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.

## Lesson: Basic Data Types



**Figure 5: Linking to the development and runtime environment**

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 2: Data objects in the ABAP Dictionary

### Unit Overview

In this chapter, you will become familiar with the different options for mapping data objects in the Dictionary



### Unit Objectives

After completing this unit, you will be able to:

- Create domains and use them in data elements
- Define data elements and use them as the basis for defining data objects in ABAP programs
- Define structures and use them as the basis for defining data objects in ABAP programs
- Define internal tables and use them as the basis for defining data objects in ABAP programs
- Define complex (nested / deep) structures and use them as the basis for defining data objects in ABAP programs
- Define global constants with the help of a type pool and use them in ABAP programs
- Create Tables
- Use the two-level domain concept
- Define the technical settings of a table
- Create and use include structures
- Describe table types in the SAP system apart from the transparent tables
- Distinguish pooled and cluster tables from one another
- Describe the advantages and disadvantages of pooled and cluster tables.

## Lesson: Basic Data Types

### **Lesson Overview**

In this lesson, you learn about the basic data types and their use with one another.



### **Lesson Objectives**

After completing this lesson, you will be able to:

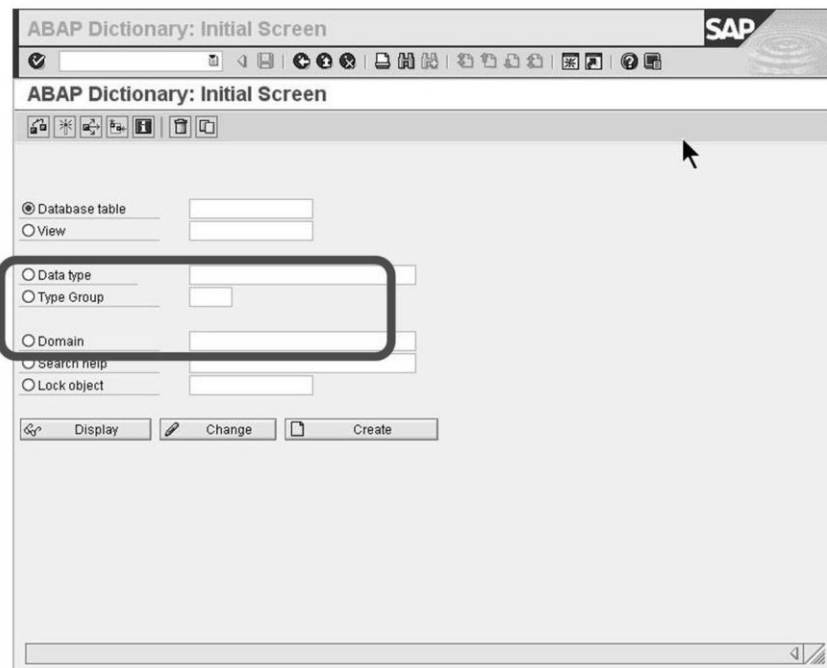
- Create domains and use them in data elements
- Define data elements and use them as the basis for defining data objects in ABAP programs
- Define structures and use them as the basis for defining data objects in ABAP programs
- Define internal tables and use them as the basis for defining data objects in ABAP programs
- Define complex (nested / deep) structures and use them as the basis for defining data objects in ABAP programs
- Define global constants with the help of a type pool and use them in ABAP programs

### **Business Example**

You should define simple and complex data types in the Dictionary and be able to use these in an ABAP program.

## Lesson: Basic Data Types

### Data types



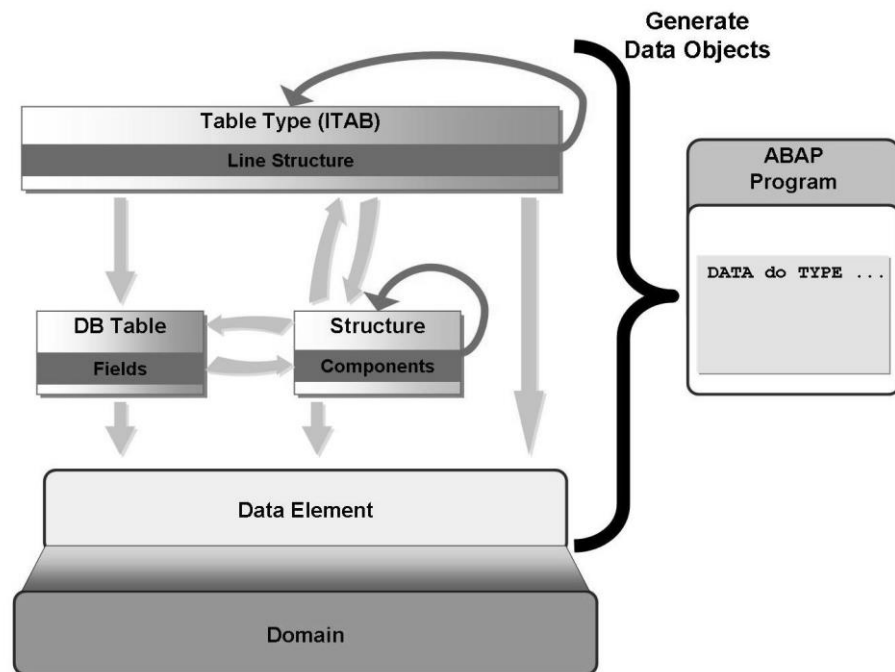
**Figure 6: Data Dictionary Initial Screen (SE11)**

Cross-program (globally known) data types can be defined in the ABAP Dictionary. You can reference these data types in every ABAP program in the SAP system with the TYPE addition for corresponding ABAP statements.

In the initial screen, the corresponding definitions are defined using the *Data Type* field. Under data type, you can find the three basic types data element, structure and table type (internal tables - ITAB).

The type group is actually a relict from past times (before Release 4.5a), when it was not yet possible to create own **global, complex data types** for declaring data objects in ABAP. The type group is still used for this. In future, when you create a new global, complex data type, you should use the options provided by the Dictionary structures. In addition, the type group for creating global constants was and is used. The use of global constants is also supported by ABAP objects in the form of constant attributes of a class. However, if you want to program in a non-object oriented way, only the type group for defining a global constant remains.

## Lesson: Basic Data Types



**Figure 7: Data Types in the ABAP Dictionary**

To define data objects in the ABAP programs, you can use the above-mentioned type definitions with the exception of the domain. The arrows show how they can be used together.

Ideally, data elements use domains to define their technical properties. An ABAP program cannot, however, access domains for defining data objects.

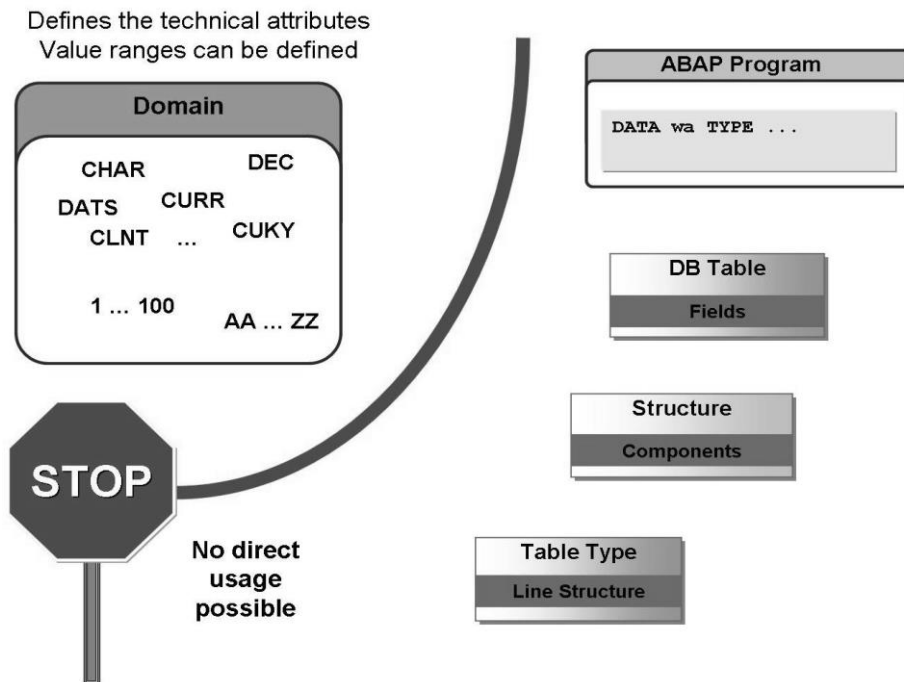
As a rule, data elements get their data types from domains. This data type can also be derived from an integrated type.

A structure consists of components that can be elementary fields, tables and also structures.

The line type of a table type (ITAB) can be a structured type or an elementary type.



## Lesson: Basic Data Types



**Figure 8: Domain**

The domains are used to manage the technical properties of data objects centrally. Domains cannot be used directly in programs, tables, etc. The data type 'Data element' (see below) is used here as a bridge from the technical properties to the different data types / data objects.

### Technical information comprises the following points:

#### Format:

In the format specifications, you will find the *Data type* and the *Number of characters* that can be entered in a dependent data object. If the format is numeric, you can also determine a value for the *decimal places*.

The data type is based on 24 integrated types. The most frequently used data types are explained below:

#### CHAR

Character string

CHAR type fields can only have a maximum length of 255 in tables. If you want to use longer character strings in tables, you have to choose the LCHR data type. In structures, there are no restrictions with regard to the length of such fields.

#### DATS

Date

## Lesson: Basic Data Types

The length for this data type has been set at 8 characters. You can define the output template by way of the user profile.

### **DEC**

Calculation or amount field with point, +/- sign and thousand-separator commas.

A DEC field can only have a maximum of 31 characters.

### **NUMC**

Character string that can only contain numbers. The length of a field of this type is restricted to a maximum of 255 characters.

### **Output properties:**

Here, the maximum field length, including commas or decimal points, is specified for the input and output of values.

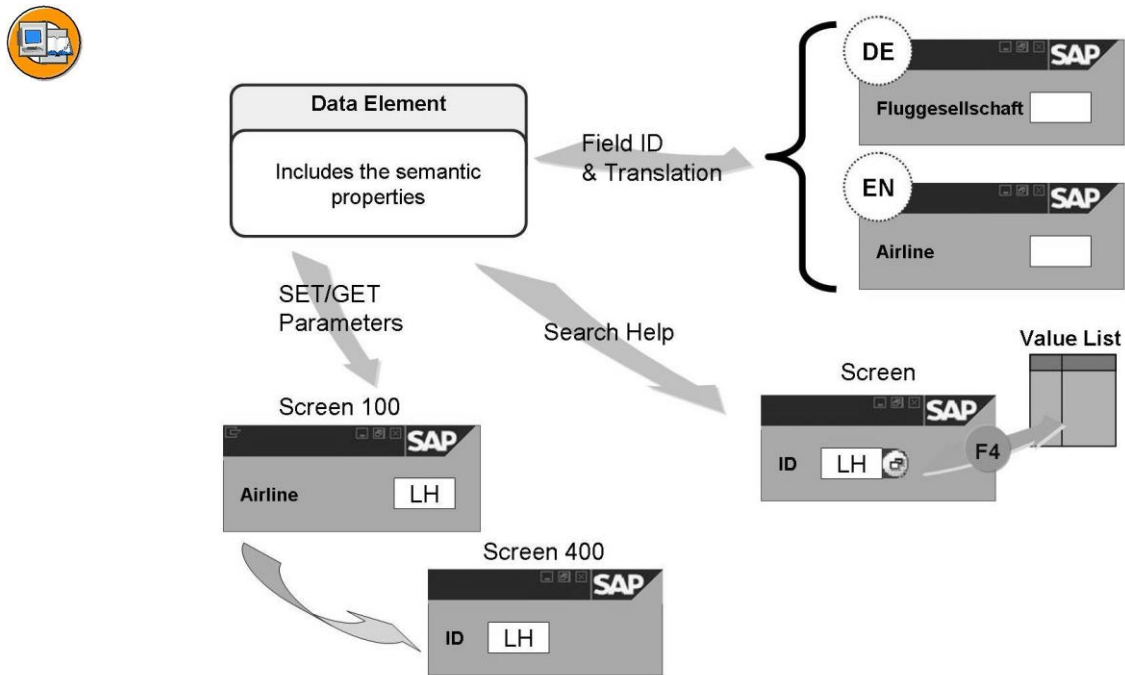
This value is usually calculated automatically once the *number of characters* has been assigned under Format, however, it can subsequently be overwritten. The output format has an effect on the output of screens and selection screens. The specifications from this area are used when integrating a field into a screen, however, they can be modified in the Screen Painter.

Furthermore, you can also define a conversion routine for this domain. This serves to change the display format (e.g. place leading zeros before a number) when converting the content of a screen field from the display format to the SAP-internal format and vice versa, as well as for output using the ABAP statement WRITE. Similarly, you can also use this conversion routine to override any unsuitable standard conversions. For certain data types (DEC, FLTP, QUAN and CURR), the check box *+/- sign* is ready for entry. If this is activated, the first character of the field is reserved for the +/- sign on a screen. The output length should be increased by 1 accordingly.

For character-based data types, you should also determine whether *lower case letters* are allowed. If this flag is not set, lower-case letters can be entered in the respective input field, but will be transformed into upper-case letters as soon as the entry is confirmed by the user (e.g. ENTER key).

In addition, you can define valid value ranges that are used for input checks. However, this subject is explained in more detail in another part of the course.

## Lesson: Basic Data Types



**Figure 9: Data Element**

As well as constituting the link between domains and the data objects, data elements also contain semantic/technical information about data objects created from these.

The field labels for the data field can and should be maintained in the data elements. These field labels (short, medium or long) can be displayed later on screens or selection screens to explain the field content.

On selection screens (e.g. ABAP command PARAMETERS), only the long version of the field label can be drawn from the Dictionary ( In the ABAP Editor menu path: *Goto -> Text Element -> Selection Text => Checkbox , Dictionary Reference* ).

If the field value is provided in a list, the entry from the field label is used for the title. You also have to specify a *length* for the respective field label. This length determines the maximum length for the field label. If you work for a company that operates throughout the world, you can translate the field labels into other languages (menu path: *Goto -> Translation* or the transaction SE63). When specifying the length, remember that, in another language, the same term in the field label might require more letters.

A search help (F4 / input help) can be appended to a data element. The subject of search helps is dealt with in more detail in the one of the later units of this course. Search helps can be integrated at different levels.

