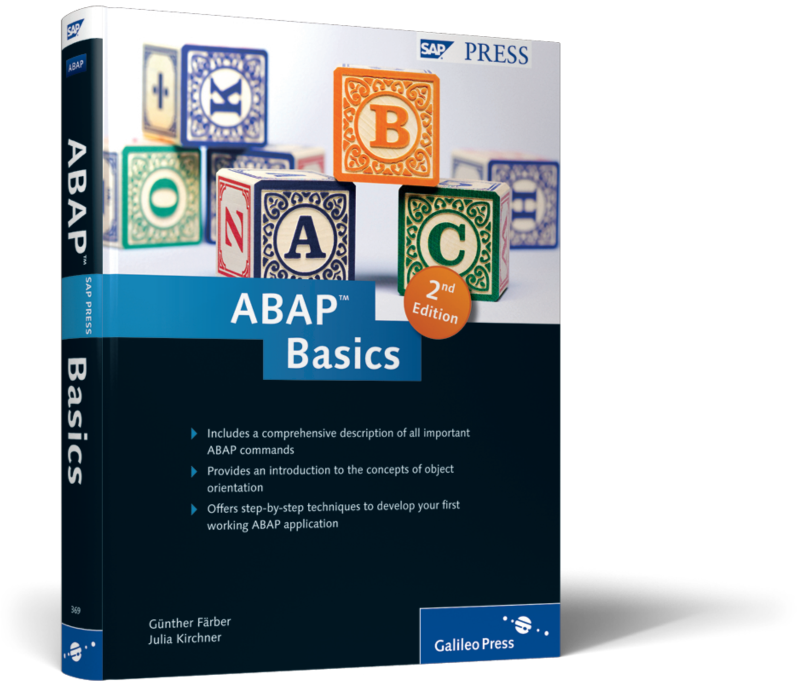
Günther Färber and Julia Kirchner

ABAP™ Basics



Galileo Press®

Bonn • Boston

Contents at a Glance

1 Technical Overview and Getting Started in the System 19

2 Working with the Development Environment: SAP Object Navigator 55

3 Procedural Language Elements 117

4 Defining and Managing Database Tables 205

5 Screen Input and Output 243

6 Object-Oriented Language Elements 343

7 Practice Scenario—Automated Accounting for Corporate Subsidiaries 421

A Programming Guidelines and Tools 479

B Glossary 491

C Sources and Literature Recommendations 495

D The Authors 497

Contents

Preface 11
Introduction 13

1 Technical Overview and Getting Started in the System 19

1.1 Overview of SAP Software and Architecture 19
1.2 Notes for Setting Up Access to SAP Software 24
  1.2.1 Regular System 25
  1.2.2 Test Systems 26
1.3 Programmer's Checklist for Getting Started 27
1.4 Logon, Getting Started, and System Setup 32

2 Working with the Development Environment: SAP Object Navigator 55

2.1 Getting Started with the SAP Object Navigator 59
2.2 Development Classes and Packages 68
2.3 Namespaces 72
2.4 First Program: "Hello World"—REPORT, INCLUDE, and Application Menu 78
2.5 Online Help, help.sap.com, service.sap.com, and SAP Notes 105

3 Procedural Language Elements 117

3.1 Basic Commands and Fields—WRITE, PARAMETERS, MOVE, SY Fields 120
3.2 Data and Data Types—DATA, PARAMETERS, CONSTANTS, FIELD SYMBOLS, TYPE, CREATE, Text Elements 127
3.3 Structures and Internal Tables—TABLE, MOVE-CORRESPONDING, INSERT, APPEND, READ, MODIFY, DELETE, LOOP 151

3.4 Subprograms and Functions—FORM, FUNCTION, EXCEPTIONS 163
3.5 Branches and Logical Expressions—IF, CASE 183
3.6 Loops—WHILE, DO 198

4 Defining and Managing Database Tables 205

4.1 Field Properties—DATA ELEMENT, DOMAIN 207
4.2 Defining and Processing Database Tables—SELECT, INSERT, UPDATE, DELETE 222

5 Screen Input and Output 243

5.1 Simple Input and Output Interfaces—SELECTION SCREEN, SELECT-OPTIONS, CALL SELECTION SCREEN, SAP List Viewer 245
5.2 Complex Input and Output Interfaces—Dynpros 278

6 Object-Oriented Language Elements 343

6.1 Classes and Objects—CLASS, CREATE OBJECT, METHOD 345
6.2 Inheritance and Polymorphism—INHERITING FROM, REDEFINITION 377
6.3 EVENTS 396

7 Practice Scenario—Automated Accounting for Corporate Subsidiaries 421

7.1 Defining the Requirements and a List of Functions for the Application 422
7.1.1 List of Functions 424
7.1.2 Selection List 425
7.2 Software Architecture 427
7.3 Software Design 430
7.3.1 Rough Design 431
7.3.2 Detailed Design 435
7.4 Implementation 445

7.4.1 Dictionary Objects 445
7.4.2 Programs 451
7.4.3 Function Groups 453

Appendices 477

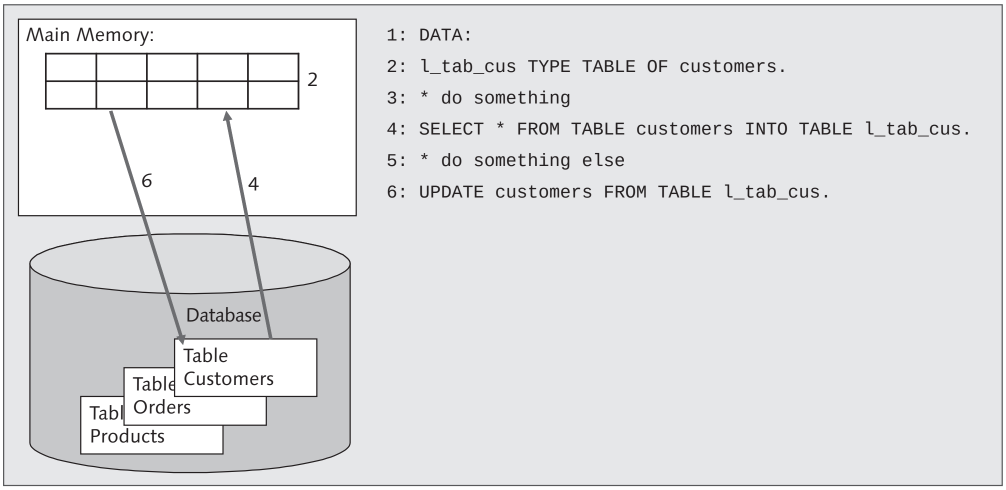
A Programming Guidelines and Tools 479
A.1 Programming guideline 479
A.2 Program Check Tools 489
B Glossary 491
C Sources and Literature Recommendations 495
D The Authors 497

Index 501

In contrast to traditional programming languages, programs and data in ABAP are always stored in a database, and only in rare exceptions are files used. This concept brings with it comparatively high costs; however, there are also clear advantages to this concept that become apparent, especially during the processing and evaluation of business mass data.

4 Defining and Managing Database Tables

Section 1.1, Overview of SAP Software and Architecture, explained that the SAP NetWeaver Application Server (SAP NetWeaver AS) manages all programs and data in a database, which is responsible for the long-term storage and correct retrieval. The strength of ABAP Objects is the integration of database accesses into the ABAP Objects language (see Figure 4.1), and this option is precisely what this chapter focuses on.



```
1: DATA:
2: l_tab_cus TYPE TABLE OF customers.
3: * do something
4: SELECT * FROM TABLE customers INTO TABLE l_tab_cus.
5: * do something else
6: UPDATE customers FROM TABLE l_tab_cus.
```

Figure 4.1 Integrating Database Accesses into the ABAP Objects Language

Of course, there are a few necessary conditions that this type of data storage needs to satisfy because, in a (relational) database, data can only be stored in the form of tables. These tables must first be created, and their internal row structure must be determined, in such a way as to meet the requirements of your application. This process is called *database design*, and in some enterprises, there are entire departments devoted to this process. In the world of science, there are academic disciplines that are concerned solely with the question of finding the optimum distribution of data in tables so that the read, processing, and write speed of the application can meet requirements optimally. A detailed excursion in that direction exceeds the scope of this book. Therefore, Section 4.2, Defining and Processing Database Tables—SELECT, INSERT, UPDATE, DELETE, simply limits itself to a few significant and basic aspects of this topic.

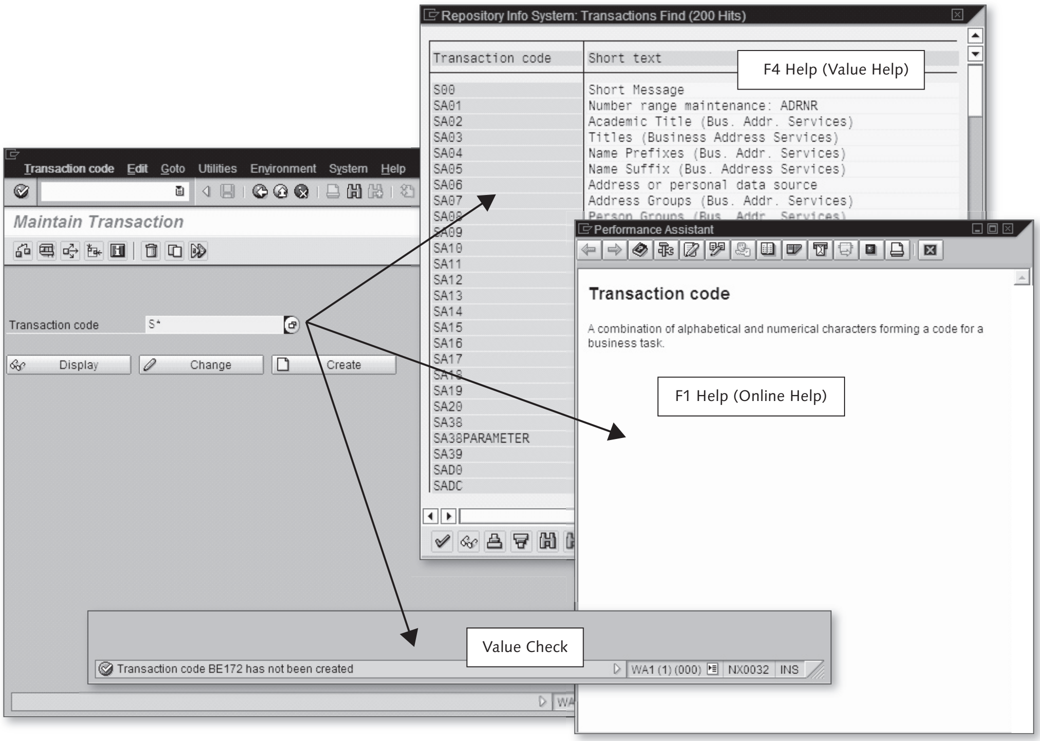
Tips for those familiar with other programming languages

In other programming languages, like Delphi or Java, the creation of database applications is normally carried out using two separate tools. On the one hand, the development interface of the programming language is used for the creation of the program; on the other hand, the administrative interface of a database is used to create the tables required. Then you use a database driver to create a connection between the program and the database; you can use special functions to exchange data between the two.

In ABAP programming, this separation doesn't exist. Instead, database tables are normal development objects in the same way that programs, include files, or text symbols are, and, all together, they are treated as an application, transported from one system to another if necessary, and so on. You don't need to set up a driver, and you need to use ABAP commands only to exchange data directly between memory in your program and the database. You don't need to worry about whether this combination of program and database table will run or not because the ABAP runtime environment ensures compatibility with a broad selection of platforms (operating system plus database management system plus hardware).

4.1 Field Properties—DATA ELEMENT, DOMAIN

The most frequently used data types in ABAP development are not the ABAP data types defined in Section 3.2, Data and Data Types—DATA, PARAMETERS, CONSTANTS, FIELD SYMBOLS, TYPE, CREATE, Text Elements, such as `i`, `c`, or `p`, but *data elements* based on the data types in the database. They ensure considerably stricter type checking, defined value lists, brief descriptions, online help, and other features that far surpass the options of the ABAP data types (see Figure 4.2). This allows them to contribute a great deal to operational convenience and runtime stability.



Transaction code	Short text
S00	Short Message
SA01	Number range maintenance: ADRNR
SA02	Academic Title (Bus. Addr. Services)
SA03	Titles (Business Address Services)
SA04	Name Prefixes (Bus. Addr. Services)
SA05	Name Suffix (Bus. Address Services)
SA06	Address or personal data source
SA07	Address Groups (Bus. Addr. Services)
SA08	Person Groups (Bus. Addr. Services)
SA09	
SA10	
SA11	
SA12	
SA13	
SA14	
SA15	
SA16	
SA17	
SA18	
SA19	
SA20	
SA38	
SA38PARAMETER	
SA39	
SAD0	
SADC	

Transaction code

A combination of alphabetical and numerical characters forming a code for a business task.

