



Dr.SNS RAJALAKSHMI COLLEGE OF ARTS AND SCIENCE
(Autonomous)

Coimbatore -641049

Accredited by NAAC (Cycle- III) with 'A+' Grade
(Recognised by UGC, Approved by AICTE, New Delhi and
Affiliated to Bharathiar University, Coimbatore)



UNIT – III

Dr.A.DEVI

Associate Professor

Department of Computer Applications

DRSNSRCAS

Debugging Programs

This chapter will introduce the ABAP debugger, and will introduce some of the tools which can be used to ensure that the programs you create function as intended. It will also show ways to highlight logic bugs in programs that cannot be identified by the syntax checker.

The first step here is to load a program which has been used previously, and which accesses the database table which has been created regarding employee records. If you have been following along with instructions, load program “Z_Employee_List_01” into the ABAP Editor.

The program contains a number of SELECT loops, which in turn write the contents of the table being read to the output screen in several ways, separated by ULINE statements:

```
REPORT z_employee_list_01 LINE-SIZE 132 .

TABLES zemployees.

*****
SELECT * FROM zemployees.           " Basic Select Loop
  WRITE zemployees.
ENDSELECT.

ULINE.

SELECT * FROM zemployees.           " Basic Select Loop with a LINE-BREAK
  WRITE / zemployees.
ENDSELECT.

ULINE.

SELECT * FROM zemployees.           " Basic Select Loop with a LINE-BREAK
  WRITE zemployees.                 " aftervthe first row is output.
  WRITE /.
ENDSELECT.

ULINE.

SKIP 2.
SELECT * FROM zemployees.           " Basic Select Loop with a SKIP statement
  WRITE / zemployees.
ENDSELECT.
```


Having examined the code, return to the front screen of the ABAP editor.

Firstly, on this screen you will notice there is a 'Debugging' button in the toolbar (also accessible with SHIFT+F5):



Click this with the program name in the program input text box to start a new debugging session. When this opens, a blue arrow should be visible, pointing at the first line of code in the program:



An alternative way of starting a debugging session is to display the code itself from the initial screen, select a line of code and set a breakpoint. This is done by, having selected a line, clicking the Stop icon: 

This sets a breakpoint for that line. When the program is then executed the execution will pause highlighting the line that has the Breakpoint set entering the debugging session. Usually, this is the easiest method to use, as one will often have a good idea of where the

issues in a program are allowing you to focus on specific areas of code straight away, rather than starting from the very beginning of a program as the previous method does:

```
SELECT * FROM zemployees.      " Basic Select Loop with a LINE-BREAK
WRITE / zemployees.
ENDSELECT.
```

```
→ [STOP] SELECT * FROM zemployees.      " Basic Select Loop with a LINE-BREAK
      WRITE / zemployees.
      ENDSELECT.
```

There are two types of breakpoint which can be set in a program. Static (which will be examined later) and dynamic. A dynamic breakpoint is the kind which was used above, and these are only valid for the current session. If one leaves the SAP GUI and returns later, any dynamic breakpoints set will no longer exist. A breakpoint can also be set by double-clicking any statement within the debugging session itself. To then remove these in the session, simply double-click the stop icon appearing adjacent to them.

You will notice that a number of buttons appear at the top of the debugging screen:



These buttons allow for different modes of the ABAP debugger to be entered. The default mode here is *Fields*.

The 'Single step' button, the first on the left in the row above the modes, also accessible with F5, allows one to go through the code within the debugger line-by-line, or indeed as its name would suggest, single steps. As one presses the button, the blue arrow on the left of the code will move down one line at a time.

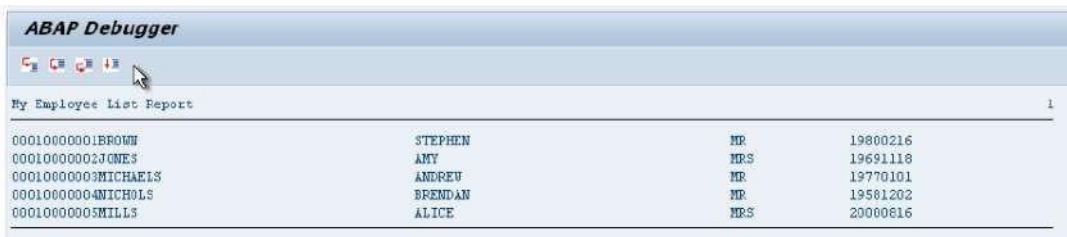
The next button along is the 'Execute' button, with a shortcut of F6. This allows for independent sections of code to be executed, such as function modules or forms. This can be very useful. If a program includes existing sections of code already created in an SAP sys-

tem which are known to be correct, there is no need to debug them. These can then be executed independently, while other parts are debugged to find specific problems.

The next button is the ‘Return’ function (F7). This can be very useful if one forgets to use the ‘Execute’ function. If one goes through the lines of a program step-by-step, using the F5 key to step into a working function module, which may contain many lines of code, it is likely the case that it does not need to be debugged (because you know this function module already exists). Pressing the F5 key endlessly to go through the lines of code here is unnecessary when one wants to step out of this function module and access the parts which require debugging. Using the ‘Return’ button, all of the code within a specific function can be executed, returning to the line of code which calls that function.

The fourth in the row is the ‘Continue’ option (F8). This allows one to continue the program without going through step-by-step, line-by-line. When this button is pressed, the program executes and the output screen is shown. This button can also be used to just access a selected line of code, where the cursor is positioned. If one positions the cursor in a line of code and presses continue, the blue arrow in the debugger will appear directly next to that line. If you then press continue again, the program will be executed.

The next option in this row of the toolbar is ‘Display list’, accessible with CTRL+F12. This takes you to the output screen as it currently stands within the debug session. Here, the code has been executed to output the result of the first SELECT statement in the program:



Employee ID	Name	Gender	Date of Birth
0001000001	BROWN	MR	19800216
0001000002	JONES	MRS	19691118
0001000003	MICHAELS	MR	19770101
0001000004	NICHOLS	MR	19581202
0001000005	MILLS	MRS	20000816

This function allows you to see the results of the reports whilst the program is in mid-flow.

The last option here is ‘Create watchpoint’ (SHIFT + F8). Watchpoints will be returned to soon.

Fields mode

The 'Fields' mode of the ABAP debugger allows the contents of fields to be checked and modified as the program is debugged. This can be accessed either by double-clicking the field name within the code itself, or entering it into the 'Field names' section below the code:

```

SKIP 2.
SELECT * FROM zemployees.  " Chain Statements
  WRITE: / zemployees-surname,
         zemployees-forename,
         zemployees-dob.
ENDSELECT.

```

Field names	Field contents
zemployees-surname	MICHAELS
zemployees-forename	ANDREW

SY-SUBRC 0 SY-TABIX 1 SY-DBCNT 3

Note that, since here a table is involved, in the field name section the name of the table must first be specified, followed by a -, then the name of the field. The field contents will be filled in automatically. As you step through code line-by-line in the SELECT loop, the text held in each field will change as each loop completes and moves onto the next record in the table. This section allows for 8 fields to be monitored at any time. Fields 5 - 8 can be made visible via the navigation buttons in the middle (*to the right of the numbers 1-4*).

Often when debugging a program, you may want to manually change the contents of fields. This can be achieved by replacing the text in the field contents area, then clicking the 'Change field contents' icon, marked with a pencil. Doing this can save a lot of time, avoiding having to exit the debugging session multiple times to enter new values into fields elsewhere:

zemployees-surname	JOHNSON	
zemployees-forename	ALICE	

System Variables





At the bottom of the debugger screen, are 3 fields, named 'SY-SUBRC', 'SY-TABIX' and 'SY-DBCNT':

SY-SUBRC 0	SY-TABIX 1	SY-DBCNT 5
------------	------------	------------

Note that the value boxes here are greyed-out, meaning that they cannot be changed manually. These are system fields, belonging to a table called SYST. This system table includes many system fields which are filled in at runtime. These system fields are filled in automatically while the program is executed. Most statements within ABAP will cause these system fields to be filled with 0 when executed successfully. It is important to remember that these fields are completely statement-dependent, meaning that they will contain different values depending on which statement is executed. These system codes and variables will be looked at in greater depth later.

Table Mode

The second mode along from the Fields button on the left of the screen is Table mode. Click this button and the code remains, but the bottom section changes to include an 'Internal table' entry, and a single row:

SELECT * FROM zemployees. " Basic Select Loop with a LINE-BREAK			
Internal table	<input type="text"/>	Type	Format E
	Change		Insert
	Append		Delete

Internal tables have not yet been covered in depth, but, put simply; an internal table is a table of records which is stored in memory while the program is running. Table mode allows one to interrogate the records and fields of each record in an internal table. As with

Fields mode, the internal table can either be double-clicked in the code, or manually entered into the 'Internal table' box.

If one does this for "zemployees", then, a new window appears, displaying the table name, its individual fields and their contents:

N.	Component name	T.	Ln...	Contents
1	<u>MANDT</u>	C	3	000
2	<u>EMPLOYEE</u>	N	8	10000005
3	<u>SURNAME</u>	C	40	JOHNSON
4	<u>FORENAME</u>	C	40	ALICE
5	<u>TITLE</u>	C	15	MRS
6	<u>DOB</u>	D	8	20000816

Things do look slightly different to normal here, as a table structure is being shown, rather than an actual internal table. This results in the debugger showing the table structure as above, listing the individual fields numbered 1 – 6 and their contents. When viewing an internal table in this mode, one will see a number of records for each internal table with their contents. These records can then be double-clicked to move to the above layout, showing the individual fields for each record. This will be returned to later.

In this screen, the code remains, but the area in which it is displayed is very small. One can continue to interrogate the code line-by-line as before still, but this may prove difficult. It is usually simpler to check Table mode for the information required, and then click back to Fields mode to continue the debug session.