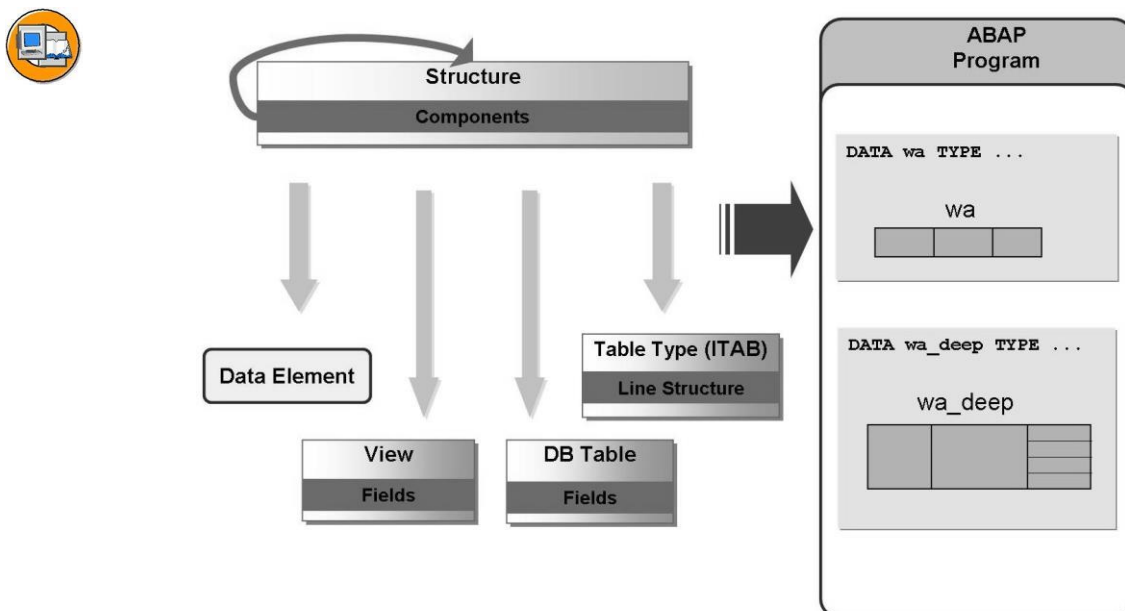
In different applications, you sometimes have to enter a particular value in several screens. To save the user having to enter the same value several times, it is possible to assign a

## Lesson: Basic Data Types

SET/GET parameter to the data element. In this parameter, the value is transferred when the screen is exited. If an input field based on the same data element exists on a subsequent screen, this value is read from the parameter and entered in the screen field. The SET/GET parameters hold the value per session. After the user has logged off, these values are not retained. In order to use a SET/GET parameter, you have to enter this in table TPARA!

You can also assign an English default name to the data element. This is only effective, however, if you use the data element as a component in BAPI structures. These should use the default name so that component names are assigned uniformly.

The technical properties for the data element are maintained on the *Data type* tab page. Here, you should use mainly domains that are predestined for technical typing. However, you can also define the data element using the same integrated types that are used to define the domains. As a special case, you can also create a data element as a reference type. The referenced type is not restricted here to the type 'DATA ELEMENT'. It can be any other reference type or even a generic reference to ANY, OBJECT or DATA. A reference of the type ANY can point to both objects and data. The definition as a reference is the same as the type declaration in an ABAP program "TYPES tr\_dt TYPE REF TO data".



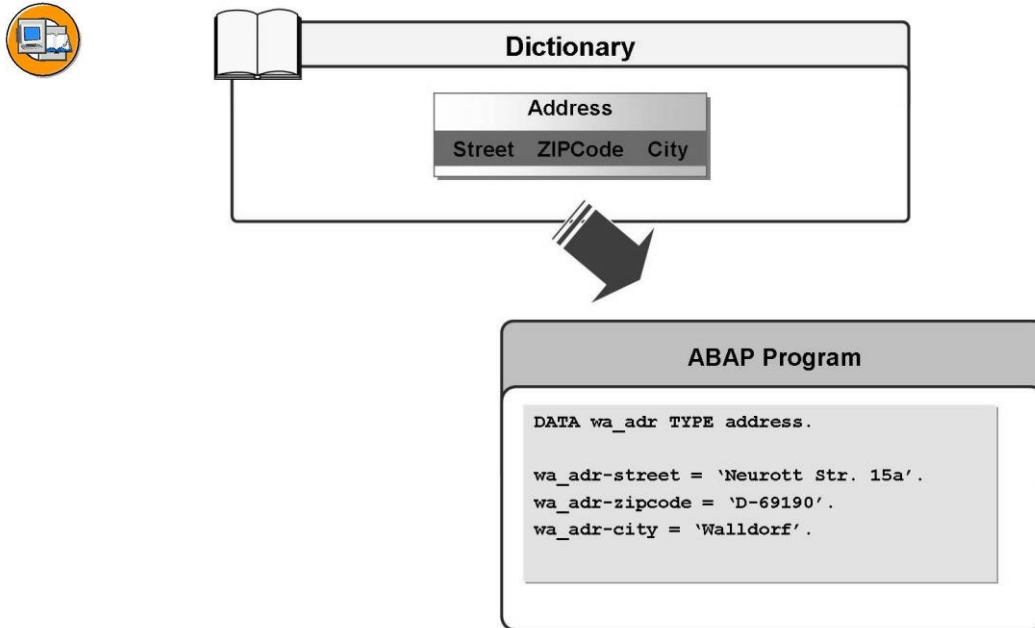
**Figure 10: Structures**

A structure consists of components in the form of data elements, integrated types, structure definitions of internal tables, DB table views or other existing structure definitions.

When the fields of an actual two-dimensional object are also taken into a structure by including a view or DB tables, the data object which can be generated from this remains flat (one-dimensional).

## Lesson: Basic Data Types

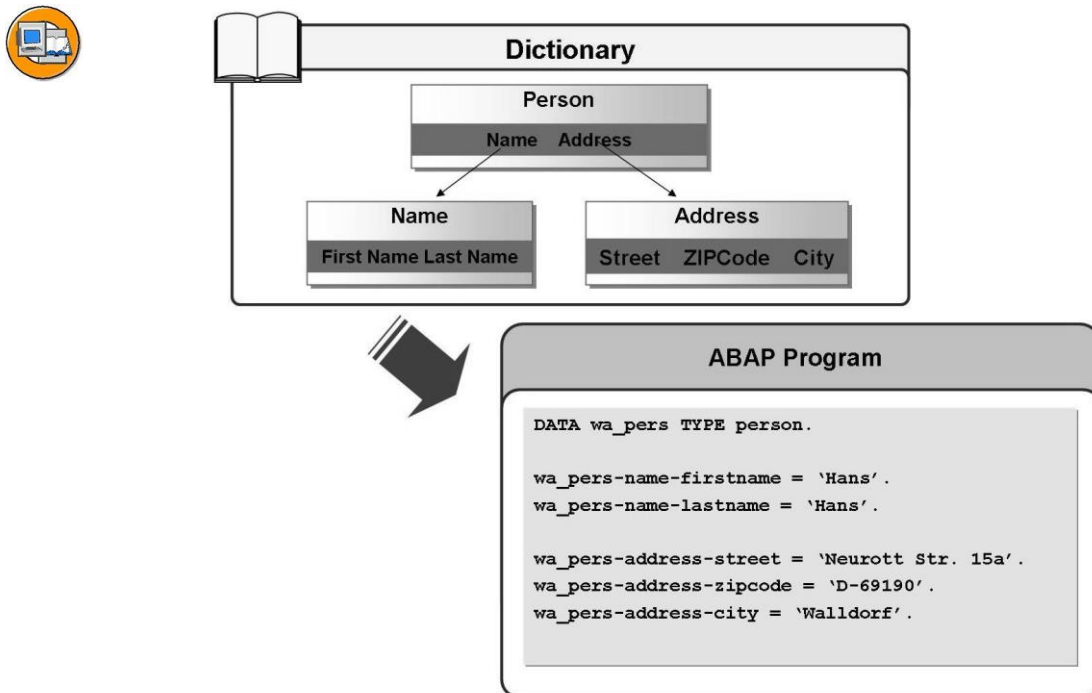
A so-called deep structure is always created when you used a table type to define a component. Although this component is then two-dimensional, the other components in the structure remain flat (one-dimensional).



**Figure 11: Using Simple Structures in ABAP**

The simplest form of a structure is the sequencing of fields through the use of data elements. This always gives rise to a so-called flat structure. A data object based on this structure type is always one-dimensional (as opposed to table-like, two-dimensional data objects). You address the individual elements (components) of the structure using the name of the structure, a hyphen and the name of the components.

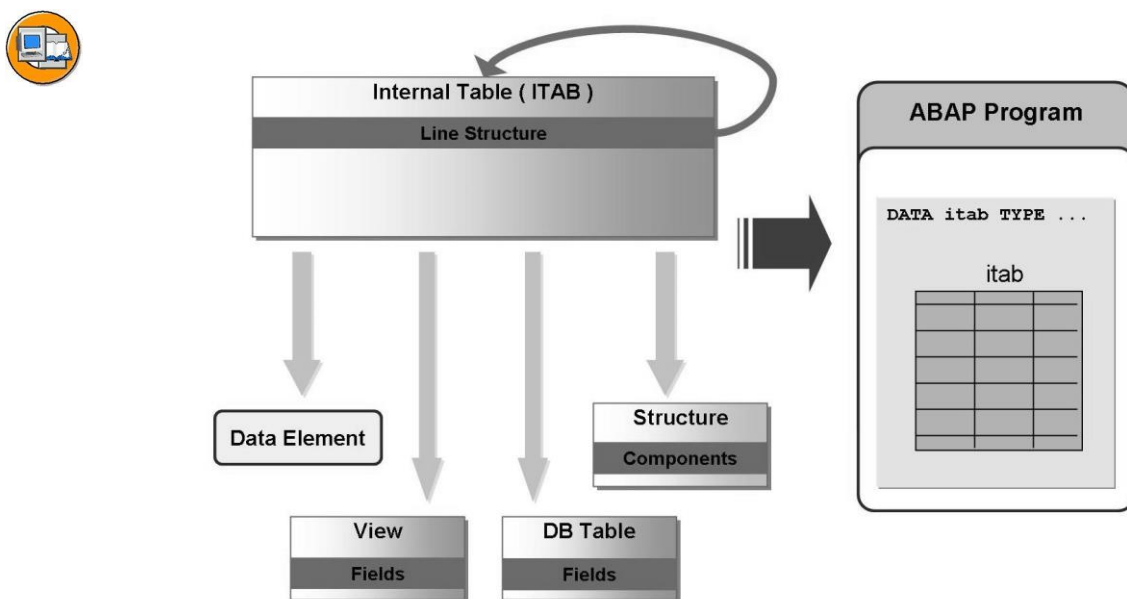
## Lesson: Basic Data Types



**Figure 12: Nested Structure**

You can include another structured object in the structure and assign it to a component. The one component refers to the structured object and the new data object is now described as a nested structure.

Structures like these can be nested into one another in any way.



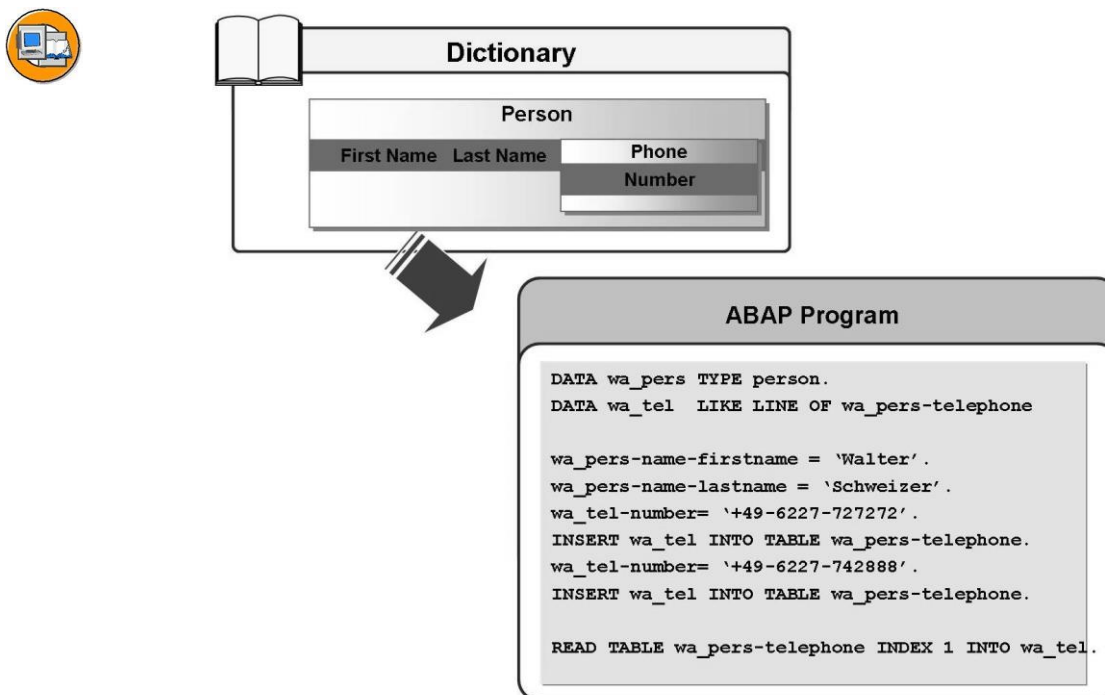
## Lesson: Basic Data Types

**Figure 13: Internal Tables**

As soon as you insert an internal table as a structure component, this becomes a deep structure.

You can define internal tables or ITABs using an existing line structure. Database tables, structure definitions, views, data elements, direct type definitions or existing table types can be used as line types.

By declaring an internal table in an ABAP program, a two-dimensional Array is created in the main memory.

