# BC - ABAP Dictionary

HELP.BCDWBDIC

**Release 4.6B**

# Copyright

# Icons

| Icon | Meaning |
| --- | --- |
|  | Caution |
|  | Example |
|  | Note |
|  | Recommendation |
|  | Syntax |
|  | Tip |

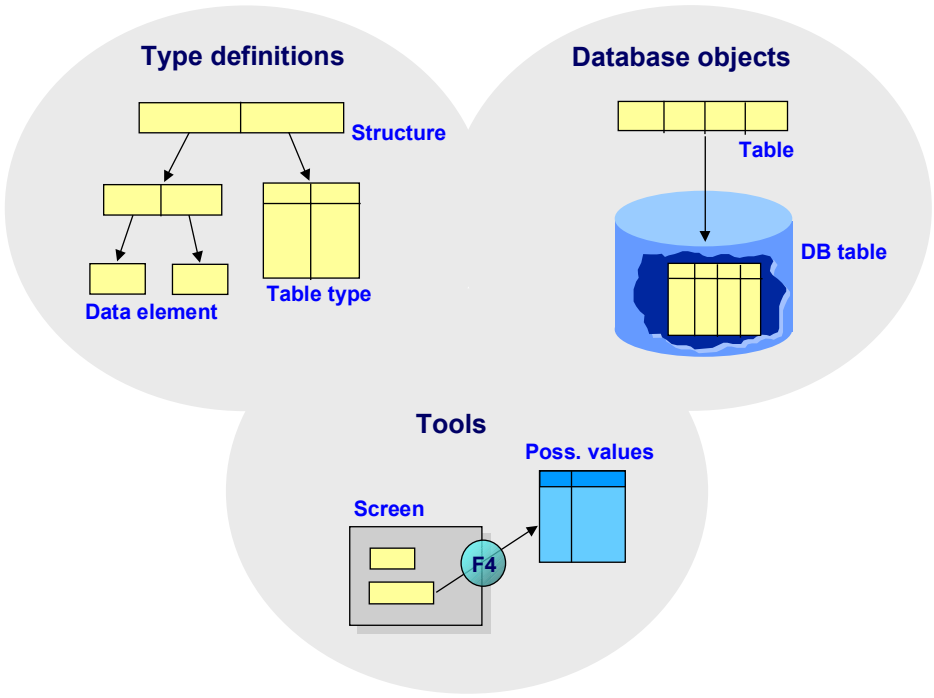
# Contents

BC - ABAP Dictionary

# BC - ABAP Dictionary

The ABAP Dictionary centrally describes and manages all the data definitions used in the system. The ABAP Dictionary is completely integrated in the ABAP Development Workbench. All the other components of the Workbench can actively access the definitions stored in the ABAP Dictionary.

The ABAP Dictionary supports the definition of user-defined types (data elements, structures and table types). You can also define the structure of database objects (tables, indexes and views) in the ABAP Dictionary. These objects can then be automatically created in the database with this definition. The ABAP Dictionary also provides tools for editing screen fields, for example for assigning a field an input help (F4 help).




The most important object types in the ABAP Dictionary are tables, views, types (data elements, structures, table types), domains, search helps and lock objects.

# ABAP Dictionary

## Purpose

Data definitions (metadata) are created and managed in the ABAP Dictionary. The ABAP Dictionary permits a central description of all the data used in the system without redundancies. New or modified information is automatically provided for all the system components. This ensures data integrity, data consistency and data security.

You can create the corresponding objects (tables or views) in the underlying relational database using these data definitions. The ABAP Dictionary therefore describes the logical structure of the objects used in application development and shows how they are mapped to the underlying relational database in tables or views.

The ABAP Dictionary also provides standard functions for editing fields on the screen, for example for assigning a screen field an input help.

## What Information is Stored in the ABAP Dictionary?

The most important object types in the ABAP Dictionary are tables, views, types, domains, search helps and lock objects.

Tables [Page 13] are defined in the ABAP Dictionary independently of the database. A table having the same structure is then created from this table definition in the underlying database.

Views [Page 95] are logical views on more than one table. The structure of the view is defined in the ABAP Dictionary. A view on the database can then be created from this structure.

Types [Page 130] are used in ABAP program. The structure of a type can be defined globally in ABAP programs. Changes to a type automatically take effect in all the programs using the type.

Lock objects [Page 202] are used to synchronize access to the same data by more than one user. Function modules that can be used in application programs are generated from the definition of a lock object in the ABAP Dictionary.

Different fields having the same technical type can be combined in domains [Page 155]. A domain defines the value range of all table fields and structure components that refer to this domain.

The ABAP Dictionary also contains the information displayed with the F1 and F4 help for a field in an input template. The documentation about the field is created for a data element [Page 132] that describes the meaning of the contents of a table field. The list of possible input values that appears for the input help is created by a foreign key [Page 19] or a search help [Page 166].

## Integration in the ABAP Development Workbench

The ABAP Dictionary is completely integrated in the ABAP Development Workbench. The R/3 System works interpretatively, permitting the ABAP Dictionary to be actively integrated in the development environment. Instead of the original objects, the interpreters see only internal representations of these objects.

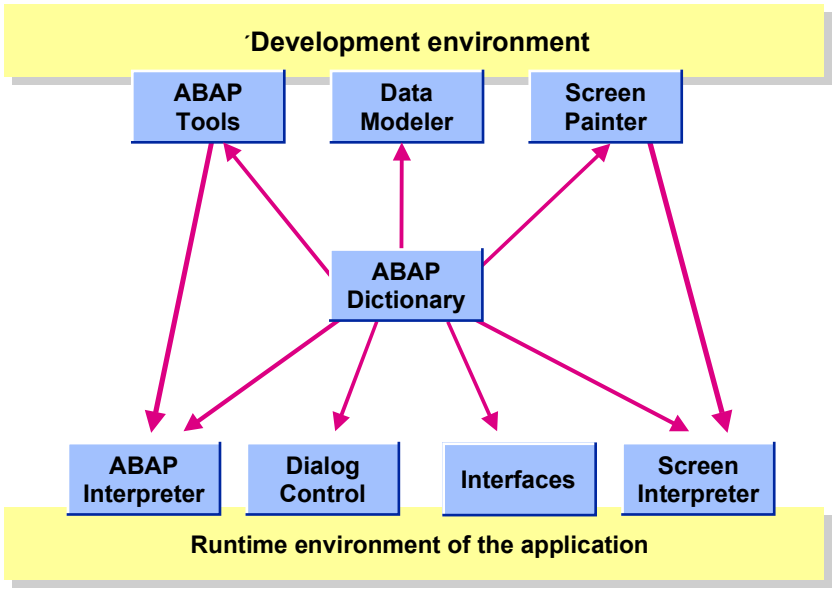
These internal representations are adjusted automatically when the system finds that changes have been made in the ABAP Dictionary. This ensures that the screen and ABAP interpreters, input help, database interface, and development tools always access current data.

The following ABAP program lists the airline carriers (see Flight model [Page 295]) and carrier IDs contained in table SCARR.

```
DATA: SCARR_TAB TYPE SCARR.
SELECT * INTO SCARR_TAB FROM SCARR.
    WRITE: / SCARR_TAB-CARRID, SCARR_TAB-CARRNAME.
ENDSELECT.
```

Only structure SCARR_TAB is declared in the program. All the information about this structure, such as the field names, data types and field lengths, are copied from table SCARR, which is defined in the ABAP Dictionary. This information about table SCARR is called from the ABAP Dictionary when the program is generated.

This means that the source text of the program need not be adjusted when a change is made to table SCARR, for example when the length of a table field is changed. The next time the program is called, the system automatically determines that the structure of table SCARR has changed. The program is simply regenerated, thereby retrieving up-to-date information about table SCARR from the ABAP Dictionary.




When you work on development projects, objects of the ABAP Dictionary can be changed any number of times before being activated [Page 230] and made available to the operative components of the system. Objects can have both an active and an inactive version in the ABAP Dictionary at the same time.

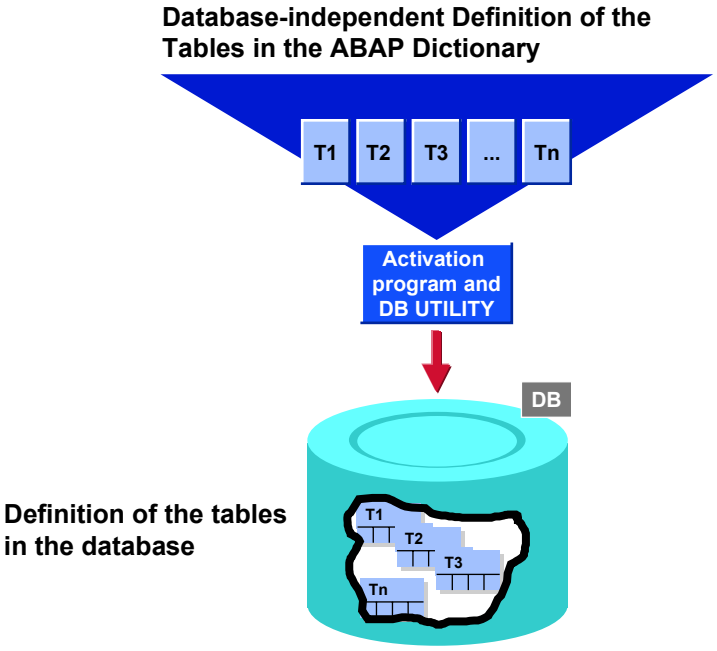
Inactive ABAP Dictionary objects have no effect on the runtime system (ABAP processor, database interface). This permits greater changes to several objects without impairing the

executability of the system. The objects can only be activated together when they have all been changed.

# Tables

Tables can be defined independently of the database in the ABAP Dictionary. The fields of the table are defined with their (database-independent) data types and lengths.

When the table is activated, a physical table definition is created in the database for the table definition stored in the ABAP Dictionary. The table definition is translated from the ABAP Dictionary to a definition of the particular database.




A table definition in the ABAP Dictionary contains the following components:

- Table fields [Page 14] define the field names and data types of the fields contained in the table
- Foreign keys [Page 19] define the relationships between the table and other tables.
- Technical settings [Page 30] control how the table should be created in the database.
- Indexes [Page 61]: To speed up data selection, secondary indexes can be created for the table

The customer can modify SAP tables with append structures [Page 69] and customizing includes [Page 68]. This kind of modification ensures that the customer enhancements are automatically merged with the new versions of the SAP tables when there is a release upgrade.

**See also:**

Creating Tables [Page 71]

Making Changes to Tables [Page 81]

# Table Fields

You must define the following for a table field in the ABAP Dictionary:

- **Field name:** The field name can have a maximum of 16 places and may contain letters, digits and underscores. The field name must begin with a letter.
- **Key flag:** determines whether the field should belong to the table key.
- **Field type:** data type of the field in the ABAP Dictionary.
- **Field length:** number of valid places in the field.
- **Decimal places:** number of places after the decimal point, specifying numeric data types.
- **Short text:** short text describing the meaning of the field.

You can also include [Page 16] the fields of a structure in the table.

## Assignment of the Data Type, Field Length and Short Text

You can assign the data type [Page 235], length and short text in different ways:

- You directly assign the field a data type, field length (and if necessary decimal places) and short text in the table definition.
- You can assign the field a data element [Page 132]. The data type, field length (and decimal places) are determined from the domain of the data element. The short description of the data element is assigned to the field as a short text.

## Other Assignment Options

- **Check table**: An input check for the field can be defined with a foreign key [Page 19]. This input check appears on all the screens in which the field is used.
- **Search help assignment**: A search help [Page 166] can be assigned to a field. This search help defines the input help flow on all the screens in which the field is used.
- Reference field and reference table [Page 15]: You must specify the table field in which the corresponding unit of measure or currency can be found for fields containing quantities (data type QUAN) or currency amounts (data type CURR).

**See also:**

Creating Tables [Page 71]

# Reference Fields and Reference Tables

You must specify a **reference table** for fields containing quantities (data type QUAN) or currency amounts (data type CURR).

This reference table must contain a field with the format for the currency key (data type CUKY) or unit of measure (data type UNIT). This field is called the **reference field** of the output field. The reference field can also reside in the table itself.

A field is only assigned to the reference field at program runtime. For example, if a field is filled with currency amounts, the corresponding currency is determined from the assigned reference field, that is the value entered in this field at the moment defines the currency.

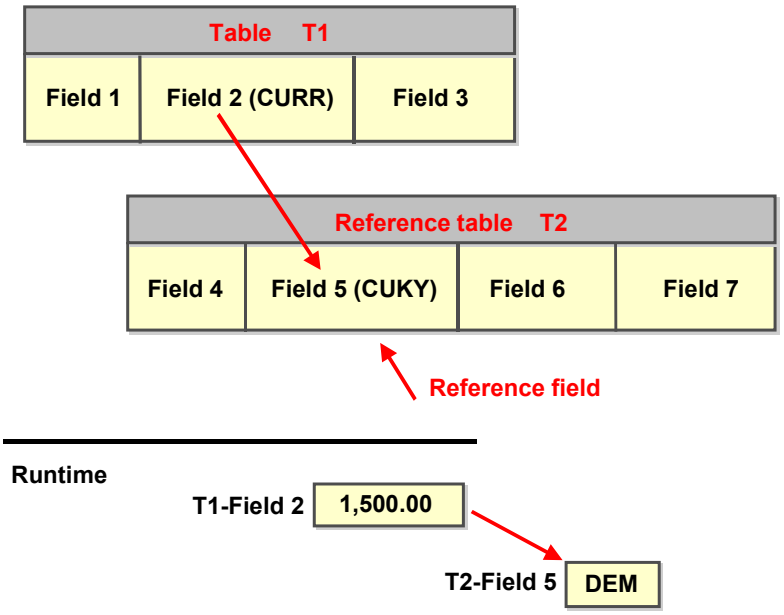
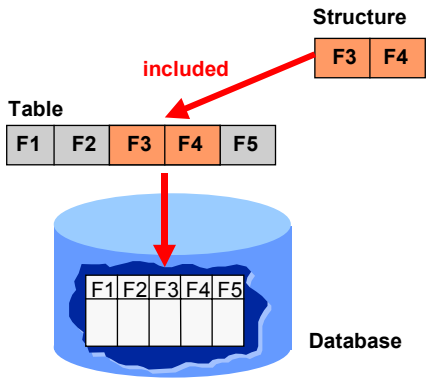



Table SBOOK in the flight model [Page 295] contains all the flight bookings made by customers. Field FORCURAM contains the price of the booking in the customer's currency. Field FORCURKEY of table SBOOK contains the corresponding currency key for this price. SBOOK is therefore the reference table for field FORCURAM and FORCURKEY is the reference field for field FORCURAM.

# Includes

In addition to listing the individual fields, you can also include the fields of another structure in tables [Page 13] and structures [Page 138]. Individual fields and includes can be mixed as required.

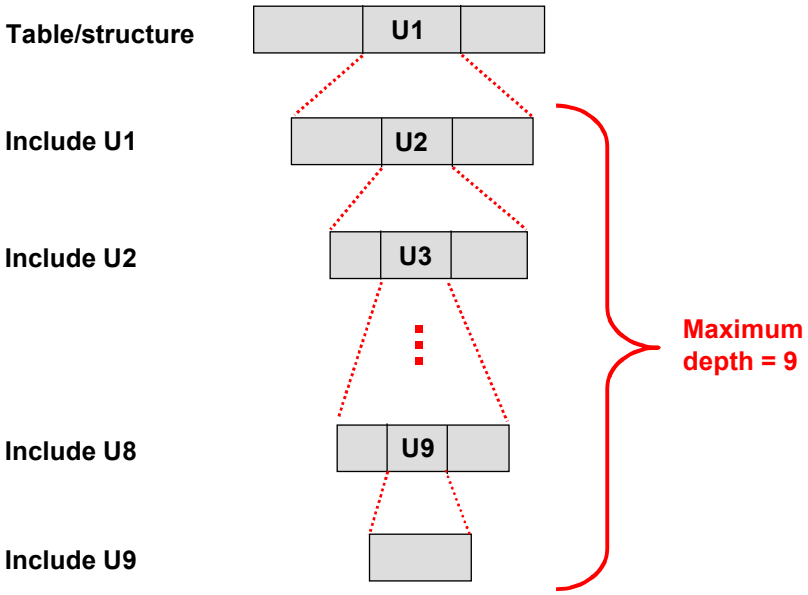



When an include is changed, all the tables and structures that include it are automatically adjusted.

Structure A was included in table B. A new field is inserted in structure A. When structure A is activated, table B is adjusted to this change, that is the new field is also inserted there.

You can assign the include a group name [Page 142] with which the group of fields in the include can be addressed as a whole in ABAP programs.

Includes can also be nested, that is structure A includes structure B which in turn includes another structure C, etc. The maximum nesting depth is limited to nine. The maximum length of a path of nested includes in a table or structure is therefore nine (the table/structure itself not included).




Only flat structures [Page 138] can be included. In a flat structure, every field either refers to a data element or is directly assigned a data type, length and possibly decimal places.

Only structures may be included in a table. Tables, structures and views may be included in a structure. However, the path of nested includes may only contain one table.

Table TAB1 includes structure STRUCT1, which in turn includes structure STRUCT2. The path of the nested includes here only contains table TAB1. It is also possible to include TAB1 in a further structure STRUCT0,

but no other table TAB2 may be included in TAB1 since in this case a path of nested includes would contain two tables (TAB1 and TAB2).

**See also:**

Inserting an Include [Page 83]

# Named Includes

If an include [Page 16] is used to define a database table or structure, a name can be assigned to the included substructure. The group of fields in the include can be addressed as a whole in ABAP programs with this name.

In ABAP programs, you can either access the fields directly with *<Table/structure name>-<Field name>* or analogously with *<Table/structure name>-<Group name>-<Field name>*. You can access the fields of the group as a whole with *<Table/structure name>-<Group name>*.

Structure PERSON includes structure ADDRESS with the name ADR. ADDRESS has a field CITY. With PERSON-ADR you can address all the fields in structure ADDRESS. The included field CITY can also be addressed with PERSON-CITY or PERSON-ADR-CITY.

You can include a structure more than once (e.g. in a period group). Since direct access by field name should be permitted here, the included field names must be renamed to ensure that they are unique.

A suffix can be assigned to each group, extending the names of the group fields. The fields can then be addressed in ABAP programs with *<Table/structure name>-<Field name (with suffix)>* or *<Table/structure name>-<Group name>-<Field name (with suffix)>*.

Structure PERSON includes structure ADDRESS twice. An address is the private address with suffix H and name ADRH. The other address is the business address with suffix W and name ADRW. You can access field CITY in the private address with PERSON-CITYH or PERSON-ADRH-CITY.

The functionality of the named includes in the ABAP Dictionary corresponds to the ABAP construction INCLUDE TYPE ... AS ... RENAMING ... .

# Foreign Keys

You can define the relationships between tables in the ABAP Dictionary by creating foreign keys.

Using foreign keys, you can easily create value checks for input fields. Foreign keys can also be used to link several tables in a view [Page 95] or a lock object [Page 202].

## Field Assignment in the Foreign Key

A foreign key links two tables T1 and T2 by assigning fields of table T1 to the primary key fields of table T2.

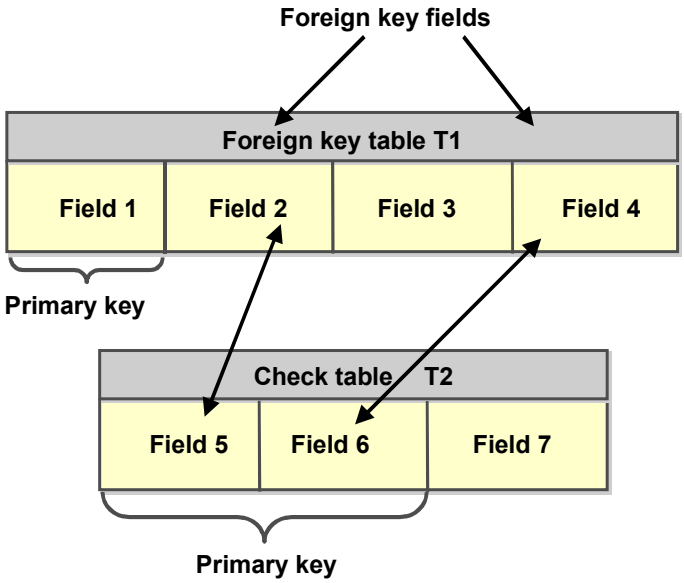



Table T1 is called the **foreign key table** (dependent table) and table T2 the **check table** (referenced table). The pair of fields for the two tables must have the same data type and length. One field of the foreign key table therefore corresponds to each key field of the check table. This field is called the **foreign key field**.