Breakpoints

Click the Breakpoint mode button in the ABAP debugger screen. This allows you to see a list of the individual breakpoints which have been set. Double-clicking any breakpoint in the Breakpoints table will remove that breakpoint from the list:

The screenshot shows the ABAP debugger interface. The top window displays the following code:

```
EVENT START-OF-SELECTION
SELECT * FROM zemployees. " Basic Select Loop with a LINE-BREAK
WRITE zemployees.        " aftervthe first row is output.
WRITE /.
```

Below the code is the "Breakpoints" table, which contains the following data:

N.	Breakpnt type	in (absolute path)	Co.
1	Point in program	Z_EMPLOYEE_LIST_01(20)	
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			

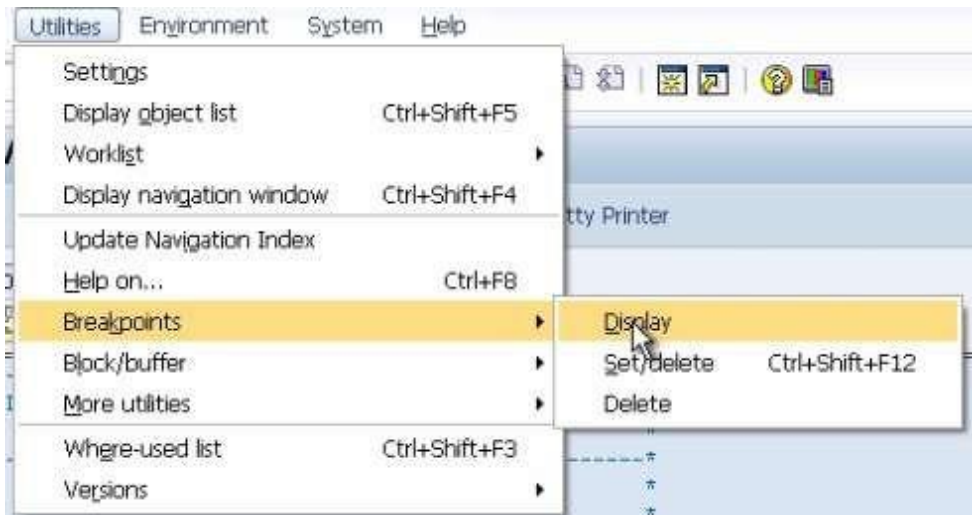
This breakpoint table can be very useful, particularly when one is in a large program with many breakpoints set. It allows one to review the breakpoint, and allows for the removal of breakpoints which are no longer desired.

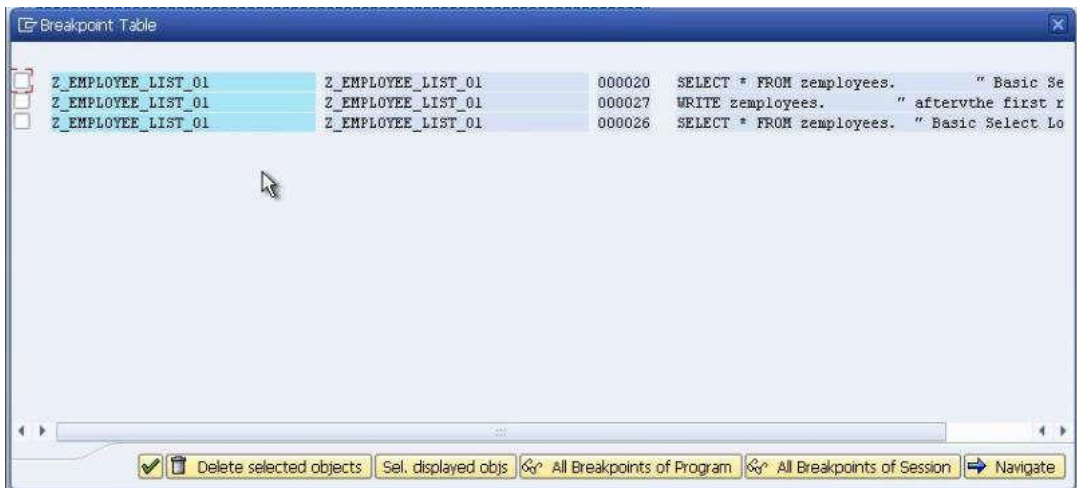
It is important to remember that breakpoints (and indeed Watchpoints) are only valid for the length of the current debug session. When you exit your session, the breakpoints will be deleted. However, an option does exist allowing you to save breakpoints (and, again, Watchpoints) before closing a debug session, keeping them active for the next time the program is to be debugged, saving the hassle of recreating them. This is done by entering

the 'Breakpoint' menu in the top toolbar and choosing 'Save'. All of the breakpoints saved will then remain until they are manually removed, or until the end of your SAP session.



If one is in the ABAP editor, it is possible to see an overview of all the dynamic breakpoints set in the program by accessing the following menu option: Utilities □ Breakpoints □ Display:





The options at the bottom of this breakpoint table allow one to delete selected break-points without entering the debugger and breakpoints can be navigated to in the program itself (within the ABAP editor) by double clicking them in this table.

Static Breakpoints

Static breakpoints were briefly alluded to earlier. These refer to a line of code written into a program which forces the program to enter debug mode at the specific line chosen. To do this, the statement **BREAK-POINT** is used. When the code is executed, the debug session will start with the usual blue arrow cursor pointing at the location of the static break-point.

```

WRITE / zemployees.
ENDSELECT.

BREAK-POINT.

ULINE.

SELECT * FROM zemployees. " Basic Select Loop with a I

```

```

ENDSELECT.
➔ BREAK-POINT.
ULINE.
SELECT * FROM zemployees. " Basic Select Loop with a LI

```

Once this statement is embedded in a program, it is active for all users. This is largely undesirable, as others running the program, who do not want to debug the code, would be faced with the breakpoint set by an individual user. Be careful not to leave this statement line in programs which will be transported to other systems.

Watchpoints

Click the Watchpoints button in the ABAP debugger. The program code will be visible above the Watchpoints table in the lower half of the screen. Breakpoints have previously been discussed, and can be very useful, but are not always the ideal tool to use to pause code execution, interrogate the contents of individual fields and internal tables and analyse the program's logic.

Imagine the program was processing a table containing 1000 records, and one wanted to debug the logic only when a certain condition occurs. This condition is dependent upon the data held in the records being processed. By using breakpoints, one would have to debug each individual record, obviously taking a huge amount of time. Here, Watchpoints become useful. Using these, one can tell the program to stop in the same manner that it would for a breakpoint, but instead of stopping at a specific line of code, it would stop based on the value in a field. In this example then, if this value occurred only in the 200th line of the table, a watchpoint would allow the first 199 records to be skipped over.

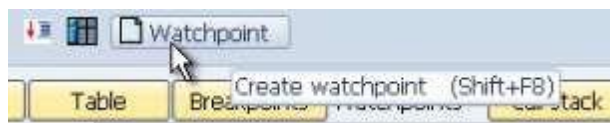
A watchpoint is created with the 'Create watchpoint' button, seen above the list of modes in the Watchpoint mode screen, or with SHIFT + F8.

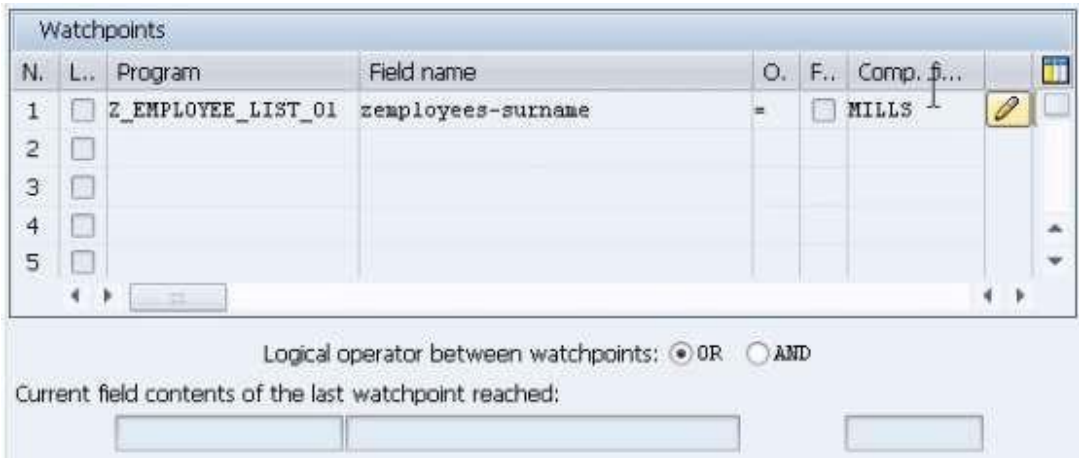
Once this is done, a dialogue box will appear, with the program name filled in automatically. Here you need to enter the name of the field to be watched. In the Z_EMPLOYEE_LIST_01 example here, we will enter the *surname* field. The format is *TABLE_NAME-FIELD_NAME*. Next, the relational operator is to be set. In this example, a sur-

name with the value “Mills” will be sought, so the operator here is an =. This can be selected from a drop-down menu, where one can also view other potential relational operators. The bottom field, then, should be filled in with the value to be watched for.

Note that one does not have to use a specific value in the bottom field, but can get a watchpoint to compare a field against another field within the program. To do this the ‘Comparison field’ box should be checked, and the field name typed into the box rather than a specific value.

Click the green tick to continue and create the watchpoint, and the entry will have been added to the list at the bottom of the screen:





Observe the boxes below the Watchpoints list here. They are currently empty, but when the program is executed, it will pause once a value of 'Mills' is reached in the 'surname' field and this will be included in the box.

The output before the program is executed looks like this:

```
My Employee List Report
```

00010000001BROWN	STEPHEN	MR	19800216
00010000002JONES	AMY	MRS	19691118
00010000003MICHAELS	ANDREW	MR	19770101
00010000004NICHOLS	BRENDAN	MR	19581202
00010000005MILLS	ALICE	MRS	20000816

00010000001BROWN	STEPHEN	MR	19800216
00010000002JONES	AMY	MRS	19691118
00010000003MICHAELS	ANDREW	MR	19770101
00010000004NICHOLS	BRENDAN	MR	19581202
00010000005MILLS	ALICE	MRS	20000816

Note that the surname Mills appears in the fifth row down. When the program is executed with the 'Mills' watchpoint set, the first four records will be written to the screen before pausing at the fifth, when Mills is displayed.

```
00010000001BROWN          STEPHEN          MR          19800216
00010000002JONES          AMY              MRS         19691118
00010000003MICHAELS      ANDREW          MR          19770101
00010000004NICHOLS       BRENDAN         MR          19581202
```

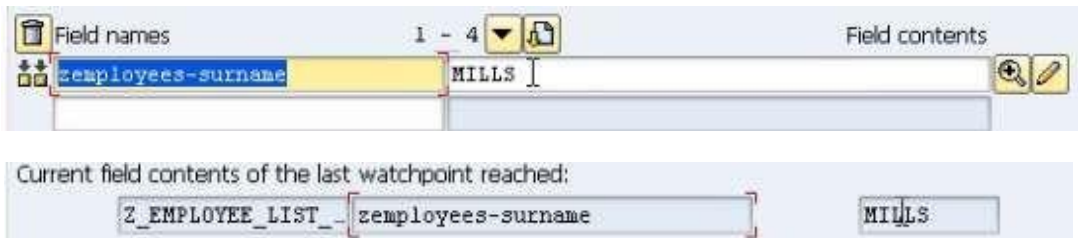
You will see that the blue arrow cursor has paused at the SELECT loop in the code.

```

➔ SELECT * FROM zemployees. " Basic Select Loop with a LINE-BREAK
    WRITE zemployees.      " aftervthe first row is output.
    WRITE /.
ENDSELECT.