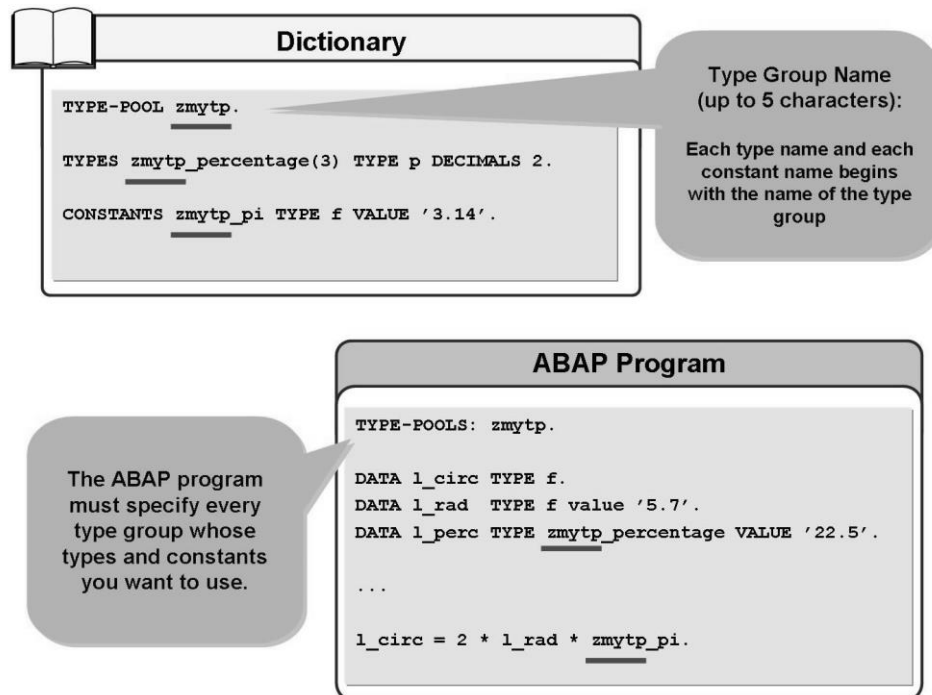


**Figure 14: Deep structure**

A deep structure contains at least one table. The component of such a table has its own name with which it can be addressed like a normal internal table (LOOP AT..., INSERT... INTO TABLE, ...).

An internal table can, in turn, have a deep structure as a line type. In this way, you can create multi-dimensional data types, as the inter-nesting of internal tables and structure is possible several times.

## Lesson: Basic Data Types



**Figure 15: Type Group**

If you want to define global constants, you have to use a type group. The name of the type group can only contain a maximum of five characters. In the type group, you can define constants using the `CONSTANTS` statement. You can use the predefined ABAP types or the global Dictionary types here.

To be able to use the types of a type group in a program, declare the type group using the `TYPE POOL` statement. From these lines on, you can use all constants of the type group.

The definition of a type group is a piece of ABAP code that you maintain either in the Dictionary (SE11) or in the ABAP Editor (SE38).

### Realization:

The first statement for the type group **zmytp** is always: **TYPE-POOL zmytp.**

This is followed by the data type definition with the statement `TYPES`, as described under local program data types. Furthermore, you can declare cross-program constants using the `CONSTANTS` statement. All names of these data types and constants must begin with the name of the type group and an underscore:

`zmytp_`

In an ABAP program, type groups must be declared with the following statement before they are used:



## Lesson: Tables in the ABAP Dictionary

### Lesson Overview



#### Lesson Objectives

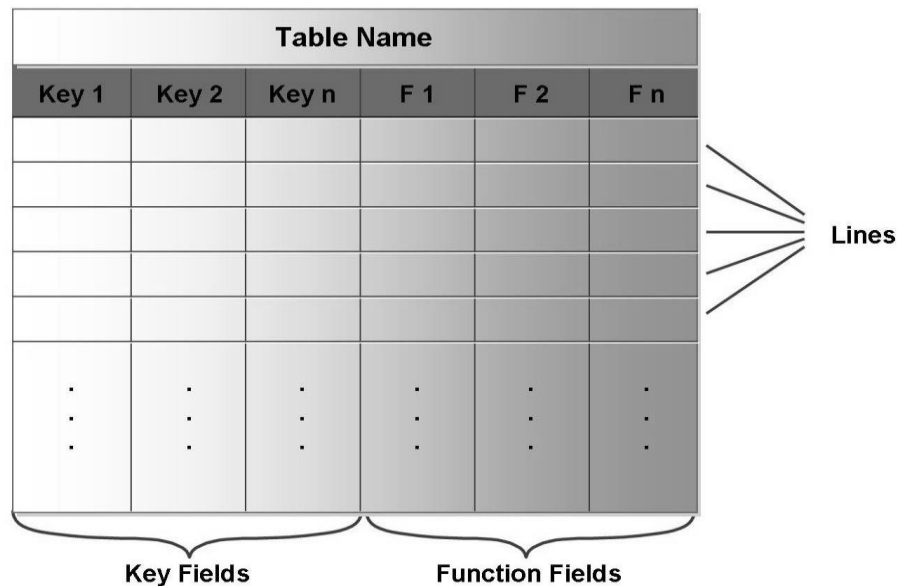
After completing this lesson, you will be able to:

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

#### Business Example

You should map information units, so-called entities in the database for your company.

#### Tables



**Figure 16: Tables and Fields**

The structure of the objects of application development are mapped in tables on the underlying relational database.

## Lesson: Tables in the ABAP Dictionary

The attributes of these objects correspond to fields of the table.

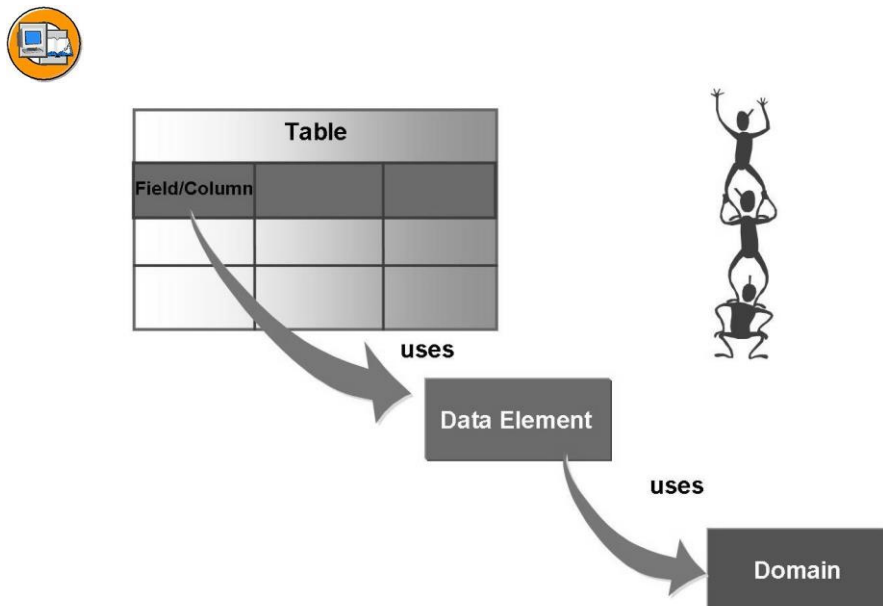
A table consists of columns (fields) and rows (entries). It has a name and different general attributes, such as delivery class and maintenance authorization.

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 CURRE) 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.



**Figure 17: 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 by its data type and length. The value range can be limited by specifying fixed values.

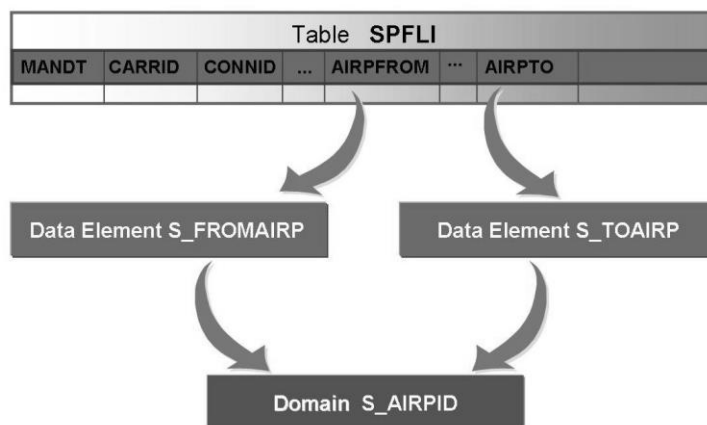
## Lesson: Tables in the ABAP Dictionary

A **data element** describes the meaning of a domain in a certain business context. The data element contains primarily the field help (F1 documentation) and the field labels in the screen.

A field is not an independent object, but it is table-dependent. The field 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**.

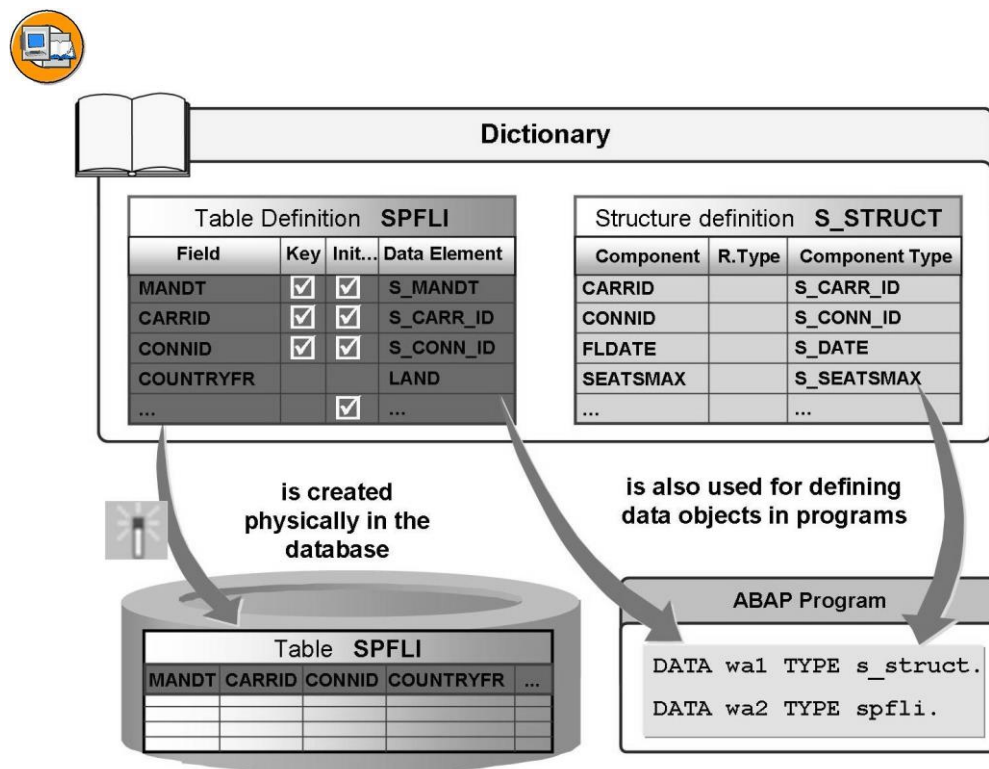
The data type attributes of a data element can also be defined by specifying a **built-in type**, where the data type and number of places is entered directly.



**Figure 18: Two-Level Domain Concept: Example**

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.

## Lesson: Tables in the ABAP Dictionary



**Figure 19: Transparent Tables and Structures**

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.

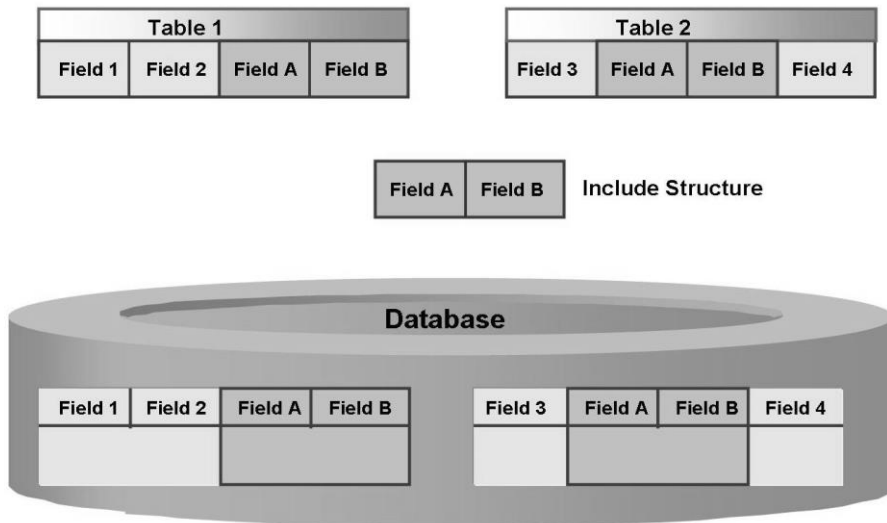
The database table has the same name as the table in the ABAP Dictionary. The fields also have the same name in both the database and the ABAP Dictionary. The data types in the ABAP Dictionary are converted to the corresponding data types of the database system.

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 you add a new field, adjust the order of the fields by changing the database catalog (ALTER TABLE). The new field is added to the database table.

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.

## Lesson: Tables in the ABAP Dictionary

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.



**Figure 20: Include Structures**

Structures can be included in tables or other structures to avoid redundant structure definitions.

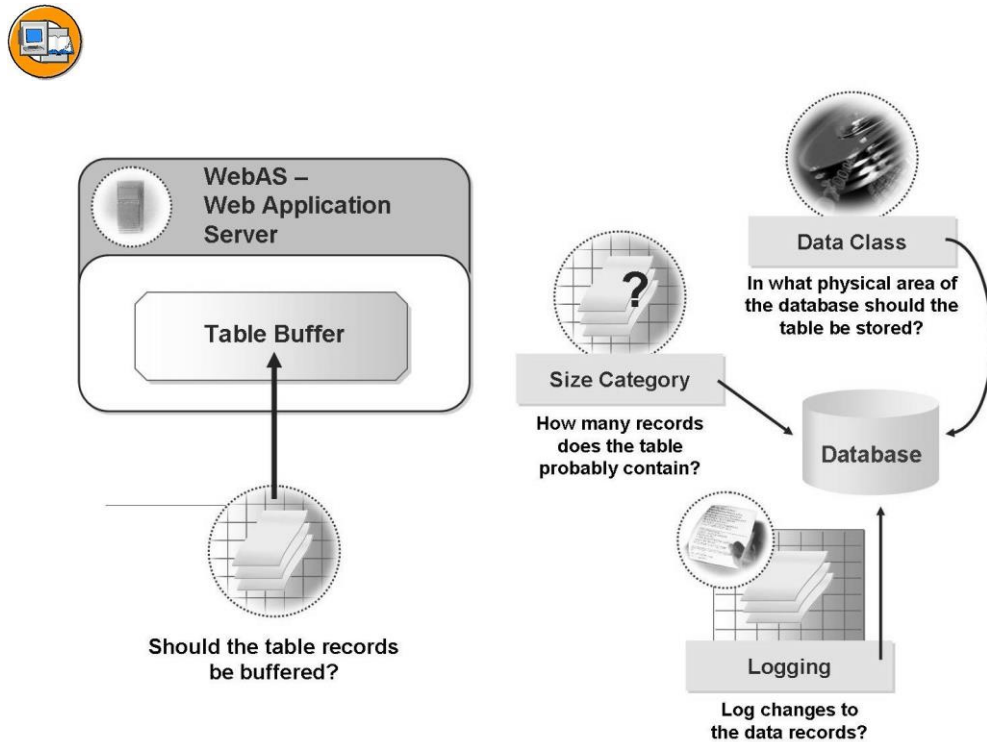
A table may only be included as an entire table.