Transaction code BE172 has not been created

Figure 4.2 Features of SAP Systems Based on Data Elements

Basics

Data elements are not based on the ABAP data types (see Section 3.2, Data and Data Types—DATA, PARAMETERS, CONSTANTS, FIELD SYMBOLS, TYPE, CREATE, Text Elements), but on the data types of the underlying database, the *dictionary data types*. They are used to define the individual columns of a database table; furthermore, they can be used as data types in the ABAP programs. The latter is covered in this chapter; Section 4.2, Defining and Processing Database Tables—SELECT, INSERT, UPDATE, DELETE, and details their use in tables.

Data elements ensure better readability and clarity of the program because the programmer has not only additional descriptions and help, but also a function of the Object Navigator, which can show all uses within source code and tables at the click of a button (the *usage explanation*, a button that is displayed on the toolbar of the SAP Object Navigator). So working with data elements in an ABAP program works entirely to your advantage. In professional ABAP, development data elements are used almost exclusively for the typing of data. The entire concept has become so fundamental that SAP is also porting it for Java development. This is reason enough to look a little more closely at data elements and their possible uses in ABAP programs.

From dictionary data type to data type in a program

The path from the definition of a data element to its use in a program generally includes several steps (see Figure 4.3), each of which will be examined in the following subsections:

1. Selection of a dictionary data type
2. Definition of a domain referring to the dictionary data type
3. Definition of a data element referring to the domain
4. Declaration of data within the program source code, referring to the data element as the data type

SAP supports this path with convenient user interfaces in the SAP Object Navigator for entry of the information required, along with the activation mechanism already familiar from working with source code, which greatly simplifies working in teams.

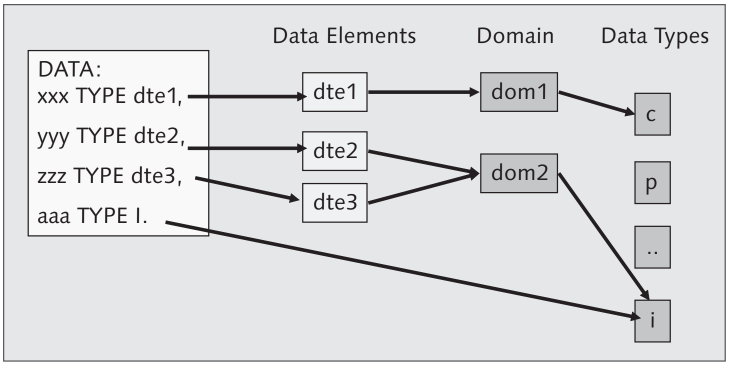


Figure 4.3 Hierarchy of Data, Data Elements, and Domains

Selecting a dictionary data type—determining basic properties

The ABAP programming language provides an entire series of dictionary data types, but they cannot be used directly in a program. Instead, they must serve as basic components. Table 4.1 shows the most important dictionary data types and the *mapping* between the dictionary data type and an ABAP data type.

Dictionary Data Type	Characters Permitted	Description	ABAP Data Type
`CHAR n`	1 – 255	Character string	`C(n)`
`CLNT`	3	Client	`C(3)`
`CUKY`	5	Currency key, referenced by `CURR` fields	`C(5)`
`CURR n, m, s`	1 – 17	Currency field, stored as `DEC`	`P((n + 2) / 2) DECIMALS m [NO-SIGN]`
`DEC n, m, s`	1 – 31, in tables 1 – 17	Computed or amount field with decimal point and sign	`P((n + 2) / 2) DECIMALS m [NO-SIGN]`
`DATS`	8	Date field (YYYYMMDD), stored as `CHAR(8)`	`D`
`FLTP`	16	Floating point number with eight bytes precision	`F`
`INT4`	10	Four-byte integer, whole number with sign	`I`

Table 4.1 Built-in Dictionary Data Types

Dictionary Data Type	Characters Permitted	Description	ABAP Data Type
`NUMC n`	1 – 255	Character string with digits only	`N(n)`
`QUAN n, m, s`	1 – 17	Quantity field, shows a unit field with format `UNIT`	`P((n + 2) / 2) DECIMALS m [NO-SIGN]`
`STRING`	256 – ...	Character string with variable length	`STRING`
`TIMS`	6	Time field (HHMMSS), stored as `CHAR(6)`	`T`
`UNIT n`	2 – 3	Unit key for `QUAN` fields	`C(n)`

Table 4.1 Built-in Dictionary Data Types (Cont.)

Defining a domain—determining technical properties

Dictionary data types serve as the basis for the definition of *domains*. Domains refer to the dictionary data type, thus defining their basic technical suitability as amount field, quantity field, numeric field, text field, and so on. Furthermore, you can also define additional technical properties (for instance, that amounts can only have positive signs, numbers may only take values from a certain range, or values must already be present in a different table, that is, a master data check). Newcomers know these kinds of technical properties as *constraints*, but what you are doing here is nothing more than defining limitations, which are largely automatically checked during execution of the programmer. Once a domain is activated, it can be used as the basis for the definition of data elements.

Defining a data element—determining descriptive properties

Data elements generally refer to a domain and determine their technical characteristics in that way. Moreover, they also define descriptive properties like text labels in different lengths for optimum output in places with little or a lot of room, brief descriptions for output as tooltips, and online help for detailed description with links to related topics. If you look at the richness of the terms used in the departments of today's enterprises and the software to support them, you will quickly realize that this kind of descriptive property is very important. It's important to know, when needed, what a "legal unit," an "accounting metric," or an "inventory conversion component" is and how that information needs to be entered. All the texts in a data element are automatically added to the worklist for the translator, and ABAP is careful when displaying input and output fields to select the proper local language. Once a data element is activated, it can be used in program code as a data type.

Declaring data—using data elements as data types

Within the program code, data elements are treated in the same way as ABAP data types, and can be used to type constants, variables, field symbols, and interface parameters. For instance, if you've defined a `zroom_number` data element to store the room number in a hotel, the declaration will hardly be different from that of the corresponding ABAP data type:

```
PARAMETERS:
  p_room_number TYPE zroom_number OBLIGATORY VALUE
    CHECK,
  p_room_number2(3) TYPE numc.
```

Assuming that the `zroom_number` is ultimately based on a dictionary data type `numc` with three characters, during processing you won't see any difference as long as you don't try to store invalid room numbers in the `zroom_number` domain. If you do, the user will automatically see an error message during input, but in the second case your program must take care of the checking and output of appropriate messages itself. This data check actually only takes place during input, and is performed by the program user interface. In the program, you can assign both parameters any numeric values with three digits without the check stopping you or even noticing. An example in Section 7.4.3, Function Groups, in Function ZPTB00_OBJ_BTC_CHECK shows how you can implement this kind of checking in the program.

Metadata, type checking, and type safety

The totality of all technical and descriptive properties of a data element is called its *metadata*, and in this respect ABAP cannot be compared with any other familiar programming language. ABAP easily gets top marks when it comes to data consistency, type safety, and runtime stability, because when only carefully selected and defined data can be input and processed, the probability that you'll also get the right results is much higher.

Take a look at the practical aspects of working with domains and data elements, as well as the processing in programs.

Exercise 4.1

Create the program ZPTB00_ROOM_CHECKER. As input parameters, request the arrival and departure dates, as well as the room number desired, and check these against those actually available, namely 001–004, 101–104, and 201–204. Use a data element to perform the check.

If the room number has been entered correctly, print a booking confirmation on the screen.

Creating domains

Let's start by creating the domain.

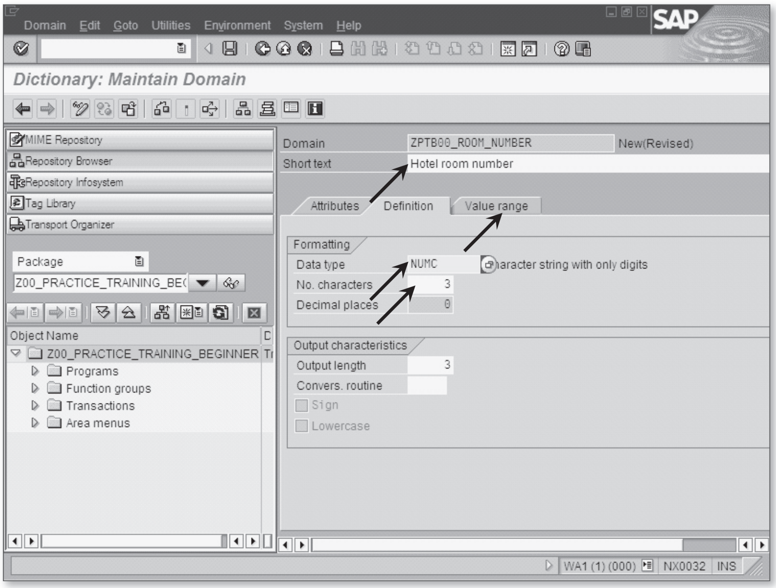
▷ In the context menu of the package or development class, select the CREATE • DDIC OBJECT • DOMAIN menu option.

A dialog box appears, where you must specify the name of the domain.

▷ Enter "ZPTB00_Room_Number" as the name of the domain.

A window is displayed in the tools area of the SAP Object Navigator where you can maintain the domain properties.

▷ As the brief description, enter "Hotel room number", enter "NUMC" as the data type, and "3" as the number of digits (see Figure 4.4).



Domain ZPTB00_ROOM_NUMBER New(Revised)
Short text Hotel room number

Data type NUMC Character string with only digits
No. characters 3
Decimal places 0

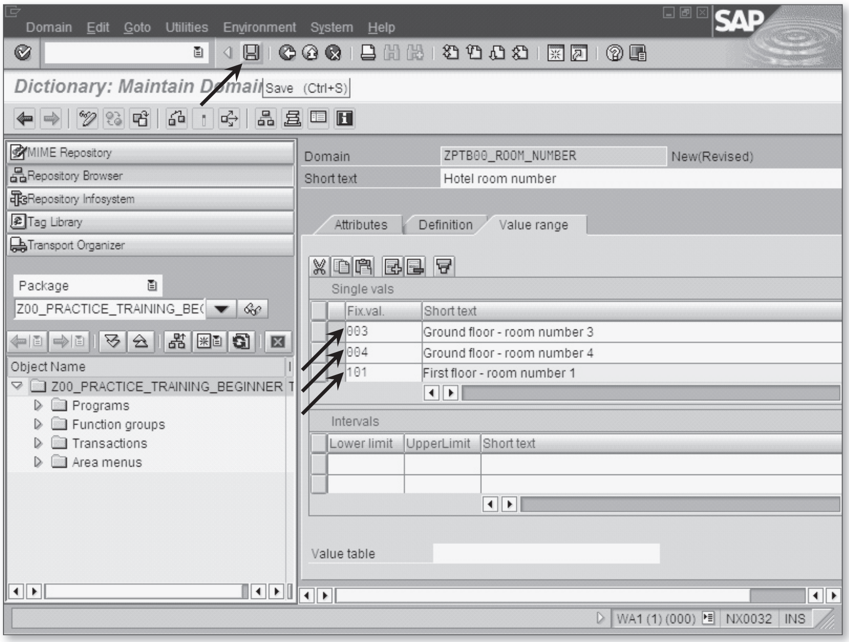
Output length 3
Convers. routine
Sign
Lowercase

Figure 4.4 Editing the Properties of Domain ZPTB00_Room_Number

▷ Select the VALUE RANGE tab.

The configuration options on the VALUE RANGE tab are displayed (see Figure 4.5). There, use the setting options under SINGLE VALS to ensure that a check is automatically performed on the entered values afterwards.

▷ Under SINGLE VALS, enter the room numbers 001 through 004, 101 through 104, and 201 through 204, together with a brief description, and click SAVE.



Domain: ZPTB00_ROOM_NUMBER New(Revised)

Short text: Hotel room number

Package: Z00_PRACTICE_TRAINING_BE(

Fix.val.	Short text
003	Ground floor - room number 3
004	Ground floor - room number 4
101	First floor - room number 1

Lower limit	UpperLimit	Short text

Figure 4.5 Editing More Properties of Domain ZPTB00_Room_Number

A transport dialog box opens, where you must define the request for logging the domain.

▷ Copy the settings unchanged, and click the CONTINUE button.

You can now check and activate the domain.

▷ Click CHECK, and then click ACTIVATE.

Creating a data element

Next create a data element based on this domain.

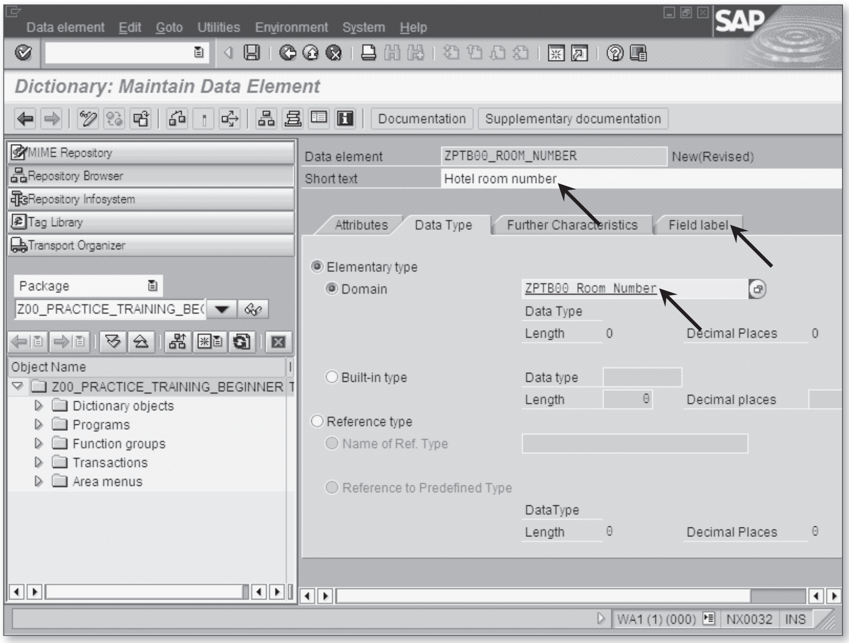
- ▷ Select CREATE • DDIC OBJECT • DATA ELEMENT in the context menu.