```

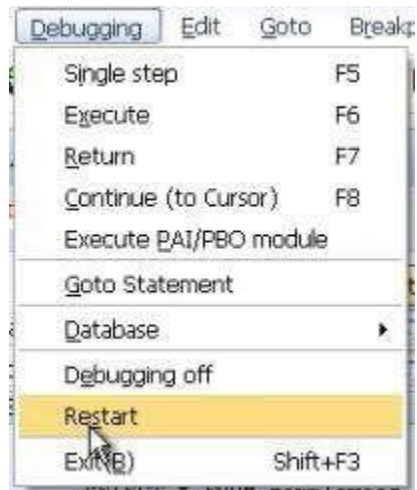
Enter `zemployees-surname` in the Fields mode of the debugger to view the contents of the field. You will see the field contains "MILLS". Also in the Watchpoints mode, the bottom field will now be filled:



Ending a Debug Session

There are two ways to stop debugging a program. The first is to use the F8 key to run the program all the way through to the end. Keep in mind though, that if any break or Watchpoints are set, the execution will likely pause and have to be started again, perhaps multiple times. Also this method depends entirely upon the program executing successfully. If any runtime errors are caused, the debug session will terminate and return you to the SAP menu screen.

The alternative way of stopping the debugger is to enter the 'Debugging' menu and choose 'Restart'. This way, no more of the program will be executed, and you can return to the ABAP Editor's initial screen:




Working with Database Tables

Making a Copy of a Table

This chapter will look at ways in which one can change the transparent tables created earlier. It is important to know how to do this, and the implications of adding and taking away fields for the underlying data in a database table.

Let's take a look at the ZEMPLOYEES table created in Chapter 2. In the SAP GUI, key in transaction code SE11 to access the ABAP dictionary, then display the table:



Field	K..	I...	Data element	DTyp	Len...	Dec...	Short text
MANDT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	MANDT	CLNT	3	0	Client
EMPLOYEE	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ZEENUM	NUMC	8	0	Employee Data Element
SURNAME	<input type="checkbox"/>	<input type="checkbox"/>	ZSURNAME	CHAR	40	0	Surname Data Element
FORENAME	<input type="checkbox"/>	<input type="checkbox"/>	ZFORENAME	CHAR	40	0	Forename Data Element
TITLE	<input type="checkbox"/>	<input type="checkbox"/>	ZTITLE	CHAR	15	0	Title Data Element
DOB	<input type="checkbox"/>	<input type="checkbox"/>	ZDOB	DATS	8	0	Date of Birth Data Element

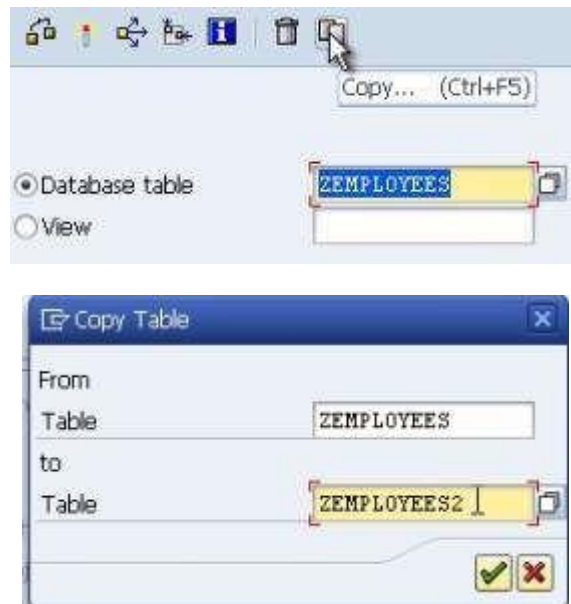
It is important to realise that whenever one wants to change a database table, there is a risk of losing data, especially where key fields in the table are being affected. The database system itself will try to determine whether adjustments can be made by deleting and creating new items which change the underlying database catalogue, or whether what has already defined has to be re-implemented.

Quite often, when working with large tables, one has to manage the manipulation of the data oneself, so as to be sure that data is not lost. Deleting fields is quite a simple task, the table structure and its contents can add certain complications. Before starting any data-base change tasks, it is important to mitigate against as many risks as possible, and start

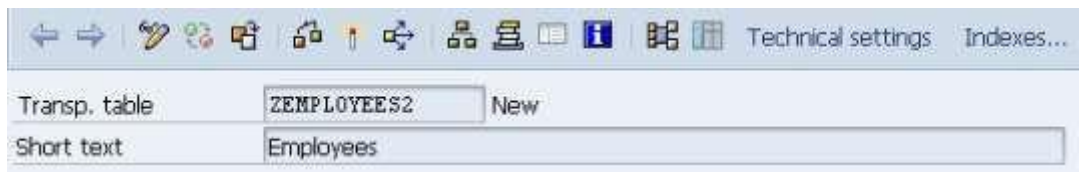
by using a copy of the database table, allowing one to test out any changes one may want to make, without affecting the initial table and its underlying data.

When you copy a database table, it is only the structure itself which is copied, meaning only its properties - fields and so on, not the actual data.

Step back to the initial SE11 screen. With ZEMPLOYEES in the Database table field, click the Copy button, then give the new table of ZEMPLOYEES2. The 'Create Object Directory Entry' box will appear and as before, select 'Local Object':

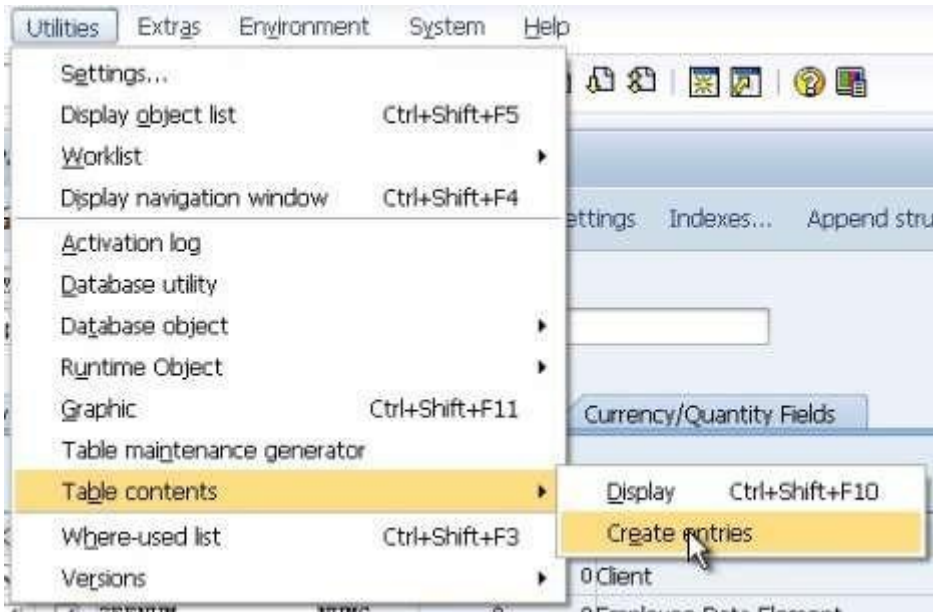


A copy of the table has now been created. Choose display at the SE11 screen and the copy will appear. The table's status will read as 'New'. It must be activated, so click the 'Change' button (the Pencil icon in the toolbar), and then Activate:



Note that all of the fields in the table, since they have been copied, are already active. This is why it is only the table itself which has to be activated here. If you try to look at the ta-

ble, you will find there are no contents, because only the structure was copied, not the underlying data. To create records, from the 'Utilities' menu, select 'Table Contents' and then 'Create Entries' to display the screen where the records for the table can be created as before.



Insert some records, click the Contents button, and then view the new table:

Table ZEMPLOYEES2 Insert	
Reset	
Client	<input type="text"/>
Employee Number	<input type="text" value="10000001"/>
Surname	<input type="text" value="Smith"/>
Forename	<input type="text" value="Paul"/>
Title	<input type="text" value="Mr"/>
Date of Birth	<input type="text" value="17.07.2012"/>

Data Browser: Table ZEMPLOYEES2 Select Entries 3

Table: ZEMPLOYEES2
Displayed fields: 6 of 6 Fixed columns: List width 0250

Client	Employee Number	Surname	Forename	Title	Date of Birth
000	10000001	SMITH	PAUL	MR	17.01.1980
000	10000002	BROWN	IAN	DR	15.07.1966
000	10000003	WILLIAMS	SARAH	MRS	11.09.1971

Add New Fields

Next, a new field will be added. This will be a non-key field and will be called **INITIALS**.

Create a new Data element for this named ZINITIALS using forward navigation. For the data element, set the short text to 'Initials' and set the domain to CHAR03 (a character string of length 3). In the Field label boxes type 'Initials', then activate the Data element. The table should now have a new field like this:

DUE	ZDUE	DAYS	Date of Birth Data Element
INITIALS	ZINITIALS	CHAR	Initials

Create another 3 more new fields and configure them as follows: •

Field Name 'GENDER'

- Set the Data element to 'ZGENDER'. Configure the data element as follows:
 - Short text: 'Gender'
 - Domain: 'CHAR01'
 - Field labels set to 'Gender'

• SALARY

- Set the Data element to ZSALARY
 - Short text: 'Salary'
 - Domain: 'CURR9' (*This has a length of 9, with 2 decimal places*)
 - Field labels set to 'Salary'.

One thing to note about the Salary field is that, because it is a currency, another field for this currency must be created and attached to ZSALARY to indicate what currency the salary is in. If you try to activate the table without doing this, an error message will appear asking for a reference field to specify the currency.

Create a new field called ECURRENCY. Currency fields should already exist in the system, so the Data element here will be a pre-existing one named **CURCY**. Type this, press enter and the remaining fields should fill in automatically, leaving the new section of the table looking like this:

<u>DOB</u>	<input type="checkbox"/>	<input type="checkbox"/>	<u>ZDOB</u>	DATS	8	0 Date of Birth Data Element
<u>INITIALS</u>	<input type="checkbox"/>	<input type="checkbox"/>	<u>ZINITIALS</u>	CHAR	3	0 Initials
<u>GENDER</u>	<input type="checkbox"/>	<input type="checkbox"/>	<u>ZGENDER</u>	CHAR	1	0 Gender
<u>SALARY</u>	<input type="checkbox"/>	<input type="checkbox"/>	<u>ZSALARY</u>	CURR	9	2 Salary
<u>ECURRENCY</u>	<input type="checkbox"/>	<input type="checkbox"/>	<u>CURCY</u>	CUKY	5	0 Currency Key

Next, the system must be told that the Salary field is referencing the Currency field. Above the table will be able to see a tab labelled 'Currency/Quantity Fields'. Click this and the table will be shown with two boxes to be filled in for the Salary field, since it has already been specified that the domain for this field is Currency. In the 'Reference table' column enter the name of the table, 'ZEMPLOYEES2' and in the 'Reference field' column, the name of the new Currency Key, 'ECURRENCY'. Now the table can be activated error free.