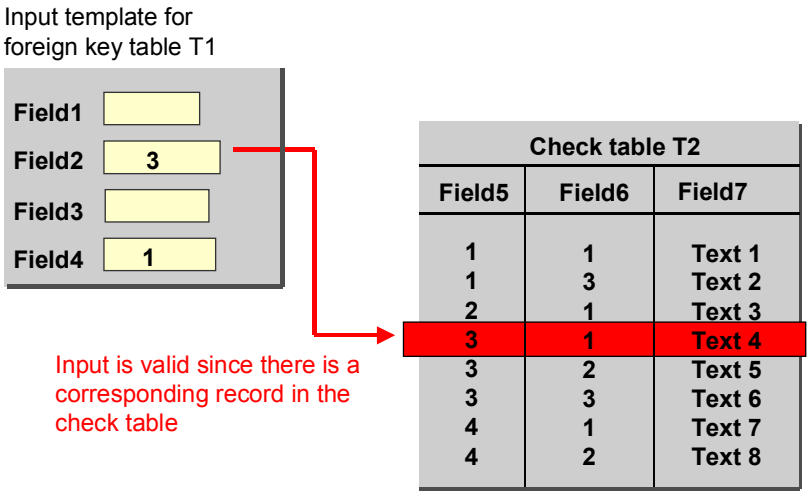
A foreign key permits you to assign data records in the foreign key table and check table. One record of the foreign key table uniquely identifies one record of the check table using the entries in the foreign key fields.

## Check Field and Value Check

One of the foreign key fields is marked as the **check field**. This means that the foreign key relationship is maintained for this field.

**Foreign Keys**

When an entry is made in the check field, there is a check whether the check table contains a record with the key defined by the values in the foreign key fields. If this is so, the entry is valid. Otherwise the system rejects the entry.




In this example the entry *Field2* = 2 and *Field4* = 2 would be rejected since T2 does not contain a record with the key *Field5* = 2 and *Field6* = 2.

If you do not want to check against all the key fields of the check table, you can exclude fields of the foreign key table from the assignment of the fields to the check table with generic and constant foreign keys [Page 22].

## How the Input Check Works

A SELECT statement is generated from the definition of the foreign key. If an entry is made in the check field, this SELECT statement is submitted. If a suitable record of the check table is found, the entry is valid. Otherwise the entry is rejected.

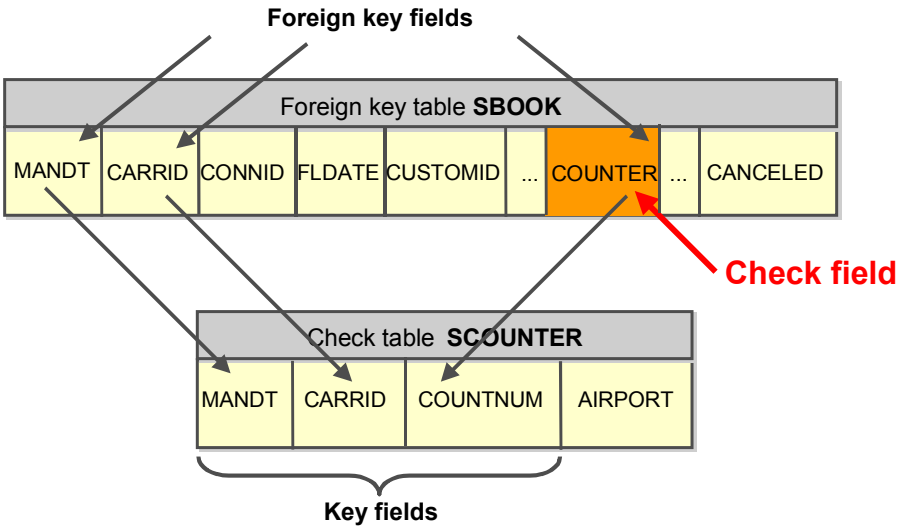
The corresponding SELECT statement has the following form for the foreign key table shown in the above graphic:

SELECT * FROM T2 WHERE T2-FIELD5 = T1-FIELD2 AND T2-FIELD6 = T1-FIELD4.

A screen entry for check field *Field2* is therefore only valid if the check table contains a record with the entries made in the screen for *Field2* and *Field4* as key.

Table SBOOK in the flight model [Page 295] contains the customer's flight bookings for a carrier. The flight bookings can be made by a travel agency or directly at the carrier's sales counter. If the booking was made at a counter, its number is stored together with the booking in field COUNTER in table SBOOK.

You must make sure that only correct counter numbers can be entered. All the counters are entered in table SCOUNTER. The necessary value check can be defined by creating a foreign key for check field COUNTNUM.




**See also:**

Multi-Structured Foreign Keys [Page 29]

Semantic Attributes of Foreign Keys [Page 24]

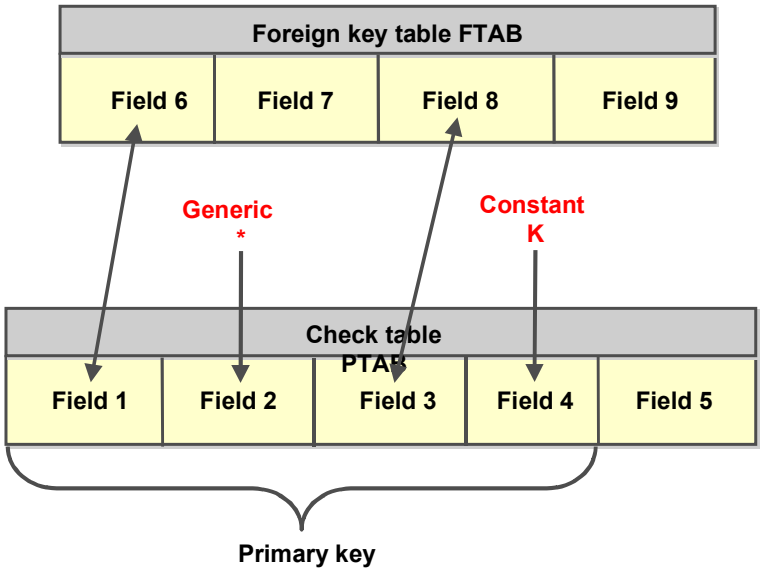
Creating Foreign Keys [Page 74]

# Generic and Constant Foreign Keys

It is not always advisable to check a foreign key against all the key fields of the check table. This is true for example for time-dependent check tables and for check tables whose version number is a component of the key.

You can use generic foreign keys in these cases. Fields are excluded from the assignment to the key fields of the check table here. The check is only against the remaining key fields.

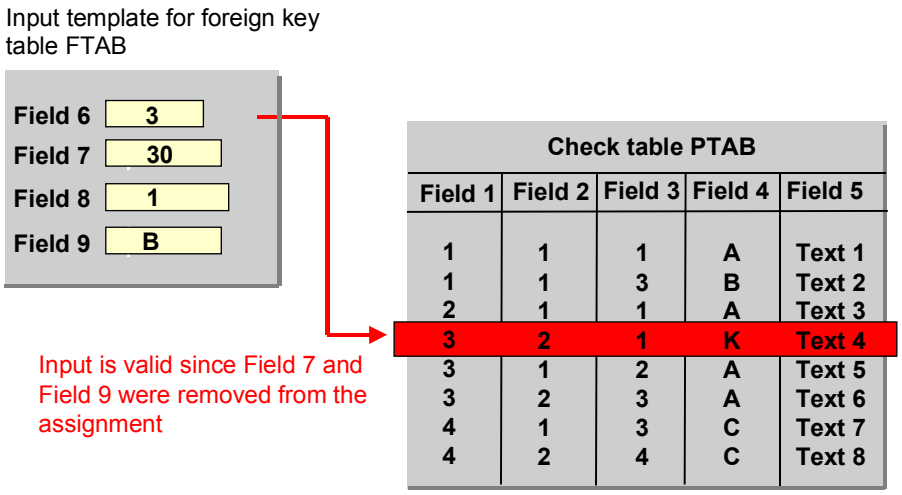
You can also assign a constant value to a key field of the check table. In this case you only have to check against the specified constant. You can use this check if only records of the check table which contain a constant value in this key field are valid.




The corresponding SELECT statement for the screen check has the following form for the foreign key definition in the graphic:

SELECT * FROM PTAB WHERE PTAB-FIELD1 = FTAB-FIELD6 AND PTAB-FIELD3 = FTAB-FIELD8 AND PTAB-FIELD4 = ‘K’.

An entry is only valid in check field *Field6* if a record of check table *PTAB* exists containing the input value for *Field6* in *PTAB-Field1*, the input value for *Field8* in *PTAB-Field3* and constant *K* in *PTAB-Field4*.




The values entered on the screen for *Field7* and *Field9* are meaningless when checking against the check table. An entry with *Field6* = 1, *Field8* = 3 and *Field9* = *B* would not be valid in this case since there is no record with *PTAB-Field1* = 1, *PTAB-Field3* = 3 and *PTAB-Field4* = *K* in the check table!

# Semantic Attributes of Foreign Keys

A foreign key describes a relationship between two tables. You can define this relationship more precisely by specifying the cardinality [Page 25] and type of foreign key fields [Page 26].

This information is **optional** and is **primarily for documentary purposes**. In particular, the definitions of the cardinality and type of the foreign key fields are **not** used in the value check for the foreign key.

The definition of the semantic attributes is only used in the following cases:

- If *Key fields of a text table* is selected as the type of the foreign key fields, the foreign key table is considered to be the text table [Page 27] for the check table. If a screen field is checked against a table, the key entries of the check table are normally displayed in the input help (F4 help) for this field. If there is a text table for the check table, each key entry displayed is enhanced with an explanatory text (contents of the first character-like field of the text table) in the user's logon language.
- Tables can only be included in a help view [Page 111] or maintenance view [Page 112] if they are linked with a foreign key. It only makes sense to create such a help or maintenance view if for each record in the primary table of the view there is no more than one corresponding record in each secondary table of the view. The system therefore checks if the foreign key with which the tables were linked in the view have suitable cardinalities when it creates a maintenance or help view. See also Restrictions for Maintenance and Help Views [Page 114].

  The foreign key between tables SBOOK and SCOUNTER ensures that only existing counters can be entered in field COUNTER (counter at which the flight was booked). See the example in Foreign Keys [Page 19] .

  A booking can be made at either a travel agency or at the carrier's sales counter. If the booking is made at a travel agency, the field COUNTER of table SBOOK remains empty. The foreign key fields do not have to be filled, that is the left side of the cardinality is *C*. Any number of bookings may be made at each counter. There may therefore be any number of entries (bookings) in foreign key table SBOOK for each record of the check table SCOUNTER. The right side of the cardinality is therefore *CN*.

  Of course several bookings can be made for the same carrier at a counter. These bookings do not differ in their foreign key fields (MANDT, CARRID, COUNTER). The entries in the foreign key fields therefore do not uniquely identify an entry in the foreign key table SBOOK (a booking). The foreign key fields therefore have the type *No key fields/candidates*.

# Cardinality

The cardinality (n:m) describes the foreign key relationship with regard to the number of possible dependent records (records of the foreign key table) or referenced records (records of the check table).

The left side (n) of the cardinality is defined as follows:

- *n=1:* There is exactly one record assigned to the check table for each record of the foreign key table.
- *n=C:* The foreign key table may contain records which do not correspond to any record of the check table because the foreign key field is empty. This can occur for example if the field of the foreign key table is optional, in which case it does not have to be filled.

The right side (m) of the cardinality is defined as follows:

- *m=1:* There is exactly one dependent record for each record of the check table.
- *m=C:* There is at most one dependent record for each record of the check table.
- *m=N:* There is at least one dependent record for each record of the check table.
- *m=CN:* There may be any number of dependent records for each record of the check table.

# Type of Foreign Key Fields

The *Type of foreign key fields* describes what the foreign key fields in the foreign key table mean.

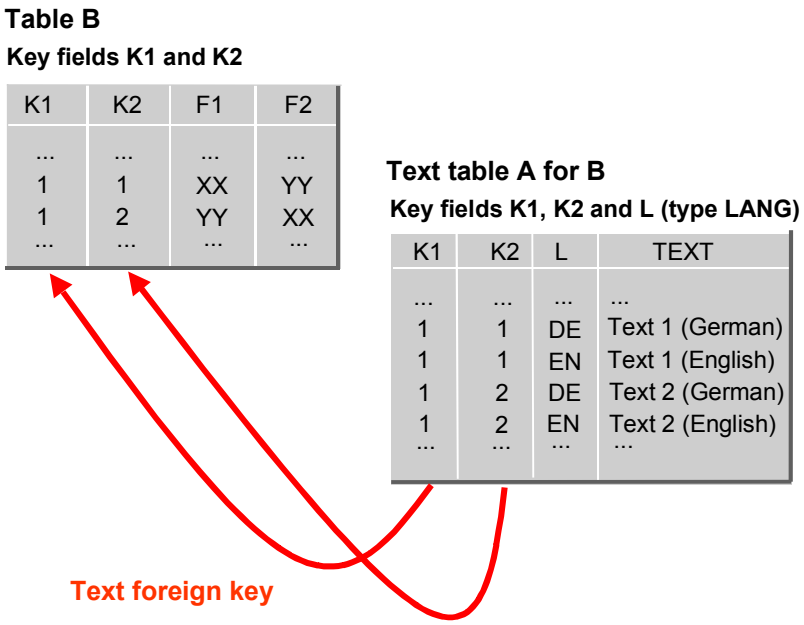
The following types of foreign key field can be defined:

- *No key fields/candidates:* The foreign key fields are neither primary key fields of the foreign key table nor do they uniquely identify a record of the foreign key table (key candidates). For this reason, the foreign key fields do not (partially) identify the foreign key table.
- *Key fields/candidates:* The foreign key fields are either primary key fields of the foreign key table or they already uniquely identify a record of the foreign key table (key candidates). The foreign key fields therefore (partially) identify the foreign key table.
- *Key fields of a text table:* The foreign key table is a text table [Page 27] for the check table, that is the key of the foreign key table only differs from the key of the check table in that it has an additional language key field. This is a special case of the type *Key fields/candidates*.

# Text Tables

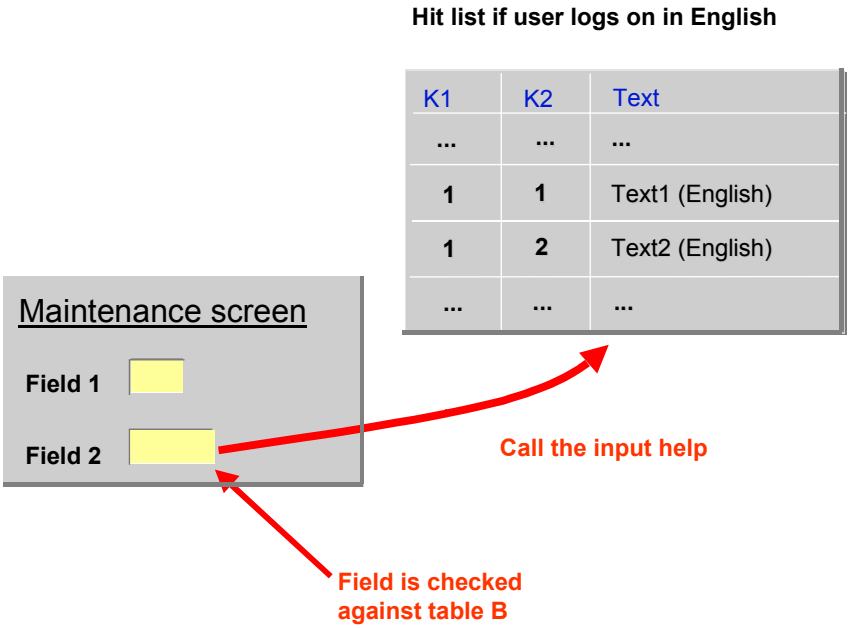
Table A is a text table of table B if the key of A comprises the key of B and an additional language key field (field of data type LANG). Table A may therefore contain explanatory text in several languages for each key entry of B.

To link the key entries with the text, text table A must be linked with table B using a foreign key. *Key fields of a text table* must be selected here for the type of foreign key fields (see Semantic Attributes of Foreign Keys [Page 24]).




If table B is the check table of a field, the existing key entries of table B are displayed as possible input values when the input help (F4) is pressed. The explanatory text (contents of the first character-like non-key-field of text table A) is also displayed in the user's logon language for each key value in table B.

## Text Tables

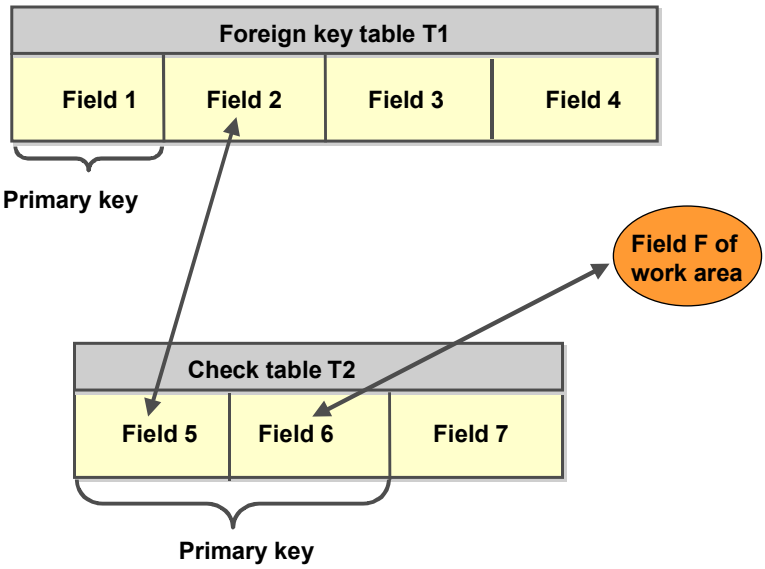



Only one text table can be created for table B! The system checks this when you attempt to activate a table with text foreign keys for B.

# Multi-Structured Foreign Keys

When you define a foreign key, a field of the work area that is not contained in the foreign key table can also be assigned to a check table (for example a field of another table). This is possible for all fields except for the check field.

Table T2 is the check table of foreign key table T1. Field *F* of the work area is assigned to key field *Field6* of check table T2.




The corresponding SELECT statement for the input check is then:

SELECT * FROM T2 WHERE T2-FIELD5 = T1-FIELD2 AND T2-FIELD6 = F.

If an entry is made in field *T1-Field2* (check field), this SELECT statement will be submitted. If a corresponding record is found, the entry is valid; otherwise it is rejected.

If a field that is not contained in the foreign key table is assigned to a field of the check table, this field must be filled at the time of the input check. Otherwise the check always fails, and no values can be entered in the check field.

# Technical Settings

The technical settings of a table define how the table will be handled when it is created in the database, that is whether the table will be buffered and whether changes to data records of the table will be logged.

The most important parameters are:

- *Data class*: The data class [Page 31] defines the physical area of the database (tablespace) in which the table should be created.
- *Size category*: The size category [Page 32] defines the size of the extents created for the table.

When the table is created in the database, the required information about the memory area to be selected and the extent size is determined from the technical settings.

- *Buffering permission*: The buffering permission [Page 33] defines whether the table may be buffered.
- *Buffering type*: If the table may be buffered, you must define a buffering type (full, single-record, generic). The buffering type [Page 34] defines how many table records are loaded into the buffer when a table entry is accessed.
- *Logging*: This parameter defines whether changes to the table entries should be logged. If logging [Page 41] is switched on, each change to a table record is recorded in a log table.

The *Convert to transparent table* flag (transparent flag [Page 42]) is also displayed for pooled tables or for tables which were converted into transparent tables earlier on with this flag.