A dialog box appears, where you must specify the name of the data element.

- ▷ Enter "ZPTB00_Room_Number" as the name of the domain.

In the tool area of the SAP Object Navigator, a window appears in which you can edit the properties of the data element (see Figure 4.6).

- ▷ Enter "Hotel room number" as the brief description, "ZPTB00_Room_Number" as the domain, and then select the FIELD LABEL tab.



Dictionary: Maintain Data Element

Package: Z00_PRACTICE_TRAINING_BE(

Z00_PRACTICE_TRAINING_BEGINNER

Data element: ZPTB00_ROOM_NUMBER New(Revised)
Short text: Hotel room number

(● Elementary type)
(● Domain) ZPTB00_Room_Number
Data Type
Length 0 Decimal Places 0

○ Built-in type
Data type
Length 0 Decimal places

○ Reference type
○ Name of Ref. Type

○ Reference to Predefined Type
DataType
Length 0 Decimal Places 0

Figure 4.6 Editing the Properties of Domain ZPTB00_Room_Number

The setting options in the FIELD LABEL tab are displayed. Specify field labels of varying lengths for the data element here in order to be able to use the data element in all possible input and output screens.

- ▷ Under SHORT enter the label "RoomNo", under MEDIUM enter "Room number", under LONG enter "Hotel room number", and under HEADING also enter "Hotel room number". The dialog box fills the length information automatically (see Figure 4.7).

▷ Click the Save button.

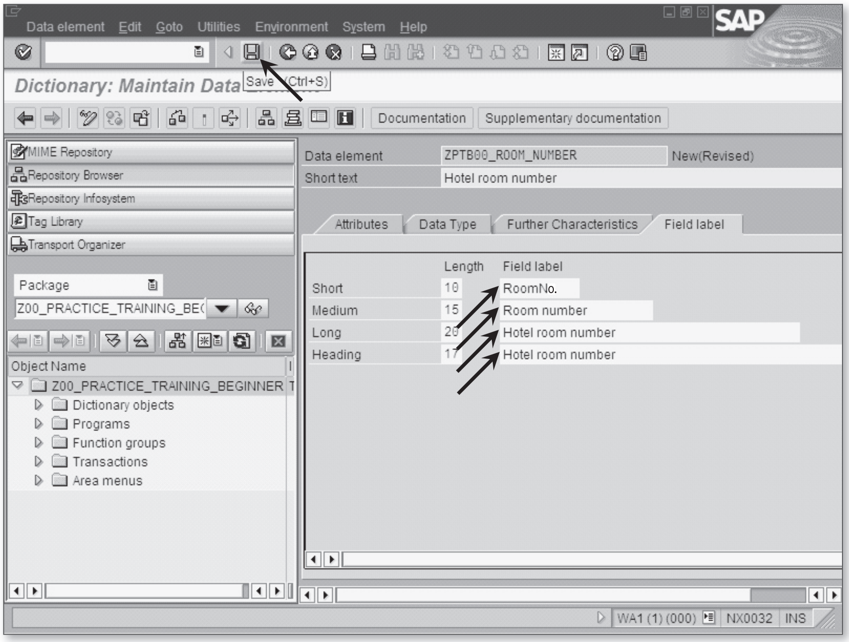
A transport dialog box opens, where you must define the request for logging the data element.

▷ Copy the settings unchanged, and click the Continue button.

You are now supposed to maintain the documentation for the data element to ensure that the user can call online help for this input field.

▷ Click the Documentation button.

A new window appears, where you can enter the documentation for the data element.



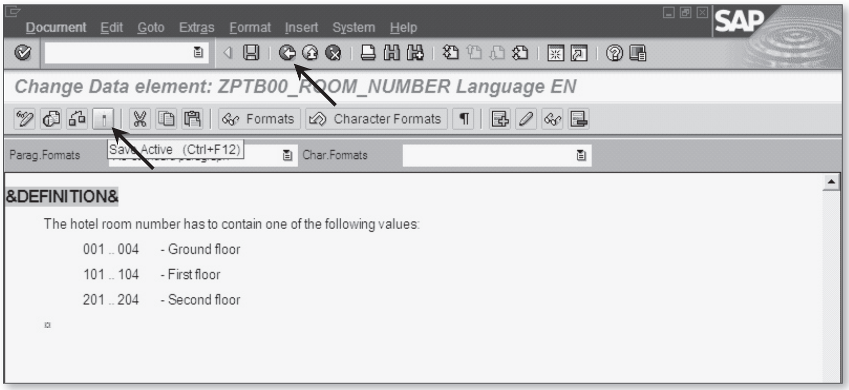
Data element ZPTB00_ROOM_NUMBER New(Revised)
Short text Hotel room number

	Length	Field label
Short	10	RoomNo.
Medium	15	Room number
Long	20	Hotel room number
Heading	17	Hotel room number

Figure 4.7 Editing More Properties of Data Element ZPTB00_Room_Number

▷ Accept the documentation from Figure 4.8.

▷ Click the Save Active button and then select the Back button.


Document Edit Goto Extras Format Insert System Help
Change Data element: ZPTB00_ROOM_NUMBER Language EN
Formats
Character Formats
Save Active (Ctrl+F12)
Parag.Formats
Char.Formats

&DEFINITION&

The hotel room number has to contain one of the following values:

001 .. 004	- Ground floor
101 .. 104	- First floor
201 .. 204	- Second floor

Figure 4.8 Editing the Documentation for the Data Element ZPTB00_Room_Number

You can now check and activate the data element.

▷ Click the CHECK button, then click ACTIVATE.

Creating a program

After this preparation, you can start with the actual program.

▷ Create a new program in the usual way, named "ZPTB00_ROOM_CHECKER", without a TOP include.

▷ Give it the title "Room checker" and leave all the other program properties unchanged. The transport request is also accepted as preset.

As usual, the SAP Object Navigator creates a skeleton program that you can fill out to check the hotel room number, check-in day, and check-out day entries for correctness, and print the data to the screen if the result is positive.

▷ Type the following code under the comment lines of the program skeleton and save:

```
REPORT  zptb00_room_checker.
PARAMETERS:
  p_room TYPE zptb00_room_number OBLIGATORY
  VALUE CHECK,
  p_chkin TYPE d OBLIGATORY,
  p_chkout TYPE d OBLIGATORY.
WRITE: / 'Room Reservation',
       / 'Hotel room number:', p_room,
        / 'Check in day      :', p_chkin,
        / 'Check out day     :', p_chkout.
```

Listing 4.1 Source Code for the Program ZPTB00_ROOM_CHECKER

Explanation of the source code

The parameter `p_room` is based on the data element `zptb00_room_number` you've defined. So that a check of the fixed values entered in the domain will take place, you have to use the `OBLIGATORY` and `VALUE CHECK` clauses. At runtime, then, entries are checked and a value help is also available that the user can call up using the F4 key. Of course, the online help for the entry field is also available, and is displayed when the F1 key is pressed. The other parameters accept the check-in and check-out dates, and, because these are date fields, a value help is automatically generated for them.

Once all the data has been correctly entered, the information collected is shown on the screen.

Tip

After activation of the program, if you still make changes to the individual allowed values in the domains, they won't appear automatically in the value help. Instead, you must provide a little manual help by temporarily renaming the parameter, activating the program again, and then changing the name back. Only then is the compiler forced to redo the generation and take into account the current individual values from the domain.

Editing the selection text

For the clarity of the program, it makes sense to give the parameters descriptive labels. Let's take care of this important step, which is essential for professional ABAP programming.

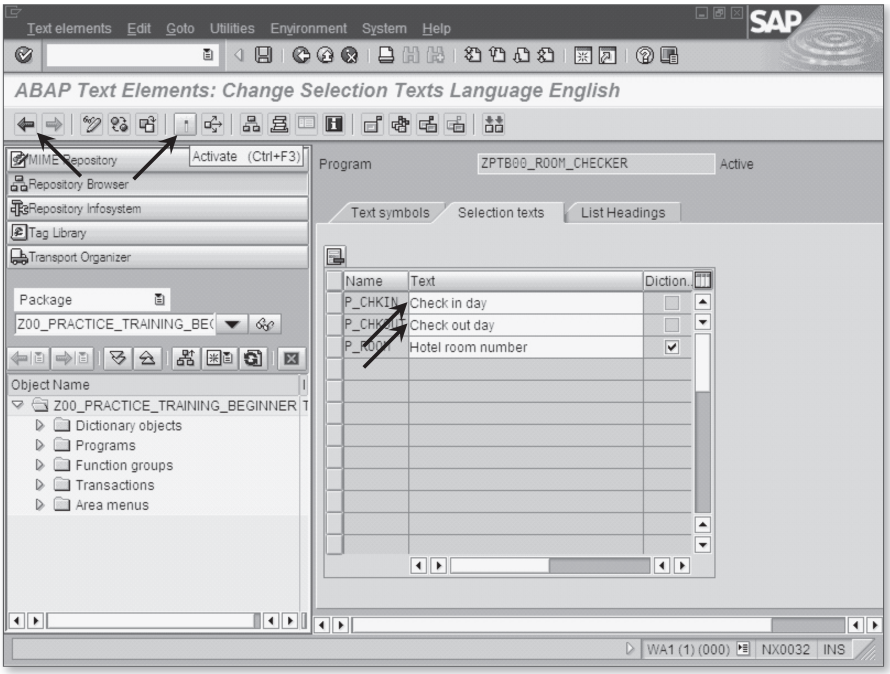
▷ From the menu, select GOTO • TEXT ELEMENTS • SELECTION TEXTS.

A dialog box opens, displaying fields in which all the parameters are already entered and only the texts need to be entered.

▷ For the parameter `p_chkin`, enter the text "Check in day", and for `p_chkout`, enter the text "Check out day".

▷ Select the DICTIONARY REFERENCE checkbox for the parameter `p_room`, which will automatically accept the heading already given in the data element, which in this example is "Hotel room number" (see Figure 4.9).

▷ Click the Activate button, and then click Back.



Name	Text	Diction..
P_CHKIN	Check in day	☐
P_CHKOUT	Check out day	☐
P_ROOM	Hotel room number	✓

Figure 4.9 Editing Selection Texts for the Program ZPTB00_ROOM_CHECKER

Testing the program

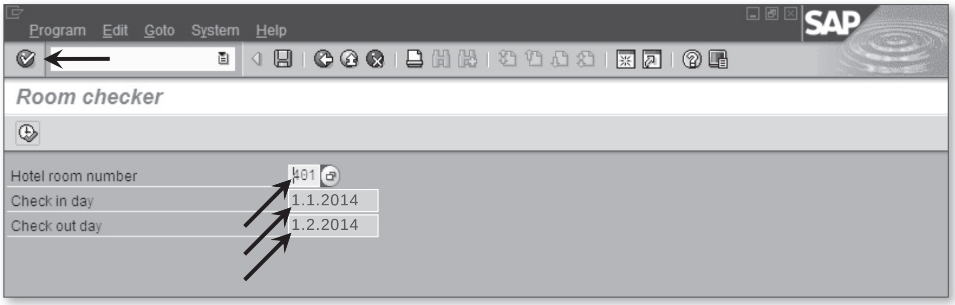
Now check the effectiveness of your settings in the running program.

▷ As usual, click the Check button, then Activate, and finally the Direct button to start the program immediately.

The program starts, prompting you for the room number and the check-in and check-out dates.

▷ Enter "401" as the room number, 1.1.2014 for the check-in date, and 1.2.2014 for the check-out date (see Figure 4.10).

▷ Confirm your input with Enter.



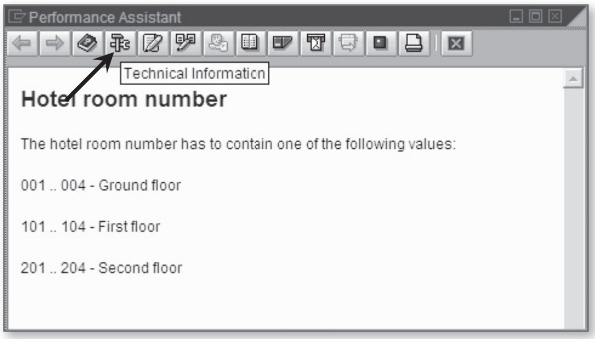
Room checker

Hotel room number 401
Check in day 1.1.2014
Check out day 1.2.2014

Figure 4.10 Entries in the Program ZPTB00_ROOM_CHECKER

The selection form determines the invalid room number, prints an error message in the status line, and locks all the input fields except for the hotel room number.