Attributes Delivery and Maintenance Fields Entry help/check Currency/Quantity Fields						
Search help 1 / 10						
Field	Data element	DType	Reference table	Ref. field	Short text	
MANDT	MANDT	CLMT			Client	
EMPLOYEE	ZEENUM	NUMC			Employee Data Element	
SURNAME	ZSURNAME	CHAR			Surname Data Element	
FORENAME	ZFORENAME	CHAR			Forename Data Element	
TITLE	ZTITLE	CHAR			Title Data Element	
DOB	ZDOB	DATS			Date of Birth Data Element	
INITIALS	ZINITIALS	CHAR			Initials	
GENDER	ZGENDER	CHAR			Gender	
SALARY	ZSALARY	CURR	ZEMPLOYEES2	ECURRENCY	Salary	
ECURRENCY	CURCY	CUKY			Currency Key	

Foreign Keys

As shown earlier enter a new record. You will see that the currency key does not offer any kind of drop-down menu, here for this example, type GBP, indicating Great British Pounds:

Client	
Employee Number	10000004
Surname	ROSE
Forename	ANN
Title	MISS
Date of Birth	04.01.1985
Initials	C
Gender	F
Salary	12345
Currency key	GBP

Save the record, and then return to the design of the table, where we can now add some error-checking to ensure that valid entries are made in the Currency key field.

To enable error-checking on the currency key field, we need to make use of a Foreign Key. These are used to ensure that only valid values can be entered into a field. Use forward navigation on the CURCY data element. Look at the Data type tab and you will see that the data element refers to a standard SAP domain, WAERS:

Data element	CURCY	Active
Short text	Currency Key	
<input checked="" type="radio"/> Elementary type <input checked="" type="radio"/> Domain		
	WAERS	Currency key
Data Type	CUKY	Currency key, referenced by CURR...
Length	5	Decimal Places 0

Double-click the WAERS domain to use forward navigation again. Look at the 'Value range' tab in this window, a 'Value table' box is visible at the bottom, labelled TCURC:

Value table	TCURC
-------------	-------


A Value table can be used to determine the entries that can be made in the field based on this domain. Double-click TCURC to again use forward navigation and this value table will be displayed.

Field	K.	I.	Data element	DTyp	Len...	Dec...	Short text
MANDT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	MANDT	CLWT	3	0	Client
WAERS	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	WAERS_CURC	CUKY	5	0	Currency Key
ISOCD	<input type="checkbox"/>	<input type="checkbox"/>	ISOCD	CHAR	3	0	ISO currency code
ALTWR	<input type="checkbox"/>	<input type="checkbox"/>	ALTWR	CHAR	3	0	Alternative key for currencies
GDATU	<input type="checkbox"/>	<input type="checkbox"/>	DATUM_CURC	DATS	8	0	Date until which the currency is valid
XPRIMARY	<input type="checkbox"/>	<input type="checkbox"/>	XPRIMARY	CHAR	1	0	Primary SAP Currency Code for ISO Code

Use the data browser to look at the data in this table. If you scroll down, the GBP value from before can be found, among a number of others. This table can be used to ensure that, in future, only entries found in this table can be entered into our new table ZEMPLOYEES2

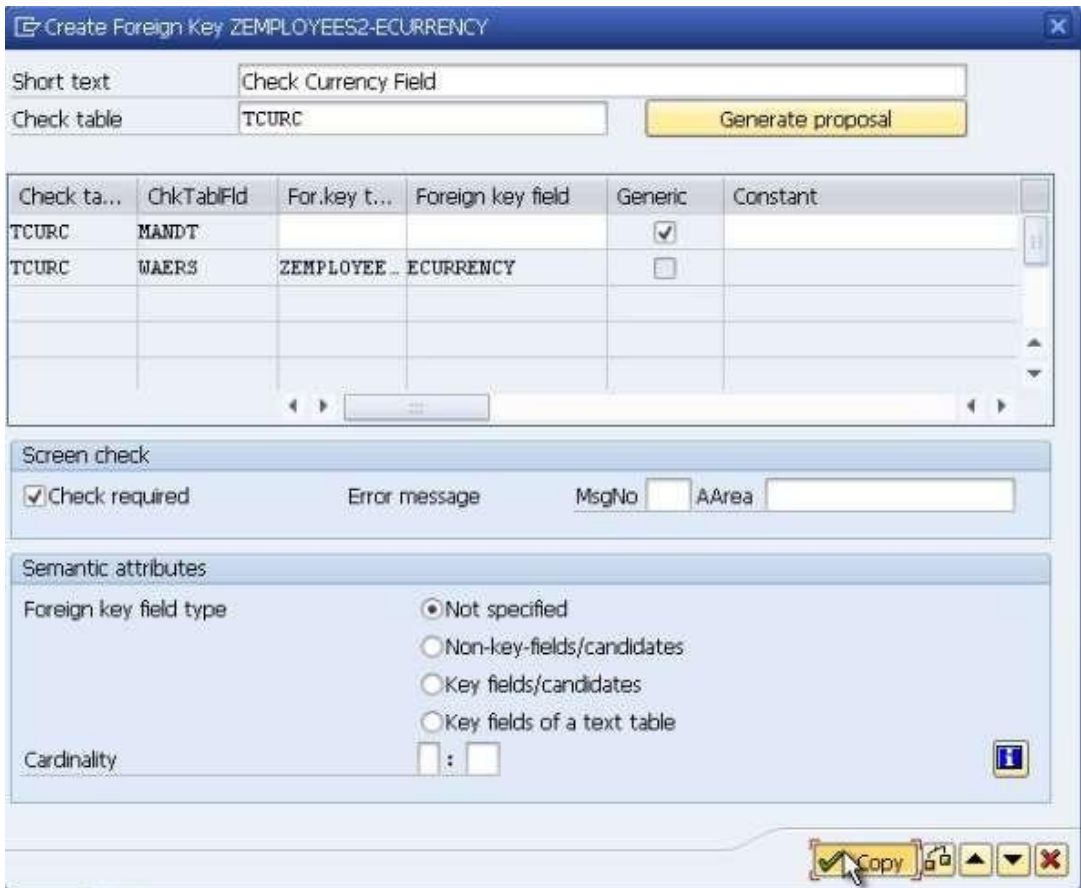
Table: TCURC
 Displayed fields: 6 of 6 Fixed columns: 2 L

	Client	Currency	ISO code	Alternative key	Valid until	Primary
<input type="checkbox"/>	000	ESP	ESP	724	00.00.0000	
<input type="checkbox"/>	000	ETB	ETB	230	00.00.0000	
<input type="checkbox"/>	000	EUR	EUR	978	00.00.0000	
<input type="checkbox"/>	000	FIM	FIM	246	00.00.0000	
<input type="checkbox"/>	000	FJD	FJD	242	00.00.0000	
<input type="checkbox"/>	000	FKP	FKP	238	00.00.0000	
<input type="checkbox"/>	000	FRF	FRF	250	00.00.0000	
<input checked="" type="checkbox"/>	000	GBP	GBP	826	00.00.0000	
<input type="checkbox"/>	000	GEL	GEL	981	00.00.0000	
<input type="checkbox"/>	000	GHC	GHC	288	00.00.0000	
<input type="checkbox"/>	000	GIP	GIP	292	00.00.0000	
<input type="checkbox"/>	000	GMD	GMD	270	00.00.0000	
<input type="checkbox"/>	000	GNF	GNF	324	00.00.0000	
<input type="checkbox"/>	000	GRD	GRD	300	00.00.0000	
<input type="checkbox"/>	000	GTQ	GTQ	320	00.00.0000	
<input type="checkbox"/>	000	GWP	GWP	624	00.00.0000	

Return to the 'Maintain table' screen for ZEMPLOYEES2, highlight the ECURRENCY field, and click the Foreign key button visible in the toolbar above: 

Choose 'Yes' in the box which appears and a 'Create Foreign Key' window will emerge. Type the short text 'Check Currency Field'. A small table is visible, detailing the two key fields from the TCURC table and the ZEMPLOYEES2 table. The option is available to ensure that the foreign key matches both fields, so that when the user is allowed to select an en- try, the records returned will only be valid for the Client which is being worked in.

Here though, the Client is not to be chosen as part of the key, so select the Check-box 'Generic' for the top row, which refers to the Client, and remove the text from the two boxes on this row where this is possible. Then click the 'Copy' button. The foreign key will be created:



Short text:

Check table:

Check ta...	ChkTabFld	For.key t...	Foreign key field	Generic	Constant
TCURC	MANDT			<input checked="" type="checkbox"/>	
TCURC	WAERS	ZEMPLOYEE ..	ECURRENCY	<input type="checkbox"/>	

Screen check

Check required Error message MsgNo AArea

Semantic attributes

Foreign key field type

- Not specified
- Non-key-fields/candidates
- Key fields/candidates
- Key fields of a text table

Cardinality :

Activate the table, and then browse the data. Now, select the currency key and either press the F4 key or select the drop-down box that appears, displaying all valid entries for this field. If you were in record change mode you will then be able to select a value from the table and see it update your employees 2 record. Try it out and select USD (US Dollar).