**See also:**

Maintaining Technical Settings [Page 76]

Buffering Database Tables [Page 43]

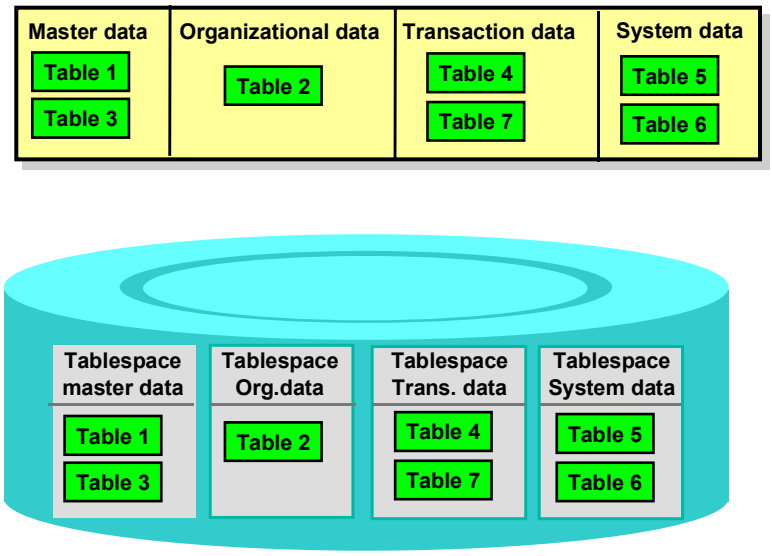
# Data Class

If you choose the data class correctly, your table is automatically assigned to the correct area (tablespace or DBspace) of the database when it is created. Each data class corresponds to a physical area in which all the tables assigned to this data class are stored. There are the following data classes:

- *APPL0* (master data): Data which is seldomly changed. An example of master data is the data contained in an address file, such as the name, address and telephone number.
- *APPL1* (transaction data): Data that is frequently changed. An example of transaction data is the goods in a warehouse, which change after each purchase order.
- *APPL2* (organizational data): Customizing data that is defined when the system is installed and seldomly changed. An example is the table with country codes.

Two further data classes, USR and USR1, are provided for the customer. These are for user developments. The tables assigned to these data classes are stored in a tablespace for user developments.

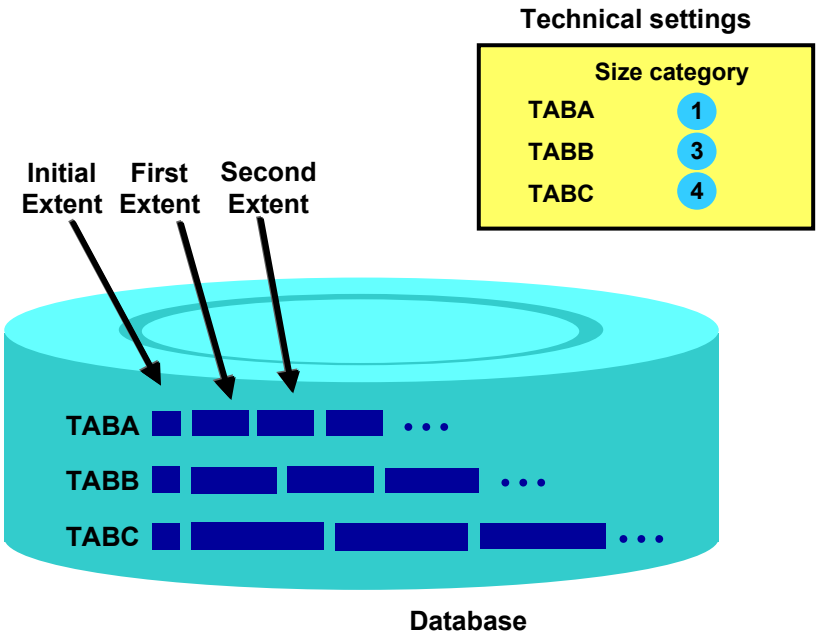
**Tables in the ABAP Dictionary**



# Size Category

The size category defines the expected space required for the table in the database. You can choose a size category from 0 to 4 for your table. Each category is assigned a certain fixed memory size in the database, which depends on the database system used.

When a table is created, initial space (an Initial Extent) is reserved in the database. If more space is required at a later time due to data entries, additional memory will be added depending on the selected size category.




Selecting the correct size category prevents a large number of very small extents from being created for a table. It also prevents space from being wasted if extents which are too large are created.

# Buffering Permission

You must define whether and how a table is buffered in the technical settings for the table. There are three possibilities here:

- *Buffering not permitted:* Table buffering is not permitted, for example because application programs always need the most recent data from the table or the table is changed too frequently.
- *Buffering permitted but not activated:* Buffering is permitted from the business and technical points of view. Applications which access the table execute correctly with and without table buffering. Whether or not table buffering will result in a gain in performance depends on the table size and access profile of the table (frequency of the different types of table access). Table buffering is deactivated because it is not possible to know what these values will be in the customer system. If table buffering would be advantageous for the table size and access profile of the table, you can activate it in the customer system at any time.
- *Buffering activated:* The table should be buffered. In this case you must specify a buffering type [Page 34].

**See also:**

Buffering Database Tables [Page 43]

Which Tables Should be Buffered? [Page 53]

# Buffering Types

The buffering type defines which table records are loaded into the buffer of the application server when a table record is accessed. There are the following buffering types:

- Full buffering [Page 35]: All the records of the table are loaded into the buffer when one record of the table is accessed.
- Generic buffering [Page 37]: When a record of the table is accessed, all the records having this record in the generic key fields (part of the table key that is left-justified, identified by specifying a number of key fields) are loaded into the buffer.
- Single-record buffering [Page 39]: Only the records of a table that are really accessed are loaded into the buffer.

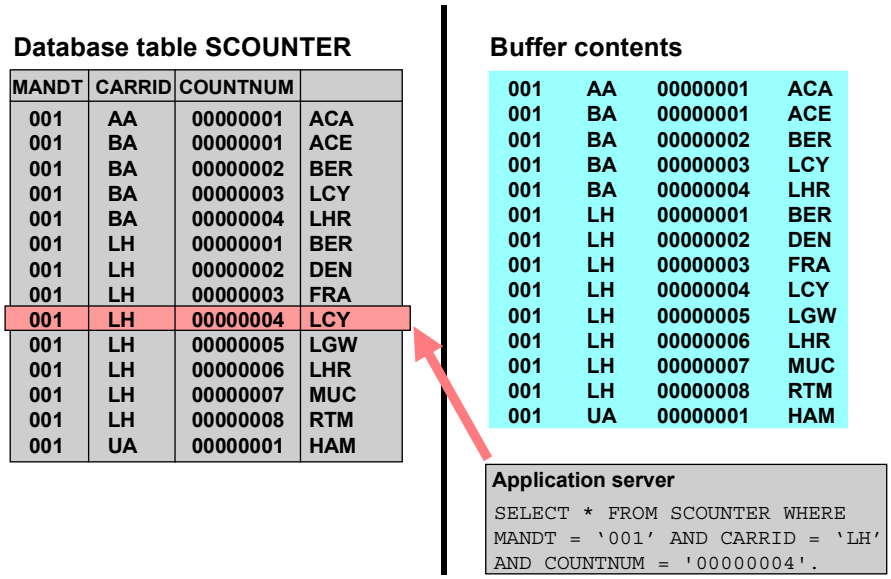
**See also:**

Buffering Database Tables [Page 43]

# Full Buffering

With full buffering, either the entire table is in the buffer or the table is not in the buffer at all. All the records of the table are loaded into the buffer when one record of the table is read.

In this example, a program reads the record highlighted in red from table SCOUNTER. If the table is fully buffered, all the records of the table are loaded into the buffer.




The buffered data records are sorted in the buffer by table key. Accesses to the buffered data can therefore only analyze field contents up to the last specified key field for restricting the dataset to be searched.

The left-justified part of the key should therefore be as large as possible in such accesses. For example, if you do not define the first key field, the system has to scan the full table. In this case direct access to the database can be more efficient if the database has suitable secondary indexes [Page 61].

## When Should you Use Full Buffering?

When deciding whether a table should be fully buffered, you should take into account the size of the table, the number of read accesses, and the number of write accesses. Tables best suited to full buffering are small, read frequently, and rarely written.

Full buffering is recommended in the following cases:

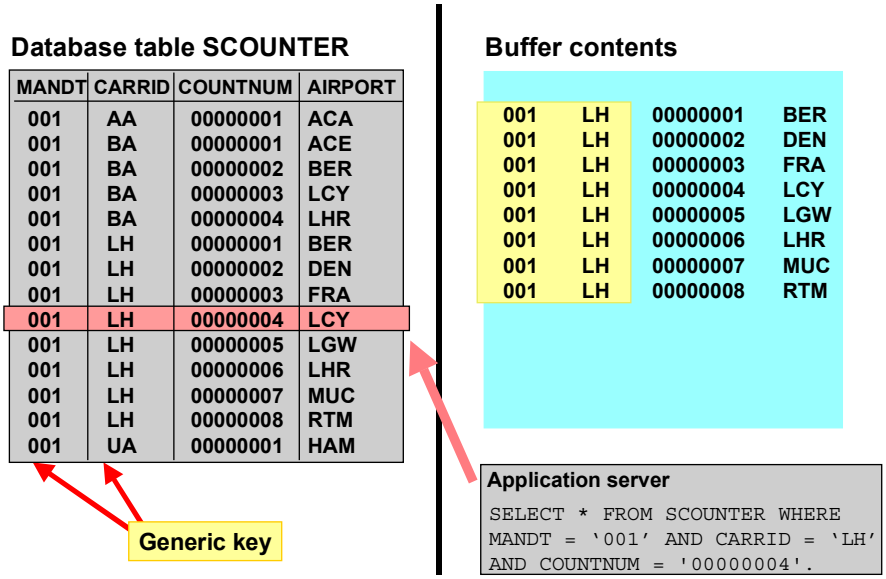
**Full Buffering**

- Tables up to 30 KB in size. If a table is accessed frequently, but all accesses are read accesses, this value can be exceeded. However, you should always pay attention to the buffer utilization.
- Larger tables where large numbers of records are frequently accessed. If these mass accesses can be formulated with a very selective WHERE condition using a database index [Page 61], it could be better to dispense with buffering.
- Tables for which accesses to non-existent records are frequently submitted. Since all the table records reside in the buffer, the system can determine directly in the buffer whether or not a record exists.

# Generic Buffering

With generic buffering, all the records in the buffer whose generic key fields match this record are loaded when one record of the table is accessed. The generic key is a part of the primary key of the table that is left-justified.

In this example, the record highlighted in red is read by a program from table SCOUNTER. If the table is generically buffered, all the records read whose generic key fields (MANDT and CARRID) agree are loaded into the buffer.




## When Should you Use Full Buffering?

A table should be buffered generically if only certain generic areas of the table are normally needed for processing.

Client-specific, fully-buffered tables are automatically generically buffered since normally it is not possible to work in all clients at the same time on an application server. The client field is the generic key.

Language-specific tables are another example where generic buffering is recommended. In general, only records of one language will be needed on an application server. In this case, the generic key includes all the key fields up to and including the language field.

## How Should you Define the Generic Key?

In generic buffering, it is crucial to define a suitable generic key.

If the generic key is too small, the buffer will contain a few very large areas. During access, too much data might be loaded in the buffer.

If the generic key is too large, the buffer might contain too many small generic areas. These can reduce buffer performance since there is an administrative entry for every buffered generic area. It is also possible that too many accesses will bypass the buffer and go directly to the database, since they do not fully define the generic key of the table. If there are only a few records in each generic area, it is usually better to fully buffer the table.

Only 64 bytes of the generic key are used. You can specify a longer generic key, but the part of the key exceeding 64 bytes is not used to create the generic areas.

## Access to Buffered Data

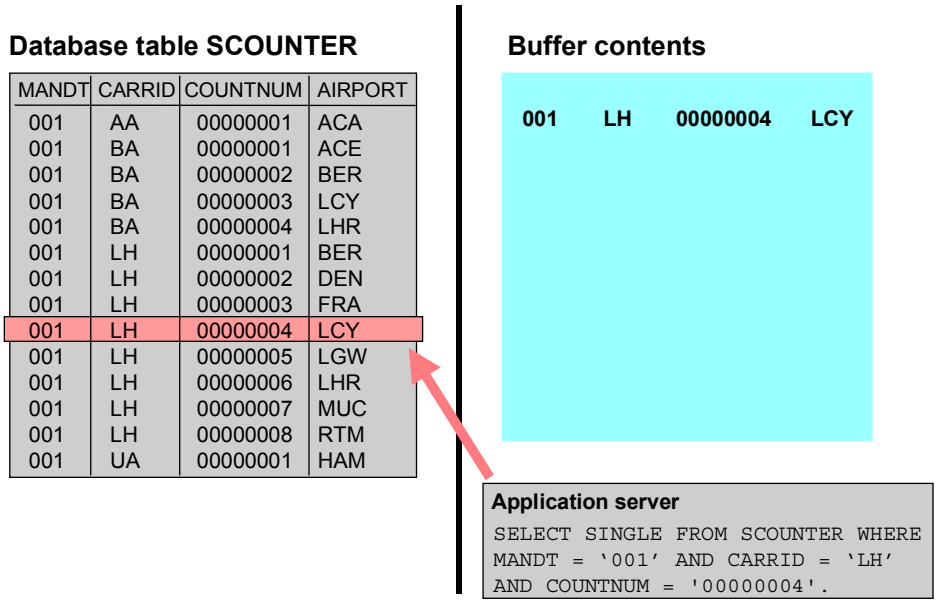
It only makes sense to generically buffer a table if the table is accessed with fully-specified generic key fields. If a field of the generic key is not assigned a value in a SELECT statement, it is read directly from the database, bypassing the buffer.

If you access a generic area that is not in the buffer with a fully-specified generic key, you will access the database to load the area. If the table does not contain any records in the specified area ("No record found"), this area in the buffer is marked as non-existent. It is not necessary to access the database if this area is needed again.

# Single-Record Buffering

With single-record buffering, only the records that are actually read are loaded into the buffer. Single-record buffering therefore requires less storage space in the buffer than generic and full buffering. The administrative costs in the buffer, however, are greater than for generic or full buffering. Considerably more database accesses are necessary to load the records than for the other buffering types.

In this example, the record highlighted in red is read by a program from table SCOUNTER. If single-record buffering is selected for the table, only the record that was read is loaded into the buffer.




## When Should you Use Single-Record Buffering?

Single-record buffering should be used particularly for large tables where only a few records are accessed with SELECT SINGLE. The size of the records being accessed should be between 100 and 200 KB.

Full buffering is usually more suitable for smaller tables that are accessed frequently. This is because only one database access is necessary to load such a table with full buffering, whereas several database accesses are necessary for single-record buffering.

## Access to Buffered Data

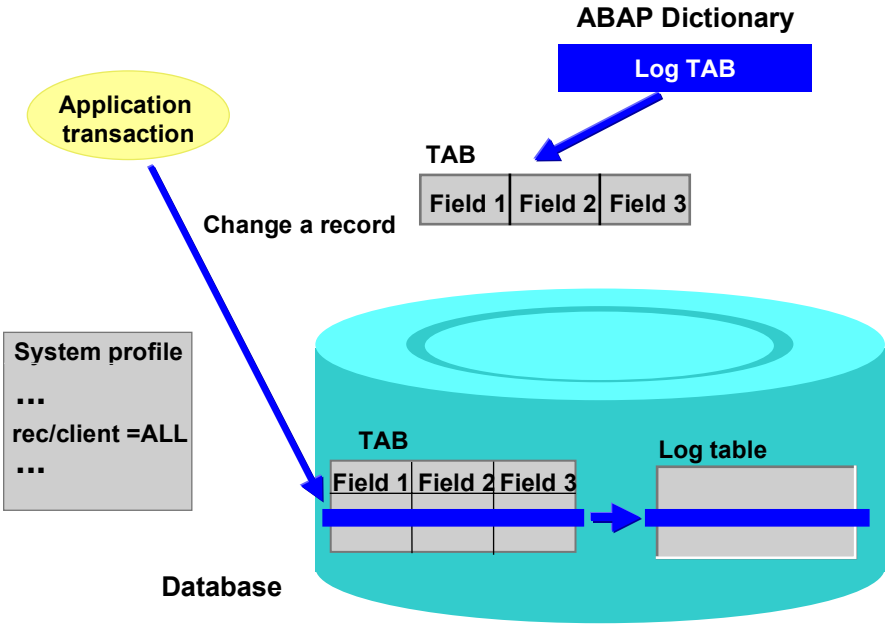
All accesses that are not submitted with SELECT SINGLE go directly to the database, bypassing the buffer. This applies even if the complete key is specified in the SELECT statement.

### Single-Record Buffering

If you access a record which is not yet buffered with SELECT SINGLE, there is a database access to load the record. This record is marked in the buffer as non-existent if the table does not contain a record with the specified key. This prevents another database access when accessing the table at a later time with the same key.

# Logging

Using the logging flag you can define whether changes to the data records of a table should be logged. If logging is switched on, each change to an existing data record (with UPDATE, DELETE) by the user or application program is recorded in the database in a log table (DBTABPRT).




To switch on logging, the R/3 System must be started with a profile containing parameter *rec/client*. This parameter defines whether all clients or only selected clients should be logged.

The parameter can have the following values:

| | |
|---|---|
| rec/client = ALL | Log all clients. |
| rec/client = 000[,...] | Log the specified clients. |
| rec/client = OFF | Do not log. |

Logging slows down accesses that change the table. First of all, a record must be written in the log table for each change. Secondly, a number of users access this log table in parallel. This can cause lock situations although the users are working with different application tables.

Logging is independent of the update.

The existing logs can be displayed with Transaction *Table History* (SCU3).

# Converting Pooled Tables to Transparent Tables

You can easily convert pooled tables to transparent table with the *transparent flag* of the technical settings. You can use this option to access a pooled table from outside the R/3 System.

## Procedure

1. From the maintenance screen of the pooled table, go to the maintenance screen for the technical settings [Page 30] of the pooled table with *Technical settings.*
2. Select *Convert to transparent table.* This will only correct the *technical settings* and not the table definition.
3. Maintain the remaining attributes of the technical settings.
4. *Activate* the technical settings.

## Result

A conversion converts the pooled table to a transparent table (see Adjusting Database Structures [Page 219]). In a dialog box you can decide whether to convert the table directly or in the background.

If SAP again delivers a table whose *transparent flag* is set as a pooled table, it will remain transparent in the user system. If you want to convert a table whose *transparent flag* is set back to a pooled table, you must first cancel the *transparent flag*. Then activate the *technical settings*. You can only change the table category [Page 90] after these actions.

If there is a revised version of a pooled table, it cannot be converted to a transparent table with the *transparent flag*. In this case the *technical settings* of the table cannot be activated when the *transparent flag* is set.

There are some restrictions for transparent tables (number of key fields, key length, table length) that are not valid for pooled tables. Only those pooled tables which satisfy all these restrictions can be converted with the *transparent flag*. If one of these restrictions is violated, the technical settings cannot be activated. In this case you can find the cause of the error in the activation log.