▷ Call the online help for this input field with the F1 key (see Figure 4.11).



Hotel room number

The hotel room number has to contain one of the following values:

001 .. 004 - Ground floor

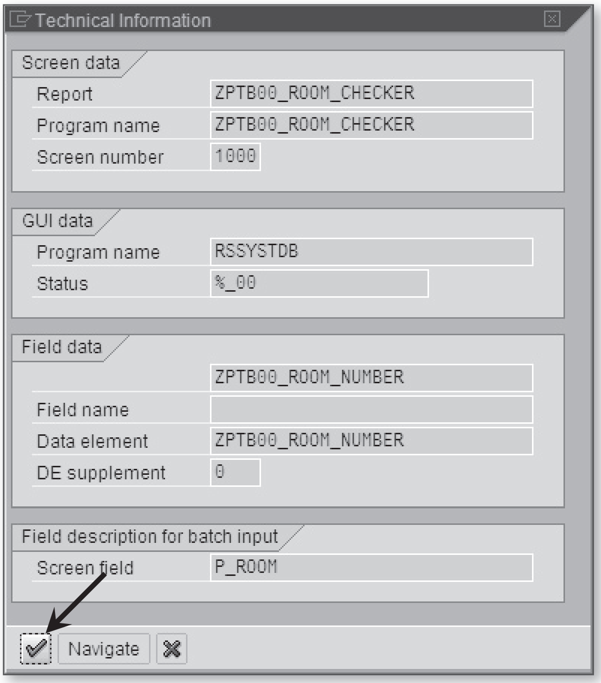
101 .. 104 - First floor

201 .. 204 - Second floor

Figure 4.11 Online Help for the Hotel Room Number Input Fields

The user can get information here about the correct input values for this input field. Particularly for developers, it can also be helpful for testing to call up information about the underlying data element.

▷ Click the Technical Information button (see Figure 4.12).


Technical Information

Screen data

Report	ZPTB00_ROOM_CHECKER
Program name	ZPTB00_ROOM_CHECKER
Screen number	1000

GUI data

Program name	RSSYSTDB
Status	%_00

Field data

	ZPTB00_ROOM_NUMBER
Field name	
Data element	ZPTB00_ROOM_NUMBER
DE supplement	0

Field description for batch input

Screen field	P_ROOM

Figure 4.12 Technical Information about the Input Field

From there, you can also navigate to the data element, for instance, to make corrections. However, you want to continue to test your program.

▷ Click the CLOSE button in the Technical Information dialog. This makes the original help window active again.

▷ Close the online help and press the F4 key or click the button to the right of the HOTEL ROOM NUMBER input field.

The value help appears on the screen and offers you the values entered in the domain for selection (see Figure 4.13).

▷ Select the hotel room number "201" and click the COPY button.

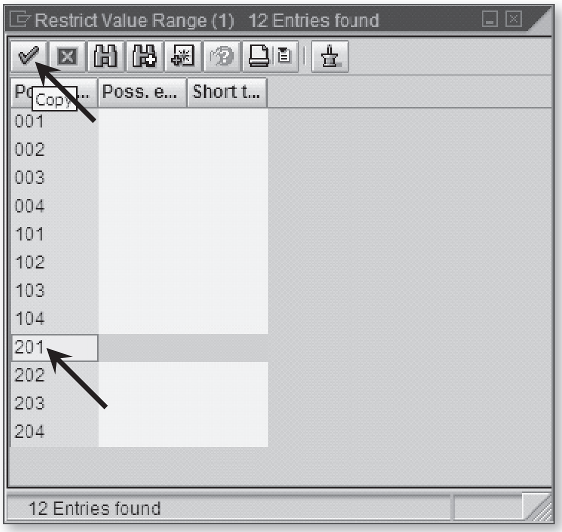
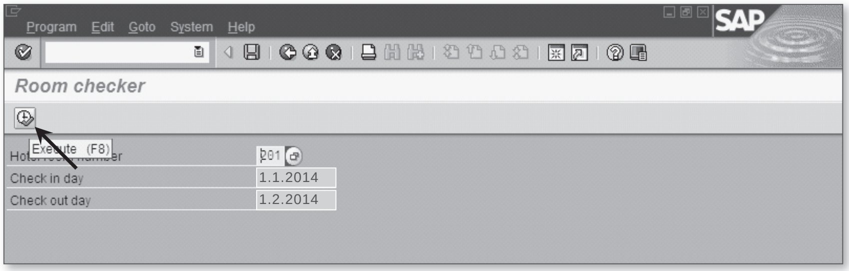


Figure 4.13 Input Help for the Hotel Room Number Field

The selection is copied into the input field.

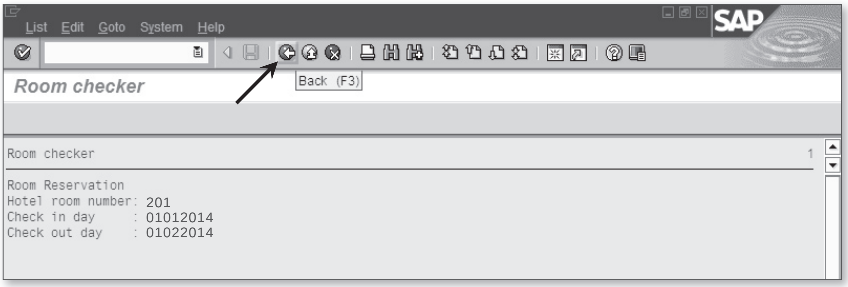
▷ Click the EXECUTE button (see Figure 4.14).



Hotel room number 201
Check in day 1.1.2014
Check out day 1.2.2014

Figure 4.14 Corrected Hotel Room Number

All entries now pass the test perfectly, and the test results are printed on the screen (see Figure 4.15).



```
Room checker                                   1

Room Reservation
Hotel room number: 201
Check in day     : 01012014
Check out day    : 01022014
```

Figure 4.15 Results of the Program ZPTB00_ROOM_CHECKER

Tip

In professional ABAP development, the importance of completely maintained data elements, domains, and text elements cannot be overstated. ABAP derives a large part of its runtime stability and flexibility from the metadata edited there. Up to 20 percent of project effort regularly goes toward the maintenance and documentation of data. Two of the most important domains used by all ABAP developers are `WAERS` and `MEINS`. The former is of type `CUKY` and uses a value table to define all the currency units used worldwide (for instance, EUR, USD, YEN, and so on). The latter is of type `QUAN` and uses a value table to define the most commonly used ISO units, like units and hours, and so on.

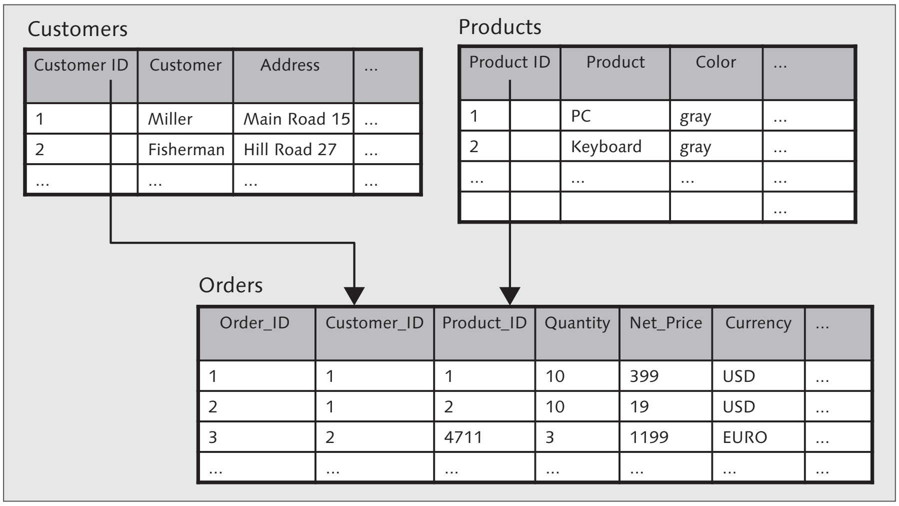
4.2 Defining and Processing Database Tables—SELECT, INSERT, UPDATE, DELETE

Relational databases manage information in tables that consist of rows (*records*) and columns (*fields*). Once stored, you can retrieve information quickly, selectively, and by different applications at the same time. They are especially designed for reading and writing access by many users, and far exceed the capabilities of files.

Basics

In the early 1980s, the triumphant success of the computer in the business word would have been impossible without relational databases; enterprises like Oracle owe their success to this very fact. While home computers and PCs stored most data as files, whose data formats were completely different and often not even published, most enterprises bought a database management system and installed it in a central location so they could work with it via the network from their individual workplaces. In the beginning, simple command line scripts and text-based user interfaces for the writing and reading out of data from perhaps a few dozen tables dominated; however, over the years, these gave way to increasingly more refined programs with Graphical User Interfaces (GUIs), which made the greater part of ever more complicated business logic and the consequent necessary multiplication of the tables transparent to the user. Today, a current installation of SAP ERP can reach 109,000 tables, required for the storage and data for over 40 business modules (such as Purchasing, Production, Sales, and so on), and that number doesn't even include the more than 20 new SAP (industry) solutions.

The greater part of the data in these tables is closely related to other parts of the data; that is, the data is *relational*. For instance, one table may store the addresses of customers, the next table the purchase orders for those customers, and another the information about the products ordered (see Figure 4.16).


Customers

Customer ID	Customer	Address	...
1	Miller	Main Road 15	...
2	Fisherman	Hill Road 27	...
...	...	...	...

Products

Product ID	Product	Color	...
1	PC	gray	...
2	Keyboard	gray	...
...	...	...	...
			...

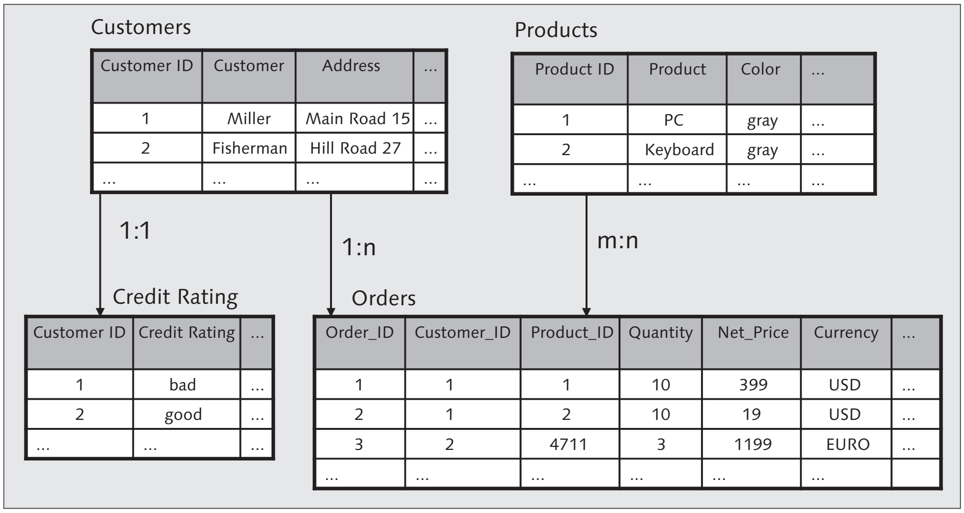
Orders

Order_ID	Customer_ID	Product_ID	Quantity	Net_Price	Currency	...
1	1	1	10	399	USD	...
2	1	2	10	19	USD	...
3	2	4711	3	1199	EURO	...
...	...	...	...	...	...	...

Figure 4.16 Tables for the Storage of Data and Relations

Relational database design

In custom-developed ABAP programs, which don't use the tables or even the data management functions of existing SAP applications, you still have to think through the same kind of relational network of tables that you'll be using in your program to store and read data. As shown in Figure 4.17, fields with identical content are used to compare data in different tables. The trick of this relational database design is nothing new for the newcomer to ABAP. Every row in a table has one field with a value unique in the table, the *key*. If you want to refer to the data in this row from another table, you only need to include a field in the other table in which this value can be stored, the *foreign key*. If you need the information referenced by the foreign key later on, you only have to look for the row with that value in the original table.



Customer ID	Customer	Address	...
1	Miller	Main Road 15	...
2	Fisherman	Hill Road 27	...
...	...	...	...

Product ID	Product	Color	...
1	PC	gray	...
2	Keyboard	gray	...
...	...	...	...

Customer ID	Credit Rating	...
1	bad	...
2	good	...
...	...	...

Order_ID	Customer_ID	Product_ID	Quantity	Net_Price	Currency	...
1	1	1	10	399	USD	...
2	1	2	10	19	USD	...
3	2	4711	3	1199	EURO	...
...	...	...	...	...	...	