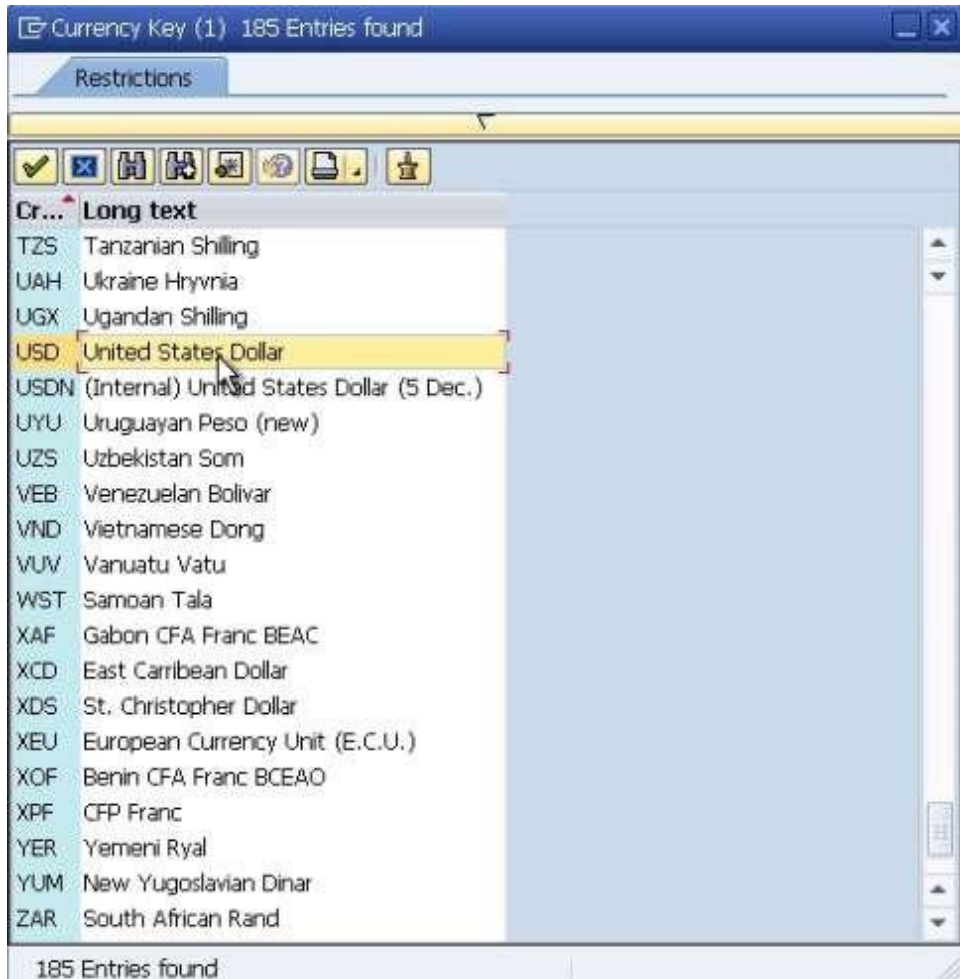


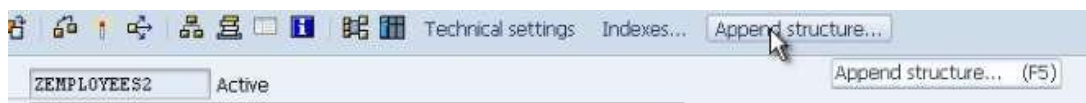
Table ZEMPLOYEES2 Change	
Check table...	
Client	000
Employee Number	10000004
Surname	ROSE
Forename	ANN
Title	MISS
Date of Birth	04.01.1985
Initials	C
Gender	F
Salary	12,345.00
Currency key	USD

Append Structures

Having looked at foreign keys, the next thing to look at are Append structures. These can be used to add additional fields. This is the preferred method for maintaining SAP delivered tables and quite often for customer-specific tables. If one does not use Append structures, problems can arise if, for example, a new version of SAP is used which does not correspond with aspects of the tables already created, resulting in serious errors.

Append structures give a safe way to enhance tables. When these are used, the initial table remains unchanged, removing any risk of changes being overwritten later if a different version of SAP is used. Quite often, a table may have multiple Append structures applied to it, because different development needs have arisen as time has gone by and people have wanted to add further fields to the standard SAP tables.

In the SE11 Maintain Table screen, go to the 'Append structure' button on the right of the top toolbar:

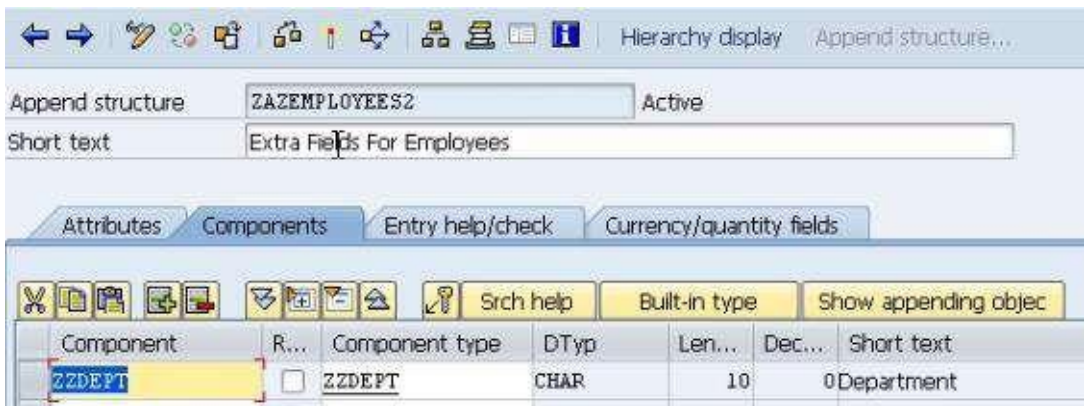


Click this, and the system will suggest a name, ZAZEMPLOYEES2 (note that this, again, must begin with a Z). Accept this and you will be presented with what looks like an empty table structure. Enter the Short text “Extra Fields For Employees”, and then move down to the table.

Note that the first field now is called ‘Component’. This is where new fields are created. However, it may be useful to differentiate between fields created in the main table, and the new components created here in the Append structure. Since both must comply with the customer name rules, where Z was used in the main table, here use ZZ.

For the first component, a ‘Department’ field will be created. Type in the ‘Component’ box ‘ZZDEPT’ and the same again in ‘Component type’. For this Component type, use forward navigation in the same way that it was used for the Data element before, double-clicking to create. Save the Append structure as a local object when prompted, and then select to create a Data element when prompted subsequently.

The familiar data element screen will now appear. Type ‘Department’ for the short text, use CHAR10 for the domain and ‘Department’ again for the Field labels, then activate the data element. Step back to the Append structure screen, then Activate:



Return to the main table screen, where a new row displaying the Append structure will have been created. To then access this structure, simply double-click the row. In Change mode only the ‘.APPEND’ line will be visible by default, but in Display mode the fields created within this will appear below:

SALARY	<input type="checkbox"/>	<input type="checkbox"/>	ZSALARY	CURR	9	2 Salary
ECURRENCY	<input type="checkbox"/>	<input type="checkbox"/>	CURCY	CUKY	5	0 Currency Key
.APPEND	<input type="checkbox"/>	<input type="checkbox"/>	ZAZEMPLOYEES2		0	0 Extra Fields For Employees

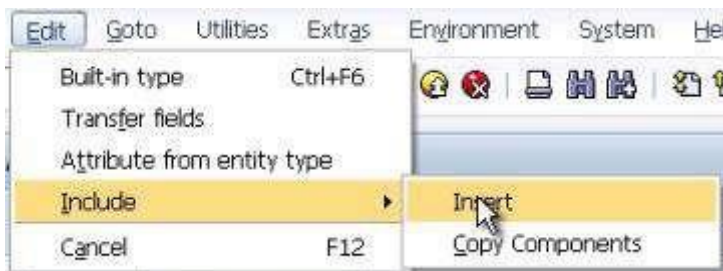
This is a very useful way to add new fields to a table without affecting the structure of the table itself. If one then browses the data as normal, a new column will have been called 'Department'. Data can then be entered into this field just like it can for any other:

Salary	Currency key	Department
0.00		
0.00		
0.00		
12,345.00	USD	

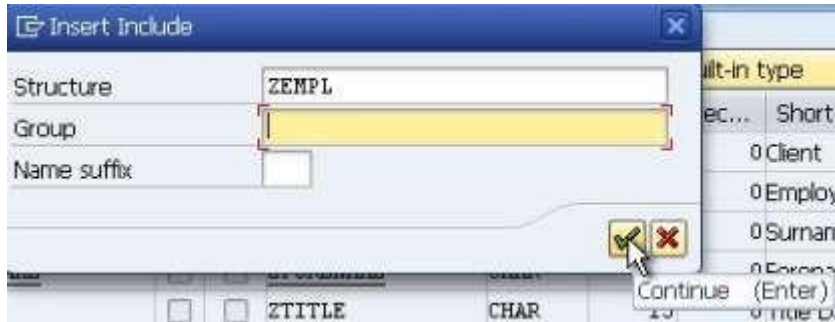
Include Structures

Include structures are similar to Append structures, with the main difference being that they are re-usable objects and can be linked to many other tables, ABAP programs, dialogue programs and structures. It is important to keep in mind that Include structures must be flat structures, meaning that they cannot hold any additional structure within them, and that the maximum length of the fields within an include structure is 16 characters.

There is no Include structure button in the way that there is an Append structure button. To create one, first ensure Change mode is selected. Where the cursor is placed is important here, as wherever the cursor is when the Include structure is created, it will be created one row above. If you want the Include structure to be part of the table key, it must appear at the top, because all table fields used as a table key need to be grouped together at the top. In this instance though, it will just be inserted above the Append structure. Place the cursor on the '.APPEND' row, select the 'Edit' menu, then 'Include' and 'Insert'.



In the window that appears, enter 'ZEMPL' in the 'Structure' field and click the continue button. A warning box will appear stating that this is not yet active, dismiss this, and the Include structure should now appear in the table:



<u>SALARY</u>	<input type="checkbox"/>	<input type="checkbox"/>	<u>ZSALARY</u>	CURR	9	2Salary
<u>ECURRENCY</u>	<input type="checkbox"/>	<input type="checkbox"/>	<u>CURCY</u>	CUKY	5	0Currency Key
.INCLUDE	<input type="checkbox"/>	<input type="checkbox"/>	<u>ZEMPL</u>		0	0
<u>.APPEND</u>	<input type="checkbox"/>	<input type="checkbox"/>	<u>ZAZEMPLOYEES2</u>		0	0Extra Fields For Employees

To add a field to this, use forward navigation as before, double-clicking where '.INCLUDE ZEMPL' appears, save and choose 'Yes' to create the structure. The screen which then appears is very similar to the Append structure screen.