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.

## Lesson: Tables in the ABAP Dictionary



**Figure 21: Technical settings**

You must maintain the technical settings when you define a transparent table in the ABAP Dictionary.

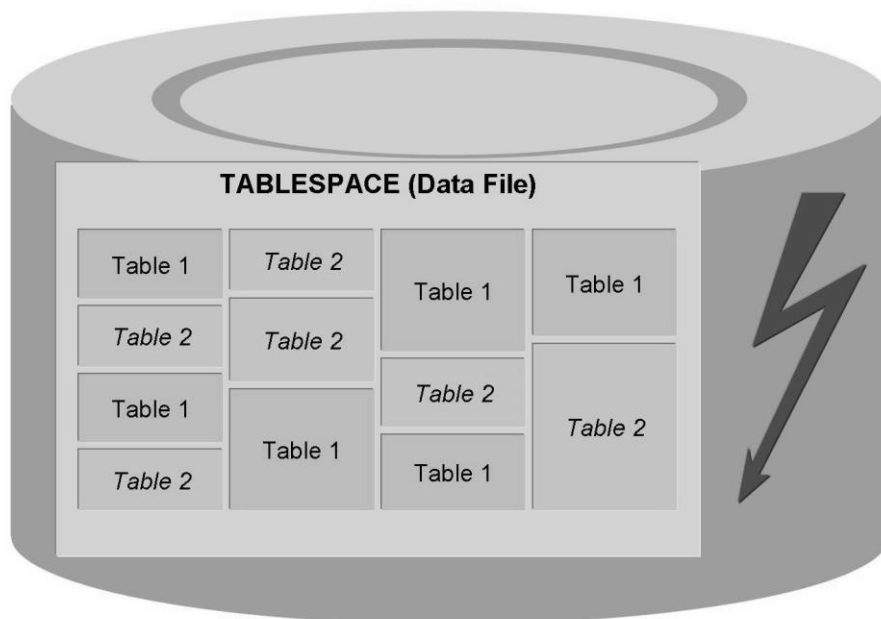
The technical settings are used to individually optimize the storage requirements and accessing behavior of database tables.

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 information required here about the storage area to be selected (tablespace) and the expected table size is 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.



**Figure 22: Fragmenting the tables in the database**

Particularly tables that include transaction data can have large gaps in their allocated memory space due to the fast growth or frequent changing of the datasets (inserting and deleting of data records). This fragmenting of the data on the hard drive of the database server always leads to repeated performance problems, which can **not** be removed by simple measures such as creating an index.

Fragmented indexes can be defragmented by simply deleting them and then entering them again. Here, the database calculates the content and the structure of the index again and creates the necessary memory space in one piece in the storage medium, where possible. While the index is being newly structured, the database tables can still be used.



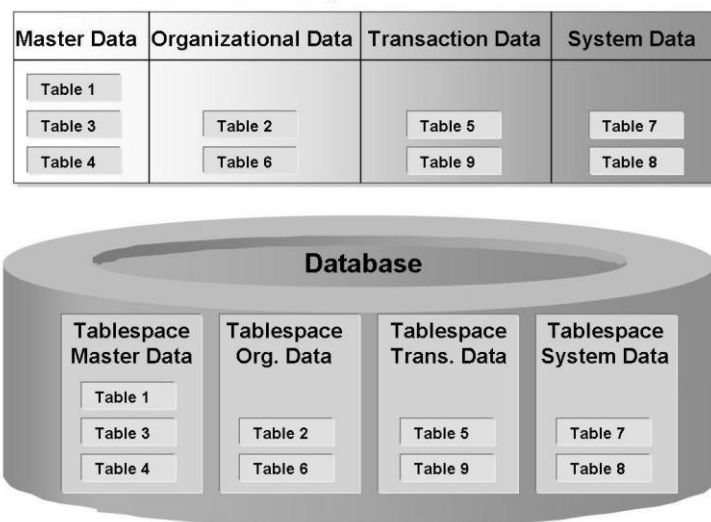
**Caution:** During the conversion process, the content of the table cannot be read or changed. The conversion process of a table must be executed in a productive environment with the greatest of care, as, depending on the size of the table, the conversion process can last for several minutes or hours.

**= > Such conversion processes (for tables or indexes) should therefore take place in times with the least load.**

In order to avoid such fragmentations as much as possible from the start, you select the tables according to size and data type in the technical settings.

## Lesson: Tables in the ABAP Dictionary

### Tables in ABAP Dictionary



**Figure 23: Data class**

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 correct 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 only seldomly 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.

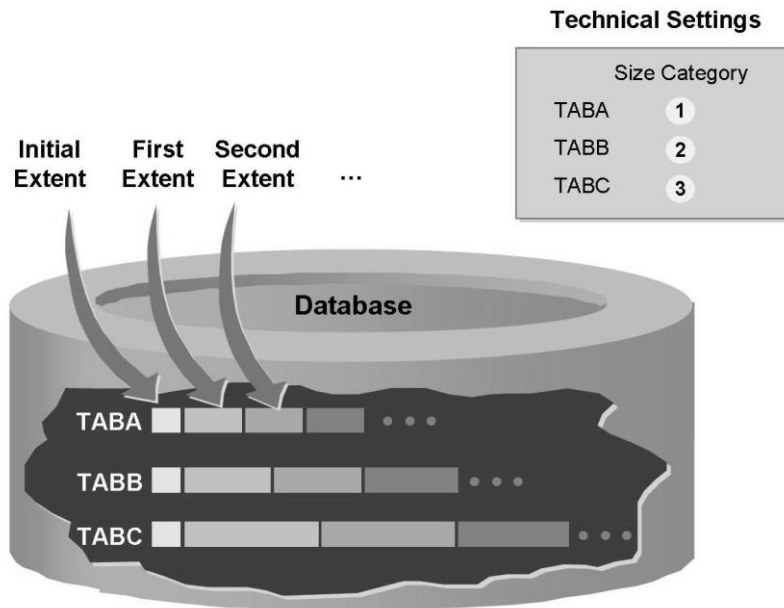
Organizational data is data that is defined during customizing when the system is installed and that is only seldomly modified thereafter. The country keys are an example.

System data is data that the SAP System itself needs. The program sources are an example.

Further data classes, called customer data classes (USER, USER1), are provided for customers. These should be used for customer developments. Special storage areas must be allocated in the database.



## Lesson: Tables in the ABAP Dictionary



**Figure 24: Size Category**

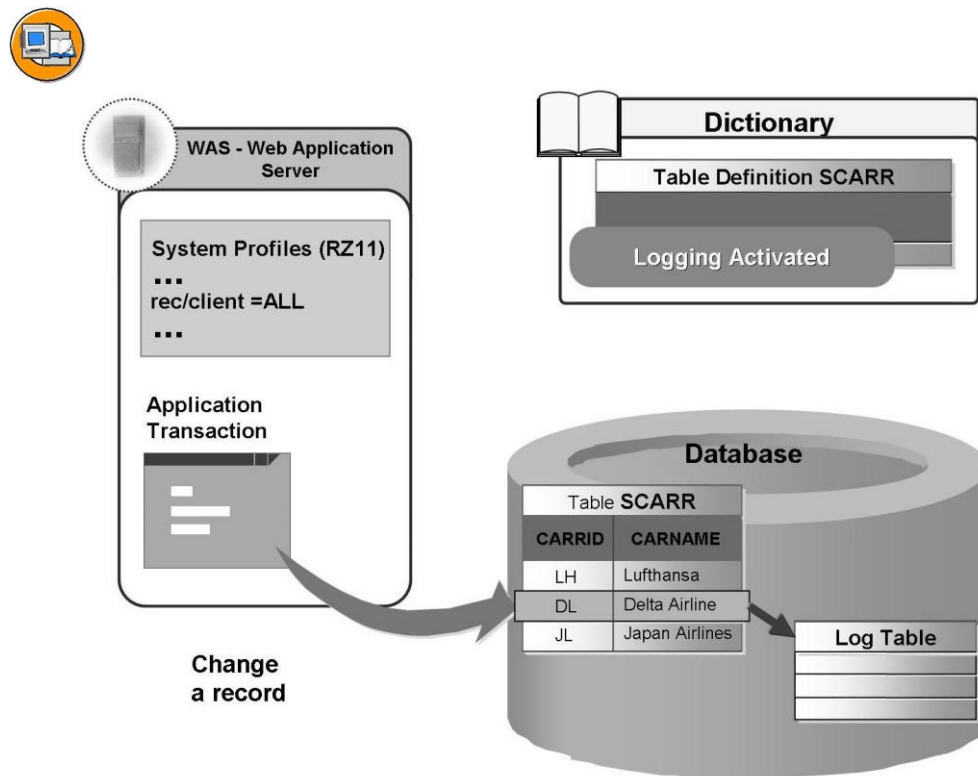
The size category describes the expected storage requirements for the table on the database.

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. Every category is assigned a fixed extent size, which depends on the database system used.

Correctly assigning a size category 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.

## Lesson: Tables in the ABAP Dictionary



**Figure 25: Logging**

You can use logging to record and store modifications to the entries of a table.

To activate logging, the corresponding field must be selected in the technical settings. Logging, however, only takes place if the SAP system was started with a profile that contains the 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 the clients should be logged. rec/client = 000[...] Only the specified clients should be logged. rec/client = OFF      Logging is deactivated in this system.

The data modifications are logged independently of the update. You can display the logs with the Table History transaction (SCU3).



### **Caution: Logging creates a bottleneck in the system:**

- Additional write access for each modification to tables being logged.

## Lesson: Tables in the ABAP Dictionary

- This can result in lock situations although the users are accessing different application tables!

### **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.

## Lesson: Special SAP Tables

### Lesson Overview

You will learn about pooled and cluster tables



### Lesson Objectives

After completing this lesson, you will be able to:

- Describe table types in the SAP system apart from the transparent tables
- Distinguish pooled and cluster tables from one another
- Describe the advantages and disadvantages of pooled and cluster tables.

### Business Example

In the performance checks of some applications, you have come across select statements, which apparently access other tables in the database than referred to in the respective ABAP coding.

### Pooled and cluster tables

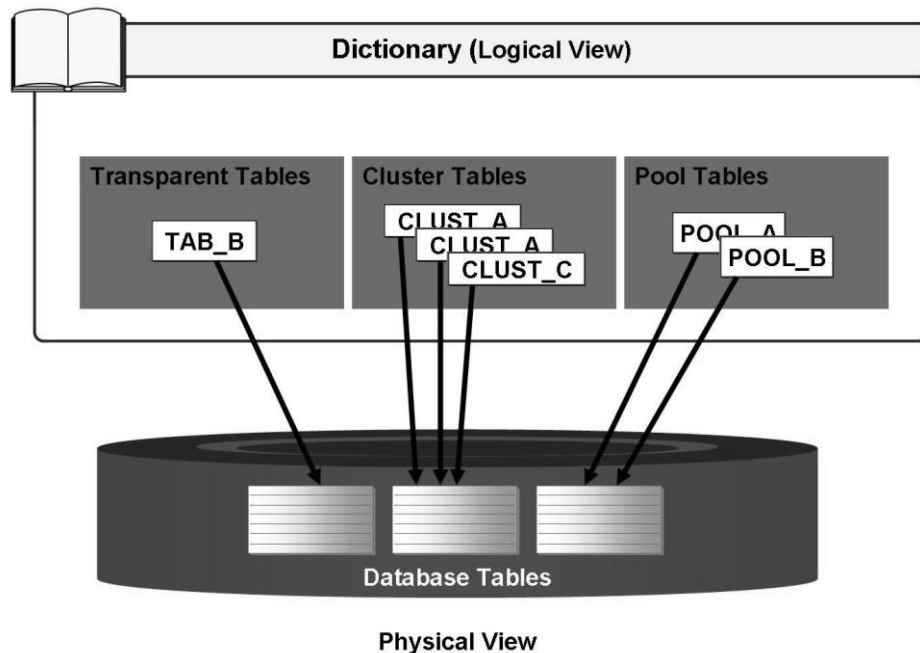
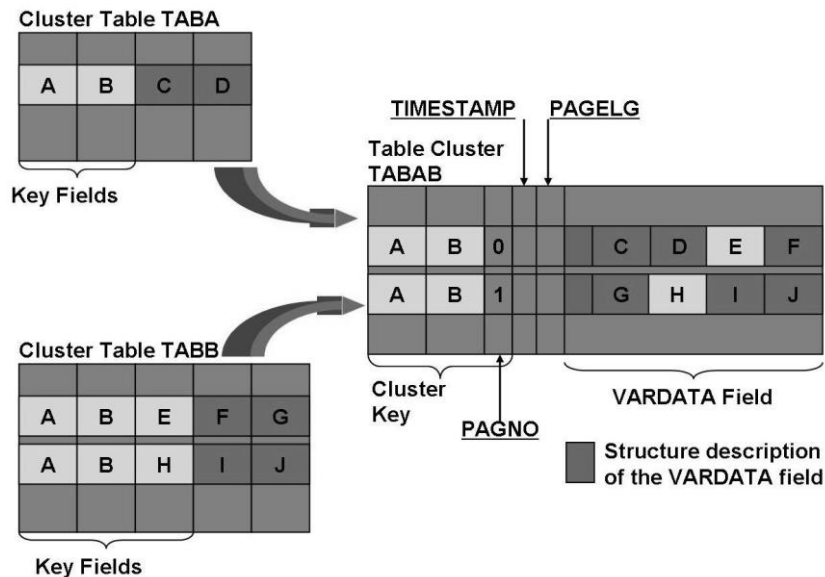


Figure 26: Overview of the DB table types

## Lesson: Special SAP Tables

In addition to transparent tables, where the definition in the ABAP Dictionary and in the database are identical, there are pooled and cluster tables in the R/3 system.