Figure 4.17 Three Types of Relation: 1:n, 1:1, and m:n

This results in relational networks between tables that break down into three types:

- **1:n relation**
  One row in table A is referenced by 0, 1, or several (n) rows in another table B. Examples are the relation between customers and orders,

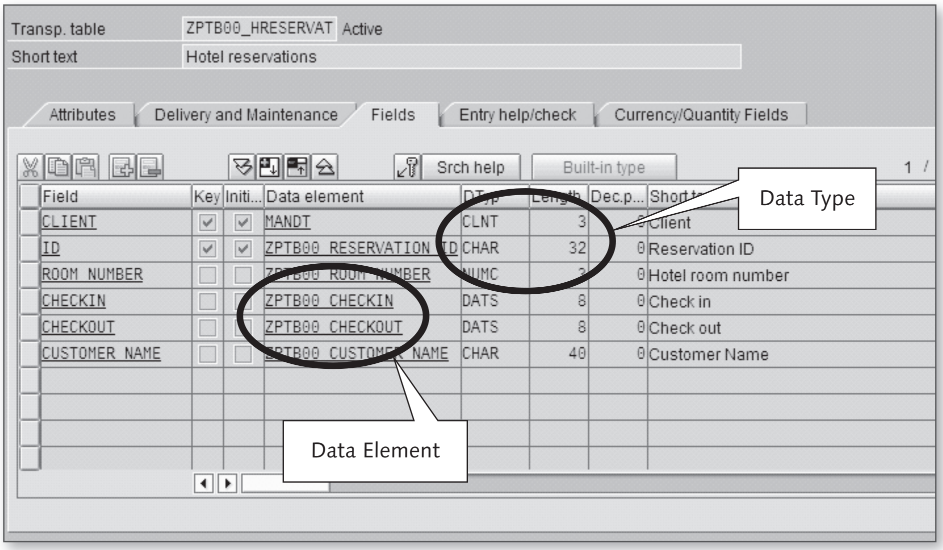
between a room in a school and the children in the class taught there, between a schoolchild and the child's books, and so on.

- **1:1 relation**
  One row in table A is referenced by exactly one row in another table B. Examples are the relation between the inhabitant of a country and his main place of residence, between the VIN number of a car and its license plate, between a corporation and its trade register entry, and so on.
- **m:n relation**
  One row in table A is referenced by 0, 1, or more rows of another table B, while one row in table B can also be referenced by 0, 1, or more rows in table A. To model this relation, you require a third table, C, which has a 1:n relation with both tables A and B, where the n side, that is, the foreign key, is stored in table C. So actually an m:n relation is split up into two 1:n relations in order to model it in a relational database.

These three basic types of relation allow all possible relationships between data in tables, regardless of how complicated the relationships are between the data in the tables. The art of database design consists of solving this task for the data at hand as elegantly as possible, without duplicated data storage (no redundancy) and suitably for fast read or write access.

Database tables and data elements

The individual columns (also called *fields*) of a database table are provided with as descriptive a name as possible, which must adhere to similar rules in ABAP as the names of variables (e.g., only a Roman letter at the start, followed by a sequence of letters, digits, and underscores, with a maximum length of 30 characters). The data type of each column is usually determined by providing a data element that controls all the technical and descriptive properties, except for its use as a key or foreign key. The latter information is added directly by the table (see Figure 4.18). Because the same data elements can also be used in the ABAP program, there is a basic compatibility when transferring values.



Transp. table ZPTB00_HRESERVAT Active
Short text Hotel reservations

Field	Key	Initi...	Data element	DTyp	Length	Dec.p...	Short text
CLIENT	✓	✓	MANDT	CLNT	3	0	Client
ID	✓	✓	ZPTB00_RESERVATION_ID	CHAR	32	0	Reservation ID
ROOM_NUMBER			ZPTB00_ROOM_NUMBER	NUMC	3	0	Hotel room number
CHECKIN			ZPTB00_CHECKIN	DATS	8	0	Check in
CHECKOUT			ZPTB00_CHECKOUT	DATS	8	0	Check out
CUSTOMER_NAME			ZPTB00_CUSTOMER_NAME	CHAR	40	0	Customer Name

Figure 4.18 Using Data Elements to Determine Technical and Descriptive Properties of Fields

Data transfer between memory and the database

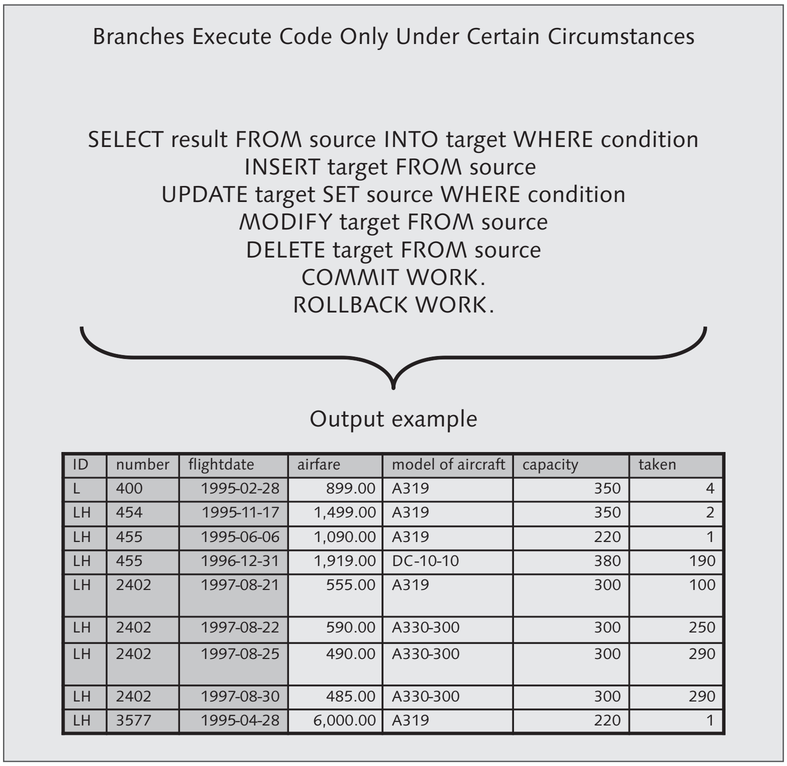
Within an ABAP program, you should prepare your data in the form of suitable internal tables and structures (see Section 3.3, Structures and Internal Tables—TABLE, MOVE-CORRESPONDING, INSERT, APPEND, READ, MODIFY, DELETE, LOOP) because only in this form can the data be written to the database and retrieved again. Ideally, you should use internal tables and structures for this purpose that correspond structurally to the database tables. In ABAP there is nothing easier because you can specify the database table as a data type. A short code fragment makes a good example:

```
DATA:
  l_tab_flight TYPE STANDARD TABLE OF sflight,
  l_str_flight TYPE sflight.
l_str_flight-planetype = 'A320-200'.
* ... more assignments ...
INSERT sflight FROM l_str_flight.
* ... or insert table instead of structure like this ...
l_str_flight-carrid = 'AA'.
APPEND l_str_flight TO l_tab_flight.
INSERT sflight FROM TABLE l_tab_flight.
```

In the first declaration instruction, the variable `l_tab_flight` is defined as an internal table of type `sflight`, so that it can accept multiple rows with the same row structure as the database table `sflight`. This allows you to write multiple records to the table at once, as you do in the two `INSERT` statements. In the second declaration instruction, the variable `l_str_flight` is defined as a structure of type `sflight`, so that it has the same row structure as the database table `sflight`. The fields in the structure only need to be assigned values, and in the next statement their content can already be inserted as a row into the database.

Open SQL

The following commands are the most important for the transfer of data between one or more database tables and data in memory, and are grouped together by SAP under the term *Open SQL* because they provide a common subset of the statements that work under all databases (see Figure 4.19).



Branches Execute Code Only Under Certain Circumstances

```
SELECT result FROM source INTO target WHERE condition
INSERT target FROM source
UPDATE target SET source WHERE condition
MODIFY target FROM source
DELETE target FROM source
COMMIT WORK.
ROLLBACK WORK.
```

Output example

ID	number	flightdate	airfare	model of aircraft	capacity	taken
L	400	1995-02-28	899.00	A319	350	4
LH	454	1995-11-17	1,499.00	A319	350	2
LH	455	1995-06-06	1,090.00	A319	220	1
LH	455	1996-12-31	1,919.00	DC-10-10	380	190
LH	2402	1997-08-21	555.00	A319	300	100
LH	2402	1997-08-22	590.00	A330-300	300	250
LH	2402	1997-08-25	490.00	A330-300	300	290
LH	2402	1997-08-30	485.00	A330-300	300	290
LH	3577	1995-04-28	6,000.00	A319	220	1

Figure 4.19 Overview of Open SQL Commands

- **SELECT—Reading data from database tables**
  The SELECT command reads data from a database table into a structure or internal table. Its basic syntax is

  ```
  SELECT result FROM source INTO target WHERE condition.
  ```

  After the WHERE clause, you can define logical expressions that the data to be read in must match, thus limiting the result set. There is also a SELECT ... ENDSELECT form, which works similarly to a LOOP instruction, reading and processing the lines from the database one by one.

  Over the years, the SELECT command has grown more and more powerful, and it now supports a whole series of clauses that can also be found in the SQL 92 standard, including nested selects, joins, and aggregation. If you would like to learn more about the functionality of the SELECT command, we recommend looking at *http://help.sap.com*. In the two following examples of SELECT commands, demo tables from SAP (flight data tables) are used, in the first case to read out all information (* wildcard) on Lufthansa flights, and in the second case, only to read the first line, and only the columns CARRID and CONNID from American Airlines:

  ```
  DATA:
    l_tab_sflight TYPE STANDARD TABLE OF sflight,
    l_str_sflight TYPE sflight.
  * read potentially more than one line into internal
  * table
  SELECT * FROM sflight INTO TABLE l_tab_sflight
  WHERE carrid = 'LH'.
  * read single line and only two fields into structure
  SELECT SINGLE carrid connid FROM sflight INTO
      l_str_sflight
  WHERE carrid = 'AA'.
  ```

- **INSERT—Inserting data into database tables**
  The INSERT command inserts data into a database table without affecting the existing data. Here, you must be absolutely sure that the key field of the data to be inserted contains a value not yet inserted into the table; otherwise, the command will stop with an error message. The INSERT command has the syntax

```
INSERT target FROM source.
```

and inserts into table `target` the data from the internal table or structure `source`. The two following examples show the insertion of values from a structure and an internal table:

```
DATA:
  l_tab_flight TYPE STANDARD TABLE OF sflight,
  l_str_flight TYPE sflight.
l_str_sflight-carrid = ‘LH’.
l_str_sflight-connid = ‘0400’.
l_str_sflight-fldate = ‘1.8.2003’.
l_str_sflight-planetype = ‘A320-200’.
APPEND l_str_sflight TO l_tab_sflight.
* ... more assignments here to fill internal table ...
* insert lines of internal table into database table
INSERT sflight FROM TABLE l_tab_sflight.
* insert structure into database table ...
l_str_sflight-carrid = ‘AA’.
INSERT sflight FROM l_str_sflight.
```

- **UPDATE—Changing data in database tables**
  The `UPDATE` command changes the content of one or more existing records in the database by overwriting it with data from individual variables, a structure, or an internal table. The syntax of the `UPDATE` command can correspondingly vary considerably, depending on whether data in the main memory (without considering the key field) should be written to multiple records in the database table or only applies to exactly one row in the database (taking the key field into consideration). In the first case, the syntax is

  ```
  UPDATE target SET source WHERE condition.
  ```

  and in the second case

  ```
  UPDATE target FROM source.
  ```

  The three following examples show the update of records between internal tables in memory and database tables, update between a structure in memory and a row in the database table, and the modification of all records from the carrier Lufthansa where the airplane should be set to “Airbus 320-200.”

```
 DATA:
  l_tab_flight TYPE STANDARD TABLE OF sflight,
  l_str_flight TYPE sflight.
* ... assignments to fill structure and internal table ...
* update identification by key ...
UPDATE sflight FROM TABLE l_tab_sflight.
UPDATE sflight FROM l_str_sflight.
* update identification by condition ...
UPDATE sflight SET planetype = l_str_sflight-planetype
    WHERE carrid = 'LH'.
```

- **MODIFY—Insert or change data in database tables**
  The `MODIFY` command is a combination of the `INSERT` and `UPDATE` commands. An attempt is first made to insert the rows from a structure or an internal table into the database. Any rows whose key fields refer to an already existing value in the table are instead modified using an `UPDATE` command. The data passed is thus always in the database in either case. The `MODIFY` command has the syntax

```
MODIFY target FROM source.
```

  Here, `target` is the name of the database table and `source`, as before, is a structure or an internal table. The following two examples show the use of the command in a single database row or multiple database rows:

```
DATA:
  l_tab_flight TYPE STANDARD TABLE OF sflight,
  l_str_flight TYPE sflight.
* ... assignments to fill structure and internal table ...
* modify one line in the database table
MODIFY sflight FROM l_str_sflight.
* modify more lines in the database table
MODIFY sflight FROM TABLE l_tab_sflight.
```

- **DELETE—Deleting data from database tables**
  The `DELETE` command is used to delete one or more rows from a database table. The syntax of the `DELETE` command differs depending on whether a structure or internal table with key values, or a logical expression, is used to identify the rows to be deleted. In the first case, the syntax is

```
DELETE target FROM source.
```

and in the second case

```
DELETE FROM target WHERE condition.
```

The next three examples show the use of the `DELETE` command using a structure, an internal table, and a logical expression:

```
DATA:
  l_tab_flight TYPE STANDARD TABLE OF sflight,
  l_str_flight TYPE sflight.
* fill keys into structure
l_str_sflight-carrid = ‘LH’.
l_str_sflight-connid = ‘0400’.
l_str_sflight-fldate = ‘1.8.2003’.
* delete line defined in structure
DELETE sflight FROM l_str_sflight.
 * delete lines defined in internal table
APPEND l_str_sflight TO l_tab_sflight.
DELETE sflight FROM TABLE l_tab_sflight.
* delete lines, where carrid is ‘LH’
DELETE FROM sflight WHERE carrid = ‘LH’.
```

For the sake of completeness, it should be mentioned at this point that besides the Open SQL commands described, SAP also has a few commands with explicit database cursor management (`OPEN CURSOR`, `FETCH`, `CLOSE CURSOR`) as well as *native SQL commands*, with which multiple databases can be used at the same time, and all the SQL commands can be used. These commands are then no longer database-independent, of course, which is why they are not further discussed in this book. SAP itself uses native SQL only in absolutely exceptional cases, for instance, when accessing customer data from other systems in SAP CRM.

