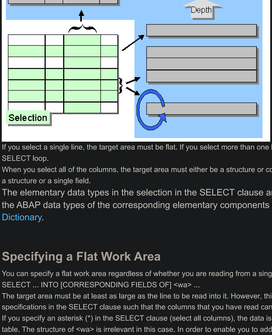
Specifying a Target Area

The INTO clause defines the target area into which the selection from the SELECT clause is written. Suitable target areas are variables whose data type is compatible with or convertible into that of the selection in the SELECT clause.
The SELECT clause determines the data type of the target area as follows:

- The <lines> specification in the SELECT clause determines the depth of the target area, that is, whether it is a flat or a tabular structure.
- The <cols> specification in the SELECT clause determines the structure (line type) of the target area.



If you select a single line, the target area must be flat. If you select more than one line, the target area may be either tabular or flat. If the target area is flat, you need to use a SELECT loop.
When you select all of the columns, the target area must either be a structure or convertible into one. When you select individual columns, the target area can be a component of a structure or a single field.
The elementary data types in the selection in the SELECT clause are Dictionary types. These Dictionary types must be able to be converted into the ABAP data types of the corresponding elementary components of the target area. For a table of data types, refer to Data Types in the ABAP Dictionary.

Specifying a Flat Work Area

You can specify a flat work area regardless of whether you are reading from a single line or from several. To read data into one, use the following in the INTO clause:
SELECT ... INTO [CORRESPONDING FIELDS OF] <wa> ...
The target area must be at least as large as the line to be read into it. However, this is the only restriction on its data type. You should base your choice of data type on the specifications in the SELECT clause such that the columns that you have read can be addressed in <wa>.
If you specify an asterisk (*) in the SELECT clause (select all columns), the data is transferred into the work area from left to right according to the structure of the database table. The structure of <wa> is irrelevant in this case. In order to enable you to address the values in the individual columns after the SELECT statement, the work area should have the same structure as the database table.
If you specified individual columns or aggregate expressions in the SELECT clause, the columns are transferred into the work area from left to right according to the structure of the work area.
When you read the data into <wa>, its previous contents are overwritten. However, the components of <wa> that are not affected by the SELECT statement retain their previous values.
If the work area is structured, you can use the CORRESPONDING FIELDS addition. This transfers only the contents of fields whose names are identical in the database table and the work area. This includes any alias column names that you specified in the selection. Working with the CORRESPONDING FIELDS option of the INTO clause does not limit the amount of data read from the database, but only the amount of data that is read from the resulting set into the ABAP program. The only way of restricting the number of columns read is in the SELECT clause.
If you use the asterisk (*) option to read all of the columns of a single database table <dbtab> in the SELECT clause, the INTO clause may be empty. The SELECT statement then writes the data by default into the table work area with the same name <dbtab> as the database table itself. You must declare this table work area using the TABLES statement. Before Release 4.0, this was necessary before you could read data from a database table at all, and was frequently used as an implicit work area. Nowadays, table work areas are still useful as interface work areas, but should no longer be used as the work area in the SELECT statement. Like internal tables without header lines, having different names for the source and target areas makes programs clearer.

Specifying Internal Tables

When you read several lines of a database table, you can place them in an internal table. To do this, use the following in the INTO clause:
SELECT ... INTO|APPENDING [CORRESPONDING FIELDS OF] TABLE <itab>
[PACKAGE SIZE <n>] ...
The same applies to the line type of <itab>, the way in which the data for a line of the database table are assigned to a table line, and the CORRESPONDING FIELDS addition as for flat work areas (see above).
The internal table is filled with all of the lines of the selection. When you use INTO, all existing lines in the table are deleted. When you use APPENDING; the new lines are added to the existing internal table <itab>. With APPENDING, the system adds the lines to the internal table appropriately for the table type. Fields in the internal table not affected by the selection are filled with initial values.
If you use the PACKAGE SIZE addition, the lines of the selection are not written into the internal table at once, but in packets. You can define packets of <n> lines that are written one after the other into the internal table. If you use INTO, each packet replaces the preceding one. If you use APPENDING, the packets are inserted one after the other. This is only possible in a loop that ends with ENDSELECT. Outside the SELECT loop, the contents of the internal table are undetermined. You must process the selected lines within the loop.

Specifying Single Fields

If you specify single columns of the database table or aggregate expressions in the SELECT clause, you can read the data into single fields for a single entry or for multiple entries in a SELECT loop. To read data into single fields, use the following in the INTO clause:
SELECT ... INTO (<f1>, <f2>, ...) ...
You must specify as many individual fields <f.> as are specified in the field list of the SELECT clause. The fields in the SELECT clause are assigned, from left to right, to the fields in the list in the INTO clause.
You do not have to use a field list in the INTO clause when you specify individual fields in the SELECT clause - you can also use the CORRESPONDING FIELDS addition to read the data into a flat work area or an internal table instead. In that case, the target area does not need to contain the same number of elements as the list in the SELECT clause.