Pooled and cluster tables are characterized by the fact that several tables logically defined in the ABAP Dictionary are combined in a physical database table (table pool or cluster).

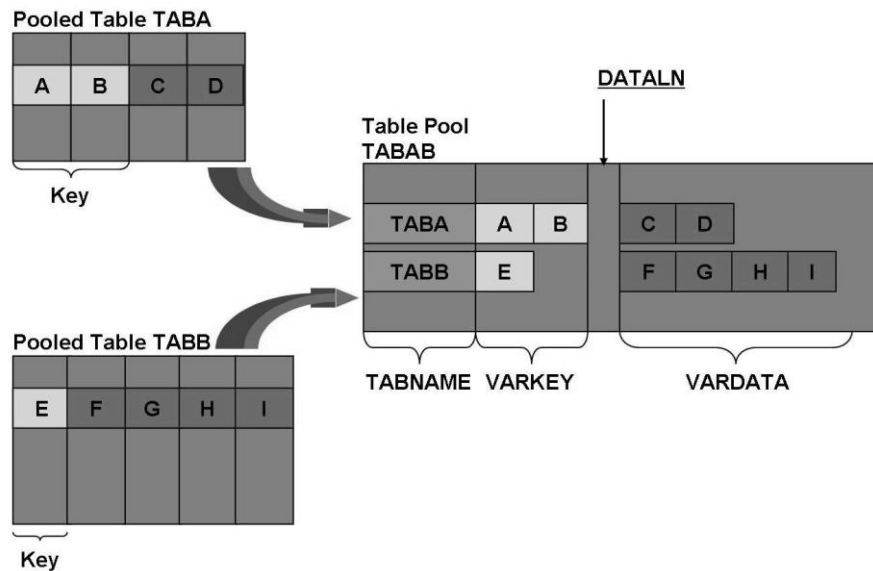


**Figure 27: Cluster tables**

The idea of cluster tables is that you store functionally dependent data which is divided among different tables in one database table. Accordingly, the intersection of the key fields of the cluster tables is formed by the key of the table cluster (cluster key).

The data dependent on one cluster key are stored in the VARDATA field of the table cluster. If the VARDATA field does not have the capacity to take on all dependent data, the database interface creates an overflow record. The uniqueness within the table cluster is guaranteed by the PAGNO field.

The content of the VARDATA field is compressed by the database interface. Accordingly, the VARDATA field contains a description for decompressing its data. The TIMESTAMP and PAGELG fields contain administrative information.



**Figure 28: Pooled Tables**

The basic idea of a table pool, as opposed to table clusters, is the storage of data records from the tables defined in the ABAP Dictionary that are not dependent on one another. You would like to combine small R/3 tables to a database table.

In the example above, you can recognize that the intersection of the key fields of TABA and TABBB is empty. Despite this, the data records from TABA and TABBB are stored in the TABAB table pool.

The key for a data record of the TABAB table pool consists of both the fields TABNAME and VARKEY. The TABNAME field assumes the name of the pooled table. The VARKEY field consists of the concatenation of the key fields of the pooled table. This arises in the necessity that the key fields of a pooled table must be of the type C.

In the VARDATA field, the non-key fields of the pooled tables are stored in an unstructured way, in compressed form by the database interface. The DATALN field contains the length of the VARDATA field.



- Fewer tables and table fields
- Data compression
- Encrypted data storage



- Limitations on database functions
  - No native SQL / No views or ABAP JOINS / No secondary indexes / No GROUP BY, ORDER BY, ...
- No table appends
- For cluster tables
  - Limited selection on clusterkey fields
- For pooled tables
  - Longer keys than necessary

### Figure 29: Describe the advantages and disadvantages of pooled and cluster tables

The decisive advantage of pooled and cluster tables is that the data can be stored in compressed form in the database. This reduces the memory space required as well as the network load.

The combination of tables into table pools to table clusters means less tables and the compressing means less fields in the database. The result is that fewer different SQL statements are carried out.

Pooled and cluster tables are not stored as separate tables in the database. In this way, the administration becomes simpler. With cluster tables, functionally dependent data are read together, which results in fewer database accesses.

The decisive disadvantage lies in the restricted database functionality. It is not possible for non-key fields to create an index. There are neither primary indices nor indices on a subset of the key fields. The use of database views or ABAP JOINS is also ruled out, as are Table Appends. You can access the data in pooled or cluster tables only via OPEN SQL (no Native SQL).

For pooled tables, only the WHERE conditions for key fields and for cluster tables, only the WHERE conditions for the fields of the cluster key (subset of the key fields) are transferred to the database. ORDER BY (or GROUP BY) clauses are not transferred for non-key fields. You need longer keys than semantically necessary for pooled tables.

## Lesson Summary

You should now be able to:

- Describe table types in the SAP system apart from the transparent tables
- Distinguish pooled and cluster tables from one another
- Describe the advantages and disadvantages of pooled and cluster tables.

## **Unit 3: Performance When Accessing Tables**

### **Unit Overview**

In this chapter, you will get to know the basic aspects of the efficient table access.

### **Unit Objectives**

After completing 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





## Lesson: Performance During Table Access

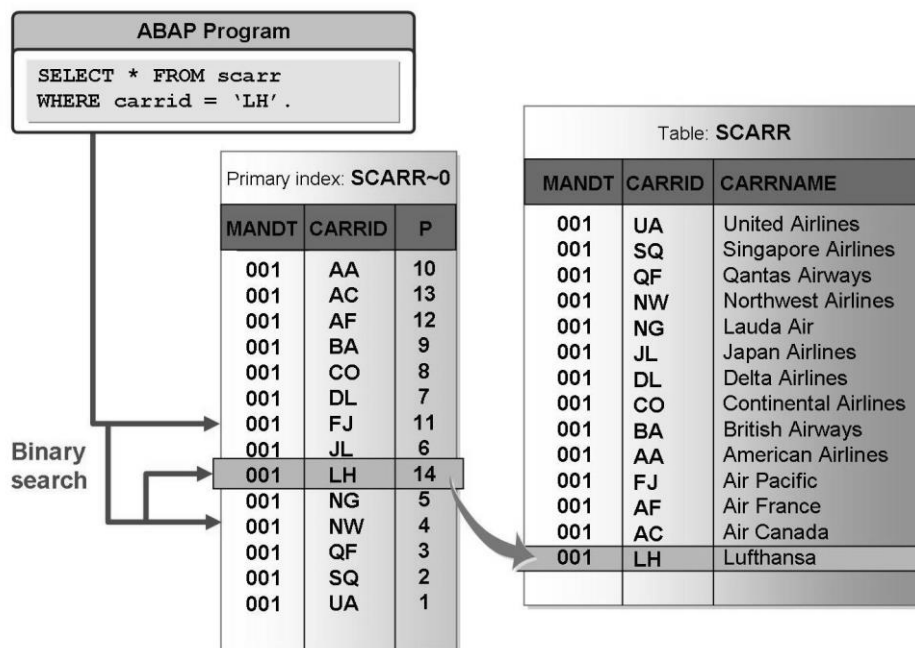
### Lesson Objectives

After completing this lesson, 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

### Business Example

Some transactions contain select statements in their applications, which cause very long runtimes. You should now improve the performance without changing the program.



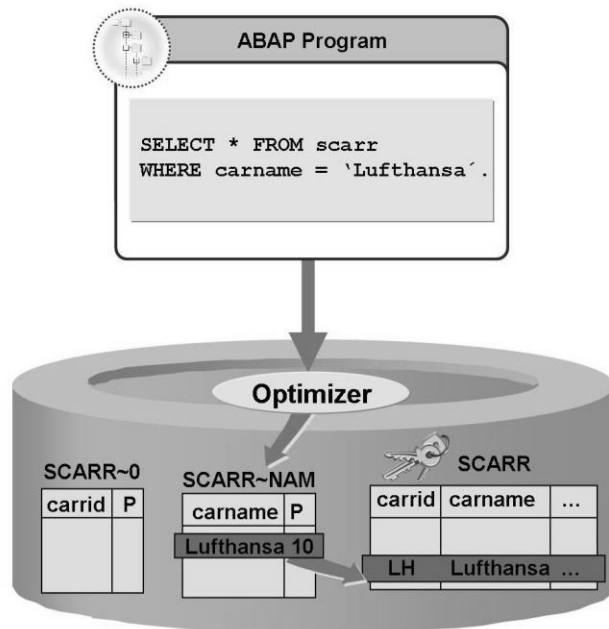
**Figure 30: Structure of an Index**

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:

- An index can only be used up to the last specified field in the selection! The fields that 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 should not have too many indexes.
- Make sure that the indexes on a table are as disjunct as possible.



**Figure 31: Access with Indexes**

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.

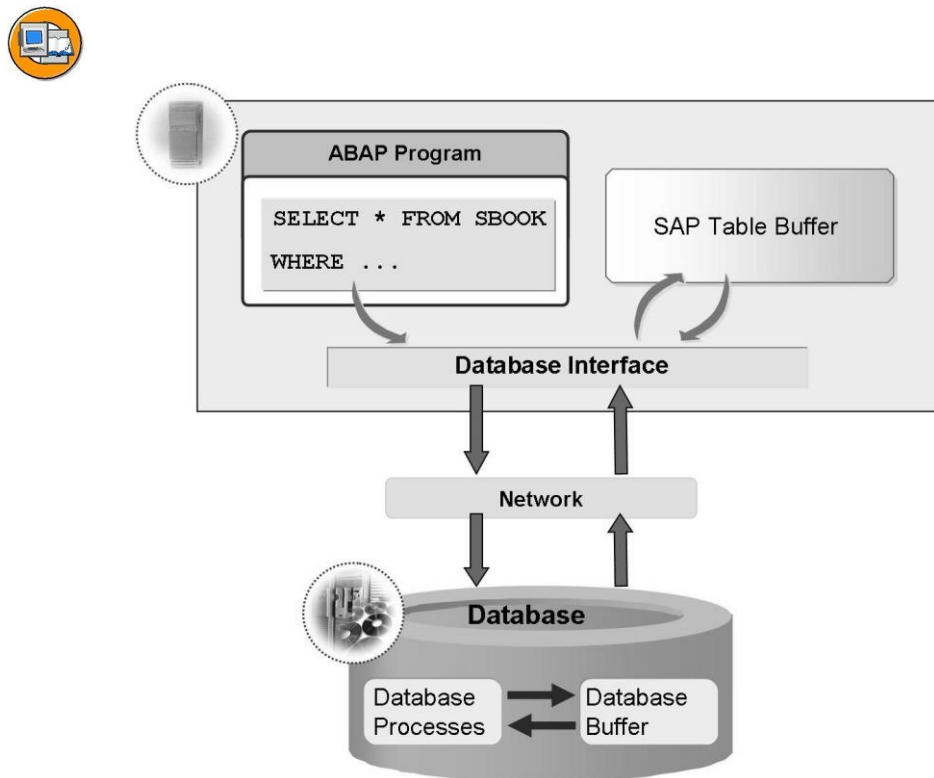
Indexes of a table have a three-place index ID. **0** is reserved for the primary index.

Customers can create their own indexes on SAP tables; their IDs must begin with Y or Z.

If the index fields have key function, for example, 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.

## Improving the Performance through Table Buffering



**Figure 32: Data Access using the Buffer**

Table buffering increases the performance when the records of the table are read.

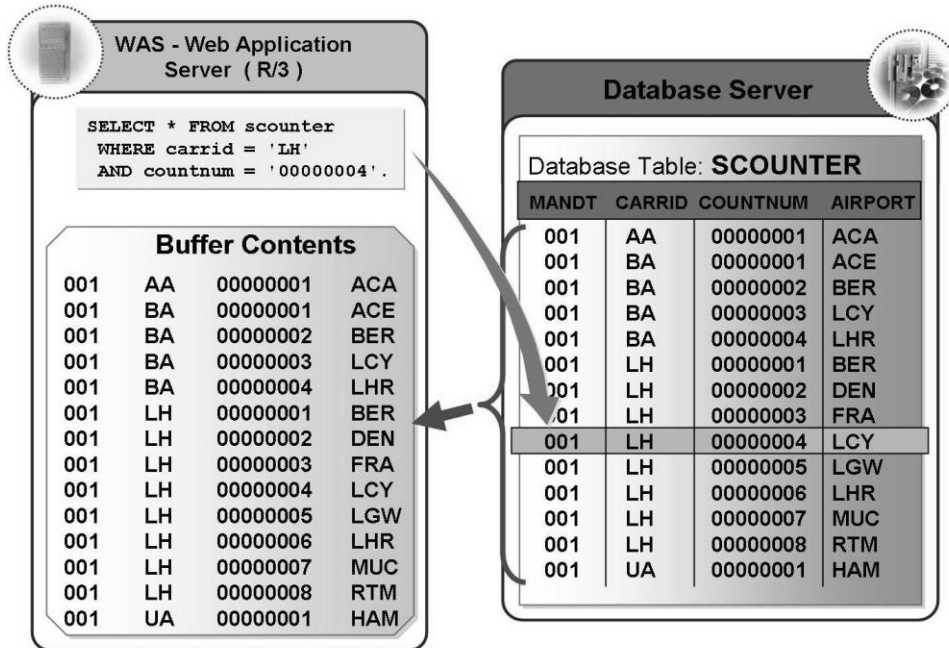
The records of a buffered table are read directly from the local buffer of the application server on which the accessing transaction is running when the table is accessed. This eliminates time-consuming database accesses. The access improves by a factor of 10 to 100. The increase in speed depends on 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.

When you enter `/$TAB` in the command field the system 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.

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 the following buffering types:

- **Full buffering:** When a record of the table is accessed, all the records of the table are loaded into the buffer.
- **Generic buffering:** When a record of the table is accessed, all the records whose left-justified part of the key is the same are loaded into the buffer.
- **Single-record buffering:** Only the record that was accessed is loaded into the buffer.