Primary key and secondary key of database tables

Each database table defined in SAP must be equipped with a primary key by marking the columns required for unique access to a row (for instance, the unique postal code, street, and house number for addressing a letter). The other columns in a table (also called *attributes*) can then be found using this primary key.

Secondary keys can be defined for database tables in SAP, but they don't have to be. They define alternative sets of columns, which can be used to limit retrieval to a few rows or one row in the database (like a P.O. box), which can be used as an alternative when addressing a letter. And just like a P.O. box, which can be used by a single person or shared by several, secondary keys can reference one row uniquely, but they don't have to.

SAP LUW concept

Another important concept regarding database access, and one of the outstanding features of ABAP, is the SAP LUW concept.

Once an application needs to use more than one database table, there is the basic problem that the data managed in these tables may not be in a consistent state at any given time. For instance, when an amount in accounting is transferred between accounts, it is first debited from one and then credited to the other. Between the two posting steps, the state of the data is inconsistent because the amount to be posted is not in the data inventory: It is "on its way." If the computer were to crash exactly between these two steps, the data would be permanently inconsistent. To avoid this kind of permanent inconsistency, SAP supports the LUW concept (*Logical Unit of Work*), also called the "all-or-nothing" principle. Its goal is to perform logically related write accesses (in actuality, the Open SQL commands `INSERT`, `UPDATE`, `MODIFY`, and `DELETE`) to tables either completely and successfully, or not at all. Thus, write accesses are largely automatically bundled in ABAP source code and performed at a time determined by the programmer, namely when the `COMMIT WORK` command is called, or they are discarded when the `ROLLBACK WORK` command is called. If the user doesn't call these commands and there was no runtime error, the `COMMIT WORK` command is automatically called at the end of a program or dialog step (see Chapter 5, Screen Input and Output). The rest of the application examples are based on the framework of this automatic mechanism.

Exercise 4.2

Create the program ZPTB00_HOTEL_RESERVATION and use the parameter declarations from the program ZPTB00_RESERVATION_CHECKER. Remove the `OBLIGATORY` clause from all the parameters. In addition to the parameter `p_name` to request the customer's name, add three radio buttons that the user can use to decide whether to insert, delete, or display reservations.

Before inserting a reservation, check whether the room is still unused.

To identify a reservation to be deleted, take the input data from the parameters.

Always show all reservations on the screen.

Let's take a practical look at the definition of tables and access using Open SQL commands.

Creating the domain and data elements

First, the domains and data elements must be created; you need them to create the table. The exact procedure is described in Section 4.1, Field Properties—DATA ELEMENT, DOMAIN, so this description here is limited to the changes, based on which you can perform the steps needed yourself. In Table 4.2, for better orientation, the specifications for the data element and domain ZPTB00_ROOM_NUMBER from Section 4.1 are shown. Create all the other domains and data elements analogously.

Data Element or Domain	Dictionary Type	Short Label	Long Label
ZPTB00_Reservation_ID	`char 32`	ID	Reservation ID
(ZPTB00_Room_Number)	`numc 3`	Room No.	Hotel room number
ZPTB00_Checkin	`dats 8`	Check in	Check in
ZPTB00_Checkout	`dats 8`	Check out	Check out
ZPTB00_Customer_Name	`char 40`	Name	Customer name

Table 4.2 Data Elements and Domains from the Example

Creating the table

After you have created and activated all the domains and data elements named, you can start creating the table.

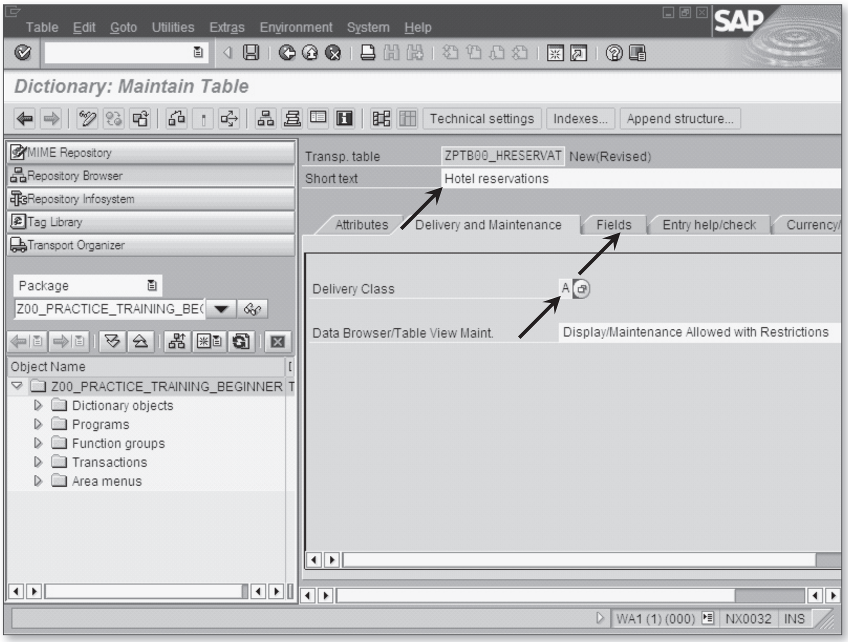
▷ In the context menu of the package or development class, select the CREATE • DDIC OBJECT • DATABASE TABLE menu option.

A dialog box subsequently opens, prompting you to enter the name of the database table that you want to create.

▷ Enter "ZPTB00_HRESERVAT" as the table name and confirm your entry with the OK button.

In the tool area, a window is displayed where you can maintain all properties of the table.

- ▷ Enter "Hotel reservations" as the brief description, and "A" as the delivery class (see Figure 4.20).
- ▷ Switch to the FIELDS tab.



Transp. table ZPTB00_HRESERVAT New(Revised)
Short text Hotel reservations
Delivery Class A
Data Browser/Table View Maint. Display/Maintenance Allowed with Restrictions

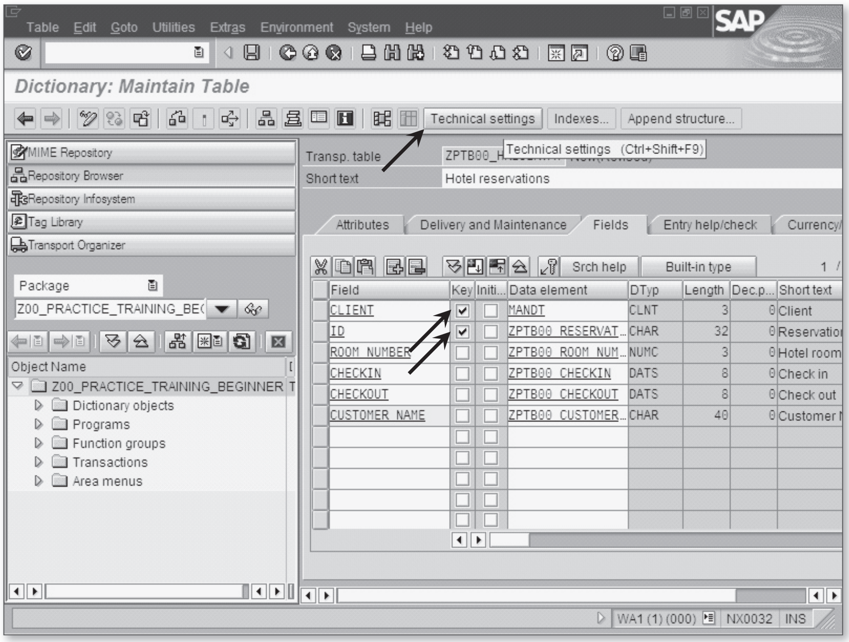
Figure 4.20 Entering Table Properties for ZPTB00_HReservat

Specify the names of all fields here that are supposed to appear as columns in the table.

- ▷ Enter the fields CLIENT, ID, ROOM_NUMBER, CHECKIN, CHECKOUT, and CUSTOMER_NAME, along with the corresponding data elements. For the CLIENT field, select the predefined data element `MANDT`.
- ▷ Check the KEY checkbox for the fields CLIENT and ID, and click the TECHNICAL SETTINGS button. You have now defined the table's primary key (see Figure 4.21).

A dialog box opens, prompting you to confirm whether you want to save the table.

- ▷ Click YES to confirm the prompt.



Transp. table: ZPTB00_H

Short text: Hotel reservations

Field	Key	Initi...	Data element	DTyp	Length	Dec.p...	Short text
CLIENT	✓		MANDT	CLNT	3	0	Client
ID	✓		ZPTB00_RESERVAT...	CHAR	32	0	Reservatio
ROOM_NUMBER			ZPTB00_ROOM_NUM...	NUMC	3	0	Hotel room
CHECKIN			ZPTB00_CHECKIN	DATS	8	0	Check in
CHECKOUT			ZPTB00_CHECKOUT	DATS	8	0	Check out
CUSTOMER_NAME			ZPTB00_CUSTOMER...	CHAR	40	0	Customer I

Figure 4.21 Entering Field Properties for ZPTB00_HReservat

You're now asked for the package to which you want to assign the database table.

▷ The package is already preset, so you can simply confirm your entries by clicking the SAVE button.

Next, you're asked for the transport request for the table.

▷ Accept the existing transport request without changing it, and confirm with the YES button.

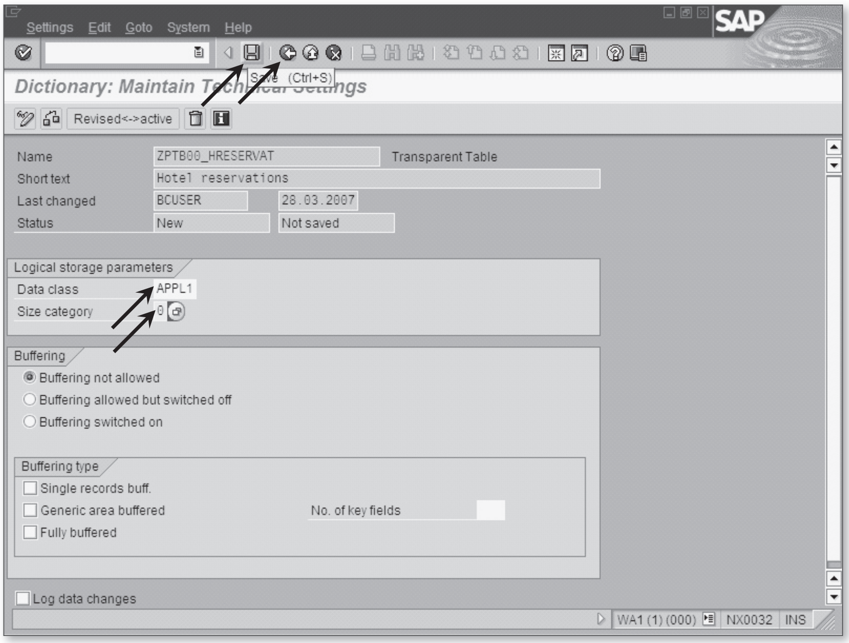
The usual transport dialog follows.

▷ Accept the existing transport request without changing it and confirm with the CONTINUE button.

The technical settings must be specified before you activate the table and contain information about the data class and size category in particular.

▷ Enter "APPL1" as the data class because this is data that is changed frequently.

▷ Select "0" as the size category (this corresponds to 0 – 3,200 records) because you can hardly expect more than a few hundred reservations at such a small hotel (see Figure 4.22).


Settings Edit Goto System Help
Save (Ctrl+S)
Dictionary: Maintain Technical Settings
Revised<->active

Name ZPTB00_HRESERVAT Transparent Table
Short text Hotel reservations
Last changed BCUSER 28.03.2007
Status New Not saved

Logical storage parameters
Data class APPL1
Size category 0

Buffering
(●) Buffering not allowed
( ) Buffering allowed but switched off
( ) Buffering switched on

Buffering type
☐ Single records buff.
☐ Generic area buffered No. of key fields
☐ Fully buffered

☐ Log data changes

WA1 (1) (000) NX0032 INS

Figure 4.22 Editing the Technical Properties for Table ZPTB00_HReservat

▷ Click the SAVE button and then on BACK.

All the necessary properties for table ZPTB00_HRESERVAT have now been set up, and you can activate it.