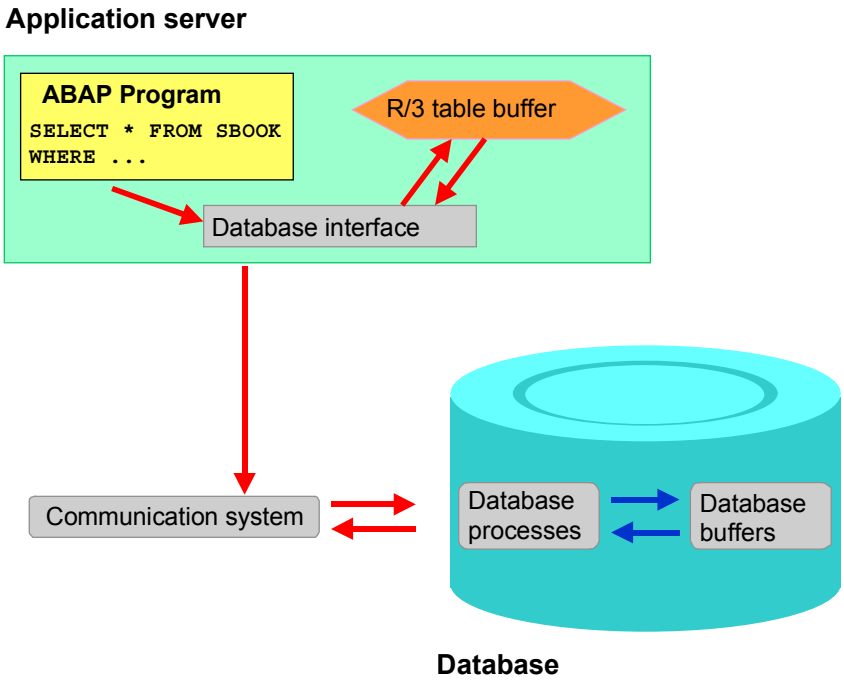
**See also:**

Pooled and Cluster Tables [Page 256]

# Buffering Database Tables

Buffering a table improves the performance when accessing the data records contained in the table.

The table buffers reside locally on each application server in the system. The data of buffered tables can thus be accessed directly from the buffer of the application server. This avoids the time-consuming process of accessing the database.



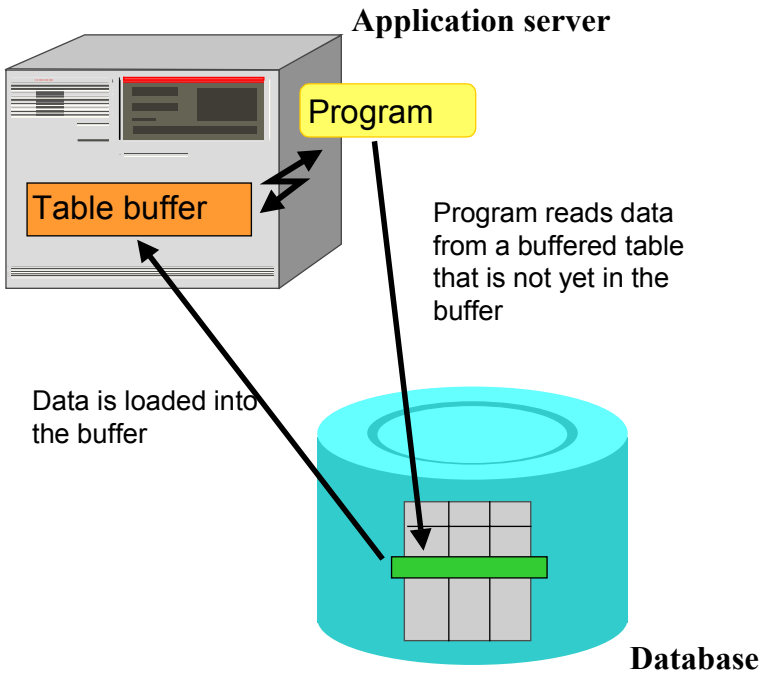

Buffering is particularly important in client/server environments, as it takes considerably longer to access a table with the network than it does to access a table that is buffered locally. Depending on the network load, this factor can lie between 10 and 100.

The difference in performance is somewhat less marked in central systems (systems with only one application server) than in local ones (systems with several application servers). However, even in central systems, a reduction in process changes and increased sophistication of the buffering over that provided by the database system have a noticeable effect on performance.

## How are the Buffers Filled?

If a program accesses the data of a buffered table, the database interface determines whether this data is in the buffer of the application server. If this is the case, the data is read directly from the buffer. If the data is not in the buffer of the application server, it is read from the database and loaded into the buffer. The next access to this data can then use the buffer.

The buffering type [Page 34] determines which records are loaded into the buffer during an access.




## How are the Local Buffers Synchronized?

A buffered table is generally read on all application servers and held in the buffer there. If a program changes the data contained in the table on an application server, this is noted in the log table by the database interface. The buffers still have the old status on all the other application servers, so that the programs might read obsolete data.

A synchronization mechanism [Page 46] runs at a fixed time interval, usually every 1-2 minutes. The log table is read and the buffer contents that were changed by other servers are invalidated. In the next access, the data of invalidated tables is read directly from the database and updated in the buffer.

## Displacement

If more space is required in the buffer due to new data, the data that has not been accessed for the longest time is displaced. The data is displaced asynchronously at certain times that are defined dynamically by the buffer accesses. The data is only displaced if at this time the free space in the buffer is less than a given value or if the access quality is not good enough.

## Resetting the Table Buffers

You can reset the table buffers on the corresponding application servers by entering $TAB in the command field. All the data in the buffer is invalidated.

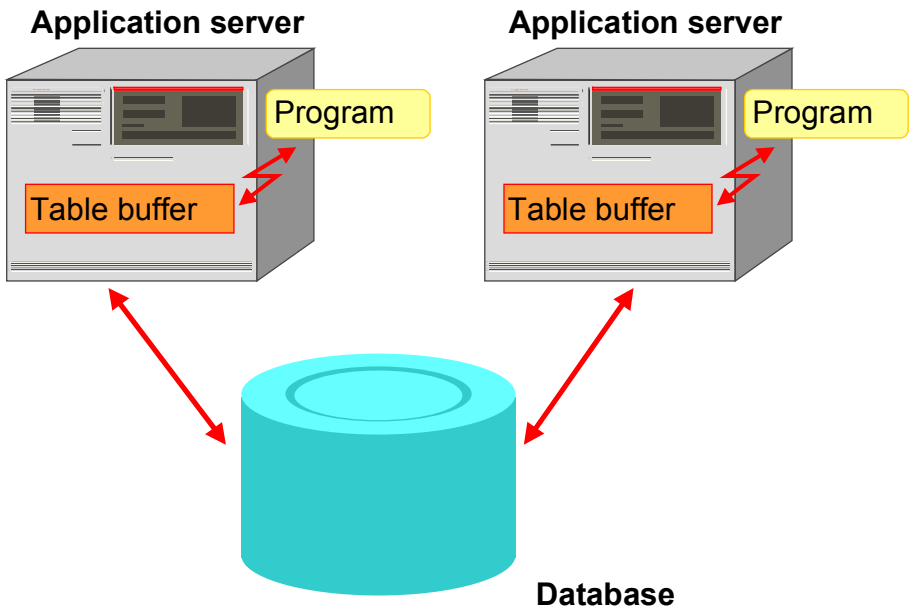
Only use this command if inconsistencies occurred in the buffer! It can take several hours to fill the buffers in large systems. Performance is considerably reduced during this time.

**See also:**

Local Buffer Synchronization [Page 46]

# Local Buffer Synchronization

The table buffers reside locally on each application server in the system. However, this makes it necessary for the buffer administration to transfer all changes made to buffered objects to all the application servers of the system.




If a buffered table is modified, it is updated synchronously in the buffer of the application server from which the change was made. The buffers of the whole network, that is, the buffers of all the other application servers, are synchronized with an asynchronous procedure.

Entries are written in a central database table (DDLOG) after each table modification that could be buffered. Each application server reads these entries at fixed time intervals.

If entries are found that show a change to the data buffered by this server, this data is invalidated. If this data is accessed again, it is read directly from the database. In such an access, the table can then be loaded to the buffer again.

To prevent a table being constantly loaded to the buffer and then immediately invalidated there, the table can only be loaded again to the buffer after a waiting time following invalidation.

The advantage of this method over a synchronous method (where the network immediately reports each change to buffered data of one server to all the other servers) is that the load on the network is kept to a minimum. If the buffers were to be synchronized after each modification, each server would have to report each modification to a buffered table to all the other servers using the network. This would have a negative effect on the performance.

The disadvantage of this method as compared with synchronous updating is that the data will not be up-to-date during the time interval between two synchronizations.

Because of the asynchronous buffer synchronization method, you should note the following when setting the table buffering:

- Only tables that are rarely written (read mostly) or for which temporary inconsistencies are of no significance should be buffered.
- Tables whose entries are frequently modified should not be buffered. Otherwise there would be constant invalidation and reloading, which would have a negative effect on the performance.

The default setting is for buffer synchronization to be activated in the system profile. Buffer synchronization is not required in central systems with only one application server and must be switched off.

**See also:**

Example for Buffer Synchronization [Page 48]

# Example for Buffer Synchronization

The following example shows how the local buffers of the system are synchronized. We have a system with two application servers, called *Server1* and *Server2*.

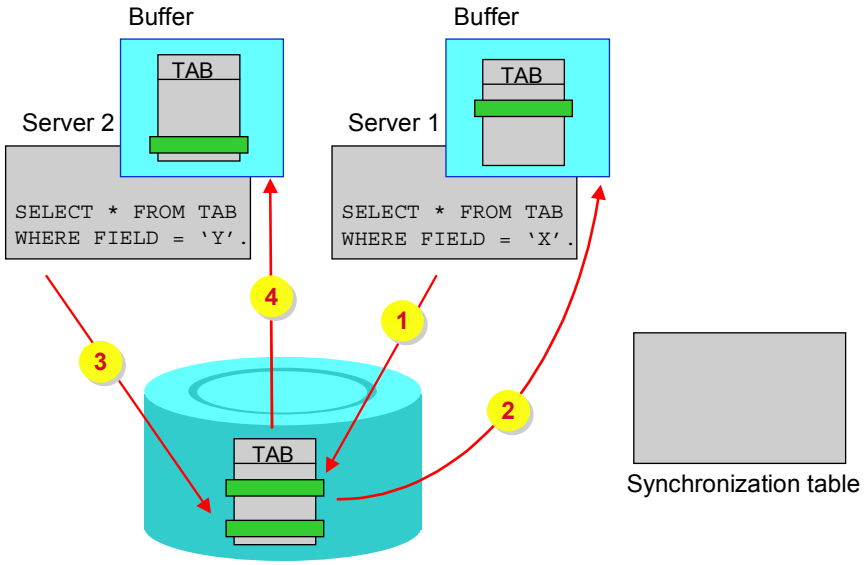
**Initial situation**: Neither server has yet accessed records of the fully buffered table TAB. The local buffers of both servers do not yet contain the records of the table.

**Event 1:** Server 1 reads records from table TAB in the database.

**Event 2:** Table TAB is completely loaded into the local buffer of Server 1. Accesses by Server 1 to the data of table TAB now use the local buffer of this server.

**Event 3:** Server 2 accesses records of the table. Since the table is not yet in the local buffer of Server 2, the records are read directly from the database.

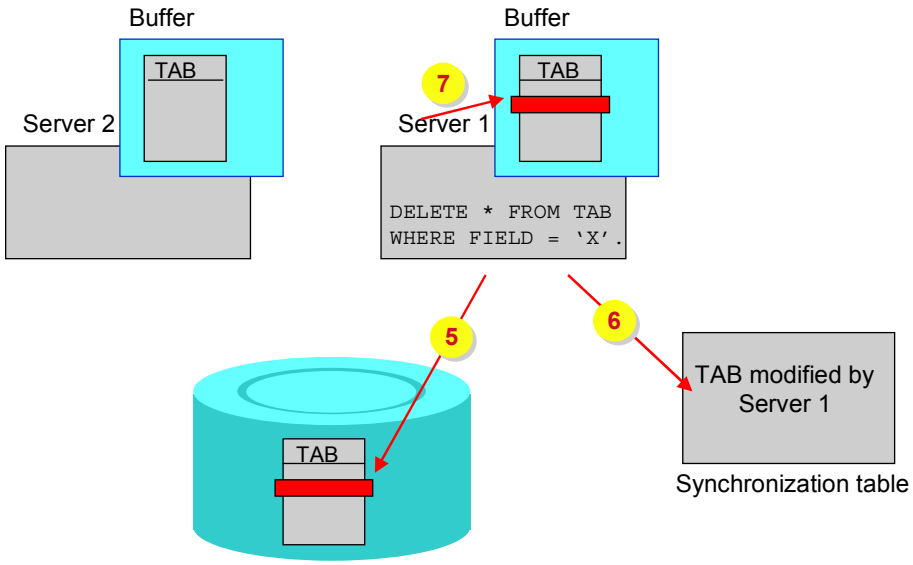
**Event 4:** Table TAB is completely loaded into the local buffer of Server 2. Server 2 then also accesses the data in TAB using its local buffer the next time it reads.




**Event 5:** Server 1 deletes records from table TAB and updates the database.

**Event 6:** Server 1 writes an entry in the synchronization table.

**Event 7:** Server 1 updates its local buffer.

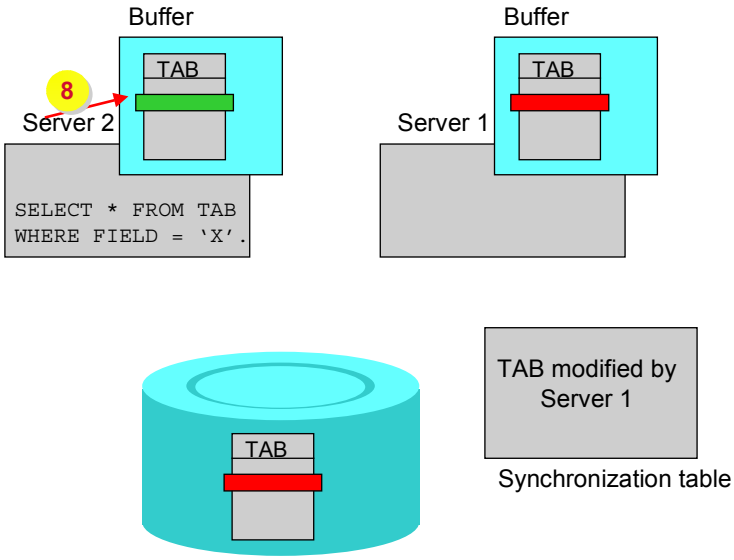



**Event 8:** Server 2 accesses the deleted data records. Since table TAB is in its local buffer, access uses this local buffer.

Server 2 therefore finds the records, although they are no longer in the database table!

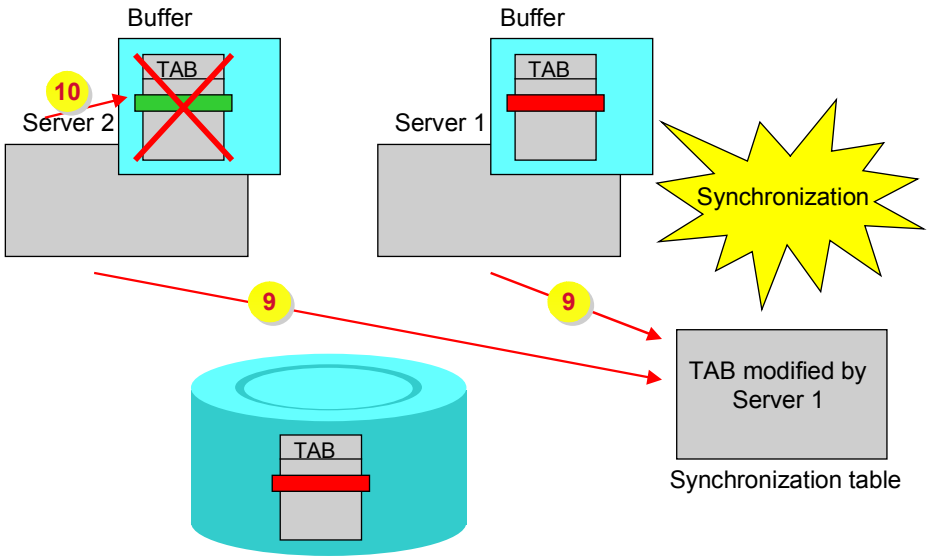
If the same access were executed from an application program on Server 1, this program would see that the records no longer exist. The behavior of an application program at this time therefore depends on the server on which it is executing!

## Example for Buffer Synchronization




**Event 9:** It is now time for synchronization. Both servers check the synchronization table to find out whether one of the tables in its local buffer has been changed in the meantime.

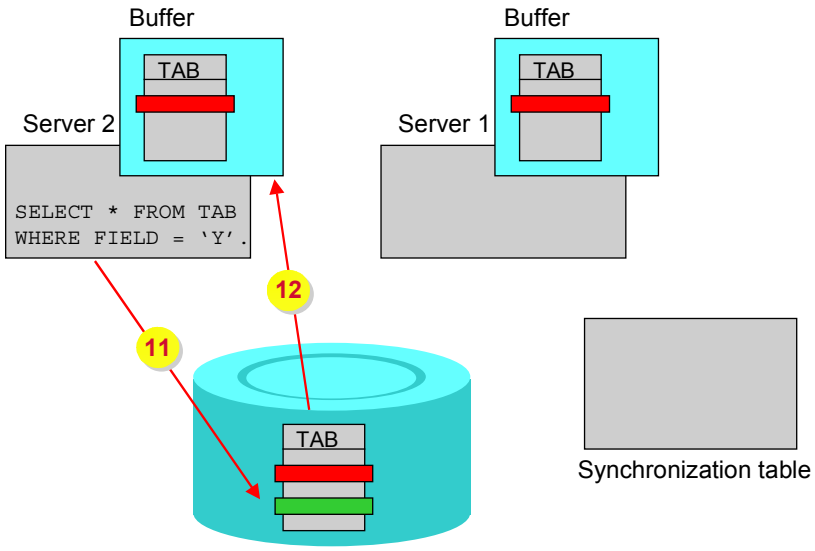
**Event 10:** Server 2 finds that table TAB was modified by Server 1 in the meantime. Server 2 therefore invalidates the table in its local buffer. The next time Server 2 accesses the data in table TAB, it therefore uses the database. Server 1 does not have to invalidate the table in its buffer because it alone changed table TAB. Server 1 therefore uses its local buffer the next time it accesses records of table TAB.




**Event 11:** Server 2 again accesses records of table TAB. Because TAB was invalidated in the local buffer of Server 2, access uses the database.

**Event 12:** The table is again loaded into the local buffer of Server 2. The information about table TAB is now consistent for both servers and the database.

## Example for Buffer Synchronization



# Which Tables should be Buffered?

Only transparent tables and pooled tables can be buffered. Cluster tables cannot be buffered.

Character data types must be assigned to all key fields of buffered tables, that is the data type of the field must be mapped to one of the ABAP types C, N, D or T (see Mapping to ABAP Data Types [Page 238]).

The following two points speak against table buffering:

- The data read by the application must always be up-to-date. Delays caused by the synchronization mechanism (see Synchronization of Local Buffers [Page 46]) cannot be accepted.
- The table data is frequently modified. In this case the cost of synchronization could be greater than the gain in performance resulting from buffering. Roughly speaking, it does not make sense to buffer a table if more than one percent of the accesses to the table are modifying accesses.

The table containing currency exchange rates is updated only once a day, but it is read frequently. Buffering is recommended in this case.

Typical candidates for buffering include customizing and system tables. In certain cases master data with customizing character can also be buffered.

The contents of buffered tables are not always up-to-date in a distributed system. You can bypass the buffer and read the data directly from the database table with the ABAP command "SELECT SINGLE ... BYPASSING BUFFER". If a buffered table is accessed frequently with this command, you should consider whether it is really necessary for the table to be buffered or whether it is essential to have the current state of the database.

You must define whether and how a table is buffered in its technical settings (see Technical Settings [Page 30]).

# Which Accesses Proceed Directly to the Database?

When programming accesses to buffered tables it is important to know which accesses read from the buffer and which accesses go directly to the database.

The following accesses always bypass the buffer and proceed directly to the database:

- SELECT... BYPASSING BUFFER
- SELECT FOR UPDATE
- SELECT with aggregate function, for example COUNT, MIN, MAX, SUM, AVG
- SELECT DISTINCT
- SELECT... WHERE… IS NULL
- ORDER BY (with the exception of PRIMARY KEY)

With generic buffering, all SELECT statements that do not fully specify the generic key also proceed directly to the database. Therefore, with generic buffering the buffer can only be used for accesses that specify the complete generic key.

With single-record buffering, each SELECT statement without the suffix SINGLE also proceeds directly to the database. This applies even if the complete key is specified in the SELECT statement. In single-record buffering, therefore, the buffer can be used only for accesses with SELECT SINGLE.