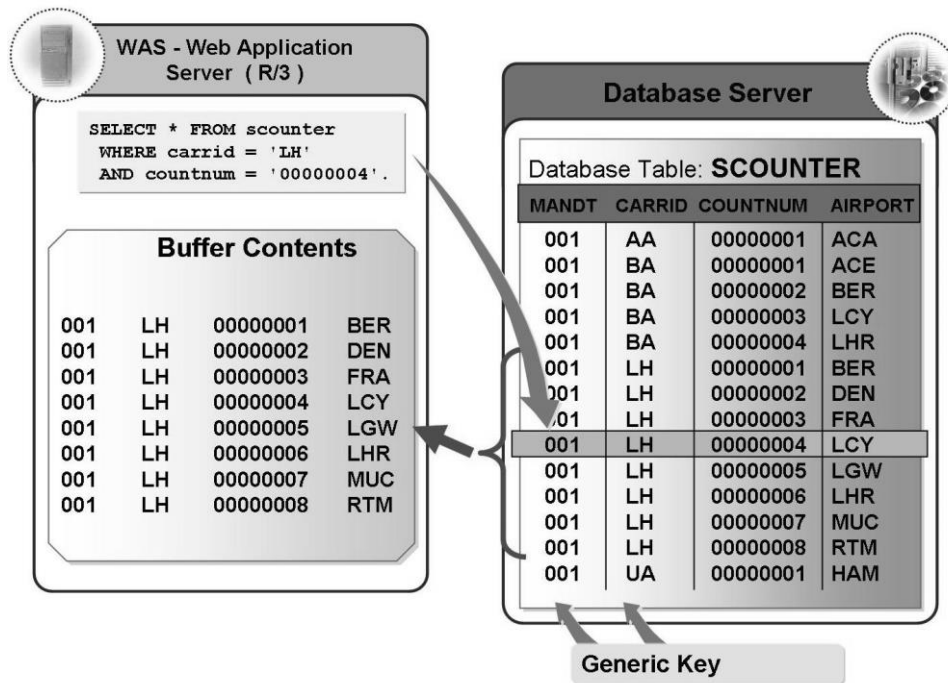
**Figure 33: Full Buffering**

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 that have 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.



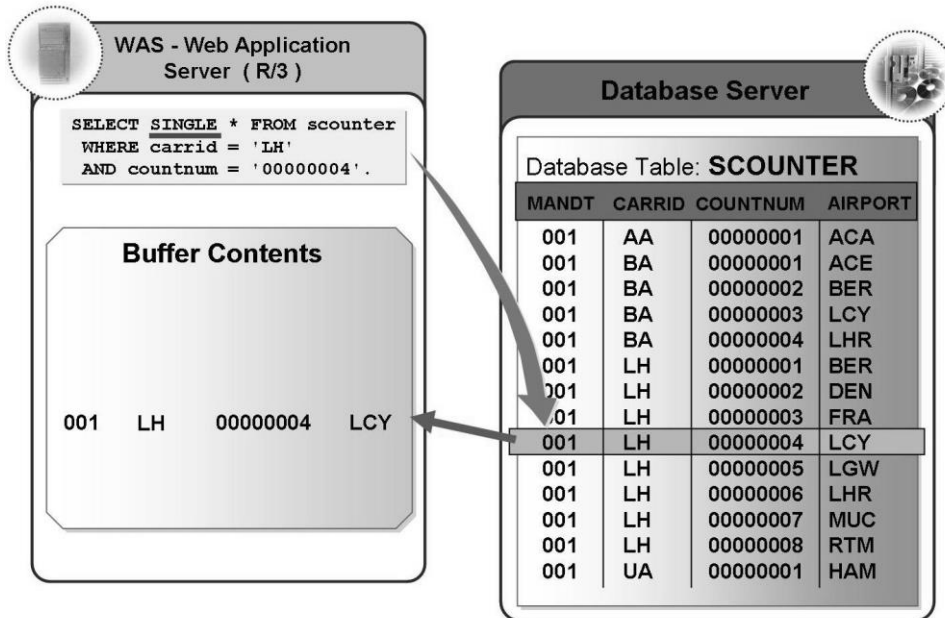
**Figure 34: Generic Buffering**

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.



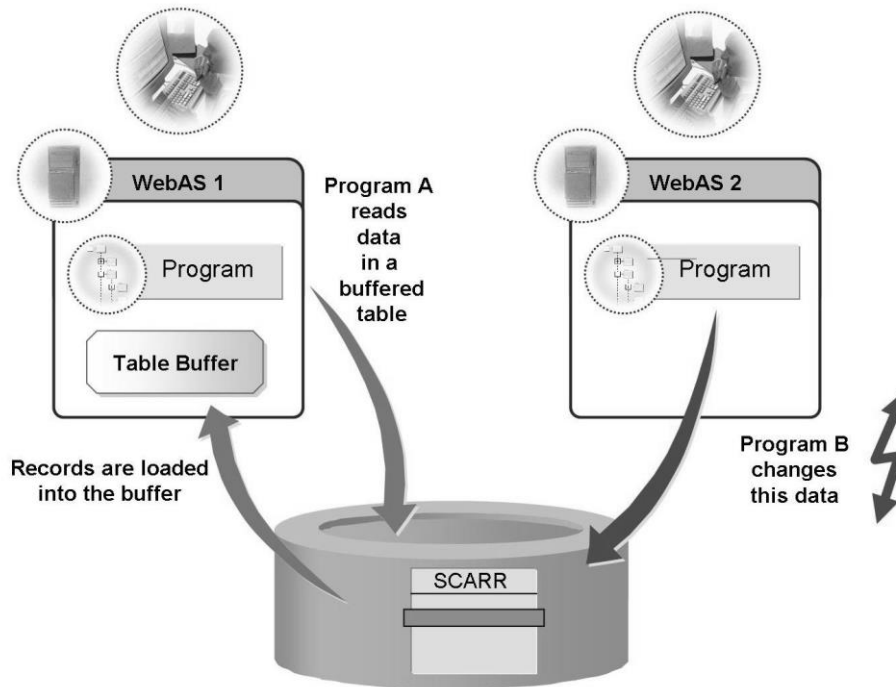
**Figure 35: Single-Record Buffering**

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.

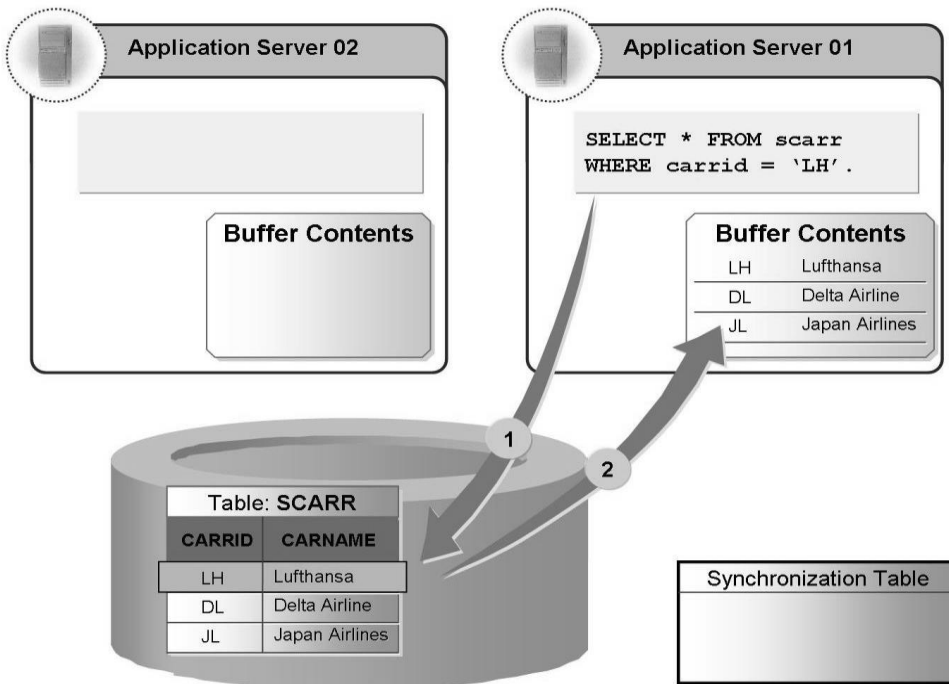
If you access a record that was not yet buffered using SELECT SINGLE, there is a database access to load the record. If the table does not contain a record with the specified key, this record is recorded in the buffer as non-existent. This prevents a further database access if you make another access 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.



**Figure 36: Table buffering**

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 directly from the database and loaded into the buffer. The buffer can therefore satisfy the next access to this data.



**Figure 37: Buffer Synchronization 1**

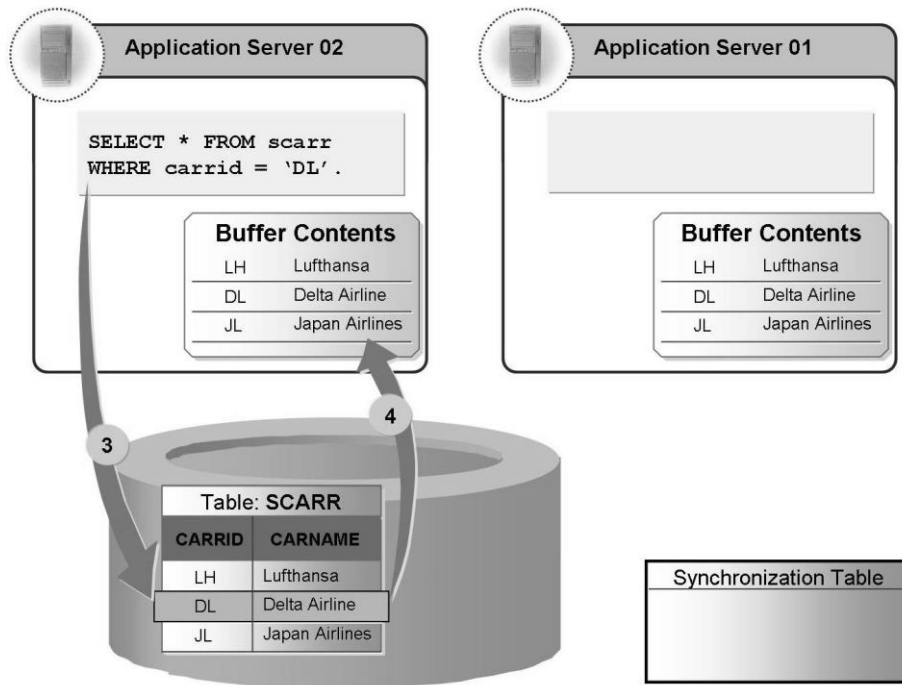
Since the buffers reside locally on the application servers, they must be synchronized after data has been modified in a buffered table. Synchronization takes place at fixed time intervals that can be set in the system profile. The corresponding parameter is “rdisp/bufreftime” and defines the length of the interval in seconds. The value must lie between 60 and 3600. We recommend a value between 60 and 240.

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 SCARR to be fully buffered. The table therefore does not yet reside in the local buffers of the two servers.

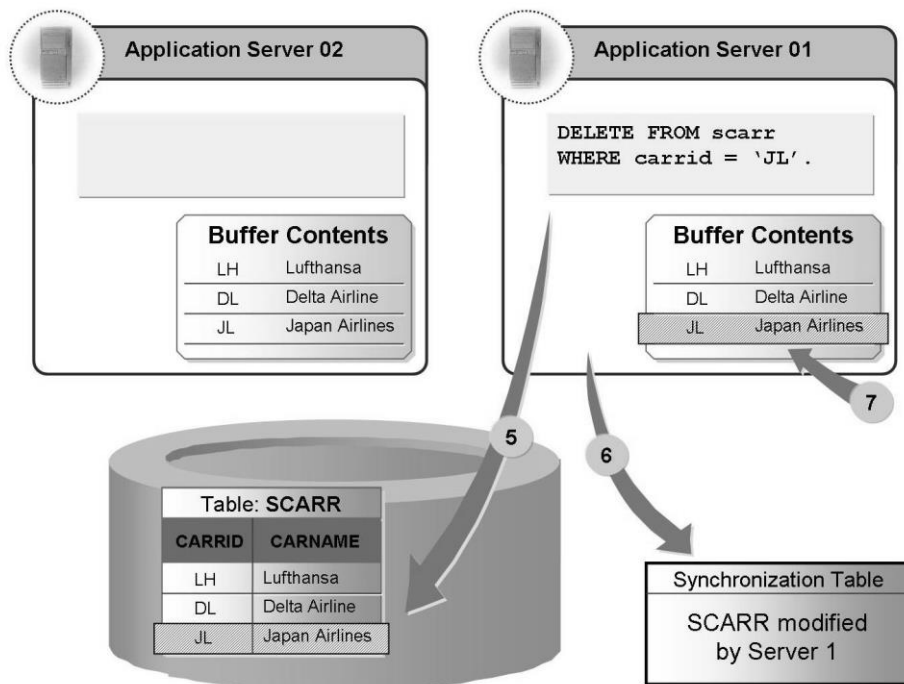
- **Timepoint 1:** Server 1 reads records from table SCARR on the database.
- **Timepoint 2:** Table SCARR is fully loaded into the local buffer of Server 1. The local buffer of this server is now used for access from Server 1 to the data of the SCARR table.





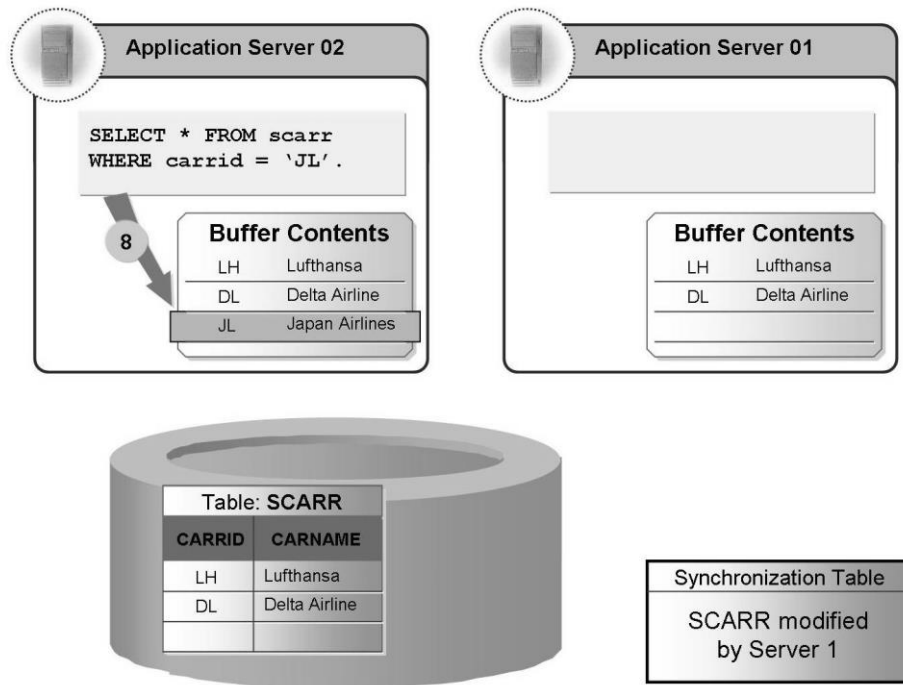
**Figure 38: Buffer Synchronization 2**

- **Timepoint 3:** A user on 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:** The SCARR table is loaded into the local buffer of Server 2. Server 2 therefore also uses its local buffer to access its data when it next reads the SCARR table.