▷ Click the BACK button, and then click ACTIVATE.

Creating a program

After these preparations, you can begin writing the program.

▷ Create a program called "ZPTB00_HOTEL_RESERVATION" in the usual way, without a TOP include. The title should be "Hotel reservation" and the rest of the program properties can stay unchanged. The transport request is also accepted as preset.

The SAP Object Navigator creates a skeleton program, which you can now fill out in order to take the inputs HOTEL ROOM NUMBER, CHECK-IN, CHECK-OUT, and CUSTOMER NAME, together with the actions ADD, DELETE, and SHOW, to build a little database application.

▷ Type the following code under the comment lines, and save it.

```
REPORT  zptb00_hotel_reservation.
PARAMETERS:
* Reservation data
  p_room TYPE zptb00_room_number VALUE CHECK,
  p_chkin TYPE zptb00_checkin,
  p_chkout TYPE zptb00_checkout,
  p_name TYPE zptb00_customer_name,
* Application menu
  p_add TYPE c RADIOBUTTON GROUP grp1 DEFAULT ‘X’,
  p_delete TYPE c RADIOBUTTON GROUP grp1,
  p_show TYPE c RADIOBUTTON GROUP grp1.
DATA:
* For working with table zptb00_hreservat
  l_str_reservation TYPE zptb00_hreservat,
  l_tab_reservation TYPE STANDARD TABLE OF
      zptb00_hreservat.

IF p_add = ‘X’.
* Check whether period is free
  SELECT * FROM zptb00_hreservat INTO TABLE
      l_tab_reservation
  WHERE ( room_number = p_room )
    AND ( ( checkin BETWEEN p_chkin and p_chkout )
    OR ( checkout BETWEEN p_chkin AND p_chkout ) ).
  IF sy-dbcnt > 0.
    WRITE: / ‘Period already reserved’.
  ELSE.
* Make reservation
    CALL FUNCTION ‘GUID_CREATE’
      IMPORTING
*       EV_GUID_16       =
*       EV_GUID_22       =
        ev_guid_32       = l_str_reservation-id.
    l_str_reservation-room_number = p_room.
    l_str_reservation-checkin = p_chkin.
```

```
      l_str_reservation-checkout = p_chkout.
      l_str_reservation-customer_name = p_name.
      INSERT zptb00_hreservat FROM l_str_reservation.
      WRITE: / 'Reservation made'.
    ENDIF.
  ELSEIF p_delete = 'X'.
  * Delete reservation
    DELETE FROM zptb00_hreservat WHERE room_number =
        p_room
    AND checkin = p_chkin AND checkout = p_chkout.
    WRITE: / 'Reservation deleted'.
  ELSEIF p_show = 'X'.
  * Show reservations
    SELECT * FROM zptb00_hreservat INTO TABLE
        l_tab_reservation
    ORDER BY room_number checkin customer_name.
    WRITE: / 'Room Reservations'.
    LOOP AT l_tab_reservation INTO l_str_reservation.
      WRITE: / 'Room:', l_str_reservation-room_number,
               'Check in:', l_str_reservation-checkin,
               'Check out:', l_str_reservation-checkout,
               'Customer name:', l_str_reservation-
                                 customer_name.
    ENDLOOP.
  ENDIF.
```

Listing 4.2 Source Code for the Program ZPTB00_Hotel_Reservation

Explanation of the source code

The parameters read the information needed to insert or delete records and are only used for the display of reservations. You define the three parameters for the radio buttons in a shared group `grp1`, so that only one radio button can be selected at any time. The two variables with the structure and the internal table for ZPTB00_HRESERVAT are needed to insert using the `INSERT` command or to read the data using the `SELECT` command.

Depending on the content of the radio buttons, use an `IF` statement to branch to different code segments. First, the insertion of a reservation is processed. For this, as required in the exercise, any possible overlap with other reservations must be ruled out, for which we use a corresponding `SELECT` statement. Only if no record with overlapping check-in/

check-out dates is found (`sy-dbcnt` is 0) does the insertion of the new reservation take place.

Deletion of a reservation, as a "sanity check," requires at least the specification of the room number and the check-in and check-out dates. Only then can the database be uniquely determined and deleted using the `DELETE` statement.

To display all reservations, they must first be read from the database into an internal table using the `SELECT` command. We use a `LOOP` to copy each record into the `l_str_hreservat` structure, where a `WRITE` command is used to write the content to the screen.

Testing the program

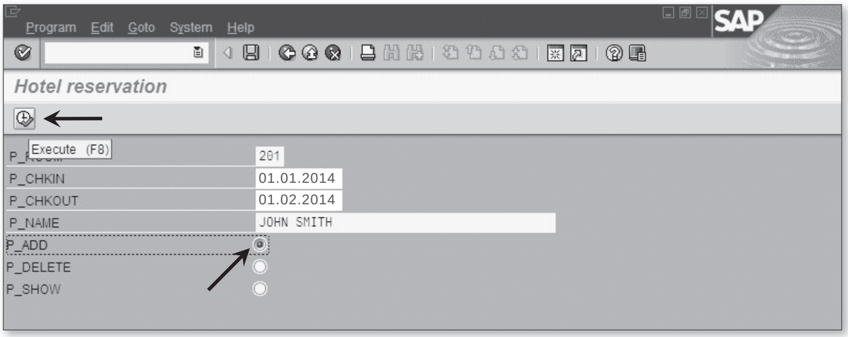
Now you need to test the functional efficiency of your source code in the running program.

▷ Click the Check button, then Activate, and finally the Direct button to start the program immediately.

The program starts and asks for the action to be performed. First, you are supposed to insert a record.

▷ Enter "201" as the room number, 1.01.2014 for the check-in date, 1.02.2014 for the check-out date, and "John Smith" as the customer name.

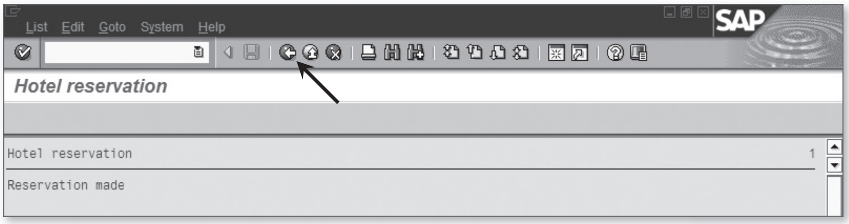
▷ Select the `p_add` parameter and click Execute.



P_ROOM	201
P_CHKIN	01.01.2014
P_CHKOUT	01.02.2014
P_NAME	JOHN SMITH
P_ADD	●
P_DELETE	○
P_SHOW	○

Figure 4.23 Performing a Reservation

The result screen appears, where the reservation is confirmed (see Figure 4.24).



```
Hotel reservation                                              1
Reservation made
```

Figure 4.24 Confirming the Reservation

▷ Click the Back button.

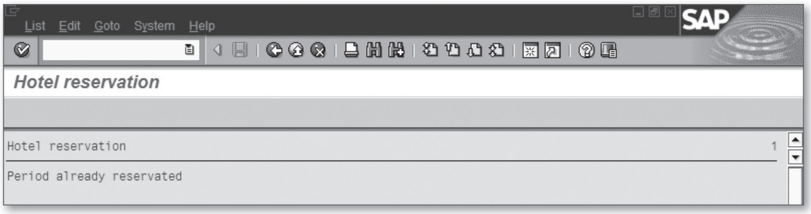
Now check whether duplicates can be detected.

▷ Enter "201" as the room number, 1.31.2014 for the check-in date, 2.11.2014 for the check-out date, and "Roman Herzog" as the customer name.

▷ Select the `p_add` parameter and click Execute.

The program determines the overlap and prints a corresponding message to the screen (see Figure 4.25).

▷ Click the Back button.



```
Hotel reservation                                              1
Period already reservated
```

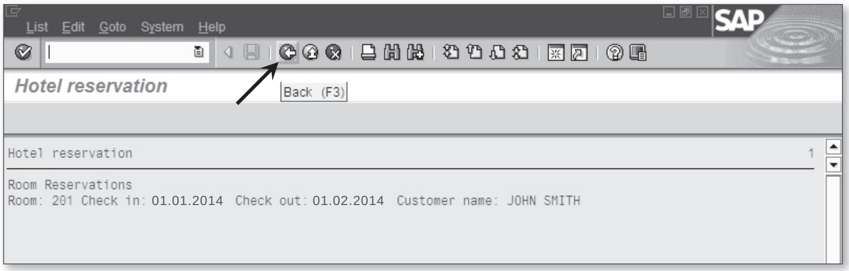
Figure 4.25 Rejection Notice for the Reservation

To find out more about the overlap, you can have the system display the reservations.

▷ Click the parameter `p_show`.

The program shows the reservations on the screen (see Figure 4.26).

▷ Click the Back button.



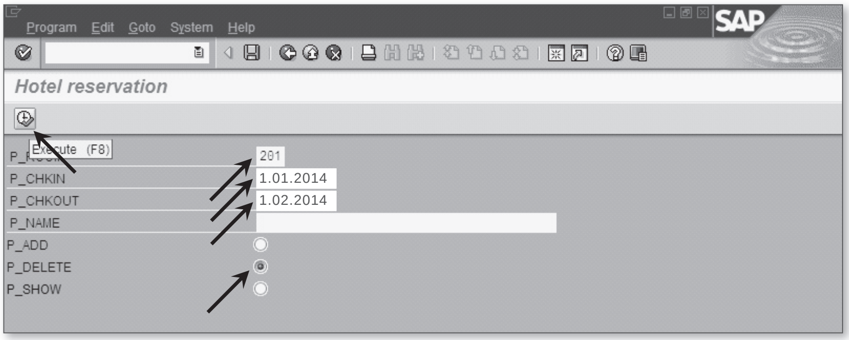
```
Hotel reservation                                                                1

Room Reservations
Room: 201 Check in: 01.01.2014  Check out: 01.02.2014  Customer name: JOHN SMITH
```

Figure 4.26 All Reservations Are Listed

You now know that you first need to delete this reservation.

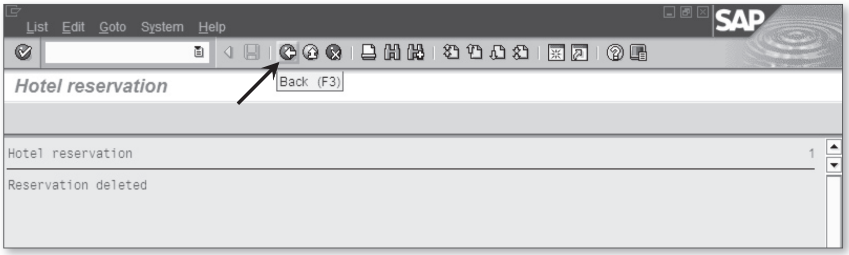
- ▷ Enter “201” as the room number, 1.01.2014 for the check-in date, and 1.02.2014 for the check-out date (see Figure 4.27).
- ▷ Select the `p_delete` parameter and click EXECUTE.



```
P_ROOM      201
P_CHKIN     1.01.2014
P_CHKOUT    1.02.2014
P_NAME
P_ADD       ○
P_DELETE    ●
P_SHOW      ○
```

Figure 4.27 Deleting the Reservation

You receive a confirmation of the deletion in the form of a message on the screen (see Figure 4.28).



```
Hotel reservation                                                                1

Reservation deleted
```

Figure 4.28 Deletion Confirmation for the Reservation

This example program could be tested much more thoroughly. However, this testing is not within the scope of this book.