Examples

Flat structure as target area

```
DATA WA TYPE SPFLI.
SELECT *
INTO   WA
FROM   SPFLI.
  WRITE: / WA-CARRID ...
ENDSELECT.
```

This example uses a flat structure with the same data type as the database table SPFLI as the target area in a SELECT loop. Within the loop, it is possible to address the contents of the individual columns.

```
DATA SPFLI TYPE SPFLI.
SELECT *
FROM   SPFLI.
  WRITE: / SPFLI-CARRID ...
ENDSELECT.
```

This example declares a structure SPFLI with the same name as the database table you want to read. This structure is used implicitly as the target area in the SELECT loop. Since the names are the same, it is possible to overlook the fact that you are working with an ABAP data object here and not the database table itself.

Internal table as target area

```
DATA: BEGIN OF WA,
        CARRID   TYPE SPFLI-CARRID,
        CONNID   TYPE SPFLI-CONNID,
        CITYFROM TYPE SPFLI-CITYFROM,
        CITYTO   TYPE SPFLI-CITYTO,
      END OF WA,
      ITAB LIKE SORTED TABLE OF WA
           WITH NON-UNIQUE KEY CITYFROM CITYTO.
SELECT CARRID CONNID CITYFROM CITYTO
INTO   CORRESPONDING FIELDS OF TABLE ITAB
FROM   SPFLI.
IF SY-SUBRC EQ 0.
  WRITE: / SY-DBCNT, 'Connections'.
  SKIP.
  LOOP AT ITAB INTO WA.
    WRITE: / WA-CARRID, WA-CONNID, WA-CITYFROM, WA-CITYTO.
  ENDLOOP.
ENDIF.
```

The output is:

```
        10  Connections

LH  2402 FRANKFURT            BERLIN
LH  0400 FRANKFURT            NEW YORK
LH  0402 FRANKFURT            NEW YORK
UA  0941 FRANKFURT            SAN FRANCISCO
AA  0017 NEW YORK             SAN FRANCISCO
AZ  0555 ROME                 FRANKFURT
QF  0805 SINGAPORE            FRANKFURT
SQ  0866 SINGAPORE            HONGKONG
SQ  0802 SINGAPORE            SAN FRANCISCO
AZ  0789 TOKYO                ROME
```

The target area is a sorted table ITAB containing four fields with the same names and data types as the database table SPFLI. The program uses the CORRESPONDING FIELDS addition to place the columns from the SELECT clause into the corresponding fields of the internal table. Because ITAB is a sorted table, the data is inserted into the table sorted by the table key of ITAB.

Reading packets into an internal table

```
DATA: WA   TYPE SPFLI,
      ITAB TYPE SORTED TABLE OF SPFLI
           WITH UNIQUE KEY CARRID CONNID.
SELECT CARRID CONNID
FROM   SPFLI
INTO   CORRESPONDING FIELDS OF TABLE ITAB
       PACKAGE SIZE 3.
  LOOP AT ITAB INTO WA.
    WRITE: / WA-CARRID, WA-CONNID.
  ENDLOOP.
  SKIP 1.
ENDSELECT.
```

The output is:

```
AA  0017
AZ  0555
AZ  0789

LH  0400
LH  0402
LH  2402

QF  0805
SQ  0802
SQ  0866

UA  0941
```

The example reads packets of three lines each into the sorted table ITAB. In each pass of the SELECT loop, the internal table has a different sorted content.
If you were to use APPENDING instead of INTO, the list would look like this:

```
AA  0017
AZ  0555
AZ  0789

AA  0017
AZ  0555
AZ  0789
LH  0400
LH  0402
LH  2402

AA  0017
AZ  0555
AZ  0789
LH  0400
LH  0402
LH  2402
QF  0805
SQ  0802
SQ  0866

AA  0017
AZ  0555
AZ  0789
LH  0400
LH  0402
LH  2402
QF  0805
SQ  0802
SQ  0866
UA  0941
```

In each loop pass, a new packet is sorted into the internal table.

Single fields as target area:

```
DATA:  AVERAGE TYPE P DECIMALS 2,
       SUM     TYPE P DECIMALS 2.
SELECT AVG( LUGGWEIGHT ) SUM( LUGGWEIGHT )
INTO   (AVERAGE, SUM)
FROM   SBOOK.
WRITE: / 'Average:', AVERAGE,
       / 'Sum    :', SUM.
```

The output is:

```
Average:              4,88
Sum    :          11.778,70
```

The SELECT clause contains two aggregate expressions for calculating the average and sum of the field LUGGWEIGHT from database table SBOOK. The target fields are called AVERAGE and SUM.

Using aliases:

```
DATA: BEGIN OF LUGGAGE,
        AVERAGE TYPE P DECIMALS 2,
        SUM     TYPE P DECIMALS 2,
      END OF LUGGAGE.
SELECT AVG( LUGGWEIGHT ) AS AVERAGE SUM( LUGGWEIGHT ) AS SUM
INTO   CORRESPONDING FIELDS OF LUGGAGE
FROM   SBOOK.
WRITE: / 'Average:', LUGGAGE-AVERAGE,
       / 'Sum    :', LUGGAGE-SUM.
```

The output is:

```
Average:              4,88
Sum    :          11.778,70
```

This example has the same effect as the previous one. The only difference is that a structure is used as the target area instead of individual fields, and that the names of the structure components are used as aliases in the SELECT clause.