Of course, all accesses with Native SQL (EXEC SQL) also bypass the buffer and proceed directly to the database. Such accesses should be avoided **at all costs** with buffered tables. First of all, read accesses always bypass the table buffers. Secondly, the R/3 System does not notice modifying accesses since no entries are made in the synchronization table here (see Synchronization of Local Buffers [Page 46]). This can lead to inconsistencies between the data in the buffers of the application server and the database.

# How are Table Buffers Implemented Technically?

The table buffers lie in shared memory. There is a single-record table buffer TABLP and a generic/full table buffer TABL. The two table buffers differ primarily in how they manage free storage areas and in their displacement mechanisms.

Detailed information on the technical buffer structure, especially on the structure, access mechanism, displacement behavior, and global synchronization in distributed systems can be found in:

- Single-Record Table Buffers [Page 56]
- Generic and Full Table Buffers [Page 58]

# Single-Record Table Buffers

Individual records of tables with single-record buffering are managed in the single-record table buffer TABLP.

## Technical Implementation and Buffer Access

The single-record table buffer contains a central administrative structure, a table directory and the data area. The data area is organized in frames of a fixed size (default value 4 KB).

The table names are sorted alphabetically in the directory. The table entries are also stored sorted in the corresponding frames of the data area.

In a buffer access, a binary search is first used to find the entry in the table directory, followed by the corresponding frame and then the record searched for within the frame.

Frames can overflow in the data area if new data is inserted while adhering to the sort sequence. Such overflow frames have to be divided up and their administrative structure must be updated. This explains why the single-record table buffer is somewhat less efficient than the generic/full table buffer. In the single-record table buffer, the data records must be added one after the other while reorganizing the frame structure. In the generic/full buffer, however, all the data of a table is transferred in one step, already sorted by the database.

## Managing Non-Existent Records

The single-record buffer also stores information about non-existent records of a table. If you attempt to access the table with a key that is not in the database, this information is stored in the buffer.

This is done with a flag that is added to every record stored in the buffer. The flag shows whether or not this record exists in the table. If you attempt to access a record that does not exist, an empty record is stored in the buffer with the corresponding key, and the flag is set to the value for a non-existent record. If you try to access this record again, the system sees in the buffer that this record does not exist in the database.

Single-record buffering is also recommended if you repeatedly try to access non-existent records of a table.

## Displacement

If records of a table with single-record buffering are to be read and stored in the buffer, it might be necessary to remove other records from the buffer for space reasons. In this case, the records of the table that were least recently accessed are removed from the buffer.

## Global Synchronization

The modifications made in the local buffers must be synchronized in distributed systems in order to keep the buffered data consistent. The general procedure for synchronization is described in Synchronization of Local Buffers [Page 46].

The effect of different ABAP commands on the local and global synchronization of the single-record buffer is described below.

If changes are made with WHERE conditions (UPDATE dbtab WHERE ..., DELETE FROM dbtab WHERE ...), the entire table is invalidated in the buffer of the local server (server on which the command was submitted) and on all other servers at the time of the synchronization.

Changes without WHERE conditions (UPDATE dbtab, INSERT dbtab, DELETE dbtab) modify the corresponding records in the buffer of the local server and delete them in the buffers of all the other servers at the time of synchronization.

Modifications with a WHERE condition therefore place a much greater load on buffer management than those without a WHERE condition.

# Generic and Full Table Buffers

Generic table areas or full tables are managed in the generic/full table buffer TABL. The individual generic areas are managed in the buffer like autonomous, fully buffered tables.

## Technical Implementation and Buffer Access

Like the single-record buffer, the generic/full table buffer has a central administration structure, a table directory and the data area.

The main difference between the generic/full buffer and the single-record buffer lies in the data area administration. The generic/full buffer divides its storage into areas (extents) of variable length. The length of an extent is a multiple of a fixed block size (256 bytes). The buffer management tries to place the contents of a generic table area or a whole table in an extent. This reduces the storage requirements for buffered data in comparison to single-record buffering.

The table names are sorted alphabetically in the table directory. The data records are transferred to the corresponding extent in one step, already sorted by the database. In a buffer access, there is first a binary search for the corresponding table name in the table directory. Then there is a binary search for the corresponding data record in the extent.

## Displacement

Unlike with single-record buffering, the displacement for generic/full buffering is carried out asynchronously at certain fixed times. These times are defined dynamically depending on the number of accesses to the buffer. The time between two displacements therefore depends on the buffer utilization factor.

Data is only displaced if the free space in the buffer falls below a defined value or the access quality (that is, the number of accesses that can be satisfied directly from the buffer) is not good enough at the time of access. An attempt is made to free a certain area of the buffer. A larger number of tables is therefore displaced at certain times.

The tables accessed least frequently are displaced. The table accesses are weighted differently, depending on the time of the access. Accesses lying further in the past are weighted less than those that occurred shortly before the displacement.

This method ensures that tables which are accessed frequently at a certain time and then not used again can be displaced from the buffer after a defined time.

The individual generic areas if the buffer of generically buffered tables are treated as independent tables. Some generic areas of a table can therefore be displaced while other generic areas of the table are kept in the buffer.

The buffer is reorganized at certain intervals once the tables have been displaced, thus reducing buffer fragmentation.

## Global Synchronization

The modifications made in the local buffers must be synchronized in distributed systems in order to keep the buffered data consistent. The general procedure for synchronization is described in Synchronization of Local Buffers [Page 46].

The effect of various ABAP commands on the local and global synchronization of the generic/full buffer is described below.

If changes are made with WHERE conditions (UPDATE dbtab WHERE ..., DELETE FROM dbtab WHERE ...), the entire generic area or the entire table is invalidated in the buffer of the local server (server at which the command was submitted) and on all other servers at the time of synchronization.

Changes without WHERE conditions (UPDATE dbtab, INSERT dbtab, DELETE dbtab) change the corresponding records in the buffer of the local server at the time of synchronization. The entire generic area or the entire table is invalidated in the buffers of all the other servers.

Modifications with a WHERE condition therefore place a much greater load on buffer management than those without a WHERE condition.

# How can you Analyze the Buffer Quality?

The buffer monitor [Ext.] permits you to look at the statistical data on the table buffer and on individual tables. The buffer monitor can be started from the initial R/3 screen with the following menu options:

*Tools → Administration*
*Monitoring → Performance*
*Setup/buffers → Buffers*

An overview of all the R/3 buffers and database calls appears.

You can look at the buffer utilization of your server during the last few days with *History*.

With *Current parameters* you can display the current valid profile parameters for the buffer sizes.

By selecting *Generic key*, you get detailed information on the generic/full table buffer. Selecting *Single record* gives you detailed information on the single-record table buffer. You can also access this detailed information with *Goto → Detail analysis menu* and then *Generic key* or *Single record* in the next screen.

Selecting *Buffered objects* takes you from the detailed display of the generic/full or single-record buffer to call statistics for the tables used. The displayed values help you optimize the buffer parameters in the system profile and search for unfavorably buffered tables. The statistics for the generic or fully buffered tables appear if you access the call statistics from the information on generic/full buffers. If you call the statistics from the information on single-record buffers, the statistics for the single-record buffered tables are displayed.

**See also:**

BC Computing Center Management System [Ext.]

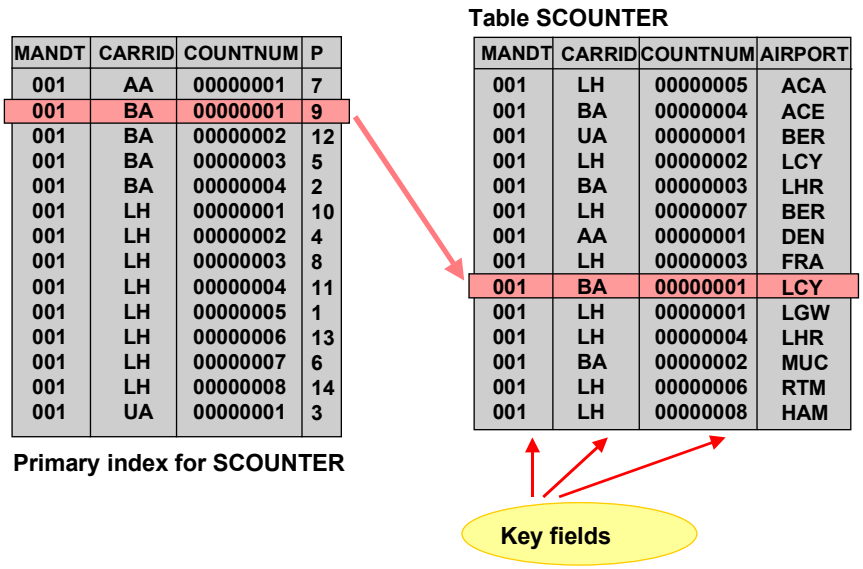
# Indexes

You can search a table for data records that satisfy certain search criteria faster using an index.

An index can be considered a copy of a database table that has been reduced to certain fields. This copy is always in sorted form. Sorting provides faster access to the data records of the table, for example using a binary search. The index also contains a pointer to the corresponding record of the actual table so that the fields not contained in the index can also be read.

The primary index is distinguished from the secondary indexes of a table. The **primary index** contains the key fields of the table and a pointer to the non-key fields of the table. The primary index is created automatically when the table is created in the database.

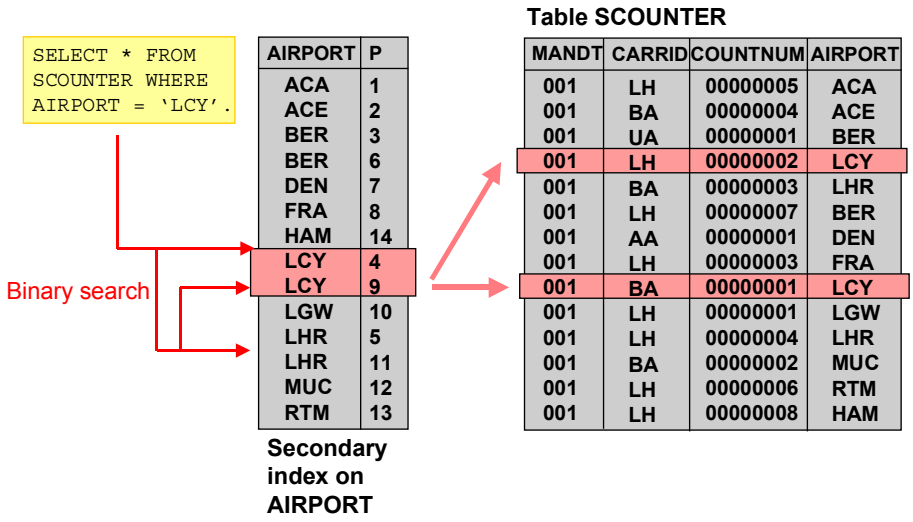
Table SCOUNTER in the flight model [Page 295] contains the assignment of the carrier counters to airports. The primary index on this table therefore consists of the key fields of the table and a pointer to the original data records.




You can also create further indexes on a table in the ABAP Dictionary. These are called **secondary indexes**. This is necessary if the table is frequently accessed in a way that does not take advantage of the sorting of the primary index for the access. Different indexes on the same table are distinguished with a three-place index identifier [Page 67].

All the counters of carriers at a certain airport are often searched for flight bookings. The airport ID is used to search for counters for such an access. Sorting the primary index is of no use in speeding up this access. Since table SCOUNTER has a large

number of entries, a secondary index on the field AIRPORT (ID of the airport) must be created to support access using the airport ID.




The optimizer of the database system decides whether an index should be used for a concrete table access (see How to Check if an Index is Used? [Page 65]). This means that an index might only result in a gain in performance for certain database systems. You can therefore define the database systems on which an index should be created when you define the index in the ABAP Dictionary (see Creating Secondary Indexes [Page 77].)

All the indexes existing in the ABAP Dictionary for a table are normally created in the database when the table is created if this was not excluded in the index definition for this database system.

If the index fields have key function, that is if they already uniquely identify each record of the table, an index can be defined as a unique index [Page 66].

**See also:**

What to Keep in Mind when Creating Secondary Indexes [Page 63]

# What to Keep in Mind for Secondary Indexes

How well an existing index supports data selection from a table largely depends on whether the data selected with the index represents the data that will ultimately be selected. This can best be shown using an example.

An index is defined on fields FIELD1, FIELD2, FIELD3 and FIELD4 of table BSPTAB in this order. This table is accessed with the SELECT statement:

SELECT * FROM BSPTAB WHERE FIELD1 = X1 AND FIELD2 = X2 AND FIELD4= X4.

Since FIELD3 is not specified more exactly, only the index sorting up to FIELD2 is of any use. If the database system accesses the data using this index, it will quickly find all the records for which FIELD1 = X1 and FIELD2 = X2. You then have to select all the records for which FIELD4 = X4 from this set.

The order of the fields in the index is very important for the accessing speed. The first fields should be those which have constant values for a large number of selections. During selection, an index is only of use up to the first unspecified field.

Only those fields that significantly restrict the set of results in a selection make sense for an index.

The following selection is frequently made on address file ADRTAB:

SELECT * FROM ADRTAB WHERE TITEL = ‘Prof.’ AND NAME = X AND VORNAME = Y.

The field TITLE would rarely restrict the records specified with NAME and FIRSTNAME in an index on NAME, FIRSTNAME and TITLE, since there are probably not many people with the same name and different titles. This would not make much sense in this index. An index on field TITLE alone would make sense for example if all professors are frequently selected.

Additional indexes can also place a load on the system since they must be adjusted each time the table contents change. Each additional index therefore slows down the insertion of records in the table.

For this reason, tables in which entries are very frequently written generally should only have a few indexes.

The database system sometimes does not use a suitable index for a selection, even if there is one. The index used depends on the optimizer used for the database system. You should therefore check if the index you created is also used for the selection (see How to Check if an Index is Used [Page 65]).).

Creating an additional index could also have side effects on the performance. This is because an index that was used successfully for selection might not be used any longer by the optimizer if the optimizer estimates (sometimes incorrectly) that the newly created index is more selective.

## What to Keep in Mind for Secondary Indexes

The indexes on a table should therefore be as disjunct as possible, that is they should contain as few fields in common as possible. If two indexes on a table have a large number of common fields, this could make it more difficult for the optimizer to choose the most selective index.

# How to Check if an Index is Used

## Procedure

1. Open a second session and choose *System* → *Utilities* → *Performance trace.*

   The *Trace Requests* screen appears.

2. Select *Trace on*.

   The SQL trace is activated for your user, that is all the database operations under your user are recorded.

3. In the first window, perform the action in which the index should be used.

   If your database system uses a *cost-based optimizer*, you should perform this action with as representative data as possible. A cost-based optimizer tries to determine the best index based on the statistics.

4. In the second session, choose *Trace off* and then *Trace list*.

## Result

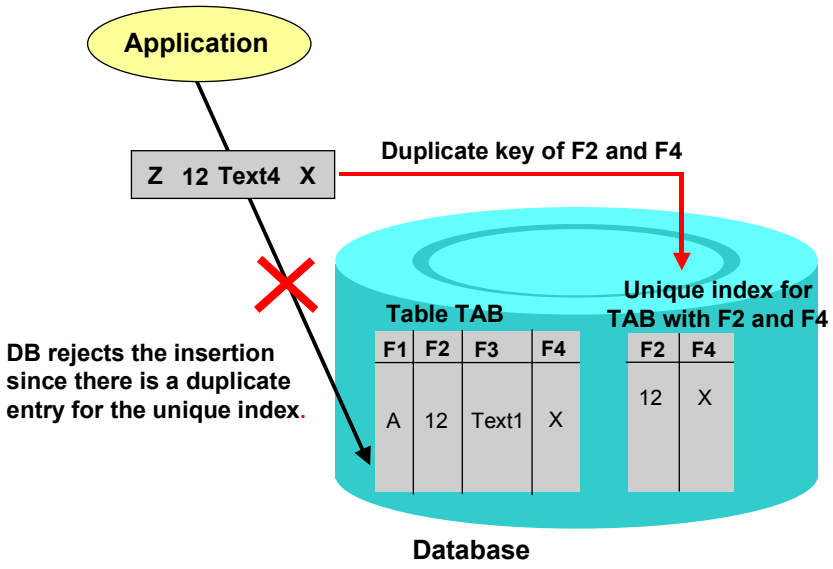
The format of the generated output depends on the database system used. You can determine the index that the database used for your action with the EXPLAIN function for the critical statements (PREPARE, OPEN, REPOPEN).

# Unique Indexes

An entry in an index can refer to several records that have the same values for the index fields. A unique index does not permit these multiple entries. The index fields of a unique index thus have key function, that is they already uniquely identify each record of the table.

The primary index of a table is always a unique index since the index fields form the key of the table, uniquely identifying each data record.

You can define a secondary index as a unique index when you create it. This ensures that there are no double records in the table fields contained in the index. An attempt to maintain an entry violating this condition in the table results in termination due to a database error.




The accessing speed does not depend on whether or not an index is defined as a unique index. A unique index is simply a means of defining that certain field combinations of data records in a table are unique.

A unique index for a client-dependent table must contain the client field.

# Index IDs

Several indexes on the same table are distinguished by a three-place **index identifier**. The index identifier may only contain letters and digits. The ID *0* is reserved for the primary index.

The index name on the database adheres to the convention <*Table name*>~<*Index ID*>.

*TEST~A* is the name of the corresponding database index in the database for table TEST and the secondary index with ID A.

Since the convention for defining the index name in the database has changed several times, some of the indexes in the database might not follow this convention.

Indexes created prior to Release 3.0 can have an 8-place name. The first 7 places (possibly filled with underlining) are for the table names, and the eighth place is for the (one-place) index identifier (for example *TEST___A*).

Indexes introduced with Release 3.0 can have a 13-place name in the database. The first 10 places (possibly filled with underlining) are for the table names, and the 11th to 13th places are for the three-place index identifier (for example *TEST_____A*).

# Customizing Includes

A Customizing include is a structure that satisfies a special naming convention. The name of a Customizing include begins with ‘CI_’ and the include lies in the customer namespace.

If enhancements are already planned in the R/3 standard using customer-specific fields, such Customizing includes are included [Page 16]. in the corresponding standard table or standard structure. The Customizing include (that is the definition of the structure [Page 138] itself) is usually first created in the customer system and filled with fields by special Customizing transactions.

Customers can thus enhance tables and structures of the R/3 standard system without themselves having to modify the table and structure definitions. This means that these enhancements will not be lost when upgrading. If a table or structure of the R/3 standard system is enhanced with customer fields using a Customizing include, these customer fields are automatically inserted in the new delivered table or structure definition during an upgrade.

Customers can but need not create a Customizing include and fill it with fields. If there is no Customizing include, there is no error message when the table or structure including it is activated.

A Customizing include can be contained in several tables or structures, so that they remain consistent when the include is modified.

The order of the fields in the ABAP Dictionary can differ from the order of the fields in the database. Inserting fields in a Customizing include does not result in a data conversion for transparent tables containing this include (see Adjusting Database Structures [Page 219]). The new fields of the table in the ABAP Dictionary are simply added to the database table.

# Append Structures

Append structures are used for enhancements that are not included in the standard. This includes special developments, country versions and adding customer fields to any tables.

An append structure is a structure that is assigned to **exactly one** table. There can be more than one append structure for a table.

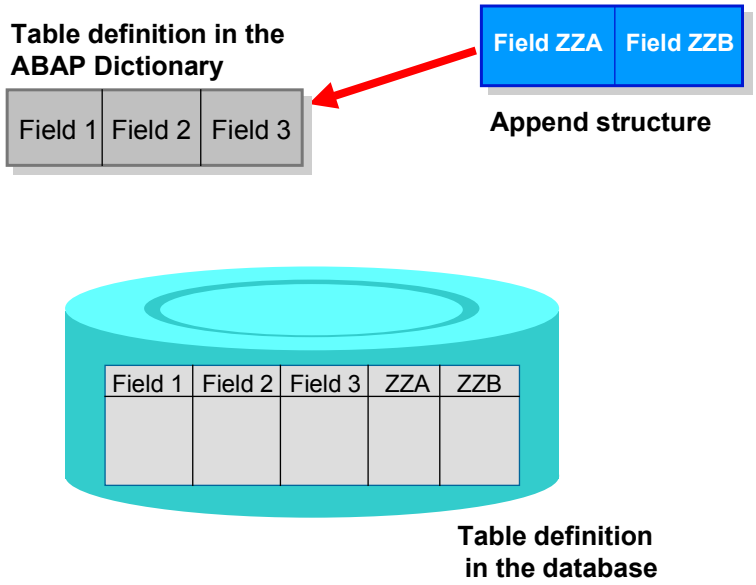
When a table is activated, all the append structures of the table are searched and the fields in these append structures are added to the table. If an append structure is created or changed, the table assigned to it is also adjusted to these changes when the append structure is activated.