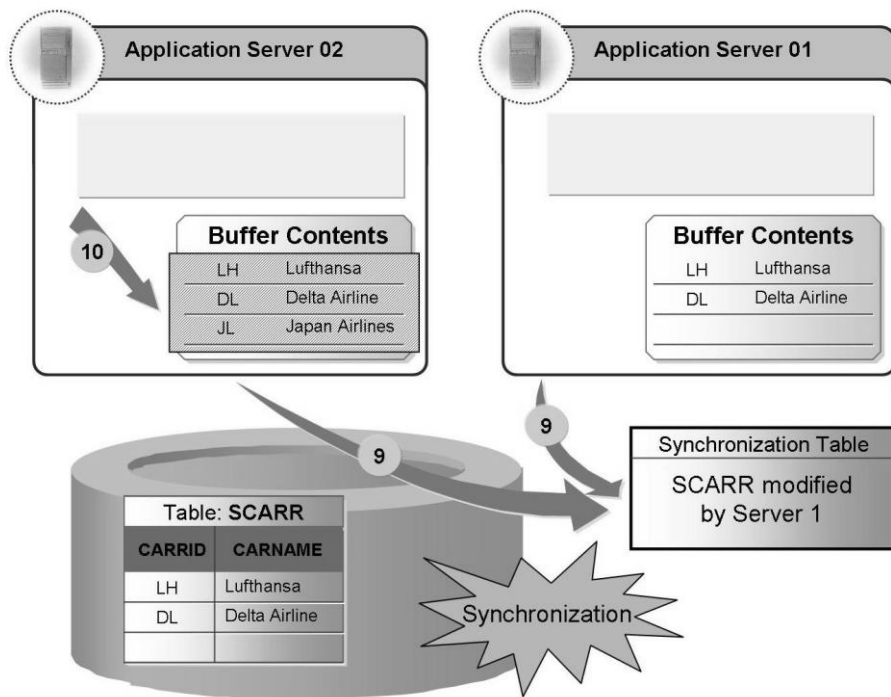
**Figure 39: Buffer Synchronization 3**

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



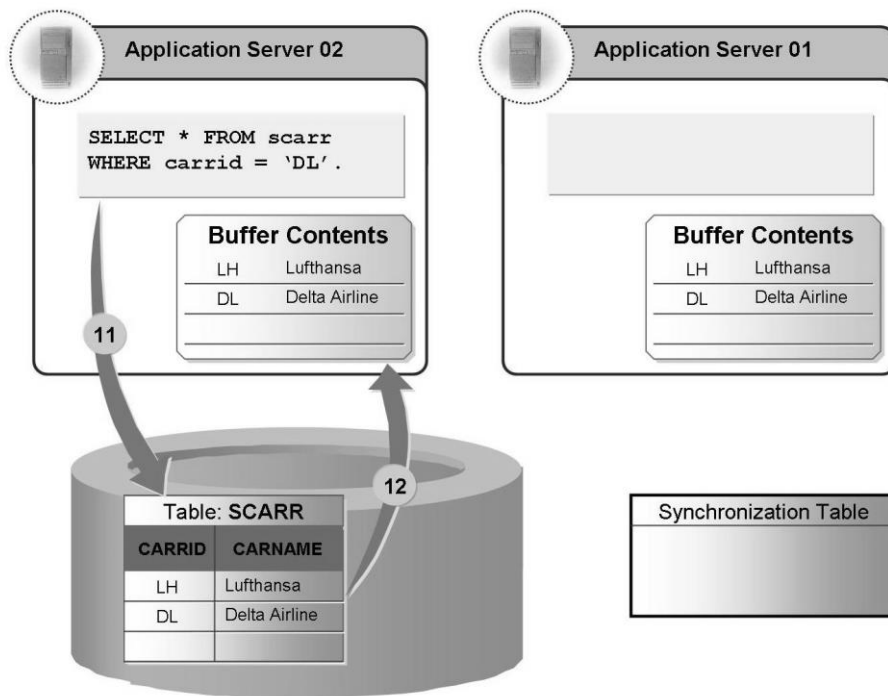
**Figure 40: Buffer Synchronization 4**

- **Timepoint 8:** A user on Server 2 accesses the deleted records. Since the SCARR table 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 depends on the server on which it is running.



**Figure 41: 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 the SCARR table 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 the SCARR table therefore uses the database. Server 1 must not invalidate the table in its buffer, as it is the only changer of the SCARR table itself. Server 1 therefore uses its local buffer again the next time to access records of SCARR table.



**Figure 42: Buffer Synchronization 6**

- **Timepoint 11:** Server 2 again accesses records of SCARR table. Since SCARR is invalidated in the local buffer of Server 2, the access uses the database.
- **Timepoint 12:** The table is reloaded into the local buffer of Server 2. The information via the SCARR table is now inconsistent again on the 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 that 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.

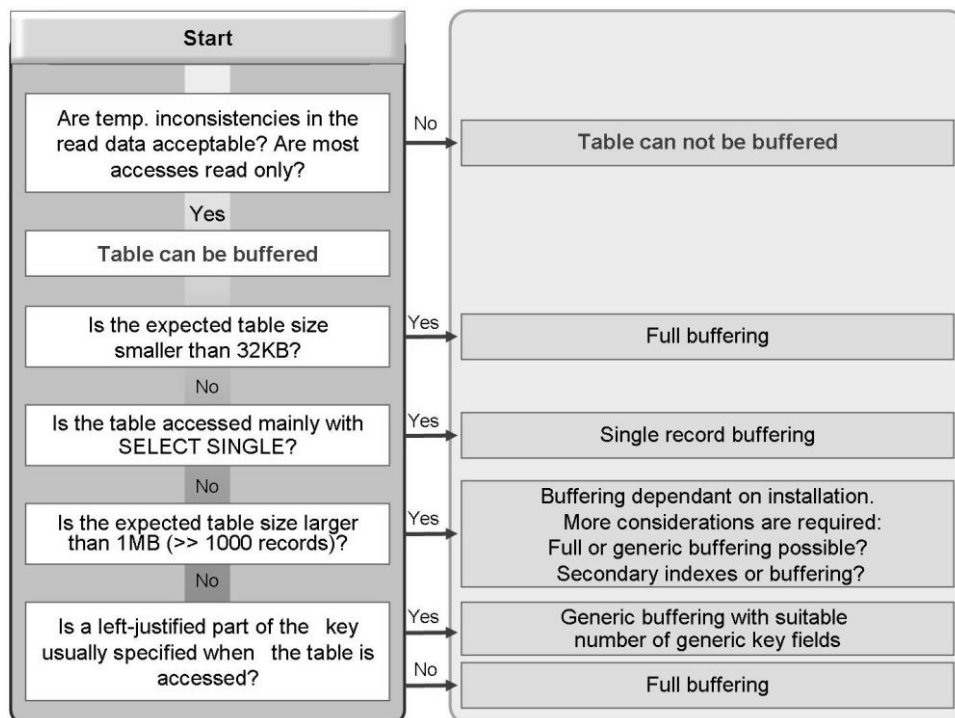
An index helps you to speed up read accesses to a table. An index can be considered a sorted copy of the table that is reduced to the index fields.

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.



**Figure 43: Decision Tree for Buffering**

The decision-making tree for buffering tables should be used for support on your development system. The information mentioned above is shown here as a diagram.

## **Unit 4: Input Checks**

### **Unit Overview**

In this chapter, you will get to know the options for defining input checks in the Dictionary.

### **Unit Objectives**

After completing 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

### **Unit Contents**

Lesson: Consistency Through Input Checks.....56

## **Lesson: Consistency Through Input Checks**

### **Lesson Overview**

This lesson deals with the implementation of input check on user dialogs. Input checks are carried out automatically if fixed values are assigned to the underlying domain or if a foreign key relationship is defined for the underlying structure field.



### **Lesson Objectives**

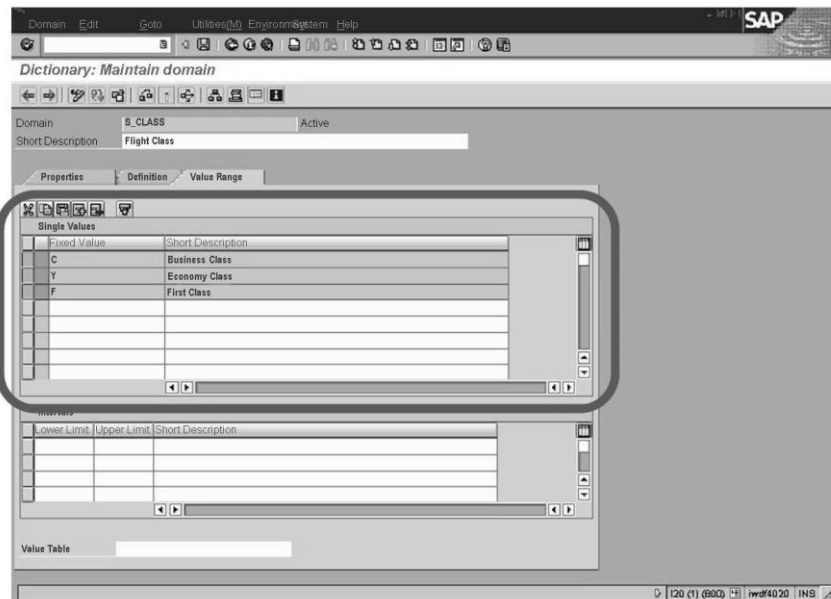
After completing this lesson, you will be able to:

- 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

### **Business Example**

When entering values in your application, further checks on the screens should already occur without additional ABAP coding.

### **Input check via the technical domains**



**Figure 44: Fixed values**

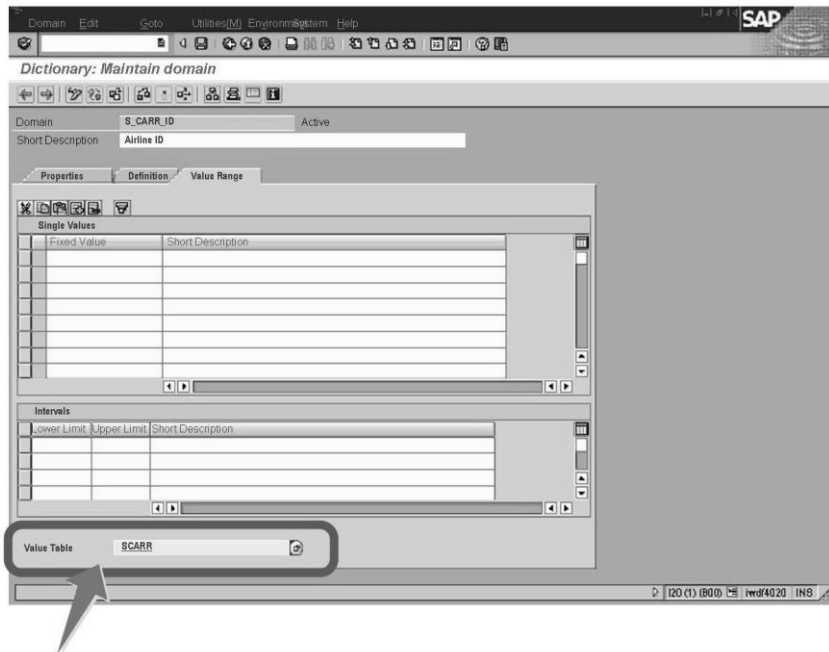


## Lesson: Consistency Through Input Checks

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, these 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 F4 help.

Fixed values can either be listed individually or defined as an interval.



**Figure 45: Value table**

The value range of a field can also be defined by specifying a value table in the domain.

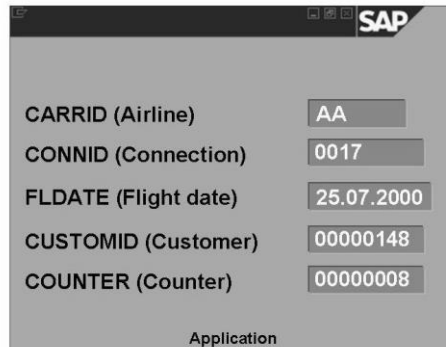
In contrast to fixed values, however, simply specifying a value table does not cause the input to be checked. There is no F4 help either.

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.**

## Unit 4: Input Checks

### Entries to fields in table SBOOK (flight booking):



SAP

CARRID (Airline) AA

CONNID (Connection) 0017

FLDATE (Flight date) 25.07.2000

CUSTOMID (Customer) 00000148

COUNTER (Counter) 00000008

Application



Can this flight be  
booked at sales  
counter 8 ?

### Check table for sales counter

SCOUNTER			
mandt	carrid	counter	...
001	AA	00000001	
001	BA	00000001	
001	BA	00000002	
001	BA	00000003	
001	BA	00000004	
001	LH	00000001	
001	LH	00000002	
001	LH	00000003	
001	LH	00000004	
001	LH	00000005	
001	LH	00000006	
001	LH	00000007	
001	LH	00000008	
001	UA	00000001	

**Figure 46: Inserting a Data Record**

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 airlines.

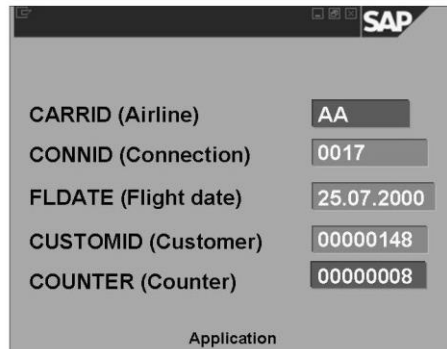
Table SCOUNTER contains all the valid counters of the airlines.

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?