Tip

In professional ABAP development, not only tables but all possible types are created and managed globally in the ABAP Dictionary. The procedure is nearly identical to the creation of tables. This form is especially popular when used for very generic applications—applications that may determine only which data they must process and store during customization for a particular customer—because it means that the customer can have a direct influence on content

Index

1:1 relation, 225
3GL language, 117
4GL language, 117

A

AB, 17
ABAP, 20, 119, 491
  debugger, 87
  Dictionary, 491
  Editor, 52, 85
  Objects, 13, 24, 344
  runtime environment, 24, 491
  Workbench, 27, 60, 62, 491
ABAP/4, 22
Adapter, 491
ALV, 251
Ancestor, 378
ANY TABLE, 156
API (Application Programming Interface), 491
  layer, 428
APPEND, 157
Application, 491
application base, 17
Application menu, 95
Application toolbar, 279
Area menu
  create, 95
Array, 117
AS CHECKBOX, 130
Assignment, 121
Attribute, 348
Authentication, 36
Avoiding redundancy, 164

B

Back end, 25, 27
Backward navigation, 51
BAdI, 32
BAPI (Business Application Programming Interface), 31, 491
BEGIN OF, 152
BEGIN OF BLOCK, 246
BEGIN OF SCREEN, 246
Branching instruction, 184
Browser area, 60
Build, decentralized, 57
Business module, 21
Business process, 491
Business scenario, 491
Business Server Page, 60, 491

C

Call by Reference, 169
Call by Value, 170
CALL METHOD, 354
CALL SCREEN, 283
CALL SELECTION-SCREEN, 249
CALL SUBSCREEN, 283
CASE ... WHEN, 192
Casting, 130
CHAIN ... ENDCHAIN, 282
CHANGING, 168
Changing parameter, 167
Character, 131
Checklist for Programmers, 27
Checkman check, 490
Child class, 378
Class, 346, 347
  attribute, 355
  create, 366
  definition, 348
  hierarchy, 377
  implementation, 351
  method, 355
  redefine method, 390
CLASS_CONSTRUCTOR, 354
CLASS-DATA, 355
CLASS-EVENTS, 399
CLASS-METHODS, 355

Client, 21, 29, 37, 492
CLOSE CURSOR, 231
COLLECT, 157
Command line, 41
Comment, 30
Comment language, 30
Comparison operator
  IS SUPPLIED, 190
  BETWEEN, 186
  CA, 187
  CN, 187
  CO, 187
  CP, 188
  CS, 188
  EQ, 185
  GE, 186
  GT, 185
  IS ASSIGNED, 189
  IS BOUND, 189
  IS INITIAL, 190
  IS REQUESTED, 190
  IS SUPPLIED, 190
  LE, 185
  NA, 187
  NE, 186
  NP, 189
  NS, 188
Component, 492
Constant, 129
Constraint, 210
CONSTRUCTOR, 354
  static, 354
CONTINUE, 199
Control example, 60
CONTROLS , 287
Counting loop, 199
CREATE DATA, 128, 136
CREATE OBJECT, 347, 352

D

Data, 127, 348
Database design , 205
Database table, 225
  fields, 225
Data element, 207
  create, 213, 233, 294
  define, 210
  edit documentation, 216
  tables, 225
Data instances, 347
Data transfer, 227
Data type, 131
  any, 149
  Boolean, 135
  c, 131
  d, 134
  f, 133
  i, 132
  n, 132
  p, 133
  string, 135
  t, 134
  x, 133
  xstring, 135
DB layer, 428
Debugging, 56, 86
Declaration, 127
DEFINITION, 349
DELETE, 155, 159, 230
Derived class , 378
Descendent, 378
DESCRIBE FIELD, 356
Descriptive properties, 210
DESTINATION , 166
Developer key, 25, 27
Development class, 68
  create, 75
Development language, 30
Development object, 55
Development system, 29
Dictionary data type, 207, 484
  built-in, 209, 210
  select, 209
Directory, 492
DNS address, 35
DO ... ENDDO, 198
Domain, 210
  create, 292
Drag-and-Drop, 32
DUPLICATES, 159

Dynamic data, 128, 136
Dynpro, 278, 492
  flow logic, 119, 330

E

ENDFORM , 164
ENDING AT, 249
END OF, 152, 246
END OF BLOCK, 246
Engine, 492
Enhancement ID, 100
EnjoySAP, 251
ERP (Enterprise Resource Planning), 21
Event, 396, 397
  declare, 399
  trigger, 401
Exception, 167, 168
EXIT, 199
EXPORTING, 168, 169, 350

F

F1 help, 61
F4 help, 65
FETCH, 231
FIELD, 281
FIELD-SYMBOLS, 130
Flat structure, 153
Floating point number, 133
Font convention, 16
Foreign key, 224
FORM, 164
Forward navigation, 51
Framework, 377
Front end, 25, 27
  integration, 33
Function, 163, 165, 348
Function group, 165
  create, 305
Function module, 165
  create, 306

G

Garbage collector, 136, 352
General report-formatting processor, 20
GET, 120
GET CURSOR, 121, 287
GET PF-STATUS, 285
GET PROPERTY OF, 120
GET REFERENCE OF, 121, 128
GET TIME, 120
Global class
  create, 389
  declare an event, 411
  define a method, 368
  define attributes, 370
  define event-handler methods, 412
  defining attributes, 390
Global table types, 365
Glue-Logic, 396
GUID (Globally Unique Identifier), 17
GUI (Graphical User Interface), 492
GUI status, 254, 256, 278
  create, 310
GUI title
  create, 309

H

Handler class, 399
Handler method, 398
  declare, 399
  register, 401
Hardware, 26
Hash algorithm, 156
HASHED TABLE, 156
help.sap.com, 105
Hexadecimal, 133
Hierarchical database, 20

I

IF ... ELSE, 192
IMPLEMENTATION, 351
IMPORTING, 168, 169, 350

IN, 248
INCLUDE, 79, 165
INDEX TABLE, 156
Inheritance, 378
INHERITING FROM, 378
Input elements, 129
Input interface, 246
Input parameter, 167
INSERT, 155, 157, 228
Instance constructor, 354
Integer, 132
Interface, 31, 166
Interface element
  create, 312
Interface parameter, 164, 166
Internal table, 151
Internal table types
  generic, 156
Internet standard, 492
Internet technology, 18

J

Java, 18
JEE (Java Platform, Enterprise Edition), 493
Just-In-Time Compiler, 52

K

Key, 224
Knowledge warehouse, 109

L

Layer model, 427
LEAVE SCREEN, 289
Library, 377
License renewal, 27
List, 78
Listener concept, 397
List header, 138
List of functions, 424
Local Class, 356
Local language, 211
Logical expression, 184
Logon, 32
Logon transaction, 36
Loop, 158, 198
LOOP AT SCREEN, 285
LOOP ... ENDLOOP, 282

M

m:n relation, 225
Main memory, 127
Main package , 69
MAPI layer, 427
Mapping, 209
Massachusetts Institute of Technology (MIT), 343
Mass data, 151
Master data check, 210
MDI window, 32
Menu bar, 278
Menu system, 40
Message, 249
  create, 260
  define, 307
  output, 249
Message class, 250
  create, 260, 307
Message server, 36
Metadata, 136, 211
Method, 345, 347, 348, 355, 399
  define transfer parameters, 369
Method pointer concept, 397
Method redefinition, 379
Microsoft, 19
Microsoft.NET, 493
MODIFY, 155, 158, 230
MODIFY SCREEN, 286
Modularization, 164
MODULE, 281
Module pool, 78
MOVE, 121
MOVE-CORRESPONDING, 153

N

n:1 relation, 224
Name conflict, 72
Namespace, 31, 72
  customer, 481
Name
  technical, 44
Naming convention
  constants, 483
  Dictionary data types, 484
  error messages, 487
  function groups, 485
  function modules, 486
  general, 480
  message classes, 487
  messages, 487
  modules, 488
  program-local data types, 484
  screens, 488
  subprograms, 487
  variables and parameters, 481
  warnings, 488
Native SQL, 231, 493
Nested structure, 152
NON-UNIQUE KEY, 156
Notation rule, 119
Numeric, 132

O

Object, 343, 346, 347
  create, 352
  dereferencing, 353
  instance, 347
  list area, 60
  orientation, 343
  referencing, 352
  type, 347
OBJ layer, 428
OBLIGATORY, 130
Offset, 121
Online help, 105, 106
OPEN CURSOR, 231
Open SQL, 227, 493
Operator, 185
  AND, 191
  NOT, 191
  OR, 191
Oracle, 19
Original class, 378
Original language, 30
Output parameter, 167

P

Package, 68
Packed, 133
PARAMETERS, 86, 120, 129, 246, 248
Parent class, 378
passed data, 166
passed parameter, 166
passed variable, 166
Password, 37
pattern, 180
PERFORM, 165
Performance example, 61
Polymorphism, 377, 379
Pretty Printer, 85
PRIVATE SECTION, 349
Procedure, 163
PROCESS, 280
  AFTER INPUT, 280
  BEFORE OUTPUT, 280
  ON HELP-REQUEST, 280
  ON VALUE-REQUEST, 280
Production system, 29
Program, 78, 80
  copy, 145
  create, 82
  enter transport request, 83
Program check
  extended, 489
  tools, 489
Programming
  guideline, 31, 479
  object-oriented, 344
  procedural, 163, 344
Programming model, declarative, 117
Program start, 48
Property, 350

PROTECTED SECTION, 348
PUBLIC SECTION, 348
Publish and subscribe, 397

R

R/1, 19
R/2, 20
R/3, 21
RAD tools, 396
RAISE EVENT, 401
RAISING, 165, 168
RAM, 127
Rapid application development, 396
READ, 155, 158
Record, 151, 222
REDEFINITION, 379
   class constructors, 381
   instance constructors, 380
   methods, 379
REDEFINITION, 355
Reference variable, 128, 136
REFRESH CONTROL, 288
REF TO, 130
   OBJECT, 352
Relational database design, 224
Relational type, 224
Release, 493
Report, 78, 79, 89
Repository, 493
Repository info system, 62
RETURNING, 350
Return value, 121
REUSE_ALV_GRID_DISPLAY, 252
REUSE_ALV_GRID_LAYOUT_INFO_rGET, 253
REUSE_ALV_GRID_LAYOUT_INFO_rSET, 254
RFC (Remote Function Call), 493
RTTI, 357

S

SAP, 13
SAP access
   setup, 24
SAP Basis, 20
SAP Basis system, 22
SAP Business Suite, 23
SAP Customer Relationship Management, 13
SAP Easy Access, 39, 494
SAP ERP, 13
SAP GUI, 36, 494
SAP Help Portal, 105, 108
SAP Library, 110
SAP List Viewer, 243, 251
SAP Menu Painter, 278
SAP NetWeaver, 22
SAP NetWeaver Application Server, 17, 22
SAP NetWeaver Process Integration, 426
SAP Notes, 106, 113
SAP Object Navigator, 55
   browser area, 60
   getting started, 59
   object list area, 60
   tool area, 60
SAP screen painter, 25, 278
SAP Service Marketplace, 106, 112
SAP Supply Chain Management, 13
SAP system, 494
   back end, 25
   front end, 25
   hardware, 25
   Linux, 26
SAP Web Application Server, 17, 24
Save, 41
Screen, 279
   components, 278
   create, 312
   flow logic, 278, 280, 326
SELECT, 228
Selection help, 65
Selection list, 425
SELECTION-SCREEN, 246
SELECTION-SCREEN COMMENT, 246
SELECTION-SCREEN PUSHBUTTON, 246
SELECTION-SCREEN ULINE, 246
Selection text, 138
   edit, 217
SELECT-OPTIONS, 247

Server cluster, 35
Session, 42
  new, 49
SET, 121
  CURSOR, 286
  EXTENDED CHECK ON/OFF, 121
  HANDLER, 401
  HOLD DATA, 289
  PARAMETER, 121
  PF-STATUS, 284
  SCREEN, 284
  TITLEBAR, 285
Signature, 166
Single server, 35
SORTED TABLE, 155
Specification, 422
SSO (Single Sign-on), 38, 494
Stack list, 88
Standard software, business, 13
STANDARD TABLE, 155
Standard toolbar, 40, 278
STARTING AT, 249
Static attribute, 354
Static method, 354
Static variable, 128
Status bar, 280
Status line, 42
Stop transaction, 40
Structure, 151, 152
  create, 303
  deep, 153
Structure package, 69
Subprogram, 163
subprogram return code, 122
SUPPRESS DIALOG, 289
SY, 121
  sy-dbcnt, 122
  sy-host, 122
  sy-index, 122, 199, 200
  sy-subrc, 121, 168
  sy-tabix, 122
  sy-uname, 122
System
  field, 121
  information, 42
  requirement, 17
  setup, 32
  status, 46

T

Table
  create, 233, 297
  internal, 154
Table control, 282
Table control wizard, 318
Table key, 156
TABLE OF, 155
TABLES, 168
Table type
  create, 301
  global, 242
  internal, 155
Table view , 282
Technical property, 210
Test system, 29
Text element, 137
Text symbol, 138
  create, 306
TIME ZONE, 126
Title bar, 279
Tool area, 60
Top-down approach, 427
Transaction, 44, 64, 92, 93
  SE80, 490
Transformation layer, 426
TRANSPORTING, 158
Transport management system, 67
Transport Organizer, 67
Transport request, 76
Transport system, 58
TYPE, 128
Type casting, 358
Type checking, 211
TYPE REF TO, 128
TYPES, 152
Type safety, 211

U

Unicode, 17
UNIQUE KEY, 156

UPDATE, 229
Usage explanation, 207
User name, 37
USING, 168

V

VALUE, 128, 350
Value help, 65
Variable, 127
Visibility, 348

W

Web Dynpro, 34, 119, 244, 494
Web service, 494
WHERE, 228
WHILE ... ENDWHILE, 199
Workplace computer, 21
WRITE, 83, 120

Z

ZPTB00_BUSINESS_TRANSACTION, 290
ZPTB00_HELLO_WORLD, 81
ZPTB00_HOTEL_RESERVATION, 232
ZPTB00_HOTEL_RESERVATION_COOL, 254
ZPTB00_INTERNAL_TABLE_JUGGLER, 159
ZPTB00_PERFORMANCE_TESTER, 171
ZPTB00_PRODUCT_CONFIGURATOR_GLOBAL, 365
ZPTB00_PRODUCT_ENHANCER_GL, 389
ZPTB00_PRODUCT_EVENT_GL, 411
ZPTB00_PROVISION_CALCULATOR, 193
ZPTB00_ROOM_CHECKER, 212
ZPTB00_SALES_ORDER, 138
ZPTB00_SALES_ORDER_DYNAMIC, 144
ZPTB00_SAVINGS_CALCULATOR, 200
ZPTB00_SYSTEM_INFORMATION, 123