Since the order of the fields in the ABAP Dictionary can differ from the order of the fields in the database, adding append structures and inserting fields in such append structures does not result in a table conversion.

The customer creates append structures in the customer namespace. The append structure is thus protected against overwriting during an upgrade. The fields in the append structure should also lie in the customer namespace, that is the field names should begin with ZZ or YY. This prevents name conflicts with fields inserted in the table by SAP.

The new versions of the standard tables are imported after an upgrade, and the fields contained in the append structures are added to the new standard tables at activation.

A standard table contains the fields *Field 1*, *Field 2* and *Field 3*. An append structure containing the fields *ZZA* and *ZZB* is defined for this table. After activating the table, the corresponding database table contains fields *Field 1*, *Field 2*, *Field 3*, *ZZA* and *ZZB*.



## Further Remarks:

- An append structure can only be assigned to exactly one table. If you want to append the same fields to several tables, you can store the fields in a structure. In this case you must create an append structure for each of these tables and include the structure there.
- Adding an append structure to an SAP standard table is supported by the Modification Assistant [Ext.].
- If you want to insert a field that is to be delivered with the R/3 standard in the next Release in the customer system in advance, you must include it in the table itself as a repair. If you include such a field in an append structure for the table, it will occur twice when the new standard table is imported. This will result in an activation error.
- No append structures may be added to tables with long fields (data types VARC, LCHR or LRAW). This is because a long field must always be the last field in the table. However, append structures can be added to structures with long fields.
- If a table with an added append structure is copied, the fields of the append structure become fields of the target table.
- Foreign keys for the appended fields must be defined within the append structure. Fields of the table assigned to the append structure may be defined when assigning the key field and foreign key field.
- Indexes on the appended fields must be defined in the original table.

**See also:**

Adding Append Structures [Page 82]

# Creating Tables

## Procedure

1. In the initial screen of the ABAP Dictionary, select object class *Database table,* enter the table name and choose *Create*.

   The maintenance screen for the table is displayed.

2. Enter an explanatory short text in the field *Short text*.

   You can for example find the table at a later time using this short text.

3. On the *Attributes* tab page, enter the delivery class [Page 78] of the table.

   On this tab page, select *Table maintenance allowed* if users with the appropriate authorization may change the data in the table with *System* → *Services* → *Table maintenance* → *Extended table maintenance*. If table data should only be maintained by program, do not set the flag.

4. On the *Fields* tab page, enter the table fields. Perform the following steps for each table field:

   Enter a name for the table field in the column *Fields*. The field name may only contain letters, digits and underlining, and it must begin with a letter.

   Select the *Key* column if the field should be part of the table key.

   Enter the name of a data element [Page 132] in field *Field type*. In this case the field takes the *data type*, *length*, *decimal places* and *short text* from this data element. If there is no suitable data element, you can go to the data element maintenance screen [Page 134] by entering a name and double-clicking.

   With *Data element/Direct type*, you can directly enter the *data type*, *field length*, *decimal places* and *short text*. Press this key again if you want to enter data elements for further fields.

   Only a limited functionality is provided for fields without data elements. No foreign keys or fixed values may be defined for such fields, and there is no F1 help.

   You must also define the reference field and reference table [Page 15] on tab page *Currency/quantity fields* for fields of types CURR (currency) and QUAN (quantity)

5. Maintain the technical settings [Page 76] for the table. The corresponding maintenance screen is displayed with *Goto* → *Technical settings*.

   The technical settings are a separate object and can be activated and transported separately from the table.

6. Maintain (if necessary) the foreign key relationships [Page 74] of the table to other tables.

   The corresponding maintenance screen is displayed if you place the cursor on the check field and choose .

7. Create (if necessary) secondary indexes for the table [Page 77].

   To do this choose *Goto* → *Indexes*.

8. *Save* the table.

A dialog box appears in which you have to assign the table a development class.

9. Choose .

## Result

During activation, the table and all the indexes on the table are automatically created in the database (if not explicitly excluded when the index was defined). You can display the table definition in the database with *Utilities* → *Database object* → *Display*.

At activation, the runtime object [Page 232] for the table is also created. You can display the runtime object with *Utilities* → *Runtime object* → *Display*.

You can find information about the activation flow in the activation log, which you can display with *Utilities* → *Activation log*. The activation log is displayed automatically if errors occur when the table is activated.

## Other Options

- **Assigning a Field a Search Help:** You can assign a field a search help with *Goto* → *Search help* → *For field*. The search help can be used by all screen fields that refer to this field. See Attaching a Search Help to a Table or Structure Field [Page 181].
- **Assigning a Table a Search Help:** You can assign a table a search help with *Goto* → *Search help* → *For table*. This search help can be used by all screen fields that are checked against the table. See Attaching a Search Help to a Check Table [Page 179].
- **Creating Documentation:** With *Goto* → *Documentation* you can create a text that describes the use of the table and maintenance of the table data more precisely.
- **Assigning the Activation Type:** With *Extras* → *Activation type* you can assign the table an activation type [Page 80]. This is **only** relevant for tables in the runtime environment.
- **Creating or Displaying Data:** If you set the *Table maintenance allowed* flag, you can enter data in the table with *Utilities* → *Table contents* → *Create entries*. You can display existing data with *Utilities* → *Table contents* → *Display*.

## Constraints

- All the key fields of a table must be stored together at the beginning of the table. A non-key field may not occur between two key fields.
- A maximum of 16 key fields per table is permitted. The maximum length of the table key is 255.
- If the key length is greater than 120, there are restrictions when transporting table entries. The key can only be specified up to a maximum of 120 places in a transport. If the key is larger than 120, table entries must be transported generically.
- A table may not have more than 249 fields. The sum of all field lengths is limited to 1962 (whereby fields with data type LRAW and LCHR are not counted).
- Fields of types LRAW or LCHR must be at the end of the table. Only one such field is allowed per table. There must also be a field of type INT2 directly in front of such a field. The actual length of the field can be entered there by the database interface.

**See also:**

Technical Settings [Page 30]

Indexes [Page 61]

Foreign Keys [Page 19]

# Creating Foreign Keys

## Procedure

1. In the field maintenance screen of the table, select the check field [Page 19] and choose 

   If the domain [Page 155] of the check field has a value table [Page 158], you can have the system create a proposal with the value table as check table. In this case a proposal will be made for the field assignment in the foreign key.

   If the domain does not have a value table or if you reject the proposal, the screen for foreign key maintenance appears without proposals. In this case, enter the *check table* and save your entries. The check table must have a key field to which the domain of the check field is assigned.

   You can then let the system make a proposal for assigning the foreign key fields to the key fields of the check fields. The system attempts to assign the key fields of the check table to fields of the table with the same domain. If you do not want a proposal, the key fields of the check table are listed and you must assign them to suitable fields of the foreign key table.

2. Enter an explanatory short text in the field *Short text*.

   The short text provides a technical documentation of the meaning of the foreign key.

3. Choose *Copy*. The foreign key is saved and you return to the maintenance screen for the table.

## Other Options

- **Removing Fields from the Assignment:** You can remove foreign key fields (with the exception of the check field) from the assignment to the key fields of the check table (see Generic and Constant Foreign Keys [Page 22]).

  Select *Generic* to remove a field from the check against the key fields of the check table. If you want to assign a constant to a foreign key field, you must enter it in the *Constant* field enclosed in apostrophes (for example: '*Constant*'). In both cases you must remove the entries in fields *For. key table* and *Foreign key field*.

- **Deactivating the Value Check:** If the foreign key should not be used for a value check, cancel the selection in the *Check required* field. This definition is valid for all screens in which the field appears. It could be advisable to deactivate the check for example if the foreign key is only used to define maintenance views [Page 112], help views [Page 111] or lock objects [Page 202].

- **Assigning Messages:** A standard message is output if the value check by the foreign key on the screen field results in an invalid input. You can replace this standard message with any message. To do this, you must enter the message class of the message in the field *AArea* and the message number in the field *MsgNo*.

- **Semantic Attributes:** Optionally you can define the semantic attributes [Page 24] of the foreign key. These are primarily for documentation purposes.

If the foreign key is derived from a field of an included table or structure (see Includes [Page 16]), the *Inherited from the include* flag is also displayed. Foreign key definitions are usually inherited from the included table or structure to the including table or structure, so that the foreign key depends on the definition in the included table.

The flag is normally set. If you cancel the selection, the link between the foreign key and the included table or structure is canceled. The foreign key no longer adjusts itself when its definition in the included table or structure changes.

**See also:**

Foreign Keys [Page 19]

# Maintaining Technical Settings

## Procedure

1. In the field maintenance screen of the table, choose *Technical settings*.

   The maintenance screen of the technical settings appears.

2. Select the data class [Page 31] and size category [Page 32] of the table.

   The input help for the *Size category* field shows how many data records correspond to the individual categories.

3. The buffering permission [Page 33] defines whether the table may be buffered.

   If you permitted table buffering, define the buffering type [Page 34] of the table.

   You can find further information about when to buffer tables and what buffering type you should choose in Buffering Database Tables [Page 43].

4. Select *Log data changes* if you want to log changes to data records of the table (see Logging [Page 41]).

   To log changes, system logging must be switched on with the profile parameter *rec/client*. The *Log data changes* flag on its own does not cause the table changes to be logged!

5. Choose .

   Errors that occurred during activation are displayed in the activation log.

## Other Options

The *Convert to transparent table* flag is displayed for pooled tables (and tables that were converted to a transparent table with this flag at an earlier time). You can convert the pooled table to a transparent table with this flag (see Converting Pooled Tables to Transparent Tables [Page 42]).

## Constraints

- If the key length of the table is larger than 64, the table cannot be buffered generically.
- If the key length of the table is larger than 120, the table cannot be buffered.
- If the key of the table has more than 86 places or the data part of the table has more than 500 places, it is not possible to log the table.

**See also:**

Technical Settings [Page 30]

Which Tables Should be Buffered? [Page 53]

# Creating Secondary Indexes

## Procedure

1. In the maintenance screen of the table, choose *Indexes*.

   If indexes already exist on the table, a list of these indexes is displayed. *Choose*  .

2. In the next dialog box, enter the index ID [Page 67] and choose 

   The maintenance screen for indexes appears.

3. Enter an explanatory text in the field *Short text*.

   You can then use the short text to find the index at a later time, for example with the R/3 Repository Information System.

4. Select the table fields to be included in the index using the input help for the *Field name* column.

   The order of the fields in the index is very important. See What to Keep in Mind for Secondary Indexes [Page 63].

5. If the values in the index fields already uniquely identify each record of the table, select *Unique index*.

   A unique index [Page 66] is always created in the database at activation because it also has a functional meaning (prevents double entries of the index fields).

6. If it is not a unique index, leave *Non-unique index* selected.

   In this case you can use the radio buttons to define whether the index should be created for all database systems, for selected database systems or not at all in the database.

7. Select *for selected database systems* if the index should only be created for selected database systems.

   Click on the arrow behind the radio buttons. A dialog box appears in which you can define up to 4 database systems with the input help. Select *Selection list* if the index should only be created on the given database systems. Select *Exclusion list* if the index should **not** be created on the given database systems. *Choose*  .

8. Choose .

## Result

The secondary index is automatically created in the database during activation if the corresponding table was already created there and index creation was not excluded for the database system.

You can find information about the activation flow in the activation log, which you can call with *Utilities* → *Activation log*. If errors occurred when activating the index, the activation log is automatically displayed.

**See also:**

Indexes [Page 61]

# Delivery Class

The delivery class controls the transport of table data for installation, upgrade, client copy and when transporting between customer systems. The delivery class is also used in the extended table maintenance.

There are the following development classes:

- **A**: Application table (master and transaction data).
- **C**: Customer table, data is only maintained by the customer.
- **L**: Table for storing temporary data.
- **G**: Customer table, SAP may insert new data records but may not overwrite or delete existing ones.
- **E**: System table with its own namespace for customer entries. The customer namespace must be defined in table TRESC.
- **S**: System table, data changes have the status of program changes.
- **W**: System table (e.g. table of the development environment) whose data is transported with its own transport objects (e.g. R3TR PROG, R3TR TABL, etc.).

## Behavior during Client Copy

Only the data of client-dependent tables is copied.

- **Class C, G, E, S:** The data records of the table are copied to the target client.
- **Class W, L:** The data records of the table are not copied to the target client.
- **Class A:** Data records are only copied to the target client if explicitly requested (parameter option). It normally does not make sense to transport such data, but this is supported nevertheless to permit the entire client environment to be copied.

## Behavior during Installation, Upgrade and Language Import

The behavior of client-dependent tables differs from that of cross-client tables.

### Client-Dependent Tables

- **Class A and C:** Data is only imported into client 000. Existing data records are overwritten.
- **Class E, S and W:** Data is imported into all clients. Existing data records are overwritten.
- **Class G:** Existing data records are overwritten in client 000. In all other clients, new data records are inserted, but existing data records are not overwritten.
- **Class L:** No data is imported.

### Cross-Client Tables

- **Classes A, L and C:** No data is imported.
- **Classes E, S, and W:** Data is imported. Existing data records with the same key are overwritten.

- **Class G:** Non-existent data records are inserted, but no existing data records are overwritten.

## Behavior during Transport between Customer Systems

Data records of tables having delivery class L are not imported into the target system. Data records of tables having delivery classes A, C, E, G, S and W are imported into the target system (for client-dependent tables this is done for the target clients specified in the transport).

## Use of the Delivery Class in the Extended Table Maintenance

The delivery class is also used in the *Extended Table Maintenance* (SM30). The maintenance interface generated for a table makes the following checks:

- It is not possible to transport the entered data using the transport connection of the generated maintenance interface for tables having delivery classes W and L.
- Data that is entered is checked to see if it violates the namespace defined in table TRESC. If the data violates the namespace, the input is rejected.

# Activation Type

The activation type defines whether the table can be activated directly from the ABAP Dictionary, or whether the runtime object of the table must first be generated with a C program. This entry is **optional** and **only** important for tables of the runtime environment.

The following entries are possible for the activation type:

- Tables having activation type **01** cannot be activated from the ABAP Dictionary. The runtime object must be generated using a C program. The table can then be activated from the ABAP Dictionary. This activation type ensures that important system tables cannot be changed and activated directly.
- Tables having activation type **02** are also used in C programs. The data structure in the C program must therefore be adjusted manually when the table is changed. There is a relevant comment in the activation log for such tables.
- Tables having activation type **10** are needed before R3TRANS runs. Such tables must exist before all other tables when upgrading.
- Tables having activation type **00** can be activated directly from the ABAP Dictionary. This is the default setting for the activation type.

# Making Changes to Tables

## Procedure

1. Enter the table name in the initial screen of the ABAP Dictionary.
2. Select object class *Database table*.
3. Choose *Change.*

The maintenance screen for the table is displayed.

The following describe how to make changes to existing tables:

- Adding an Append Structure [Page 82]
- Inserting an Include [Page 83]
- Inserting New Fields [Page 85]
- Deleting Existing Fields [Page 88]
- Changing the Data Type, Field Length and Decimal Places of Existing Fields [Page 89]
- Changing the Table Category [Page 90]
- Moving Fields [Page 91]
- Copy Fields from Another Table [Page 92]
- Copying Fields from an Entity Type [Page 93].
- Deleting Tables [Page 94]

# Adding an Append Structure

## Procedure

1. Go to change mode in the maintenance screen of the table where you want to add the append structure.

   If the table contains a long field (data type VARC, LCHR or LRAW), you cannot add an append structure.

2. Choose *Goto* → *Append structures*.

   Either a dialog box appears in which you can enter the name of the append structure or a list of all the append structures [Page 69]. that were already created for the table appears. If append structures already exist, you have to choose  to get the dialog box for entering the name.

3. Enter the append structure name and choose  . The name must lie in the customer namespace.

   You go to the field maintenance screen of the append structure.

4. Create the append structure. You can proceed as when creating a normal structure with two restrictions (see Creating Structures [Page 140]).

   The fields of an append structure must lie in the customer namespace, that is the field names must begin with ZZ or YY. This prevents conflicts with fields inserted in the table by SAP.

   An append structure must be flat, that is each field of the append structure must either refer to a data element or be directly assigned a data type, length, decimal places and short text.

5. Choose .

## Result

The table is also activated when the append structure is activated. The fields of the append structure are added to the table in the database.

You can display information about the activation process with *Utilities* → *Activation log*. Errors occurring when the append structure is activated are displayed directly in the activation log.

**See also:**

Append Structures [Page 69]

# Inserting an Include

## Prerequisites

Only flat structures may be included in a table. No field refers to another structure in a flat structure. All the fields of a flat structure therefore refer either to a data element or were directly assigned a data type, field length and decimal places.

## Procedure

1. Place the cursor under the line in which you want to insert the include [Page 16] and choose *Edit* → *Include* → *Insert*.

   A dialog box appears.

2. Enter the structure name. You can optionally enter a group name (see Named Includes [Page 142]) or a three-place suffix.

   With the group name, you can access the fields in the include together in ABAP programs.

   The suffix can be used to avoid name collisions between fields of the include and fields already in the table. The suffix is added to all the fields of the include, whereby the field name is first truncated if necessary.

3. Choose  .

   A line with .INCLUDE in the *Fields* field and the name of the include in the *Field type* field is inserted in the field maintenance screen for the table.

4. Select column *Key* if all the fields in the include should be key fields of the table.

   The table key must be at the start of the field list. If you select column *Key*, you must insert the include after the last key field or between the existing key fields of the table.

   If you do not select column *Key*, none of the included fields is a key field of the table.

5. Choose  .

## Result

The fields of the include are added to the table in the database. If you inserted the fields of the include as key fields, the primary index [Page 61] of the table is built again.

You can find information about the activation flow in the activation log, which you can display with *Utilities* → *Activation log*. The activation log is displayed immediately if errors occur when the table is activated.

## Other Options

You can display the fields contained in an include by placing the cursor on the line of the include and choosing  . The fields of the include are now shown below this line. You can cancel this action with  .

You can expand all the includes contained in a table with  . Cancel this action with  .

### Inserting an Include

You can copy the fields contained in the include directly to the table with *Edit* → *Include* → *Copy components*. The fields of the include become table fields. They are no longer adjusted to changes in the include.

**See also:**

Includes [Page 16]

# Inserting New Fields

## Procedure

1. In the maintenance screen for the table, place the cursor on the field in front of which the new field should be inserted and choose *Insert line*. You can append several new fields at the end of the table with *New lines*.

   An empty line appears in which you can insert the new field. Creating Tables [Page 71]. describes how to add a field to a table.

2. Set the *Init* flag if the new field should be created in the database as NOT NULL.

   In this case the entire is scanned during activation and the new field is filled with the initial value [Page 86]. **This can be very time-consuming for large tables.**

   Key fields are always defined as NOT NULL. When a new key field is inserted, the primary index of the table is built again in the database.

3. Choose .

## Result

The new field is added to the table when it is activated in the database. This does not depend on the position of the new field in the field list of the table, that is the order of the fields in the ABAP Dictionary does not have to be the same as in the database.

If the table is a check table of a foreign key [Page 19], new key fields can only be appended to the previous primary key. When the table is activated, the foreign keys concerned are defined as generic with regard to the new key fields. The foreign keys changed in this way are listed in the activation log.

Inserting a client field results in a table conversion (see Adjusting Database Structures [Page 219]). The data in the table is copied to all the clients listed in client table T000.

If the table is a check table of a foreign key, it is not possible to insert a client field. In this case you must first delete the existing foreign keys.