## Consistency Through Input Checks

### Entries to fields in table SBOOK (flight booking):



SAP

CARRID (Airline) AA

CONNID (Connection) 0017

FLDATE (Flight date) 25.07.2000

CUSTOMID (Customer) 00000148

COUNTER (Counter) 00000008

Application



Effect of the foreign key definition:  
A data record containing:  
MANDT = '001', CARRID = 'AA',  
COUNTNUM = '000000008' does not  
exist in table SCOUNTER.

### Check table for sales counter

SCOUNTER			
mandt	carrid	counter	...
001	AA	00000001	
001	BA	00000001	
001	BA	00000002	
001	BA	00000003	
001	BA	00000004	
001	LH	00000001	
001	LH	00000002	
001	LH	00000003	
001	LH	00000004	
001	LH	00000005	
001	LH	00000006	
001	LH	00000007	
001	LH	00000008	
001	UA	00000001	

**Figure 47: Violation of the Foreign Key Check**

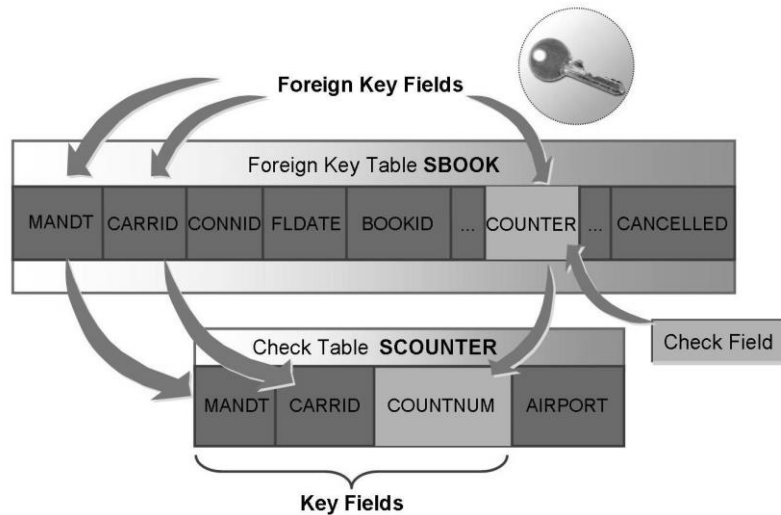
The flight cannot be booked because American Airlines (AA) does not have a counter 8.

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.

## Consistency Through Input Checks



**Figure 48: 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. The **check table** SCOUNTER contains exactly this information. Each counter is identified with three key fields in this table: MANDT, CARRID, and COUNTNUM.

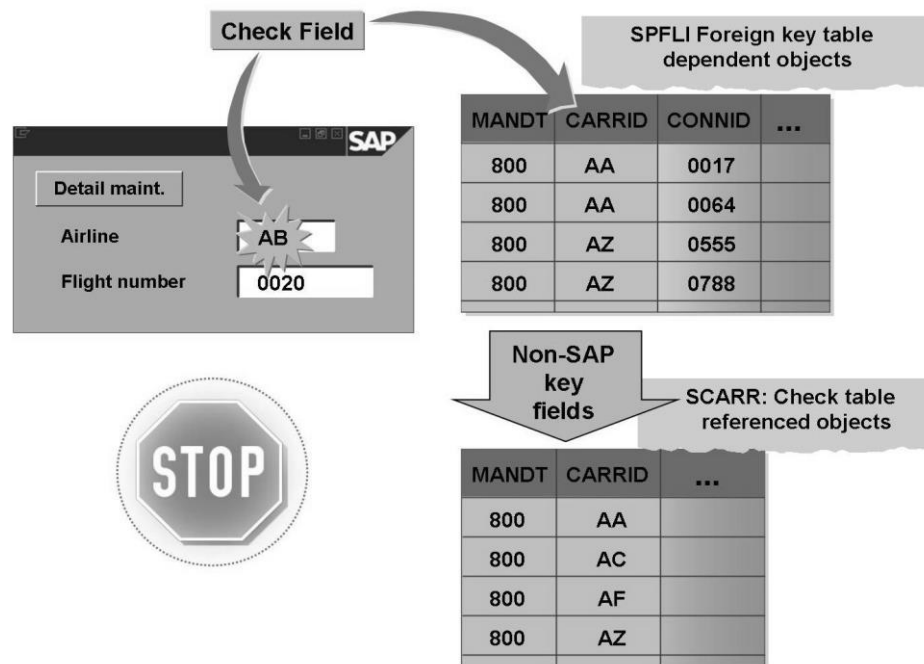
In order to define the foreign key, these three fields are assigned to the fields of the foreign key table (foreign key fields) with which the input to be checked is entered on the screen. In table SBOOK, these are the fields: MANDT, CARRID, COUNTER. The entry is accepted if it represents a valid counter; 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. The COUNTER field 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-CLIENT	SBOOK-MANDT
SCOUNTER-CARRID	SBOOK-CARRID
SCOUNTER-COUNTNUM	SBOOKCOUNTER

## Consistency Through Input Checks



**Figure 49: Data Consistency through Foreign Keys**

A combination of fields in 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*.

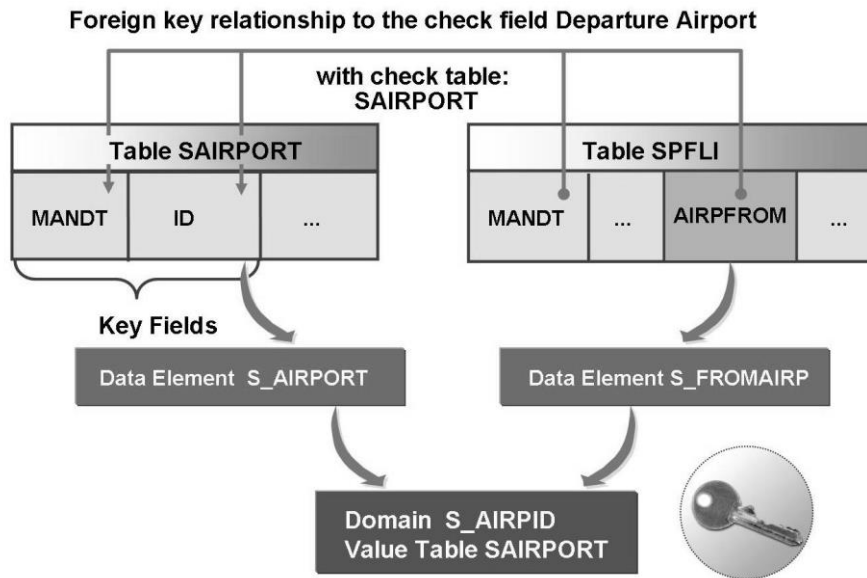
An entry is to be written in the foreign key table. This entry must be consistent with the key fields of the check table.

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), for example, the checks are on this field. The

## Consistency Through Input Checks



corresponding check table is table SCARR with the primary key fields CLIENT and CARRID.

**Figure 50: Foreign Key Definitions in the Check Field**

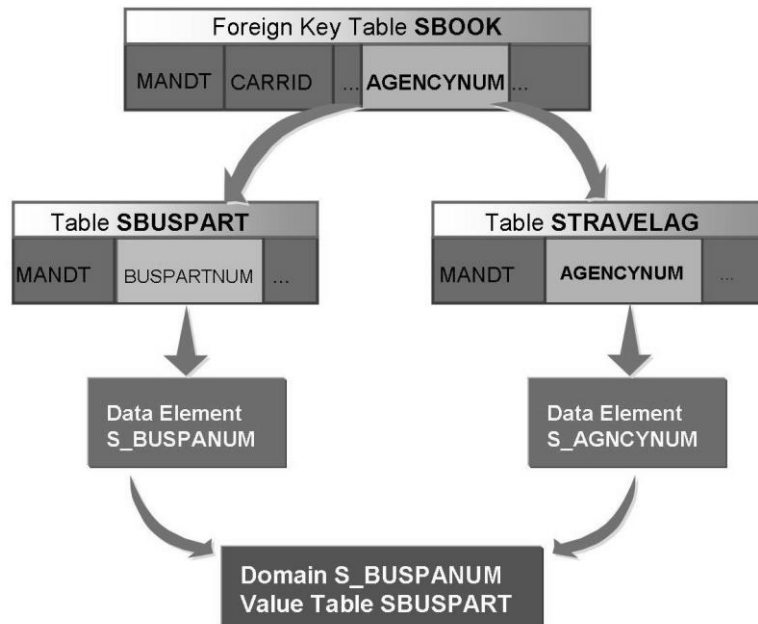
In the ABAP Dictionary, the same domain is required for the check field and referenced key field of the check table so that you do not compare fields with different data types and field lengths. **Domain equality is essential. Different data elements can be used, but they must refer to the same domain.**

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 would be inconsistent if, for example, the field length were 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-ID is correct! However, a foreign key is never defined on this field (avoiding a loop).**

## Consistency Through Input Checks



**Figure 51: Check Table not Equal to Value Table**

If the reusability of the domains is also inserted at unsuitable places, the proposal may be unsuitable for the check table. In the above case, a primary key field exists in several tables that is based on the same domain as the check field. The system proposal is then the value table of the domain. In order to avoid this problem from the beginning

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:**

<b>Check table</b>	<b>Foreign key table</b>
SBUSPART-CLIENT	SBOOK-MANDT
SBUSPART-BUSPARTNUM	SBOOK-AGENCYNUM

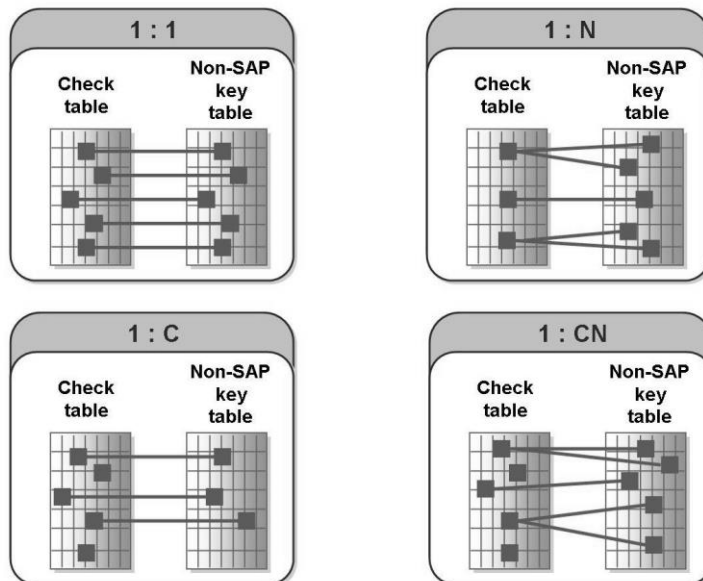
### **This proposal doesn't do what you want:**

The SBUSPART table 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.

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.



## Cardinality (Multiplicity)



**Figure 52: Semantic Attributes**

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.

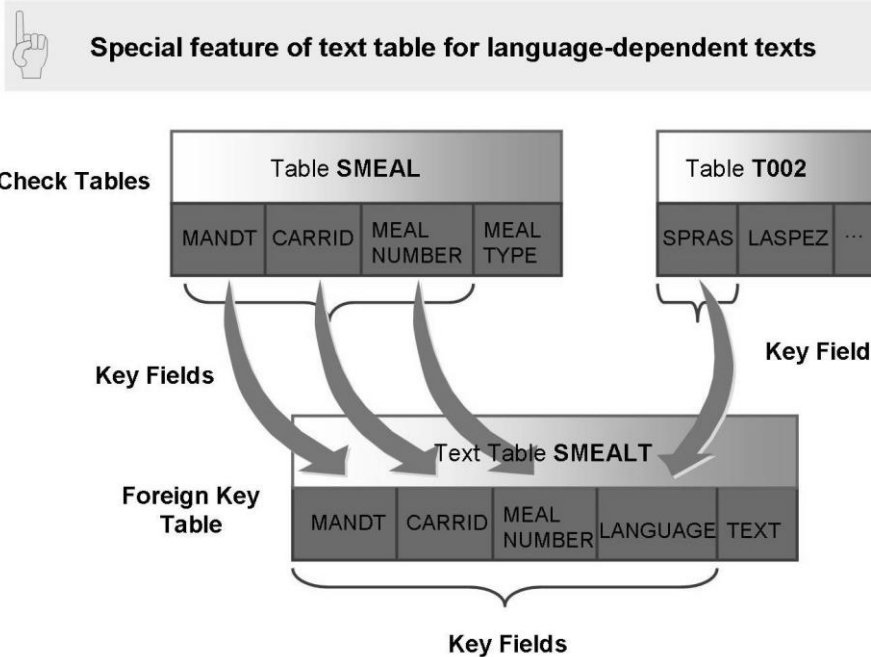
There are the following kinds of foreign key fields:

- **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 in the check table. For example, the key of the foreign key table differs from the key of the check table only in



## Consistency Through Input Checks

that it has an additional language key field. This is a special case of the category *Key fields / candidates*.



**Figure 53: Text table**

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 linked with a table.

## **Lesson Summary**

You should now 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

## Appendix

### Appendix

```
REPORT sapbc430s_struct_deep.
DATA wa_person TYPE zperson01.
DATA wa_phone TYPE str_phone.
START-OF-SELECTION.
*Fill deep structure with data
wa_person-name-firstname = 'Harry'.
wa_person-name-lastname = 'Potter'.
wa_person-street = 'Privet Drive'.
wa_person-nr = '3'.
wa_person-zip = 'GB-10889'.
wa_person-city = 'London'.
wa_phone-p_type = 'P'.
wa_phone-p_number = '+31-10-9938990'.
INSERT wa_phone INTO TABLE wa_person-phone.
wa_phone-p_type = 'F'.
wa_phone-p_number = '+31-10-9938991'.
INSERT wa_phone INTO TABLE wa_person-phone.
wa_phone-p_type = 'M'.
wa_phone-p_number = '+31-79-12211433'.
INSERT wa_phone INTO TABLE wa_person-phone.
*Write on List
WRITE: / wa_person-name-firstname ,
wa_person-name-lastname ,
wa_person-street ,
wa_person-nr ,
wa_person-zip ,
wa_person-city .
WRITE: / 'Phone-Numbers:'.
LOOP AT wa_person-phone INTO wa_phone.
WRITE: AT 20 wa_phone-p_type, wa_phone-p_number.
NEW-LINE.
ENDLOOP.
```