Type the Short text "Employee Include" and begin to create a field (the boxes are, like in the Append structure, labelled 'Component'), this time for location, called ZZLOCAT, and use ZLOCAT for the 'Component type'. Use forward navigation again to create this Data element with Short text 'Location', the domain CHAR10 and 'Location' again for the Field labels, then Activate this as usual. Activate the Include structure once the field has been created and return to the main table to see the Include structure located just where we wanted it, above the Append structure:

<u>ECURRENCY</u>	<input type="checkbox"/>	<input type="checkbox"/>	<u>CURCY</u>	CUKY	5	0Currency Key
.INCLUDE	<input type="checkbox"/>	<input type="checkbox"/>	<u>ZEMPL</u>		0	0Employee Include
<u>.APPEND</u>	<input type="checkbox"/>	<input type="checkbox"/>	<u>ZAZEMPLOYEES2</u>		0	0Extra Fields For Employees

Activate the table now, and view the contents. The Location column should now be visible, and these records can now be edited and created like any other:

key	Location	Department

Client	000
Employee Number	10000005
Surname	GREEN
Forename	ANDREW
Title	MR
Date of Birth	04.01.1982
Initials	P
Gender	M
Salary	245,200
Currency key	HUF
Location	LONDON
Department	IT

If one switches to Display mode, the field created in the Include structure can be seen in the context of the main table, albeit in a different colour:

Field	K..	I...	Data element	DTyp	Len...	Dec...	Short text
<u>SURNAME</u>	<input type="checkbox"/>	<input type="checkbox"/>	ZSURNAME	CHAR	40	0	Surname Data Element
<u>FORENAME</u>	<input type="checkbox"/>	<input type="checkbox"/>	ZFORENAME	CHAR	40	0	Forename Data Element
<u>TITLE</u>	<input type="checkbox"/>	<input type="checkbox"/>	ZTITLE	CHAR	15	0	Title Data Element
<u>DOB</u>	<input type="checkbox"/>	<input type="checkbox"/>	ZDOB	DATS	8	0	Date of Birth Data Element
<u>INITIALS</u>	<input type="checkbox"/>	<input type="checkbox"/>	ZINITIALS	CHAR	3	0	Initials
<u>GENDER</u>	<input type="checkbox"/>	<input type="checkbox"/>	ZGENDER	CHAR	1	0	Gender
<u>SALARY</u>	<input type="checkbox"/>	<input type="checkbox"/>	ZSALARY	CURR	9	2	Salary
<u>ECURRENCY</u>	<input type="checkbox"/>	<input type="checkbox"/>	CURCY	CUKY	5	0	Currency Key
<u>.INCLUDE</u>	<input type="checkbox"/>	<input type="checkbox"/>	ZEMPL		0	0	Employee Include
<u>ZZLOCAT</u>	<input type="checkbox"/>	<input type="checkbox"/>	ZLOCAT	CHAR	10	0	Location
<u>.APPEND</u>	<input type="checkbox"/>	<input type="checkbox"/>	ZAZEMPLOYEES2		0	0	Extra Fields For Employees
<u>ZZDEPT</u>	<input type="checkbox"/>	<input type="checkbox"/>	ZZDEPT	CHAR	10	0	Department

In Change mode, these fields can be seen by selecting the ‘.INCLUDE’ row and clicking the ‘Expand include’ icon (the same works for the Append structure also):

Field	K..	I...	Expand include	DTyp	Len...	Dec...	Short text
<u>SURNAME</u>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	CHAR	40	0	Surname Data Element
<u>FORENAME</u>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	CHAR	40	0	Forename Data Element
<u>TITLE</u>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	CHAR	15	0	Title Data Element
<u>DOB</u>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	DATS	8	0	Date of Birth Data Element
<u>INITIALS</u>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	CHAR	3	0	Initials
<u>GENDER</u>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	CHAR	1	0	Gender
<u>SALARY</u>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	CURR	9	2	Salary
<u>ECURRENCY</u>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	CUKY	5	0	Currency Key
<u>.INCLUDE</u>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		0	0	Employee Include
<u>ZZLOCAT</u>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	CHAR	10	0	Location

Key Fields

If you want to add or remove fields which are designated key fields, then it is important to take into consideration what will be going on in the database itself. All of the new elements which have been created for this table have their features applied by the system to the ABAP dictionary, not the underlying database. When any key field is adjusted, the system has to apply changes to the underlying database itself. If there is data in the table, and key fields are changed, this can have unintended consequences.

If you introduce a new key field, this will probably not have a large effect. However, if one makes a key field no longer a key field, this will require consideration, because if there is a lot of data in the underlying database, by taking away a key field, duplicate records could be introduced. Corrupt data or records being deleted from the table can also happen here.

Let's see how we can add, remove and alter fields without these hazards.

Open the full ZEMPLOYEES2 table in the ABAP Dictionary 'Maintain Table' screen. Let's change the 'Surname' field by turning it into a key field.

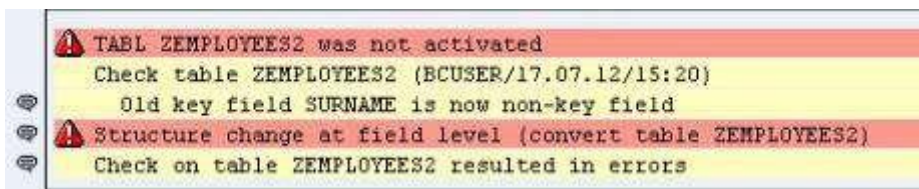
Check the two boxes (key and Index) by 'SURNAME' and Activate the table. When you now view the table contents, the surname column will be a darker colour, indicating that it is now a key field. Beyond this though, it appears very little has changed:

Field	Key	I...	Data element
MANDT	(Key)	<input checked="" type="checkbox"/>	MANDT
EMPLOYEE	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ZEENUM
SURNAME	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ZSURNAME
FORENAME	<input type="checkbox"/>	<input type="checkbox"/>	ZFORENAME

Displayed fields: 12 of 12 Fixed columns: List width 0250

Client	Employee Number	Surname	Forename
000	10000001	SMITH	PAUL
000	10000002	BROWN	IAN
000	10000003	WILLIAMS	SARAH
000	10000004	ROSE	ANN
000	10000005	GREEN	ANDREW

Now, uncheck the boxes on the 'Maintain Table' screen, to make it no longer a key field. When you try to activate the table an error message appears, refusing to activate the table as data may be lost with the removal of a key field:



To activate the table against what seem to be the wishes of the system (after all, one knows the data will be fine as the surname field has not been operating as a key field at any point previously), a different transaction must be used.

From the ‘Utilities’ menu, select ‘Database utility’, or use transaction code SE14. A new screen will appear:

ABAP Dictionary: Utility for Database Tables

Indexes... Storage parameters Check... Object log **i**

Name: ZEMPLOYEES2 Transparent table

Short text: Employees

Last changed: BCUSER 17.07.2012

Status: Revised Saved

Exists in the database

Execute database operation

Processing type

Direct

Background

Enter for mass processing

Create database table

Delete database table

Activate and adjust database Save data Delete data

This transaction lets us automatically adjust the data held in our table when making adjustments to the database table structure. Environments where tables are being worked on may contain a huge number of records. With this in mind, this transaction can be executed as a background process. However, for our example the ‘Direct’ option is the option

to choose because we know we have very few records in our database table. Select this, and then click 'Activate and adjust database' with 'Save data' radio button selected. Say 'Yes' when the box asks "Request: 'Adjust'" and notice the status bar should indicate the success of this execution. Then, step back to the 'Maintain Table' screen and you will see the table should be Active with the surname field no longer key.

To insert a new field as part of the table key, you must be able to adjust the location of fields on the screen. For example, if you wanted to create a new field above the surname field, you would highlight the row and then click the 'Insert row' icon in the toolbar. *This toolbar also includes 'Cut', 'Copy' and 'Paste' options, allowing for rows to be moved up and down if there is a need to do this:*



Field	K.	I.	Data element	DTyp	Len
MANDT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	MANDT	CLNT	
EMPLOYEE	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ZEENUM	NUMC	
SURNAME	<input type="checkbox"/>	<input type="checkbox"/>	ZSURNAME	CHAR	

Deleting Fields

While infrequent, occasionally there may be a need to remove a field from a table. When doing this, it is important to take special care, as data can be lost in the process. Certainly in the case of key fields.

If, for example, the Currency key field was removed from our table, the foreign key relationship to the TCURC table would be removed. As the SALARY field has to have a related Currency Key this would cause the table to no longer continue working, and likely make the ZEMPLOYEES2 table become inactive.

When deleting fields it is important to ask oneself whether the data being held in the table is being used elsewhere, and whether its deletion will have further consequences. If you do try to delete fields which are being used elsewhere, the SAP system should try to prevent this, or at least issue a stern warning. This is not necessarily to be relied upon though, so always ensure to check manually what the effects of deletion are likely to be. Also, if

you do delete fields, the table will have to be adjusted via the SE14 transaction to be activated again.

Create a new field, above '.INCLUDE', named 'ZAWESOME'. Use a previously created Data element, here ZTITLE just to save time, and activate the table:

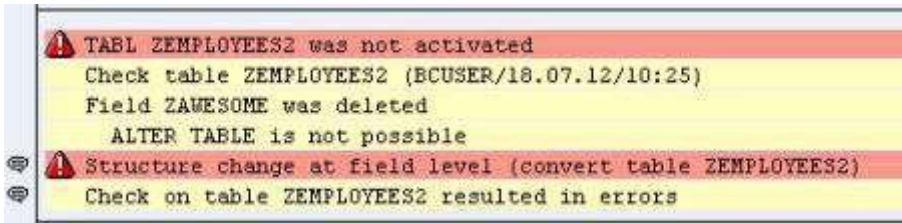
ECURRENCY	<input type="checkbox"/>	<input type="checkbox"/>	CURCY	CURY	5	0Currency Key
ZAWESOME	<input type="checkbox"/>	<input type="checkbox"/>	ZTITLE	CHAR	15	0Title Data Element
.INCLUDE	<input type="checkbox"/>	<input type="checkbox"/>	ZEMPL		0	0Employee Include

Create a new record in the table. The data here is not important and will be deleted, so the content can be anything:

Client	
Employee Number	10000010
Surname	qwe
Forename	qwe
Title (TITLE)	qwe
Date of Birth	04.01.1982
Initials	q
Gender	M
Salary	1234
Currency key	GBP
Title (ZAWESOME)	awesome
Location	LONDON
Department	HR

Currency key	Title	Location	
USD		PARIS	F
HUF		LONDON	I
GBP	AWESOME	LONDON	F

Now, to delete the field, highlight it in the ‘Maintain Table’ screen, and click the ‘Remove row’ icon, in the toolbar next to ‘Insert row’. The row will disappear, but when you try to activate the table, an error message will appear:



Transaction SE14 must again be used to adjust the table so the change can be applied. Follow the same steps as in the previous section to perform this task. Once this is complete, view the table again. The column has disappeared, and the data which was contained within it lost:

Currency key	Location
USD	PARIS
HUF	LONDON
GBP	LONDON

To see what happens when a key field is deleted, return to the ABAP Dictionary initial screen and make a copy of ZEMPLOYEES2, called, unsurprisingly, ZEMPLOYEES3. *Doing this will allow the ZEMPLOYEES2 table to not be damaged in this risky procedure.* Activate the new table (which, don’t forget, will be empty of records). As before, again make the Sur- name field a key field. Now create some records for this table:

Client	Employee Number	Surname	Forename	Title
000	10000001	SMITH	PAUL	MR
000	10000001	SMITH2	PAUL	MR
000	10000002	ANDREWS	PAUL	MR
000	10000002	ANDREWS-2	PAUL	MR

To save time creating new records, the same data was replicated here, with only slight changes to the key fields. Remember that it is only one key field per entry which must be unique for that particular record to be unique itself.