# Initial Values

If the value of a field in a data record is undefined or unknown, it is called a NULL value. NULL values occur when new fields are inserted in existing tables or with inserts on database views [Page 105].

If a new field is inserted in a table which already exists in the database, this operation is performed with the DDL statement

ALTER TABLE *table name* ADD FIELD *field name* ...

in the database. The new field of records that already exist have a NULL value. If there is an INSERT on a table using a database view, NULL values are inserted in all the fields of the table that are not contained in the view.

NULL values do not have any disadvantages as long as the corresponding field is not selected. If a field with NULL values is selected, some of the entries satisfying the selection condition might not be found. This is because NULL values only satisfy the selection condition WHERE FIELD IS NULL.

Field *Field1* is inserted in table *TAB*. If this table is accessed with the SELECT... FROM *TAB* WHERE *Field1* <> 5... statement, records with NULL values are not found in *Field1* even though they logically correspond to the WHERE condition of the SELECT statement.

The case described in the example can be avoided by defining the inserted field as *Initial*. The field is thus created in the database as NOT NULL. The entire table is scanned when it is activated and the new field is filled with the initial value. This can be very time-consuming! You should therefore only use the Initial flag if it is really needed or if the table only has a few entries.

The initial values depend on the data type of the field.

| Data type | Initial Value |
|---|---|
| ACCP, CHAR, CUKY, LANG, UNIT | ” ” space |
| CURR, DEC, FLTP, INT1, INT2, INT4, QUAN | 0 |
| CLNT | 000 |
| TIMS | 000000 |
| DATS | 00000000 |
| NUMC | 00000... for field length <= 32<br>no initial value for field length > 32 |
| LRAW, LCHR, RAW, VARC | no initial value |

A field can also be created in the database as NOT NULL if the initial flag is **not** set in the ABAP Dictionary! You can find out if a field is defined as NOT NULL in the database with *Utilities* → *Database object* → *Display* in the maintenance screen of the corresponding table.

When a table is created, **all** the fields of the table are defined as NOT NULL and filled with the default value. The same is true when the table is converted. The fields are only created as NOT NULL if new fields are added or inserted. An exception is key fields, as the Initial flag is automatically set here.

If the Initial flag for an include [Page 16] is set, the structure fields whose initial flag is set also have this attribute in the table. If the Initial flag is not set, NULL values are permitted for all the fields of the include.

Fields with data types LCHR, LRAW, RAW and NUMC fields with a field length greater than 32 cannot be defined as Initial.

# Deleting Existing Field

To delete **one** field from a table, place the cursor on the corresponding line and choose *Delete line*. Choose .

If the table in the database already contains data, you have to convert the table after deleting existing fields (see Adjusting Database Structures [Page 219]).

If you delete key fields of the table, data could be lost during conversion. If the table contains data records that only differ in the deleted key field, only one of these data records can be loaded back into the table.

You cannot delete a field used as a reference field in another table. In this case you must first remove all the uses of the field as a reference field (see Reference Fields and Reference Tables [Page 15]).

You cannot delete key fields of the table if the table is the check table. In this case you must first delete the corresponding foreign keys.

You can determine all the tables that use the particular table as check table or reference table with .

# Changing Data Types and Lengths of Existing Fields

If a data element [Page 132] is assigned to the field, you must change the technical field attributes (data type, length, number of decimal places) in the domain of the data element.

## Procedure for Fields with Data Elements

1. Double-click on the name of the data element.

   The data element maintenance screen appears.

2. Double-click on the name of the domain.

   The domain maintenance screen appears.

3. Choose *Domain* → *Display* → *Change*.

   Make the required changes and save your entries.

   Note that this change affects **all** the table fields that refer to the particular domain. Foreign keys could become inconsistent and conversions required. You should therefore consider the effects of this action **before** changing the domain by choosing  in the maintenance screen of the domain and checking if the domain is used in other tables.

4. Choose .

   The table is activated again and the table field is adjusted to the domain change.

## Procedure for Fields with Direct Type Entry

1. Place the cursor on the field and choose *Data element* / *Direct type*.

   Values can now be entered in fields *Data type*, *Length, DecPlaces* and *Short text*.

2. Change the entries for the data type, length and possibly the number of decimal places.

   Save your entries.

3. Choose .

   Note that a table conversion will normally be necessary (see Adjusting Database Structures [Page 219]) if you change the field attributes.

# Changing the Table Category

## Prerequisites

Depending on the category of the change, note the following features:

- **Transparent to structure**: The table is deleted from the database at activation. Existing data is lost. Any technical settings for the table are also lost.
- **Pooled/cluster to transparent**: Since technical settings are meaningless for pooled and cluster tables, you must maintain them before activating the transparent table.
- **Transparent to pooled/cluster:** You must define the assignment to the table pool or table cluster in which the table data should be stored.

## Procedure

1. In the field maintenance screen of the table, choose *Extras* → *Change table category*.

   A dialog box appears in which the current table category (*transparent database table, structure, pooled table, cluster table*) is selected.

2. Select your table category and choose *Select*.

   You return to the field maintenance screen for the table.

3. Choose .

   In addition to the above procedure, you can also change the table category for pooled tables with the transparent flag [Page 42] .

   Changing a pooled/cluster table to a transparent table and vice versa always results in a table conversion (see Adjusting Database Structures [Page 219]).

**See also:**

Pooled and Cluster Tables [Page 256]

# Moving Fields

## Procedure

1. Place the cursor on the field to be moved and choose  .

   The selected field is deleted and copied to the clipboard.

2. Place the cursor on the field in front of which you want to insert the selected field and choose  .

   The field is inserted in the table at the new position.

   The order of the fields in the ABAP Dictionary can differ from the order of the fields in the database. Changing the order of the fields (with the exception of key fields) in the ABAP Dictionary therefore does not result in a change in the database. In particular, a table conversion is not required (see Adjusting Database Structures [Page 219]).

   If the table is a check table, you cannot change the order of the key fields. In this case you must first delete the corresponding foreign keys.

# Copying Fields from Another Table

You can copy fields or blocks of fields from another table.

## Procedure

1. In the maintenance screen of the table choose *Edit* → *Copy fields.*

   Enter the name of the table from which you want to copy the fields in the next dialog box.

2. Choose *Field selection*.

   A list of the fields contained in this table appears.

3. Place the cursor on the first field of the block to be copied and choose *Select block*.

   If you want to copy only single fields, place the cursor on these fields and choose *Select*.

4. Place the cursor on the last field of the block to be copied and again choose *Select block*.

   All the fields of the selected block are highlighted.

5. Choose *Copy*.

   The selected block or the selected single fields are copied to a buffer, and you return to the field maintenance screen.

6. Place the cursor on the field in front of which you want to insert the new fields and choose .

   The selected fields are inserted in the table.

7. Choose .

**See also:**

Inserting New Fields [Page 85]

# Copying Fields from an Entity Type

## Procedure

1. In the field maintenance screen of the table choose *Edit* → *Att. from ent. type.*

   A dialog box appears.

2. Enter the name of the entity type from which you want to copy attributes to the table. Choose *Field selection*.

   A dialog box appears listing all the attributes of the entity type.

3. Select the attributes that you want to copy. Select the corresponding entry in the list and choose *Choose*.

   The entry is highlighted.

   Note that only the *explicit attributes* of the entity type can be copied with this function. Only these explicit attributes are displayed. Explicit attributes are the attributes that were not copied from the table assigned to the entity type. This means that only those attributes that were entered directly in the *Data Modeler* can be copied.

4. Select all the fields you want and choose *Copy*.

   The attributes are now copied to the clipboard.

5. Place the cursor on the field in front of which you want to insert the fields and choose .

   The selected fields are inserted in the table.

**See also:**

Inserting New Fields [Page 85]

Data Modeler: Overview and Modeling Principle [Ext.] (data modeling, entity types)

# Deleting Tables

## Prerequisites

A table can only be deleted in the ABAP Dictionary when it is no longer used in any other objects (for example views or programs).

## Procedure

1. In the initial screen of the ABAP Dictionary, select the object type *Database table* and enter the name of the table.

   With the where-used list  , check if the table is still used in programs or other objects of the ABAP Dictionary.

2. *Choose*  .

   A dialog box appears in which you are asked to confirm the deletion request. This dialog box tells you if the table still contains data.

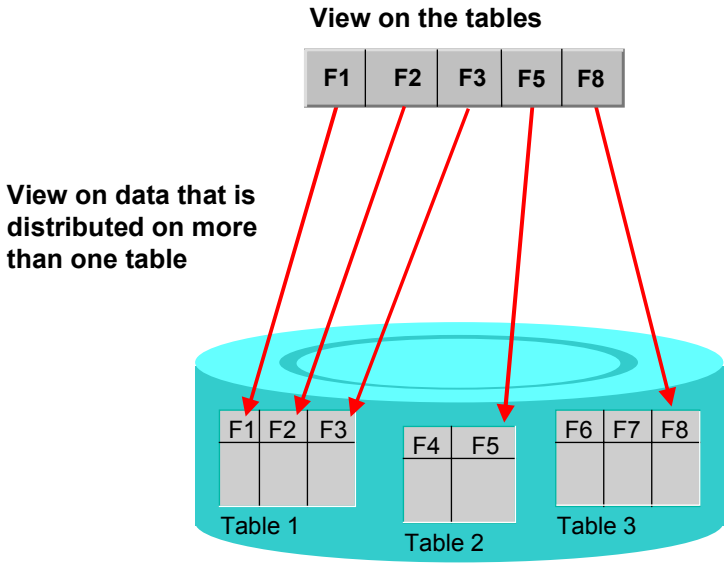
3. Confirm the deletion request.

## Result

The table is now deleted if it is no longer being used in other objects of the ABAP Dictionary.

# Views

Data about an application object is often distributed on several tables. By defining a view, you can define an application-dependent view that combines this data. The structure of such a view is defined by specifying the tables and fields used in the view. Fields that are not required can be hidden, thereby minimizing interfaces. A view can be used in ABAP programs for data selection.




The data of a view is derived from one or more tables, but not stored physically. The simplest form of deriving data is to mask out one or more fields from a base table (projection) or to include only certain entries of a base table in the view (selection). More complicated views can comprise several base tables, the individual tables being linked with a relational join operation. See also Join, Projection and Selection [Page 97].

The base tables of the view must be selected in the first step of a view definition. In the second step, these tables must be linked by defining the join conditions. It is also possible to use the join condition from a foreign key defined between the tables (see Foreign Key Relationship and Join Condition [Page 102]). In the third step, you must select the fields of the base tables to be used in the view. Selection conditions that restrict the records in the view can be formulated in the fourth step.

Four different view types are supported. These differ in the way in which the view is implemented and in the methods permitted for accessing the view data.

- Database views [Page 106] are implemented with an equivalent view on the database.
- Projection views [Page 110] are used to hide fields of a table (only projection).
- Help views [Page 111] can be used as selection method in search helps [Page 166].
- Maintenance views [Page 112] permit you to maintain the data distributed on several tables for one application object at one time.

## Views

Database views implement an **inner join**. The other view types implement an **outer join** (see Inner and Outer Join [Page 101]).

The join conditions for database views can be formulated using equality relationships between any base fields. The join conditions for the other view types must be obtained from existing foreign keys. Tables therefore can only be combined in a maintenance view or help view if they are linked to one another with foreign keys.

The maintenance status [Page 103] defines whether you can only read data with the view or whether you can also insert and change data with it.

**See also:**

Example for Views [Page 127]

Creating Views [Page 115]

Deleting Views [Page 126]

# Join, Projection and Selection

This example shows the structure of a view with the relational operators **join**, **projection** and **selection**.

Given two tables TABA and TABB. Table TABA has 2 entries and table TABB has 4 entries.

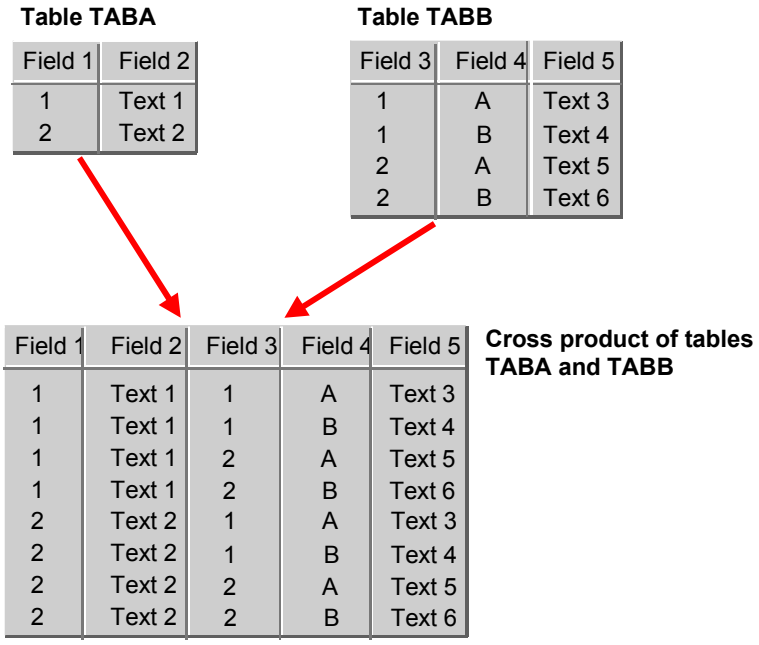


**Table TABA**

| Field 1 | Field 2 |
|---|---|
| 1 | Text 1 |
| 2 | Text 2 |

**Table TABB**

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

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

**Cross product of tables TABA and TABB**

Each record of TABA is first combined with each record of TABB. If a join condition is not defined, the cross product of tables TABA and TABB is displayed with the view.

## Join Conditions

The entire cross product is not usually a sensible selection. The cross product must therefore be restricted with join conditions. A join condition describes how the records of the two tables are connected.

In our example, Field 1 of TABA must be identified with Field 3 of TABB. The join condition is therefore **TABA-Field 1 = TABB-Field 3**. This join condition removes all records for which the entry in Field 1 is not identical to the entry in Field 3 from the cross product. The column for Field 3 in the view is not required.

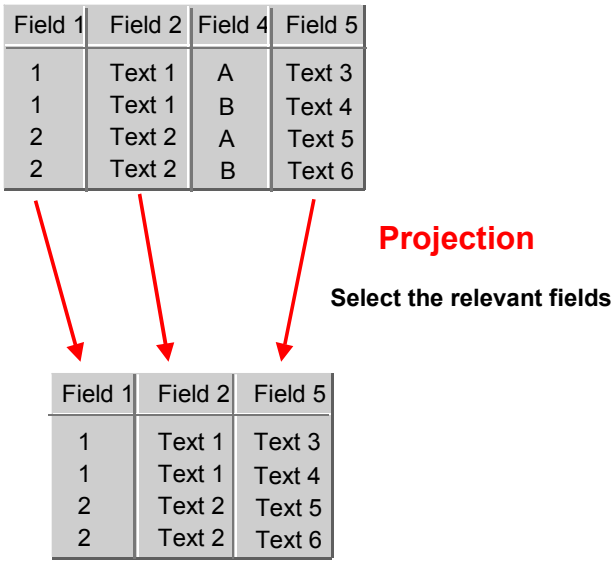
## Join condition: TABA - Field 1 = TABB - Field 3

**Reduce the cross product by all records in which the entry in Field 1 is not the same as the entry in Field 3.**

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

# Projection

Sometimes some of the fields of the tables involved in a view are not of interest. The set of fields used in the view can be defined explicitly (projection). In our example, Field 4 is of no interest and can be hidden.




## Selection Conditions

Selection conditions that are used as a filter can be defined for a view.

Restrictions can be defined for the contents of the view fields in selection conditions. In this case, only the data records that satisfy these restrictions can be selected with the view. In a selection condition, the contents of a view field are compared with a constant using a comparison operator. Several selection conditions can be linked with the logical operators AND and OR.

In our example, only the records that have the value A in Field 4 should be displayed with the view. In this case the selection condition is therefore TABB-Field 4 = 'A'.

A selection condition can therefore be formulated with a field that is not contained in the view.

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

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

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

**See also:**

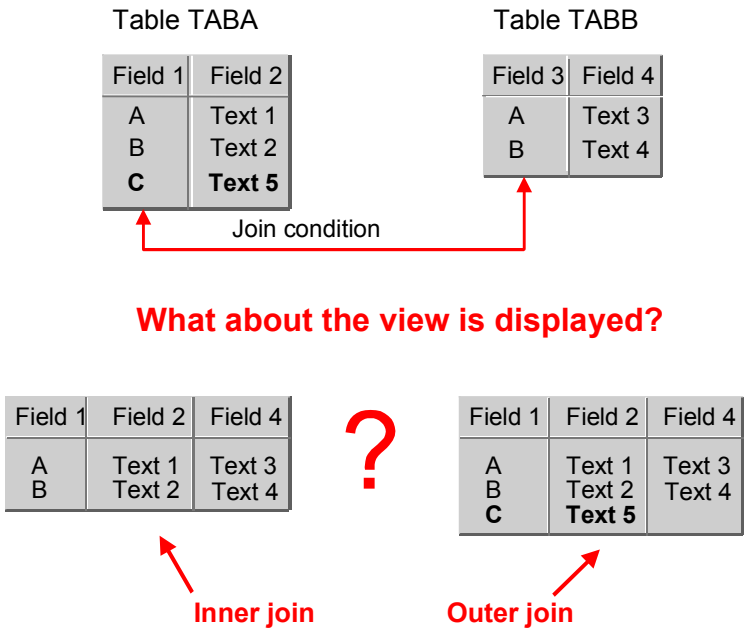
Inner Join and Outer Join [Page 101]

# Inner Join and Outer Join

The data that can be selected with a view depends primarily on whether the view implements an **inner join** or an **outer join**. With an inner join, you only get the records of the cross-product for which there is an entry in all tables used in the view. With an outer join, records are also selected for which there is no entry in some of the tables used in the view.

The set of hits determined by an inner join can therefore be a subset of the hits determined with an outer join.

Database views implement an inner join. The database therefore only provides those records for which there is an entry in all the tables used in the view. Help views and maintenance views, however, implement an outer join.



# Foreign Key Relationship and Join Condition

If there is already a suitable foreign key between two tables used in the view, these tables can be linked with a join condition from this foreign key.

Create a view on tables TAB1 and TAB2. TAB1 is the primary table of the view. TAB2 is the secondary table of the view. TAB1 is the check table for TAB2. The foreign key fields are assigned to the check table fields as follows:

TAB1-FIELD_A assigned to TAB2-FIELD_1
TAB1-FIELD_A assigned to TAB2-FIELD_1

The join condition of the view generated from the foreign key is then:

CREATE VIEW ... AS SELECT ... WHERE TAB2-FIELD_1 = TAB1-FIELD_A AND TAB2-FIELD_2 = TAB1-FIELD_B.

Join conditions can also be copied from generic and constant foreign keys. If a constant is assigned to a field in the foreign key, it is also assigned to the field in the join condition. There is no join condition for a generic relationship in the foreign key.

The foreign key between tables TAB1 (check table) and TAB2 (foreign key table) is defined as follows:

TAB1-FIELD_A assigned to TAB2-FIELD_1
TAB1-FIELD_B generic
TAB1-FIELD_C assigned to constant ‘C’

The join condition for the view generated from the foreign key is in this case:

CREATE VIEW ... AS SELECT ... WHERE TAB2-FIELD_1 = TAB1-FIELD_A AND TAB2-FIELD_2 = ‘C’.

# Maintenance Status

The maintenance status of a view controls whether data records can also be changed or inserted in the tables contained in the view.

The maintenance status can be defined as follows:

- **read only**: Data can only be read with the view.
- **read, change, delete and add**: Data of the tables contained in the view can be changed, deleted and inserted with the view.

Only read access is permitted for database views [Page 106] that are defined over several tables. If a database view contains only one single table, data can be inserted in this table with the view (see Inserts in Database Views [Page 105]).