Now, the surname field will be deleted, and the effects of deleting this key field observed. By removing this key field, the only unique data which will be held for each record will be the Employee Number and Client. Since SMITH and SMITH2, and ANDREWS and ANDREWS-2 have the same Employee Number and Client, these will no longer hold unique key field data, leaving duplicate records, which the system will not allow.

Remove the Surname field; try to activate the table, and error messages will appear. Go through SE14 to adjust the table for activation. When you now view the table, the Surname field is gone, and two records have been lost, leaving only one of the two records for each of the two Employee Numbers used:

Client	Employee Number	Forename	Title	Date
<input type="checkbox"/> 000	10000001	PAUL	MR	01.01
<input type="checkbox"/> 000	10000002	PAUL	MR	01.01

Deleting Tables

One will not often have to delete an entire database table, for largely the same reasons as were outlined above for fields. If this does have to be done it is important to remember that one's own customer-specific tables are the only ones which can be deleted, SAP delivered tables cannot be deleted. Because ZEMPLOYEEES3 has only just been created, and nothing else depends on this table, it can be deleted without consequences.

To check whether a table can be deleted without causing unintended consequences elsewhere in the system, return to the ABAP Dictionary's initial screen. Because the original ZEMPLOYEEES table was used in the programs which have been created, use this as a test.

Insert this into the Database table field on the screen and then click the 'Where-used list' icon from the toolbar.



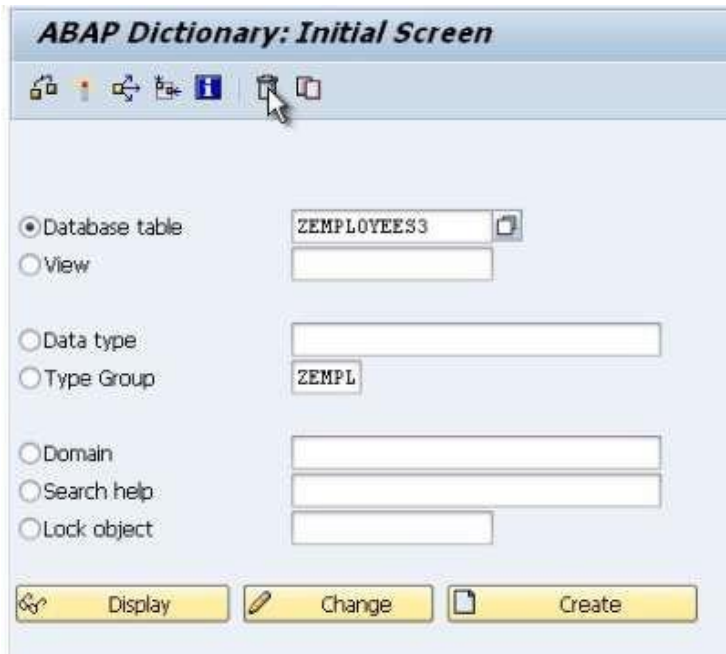
Once this is clicked, a dialogue box will appear offering a list of check-boxes. This will then search all of the different areas of the SAP system selected for references to the table ZEMPLOYEEES. To execute this search click the Continue icon. Choose 'Yes' to the pop-up box, and wait while the system compiles the search results, which here show that this table is being used currently by 2 programs:

Program	Short description
<input type="checkbox"/> Z_EMPLOYEE_LIST_01	My Employee List Report
<input type="checkbox"/> Z_RELEASE_4	Release 4

Having done this, one now knows that if the ZEMPLOYEEES table were to be deleted, these programs would become inactive. By double-clicking these entries, one can see the code in the program where ZEMPLOYEEES is referred to, and if you double-click on any line of the program, it will open the program at that line of code in the ABAP Editor. The Where-used button is a very useful tool, which can be invaluable not just when deleting programs, but in many other scenarios.

If you were to try to delete ZEMPLOYEEES, the system would not allow this course of action and would prevent it from happening until all the programs that are dependent upon it were either edited to remove references or deleted altogether themselves.

Since nothing depends upon ZEMPLOYEEES3, this can be deleted. With the correct name in the 'Database table' field, click the 'Delete' button in the toolbar:



A box appears stating that the data contained in the table would also be deleted. If you click the green tick icon this time, the system would return to the main screen with the table still intact. If the middle button, illustrated with the trashcan icon is clicked, this will proceed with the deletion. Once this is done, the status bar should confirm the action. If you try to display the table now, it does not exist. Once the deletion is completed, it can- not be undone:

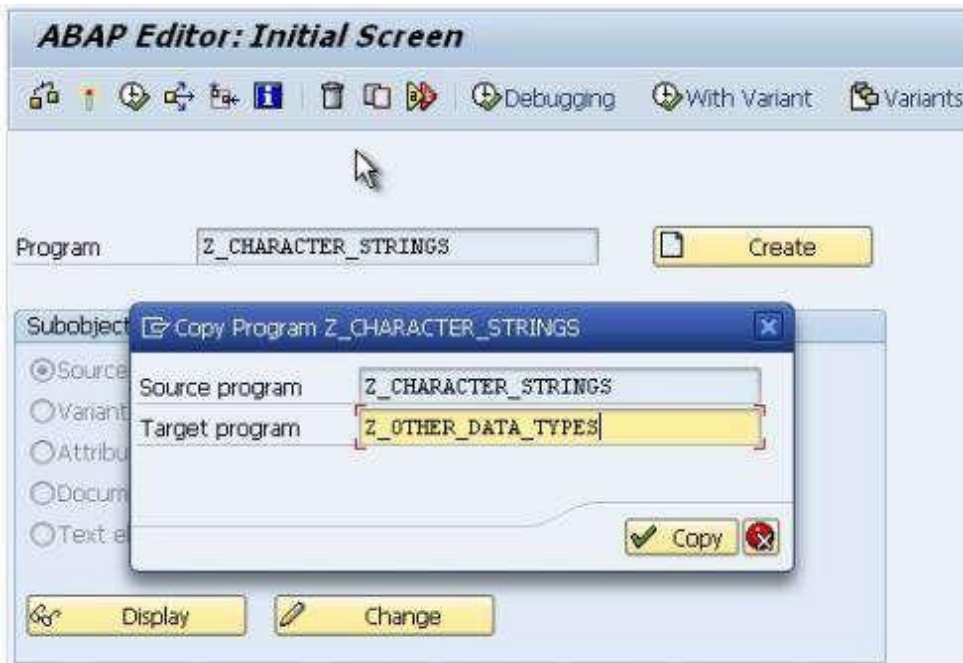


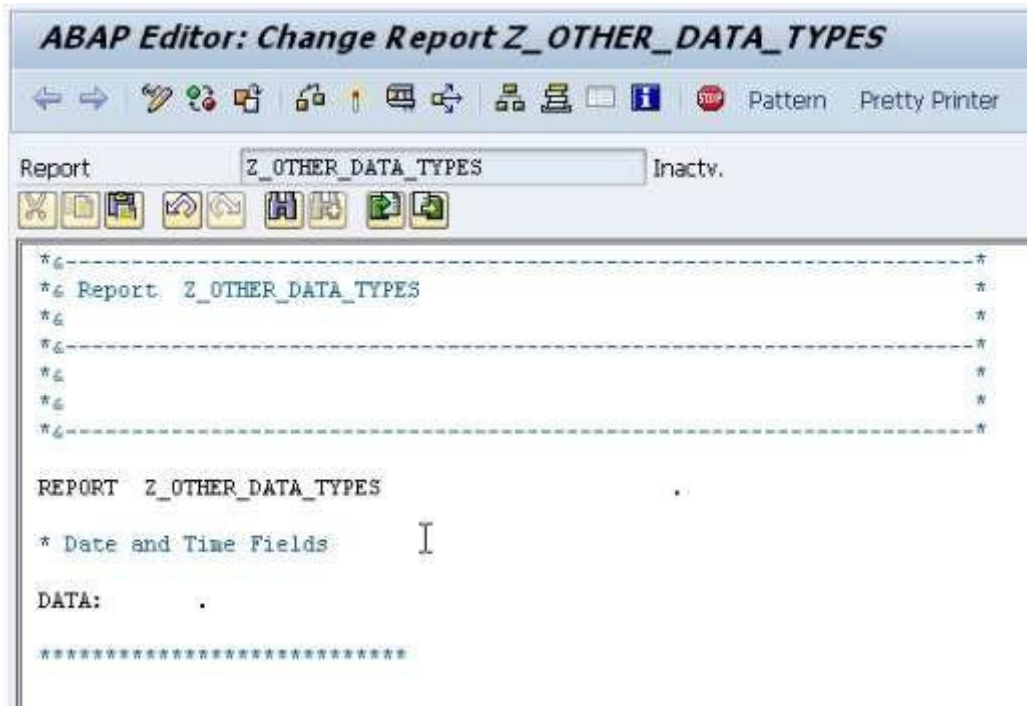
Working with Other Data Types

Date and Time Fields

This section will look at some other data types which can be used in ABAP. So far, numeric fields have been used for performing calculations, and character strings have been examined along with the ways these can be manipulated with ABAP statements. Now, date and time fields will be looked at.

Enter the ABAP editor (with transaction SE38) and make a copy of the previous program, alter the comment sections, and remove most of the code:





Date and time fields are not stored as numeric data types, but instead as character data types. Effectively, they are character strings which can be used in calculations. This is made possible by the inbuilt automatic data type conversions which have previously been discussed. Just like any other data type, the DATA statement is used to declare these fields.

For a date field, the data type is referred to with 'd', and is limited to 8 characters. The first 4 of these represent the year, the next 2 the month, and the final 2 the day. The VALUE addition is used to specify this, and if it is not used then the value, by default, is assigned as 8 zeros. In the example below, the date is the 1st of January, 2012:

```
REPORT z_other_data_types .
* Date and Time Fields
* Date fields format: YYYYMMDD with initial value of '00000000'
DATA my_date TYPE d VALUE '20120101'.
*****
```

The LIKE statement, of course, can also be used. SY-DATUM is a system variable, which always holds the value of the system's date. Below, "my_date2" is defined in the same way as this system variable:

```
DATA my_date2 LIKE SY-DATUM.
```

Time fields work similarly, but this time are limited to 6 characters. The first 2 refer to the hour, the second 2 the minute, and the final 2 the second. Again, the default value will be 6 zeros. The data type this time is 't'. Again, the LIKE statement can be used, here for the system's time field, referred to with SY-UZEIT:

```
* Time fields format: HHMMSS with initial value of '000000'
DATA my_time TYPE t VALUE '111005'.

DATA my_time2 LIKE sy-uzzeit.

*****
```

We can then use the WRITE statement to output the field contents:

```
WRITE: my_date,
      / my_date2,
      / my_time,
      / my_time2.
uline.
*****
```

```
01012012
00.00.0000
111005
00:00:00
```

Note that in the first row the my_date field has reversed itself to the format DDMMYYYY. In the second, no value was assigned to the field, so the system has output the default zeros. However, as this was defined like the system's date variable, it has included periods in the formatting. This also applies to the my_time2 field, where colons have appeared between the places where the time values would ordinarily be.

Date Fields in Calculations

Some examples of performing calculations with date and time fields will now be looked at. Using these fields in calculations is common practice within programming business systems, as one will often have to, for example, find the difference between two dates to deliver invoice dates, delivery dates and so on. Here, examples will be looked at so as to find new dates, and find the difference between two dates.

Use the DATA statement to declare a start date for an employee, called “empl_sdate”, and then give this a value of ‘20090515’. Then create another field called “todays_date” and define the value of this as ‘sy-datum’, the system variable, which should then include the date on that particular day:

```
DATA empl_sdate TYPE d.
DATA todays_date TYPE d.
```

```
empl_sdate = '20090515'.
todays_date = sy-datum.
```

Next, a calculation will be added, so as to work out this employee’s length of service. Create a new variable named “LOS”, include a DATA statement giving “LOS” a data type ‘i’ and then define LOS as the calculation ‘todays_date – empl_sdate’. Then, add a WRITE statement for this variable, which will include the employee’s length of service in the output. Once this is complete, execute the code:

```
DATA empl_sdate TYPE d.
DATA todays_date TYPE d.
DATA LOS type i.
```

```
empl_sdate = '20090515'.
todays_date = sy-datum.
los = todays_date - empl_sdate.
WRITE / los.
```

```
1,160
```

If one wants to add, for example, 20 days to today’s date, the same value is used for todays_date (the system variable, sy-datum). Create another variable, called “days_count” with an integer value of 20, and another called “fut_date”. This variable’s value should then be defined as ‘todays_date + days_count’, then add a WRITE statement to output the

fut_date. Don't forget also to add the data types above ('i' for days_count and 'd' for fut_date). The output should give the date 20 days on from today's date, which here is the 7th of August, 2012:

```

todays_date = sy-datum.
days_count = 20.
fut_date = todays_date + days_count.
WRITE / fut_date.

```

```

DATA days_count TYPE i.
DATA fut_date TYPE d.
.....

```

```

07082012

```

Subfields can be used for date fields in exactly the same way as they were used before. In the next example, a date field will be changed to represent the 20th day of the current month. Copy the todays_date variable, then add a new line of code which changes the last two figures of todays_date to the value '20', and a WRITE statement. Also, output the system date so as to compare the two:

```

todays_date = sy-datum.
todays_date+6(2) = '20'.
WRITE / sy-datum.
WRITE / todays_date.

```

```

18.07.2012
20072012

```

In this next example, the last day of the previous month will be established. Use the todays_date variable again, this time using the subfield method above to change this to represent the first day of the current month. Then on a new line of code, subtract one from this, so that the todays_date variable is now the final day of the previous month:

```

todays_date = sy-datum.
todays_date+6(2) = '01'.
todays_date = todays_date - 1.
WRITE / todays_date.

```

```

30062012

```

Time Fields in Calculations

Calculations like those above can also be performed with time fields.

In the examples, employees' clocking in and out times will be used. Use DATA statements to declare the variables "clock_in" and "clock_out" as type 't', along with others seen in the image below, which will be used for calculations to work out the differences between times in seconds, minutes and hours, all of an integer type:

```
* Field for Time Calculations
DATA clock_in      TYPE t.
DATA clock_out     TYPE t.
DATA seconds_diff TYPE i.
DATA minutes_diff TYPE i.
DATA hours_diff   TYPE i.
```

Assign values to clock_in and clock_out of '073000' and '160000' respectively. Then, to work out the difference between the two in seconds, use the calculation 'clock_out - clock_in' and assign this value to "seconds_diff". Then include some WRITE statements to output this information:

```
* TIME CALCULATIONS
clock_in = '073000'.
clock_out = '160000'.
seconds_diff = clock_out - clock_in.

WRITE: / 'clock in: ', clock_in, '   clock out: ', clock_out.
WRITE / seconds_diff.
```

```
clock in: 073000   clock out: 160000
        30,600
```

To establish the difference in minutes, simply use the seconds_diff value, and divide this by 60, and then to establish the hour's difference, follow this by dividing minutes_diff by 60:

```
minutes_diff = seconds_diff / 60.
WRITE: / 'difference in minutes: ', minutes_diff.

hours_diff = minutes_diff / 60.
WRITE: / 'difference in hours: ', hours_diff.
```

```
difference in minutes:      510
difference in hourss:      9
```

Note that here, the 510 minutes do not, in fact, equal 9 hours exactly, the system has rounded the number. This is because the `hours_diff` variable was declared as an integer. If the data type for this is changed to a packed decimal, the value would have been established as the more accurate 8.5 hours:

```
DATA minutes_diff TYPE i.
DATA hours_diff   TYPE p decimals 2.
```

```
difference in hours:      8.50
```

Quantity and Currency Fields in Calculations

Now, a look will be taken at using quantity and currency fields in calculations. In ABAP, these are treated the same as packed number fields. Currency fields must be declared as data type ‘p’, bearing in mind how many decimal places are required. This is important, as having the right number of decimal places can have a large impact on the accuracy of calculations.

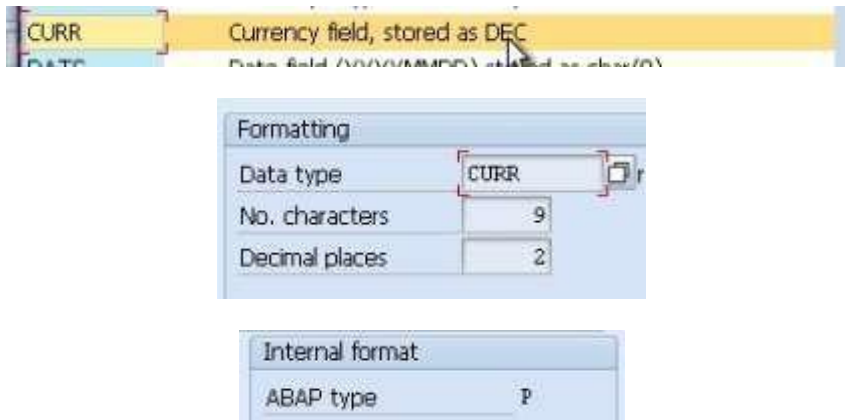
Quite often in a program, one wants to create one’s own variables for quantity and currency fields. It is usually better, however, to associate these fields with the data types of those in a table created in the ABAP dictionary. This is because the ABAP dictionary will already have defined the correct field length and number of decimal places for these. For example, the Salary field in the table created previously had defined two decimal places. If a currency field in a program is declared to match this field but the data type in the program is set manually to 2 decimal places and the number of decimal places in the table was to change, the program would no longer operate properly here. For this reason, it is usually preferable to use the LIKE statement for these fields.

In this example a new variable named “my_salary” has been declared using the LIKE statement:

```
* Field for Currency Calculations
DATA my_salary LIKE zemployees2-salary.
```

Because this field in the program is linked to the field in the table, the system will ensure these data types are kept in sync. There are two aspects to this process, the number of

decimal places, and the associated currency (or quantity) keys. If you look at the CURR data type in the ABAP dictionary, you will see that this is stored as a decimal - 9 characters and 2 decimal places. You can also see that its internal format is ABAP type p, packed decimal:



Additionally, don't forget that the salary field and its currency data type always refer to the currency key field, in the table called ECURRENCY. Ultimately, then, when one is declaring fields in ABAP, it is important to reference these to the associated fields in a table, and when working with currencies, the currency key field will always be there and should be taken into account. The same applies to quantity fields. The only difference is their data type is QUAN, and rather than a currency key, will always have a UNIT associated with them.

Now, using calculations from the currency field, an employee's tax and net pay amounts will be established, so declare two more DATA statements for these fields, again referencing the salary field in the table. Also add a tax percentage variable, of type p with 2 decimals:

```
* Field for Currency Calculations
DATA my_salary LIKE zemployees2-salary.
DATA my_tax_ant LIKE zemployees2-salary.
DATA my_net_pay LIKE zemployees2-salary.
DATA tax_perc TYPE p decimals 2.
```

Add a TABLES statement so that the program knows to refer to the ZEMPLOYEES2 table, then observe the calculations in the code below:


```

REPORT z_other_data_types

TABLES: zemployees2.

tax_perc = 0.20.
SELECT * FROM zemployees2.
WRITE: / zemployees2-surname, zemployees2-salary, zemployees2-ecurrency.
    my_tax_amt = tax_perc * zemployees2-salary.
    my_net_pay = zemployees2-salary - my_tax_amt.
WRITE: / my_tax_amt, zemployees2-ecurrency,
        my_net_pay, zemployees2-ecurrency.
ENDSELECT.

```

First, the tax percentage is established. This is in this example 20%, so for the means of the calculations is written as 0.20. Then the code will select records from the ZEMPLOYEES2 table, and write the surnames, salaries and currencies for these. Next, the tax amount is established, by multiplying the tax percentage by the salary. Net pay is equal to the salary, minus the tax amount. Then add a WRITE statement to output the results the end of the SELECT loop. The output should look like this (where salaries and currencies are not present in the table, go back and edit the records in your table to put some values):

SMITH				1,111.00	ATS
	222.20	ATS	888.80	ATS	
BROWN				2,222.00	BDT
	444.40	BDT	1,777.60	BDT	
WILLIAMS				6,423.00	FJD
	1,284.60	FJD	5,138.40	FJD	
ROSE				12,345.00	USD
	2,469.00	USD	9,876.00	USD	
GREEN				2,452.00	HUF
	490.40	HUF	1,961.60	HUF	
QWE				1,234.00	GBP
	246.80	GBP	987.20	GBP	

The surname, salary and currency for each record are written on the first line, followed by the tax amount and net pay on the following line. To make this look tidier, descriptive text can be added to the WRITE statements in the code:

SMITH			1,111.00	ATS
tax amount:	222.20	ATS	net amount:	888.80
BROWN			2,222.00	BDT
tax amount:	444.40	BDT	net amount:	1,777.60
WILLIAMS			6,423.00	FJD
tax amount:	1,284.60	FJD	net amount:	5,138.40
ROSE			12,345.00	USD
tax amount:	2,469.00	USD	net amount:	9,876.00
GREEN			2,452.00	HUF
tax amount:	490.40	HUF	net amount:	1,961.60
QWE			1,234.00	GBP
tax amount:	246.80	GBP	net amount:	987.20