The following status definitions are possible for maintenance views [Page 112]:

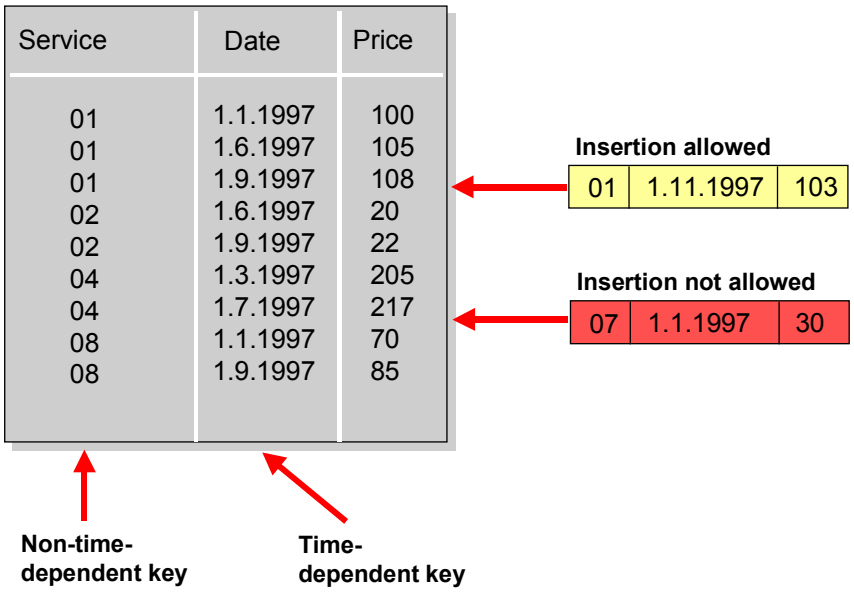
- **read and change**: Existing view entries can be changed. Records cannot be deleted or inserted.
- **read and change (time-dependent views)** : Only entries whose non-time-dependent part [Page 104] of the key is the same as that of existing entries may be inserted.

# Time-Dependent Key Components

The key of some views can be divided into a non-time-dependent and a time-dependent component. The records of these views have a time-dependent meaning.

The prices of services change depending on the date. A view contains the service code, a date field and a field for the price. The field for the service code is the **non-time-dependent component of the key** and the field for the date is the **time-dependent component of the key**.

If maintenance status **read and change (time-dependent views)** was selected for the view, only records that have the same service code as existing records can be inserted with the view. You can also enter a new price for an existing service as of a new date. However, you cannot insert records if they do not yet have a service code. New services therefore cannot be inserted with the view.



# Inserts with Database Views

If a database view contains only one single table, data can be inserted in this table with the view (see maintenance status [Page 103]). You have the following options for the contents of the table fields not contained in the view:

- If the field is defined on the database with NOT NULL as initial value, the field is filled with the corresponding initial value.
- If the field is defined on the database as NOT NULL without initial value, an insert is not possible. This results in a database error.
- If the field is not defined on the database as NOT NULL, there will be a NULL value in this field.

You should therefore only insert data in the table with a database view if initial values [Page 86] are defined for all the table fields not contained in the view.

Changing data records that already exist with a database view is not critical if the database view contains all the key fields of the table.

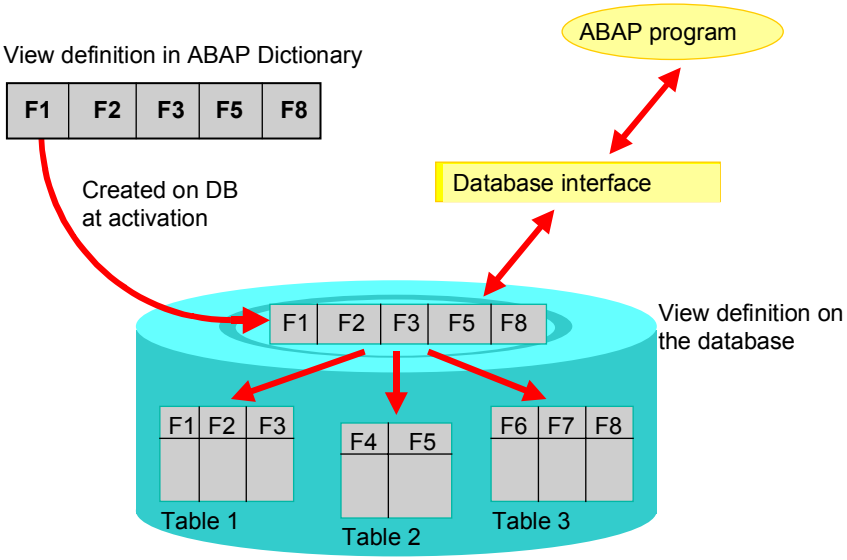
When you insert data records with maintenance views [Page 112] or projection views [Page 110], all the table fields not contained in the view are assigned the default value of the field. This does not depend on whether the field is defined in the database as NOT NULL. This avoids NULL values in these fields.

# Database Views

Data about an application object is often distributed on several database tables. A database view provides an application-specific view on such distributed data.

Database views are defined in the ABAP Dictionary. A database view is automatically created in the underlying database when it is activated.

Application programs can access the data of a database view using the database interface. You can access the data in ABAP programs with both OPEN SQL and NATIVE SQL. However, the data is actually selected in the database. Since the join operation is executed in the database in this case, you can minimize the number of database accesses in this way. Database views implement an **inner join** (see Inner and Outer Join [Page 101]).




If the database view only contains a single table, the maintenance status [Page 103] can be used to determine if data records can also be inserted with the view. If the database view contains more than one table, you can only read the data.

Database views should be created if want to select logically connected data from different tables simultaneously. Selection with a database view is generally faster than access to individual tables. When selecting with views, you should also ensure that there are suitable indexes on the tables contained in the view.

Since a database view is implemented in the database, a database view may only contain transparent tables.

The technical settings of a database view [Page 109] control whether the view data should be buffered.

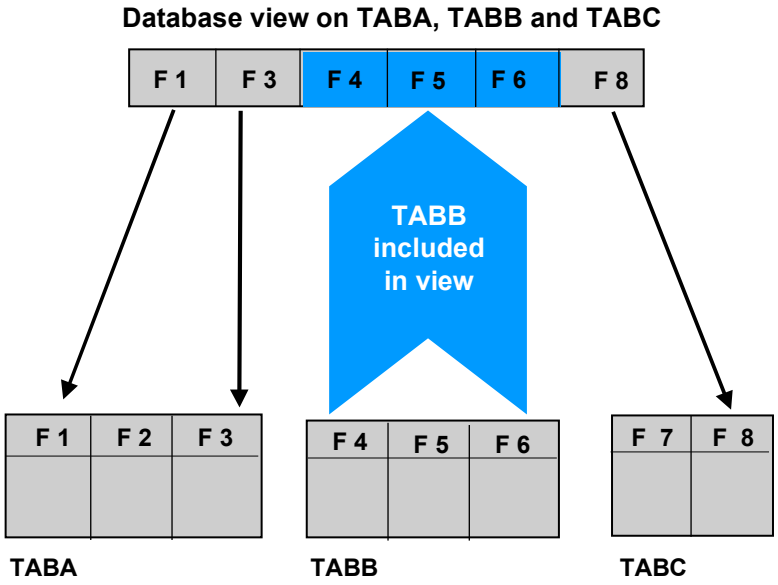
**See also:**

Creating Database Views [Page 116]

Includes in Database Views [Page 108]

# Includes in Database Views

An entire table can be included in a database view. In this case all the fields of the included table will become fields of the view (whereby you can explicitly exclude certain fields). If new fields are included in the table or existing fields are deleted, the view is automatically adjusted to this change. A new or deleted field is thus included in the view or deleted from the view in this case.




To include one of the tables in the view, enter character * in field *View field*, the name of the table to be included in field *Table* and character * again in field *Field name* on the *View fields* tab page of the maintenance screen of the view.

You can also exclude individual fields of an included table. If you do not want to include a field of the included table in the view, enter - in field *View field*, the name of the included table in field *Table* and the name of the field to be excluded in field *Field name*.

# Technical Settings of a Database View

You can buffer the data read from an ABAP program with a database view. View data is buffered completely analogously to table data (see Buffering Database Tables [Page 43]).

The technical settings of a database view control whether view data may be buffered (see Buffering Permission [Page 33]) and how this should be done (see Buffering Types [Page 34]). The same settings are possible as when buffering tables.

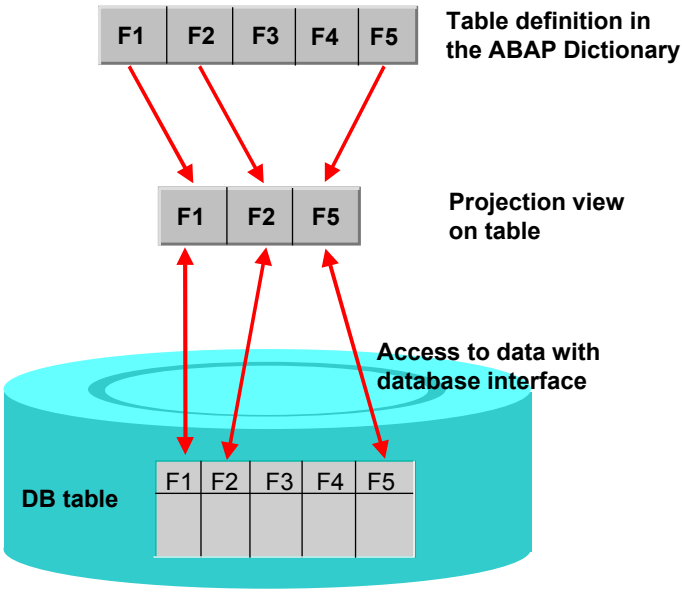
The buffered view data is invalidated as soon as data in one of the base tables of the view changes.

# Projection Views

Projection views are used to hide fields of a table. This can minimize interfaces; for example when you access the database, you only read and write the field contents actually needed.

A projection view contains exactly one table. You cannot define selection conditions for projection views.

There is no corresponding object in the database for a projection view. The R/3 System maps the access to a projection view to the corresponding access to its base table. You can also access pooled tables and cluster tables with a projection view.




The maintenance status [Page 103] of the view controls how the data of the table can be accessed with the projection view.

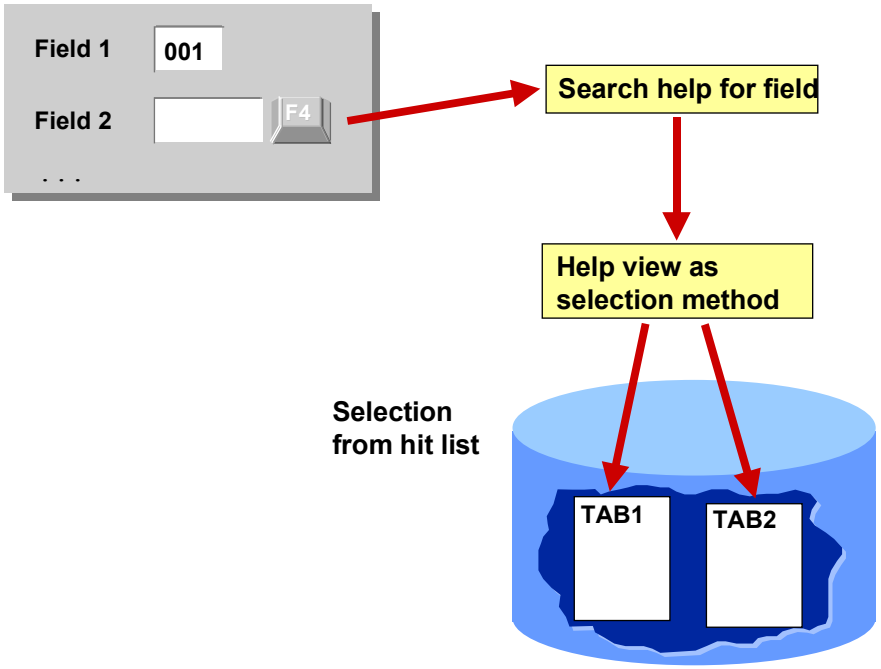
**See also:**

Creating Projection Views [Page 120]

# Help Views

You have to create a help view if a view with outer join [Page 101] is needed as selection method of a search help [Page 166].

The selection method of a search help is either a table or a view. If you have to select data from several tables for the search help, you should generally use a database view as selection method. However, a database view always implements an inner join. If you need a view with outer join for the data selection, you have to use a help view as selection method.




All the tables included in a help view must be linked with foreign keys. Only foreign keys that have certain attributes can be used here (see Restrictions for Maintenance and Help Views [Page 114]).

The functionality of a help view changed significantly between Release 3.0 and Release 4.0! In Release 3.0, a help view was automatically displayed for the input help (F4 help) for all the fields that were checked against the first table (primary table) of the help view. This is no longer the case in Release 4.0.

As of Release 4.0, you must explicitly create a search help that must be linked with the fields for which it is offered (see Linking Search Helps with Screen Fields [Page 176]).

**See also:**

Creating Help Views [Page 118]

# Maintenance Views

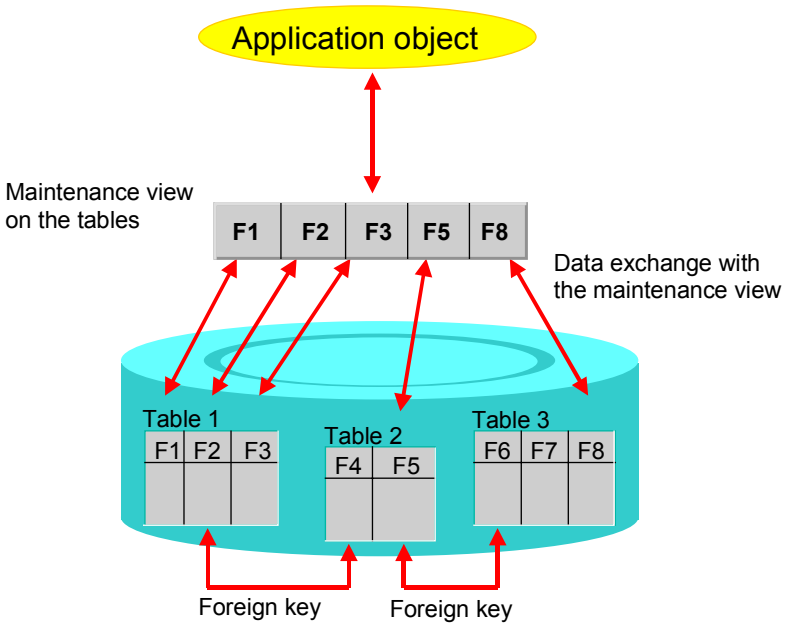
Maintenance views offer easy ways to maintain complex application objects.

Data distributed on several tables often forms a logical unit, for example an application object, for the user. You want to be able to display, modify and create the data of such an application object together. Normally the user is not interested in the technical implementation of the application object, that is in the distribution of the data on several tables.

A maintenance view permits you to maintain the data of an application object together. The data is automatically distributed in the underlying database tables. The maintenance status [Page 103] determines which accesses to the data of the underlying tables are possible with the maintenance view.

All the tables in a maintenance view must be linked with foreign keys, that is the join conditions for maintenance views are always derived from the foreign key (see Foreign Key Relationship and Join Condition [Page 102]). You cannot directly enter the join conditions as for database views.

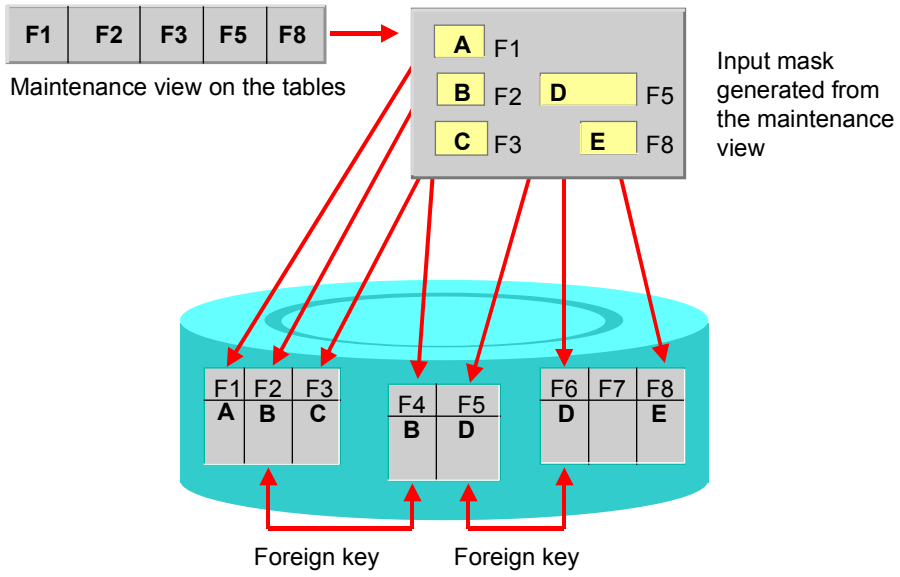
There are some restrictions for the attributes of the foreign keys with which the tables in a maintenance view can be linked (see Restrictions for Maintenance and Help Views [Page 114]).




A standardized table maintenance transaction is provided (SM30), permitting you to maintain the data from the base tables of a maintenance view together.

Maintenance mechanisms, like screens and processing programs, must be created from the view definition with the transaction *Generate Table View* (SE54). This makes it possible to create easy-to-use maintenance interfaces in a simple manner.

You can find out how to create such maintenance mechanisms in the documentation BC - Generate Table Maintenance Dialog [Ext.].



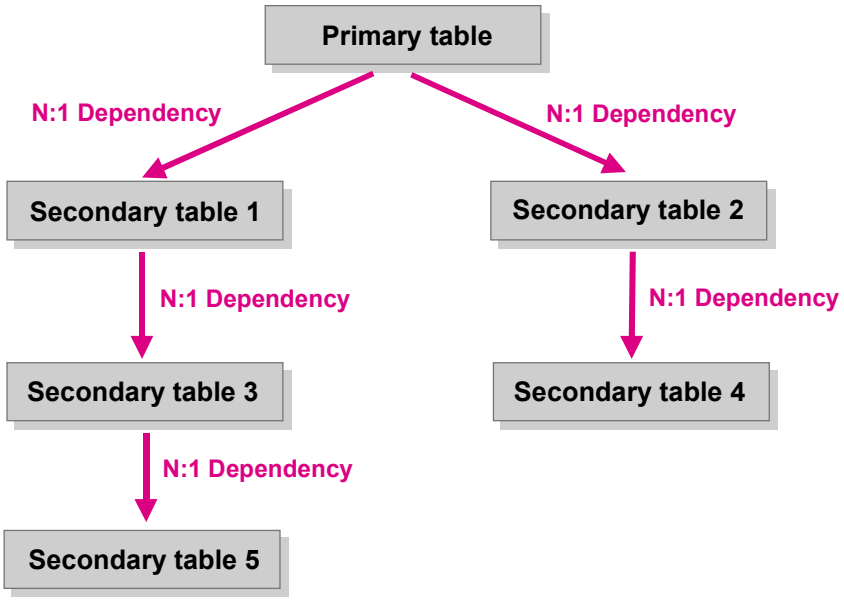

**See also:**

Creating Maintenance Views [Page 121]

Maintenance Status [Page 103]

# Restrictions for Maintenance and Help Views

There are some restrictions for selecting the secondary tables of a maintenance view or help view. The secondary tables have to be in an N:1 dependency to the primary table or directly preceding secondary table. This ensures that there is at most one dependent record in each of the secondary tables for a data record in the primary table.




A N:1 dependency exists if the secondary table is the check table in the foreign key [Page 19] used. If the secondary table is the foreign key table, the foreign key fields must be *Key fields of a table* or the foreign key must have the cardinality N:1 or N:C (see Semantic Attributes of a Foreign Key [Page 24]).

# Creating Views

The procedure for creating a view depends on the view type.

## Procedure

1. Select object class *View* in the initial screen of the ABAP Dictionary, enter the view name and choose *Create*.

   A dialog box appears in which you must select the view type.

2. You go to the maintenance screen of the selected view type with *Choose*.

The procedure to be followed (depending on the view type) is described in:

- Creating Database Views [Page 116]
- Creating Maintenance Views [Page 121]
- Creating Help Views [Page 118]
- Creating Projection Views [Page 120]

# Creating a Database View

## Procedure

1. Enter an explanatory short text in the field *Short text*.

   You can for example find the view at a later time using this short text.

2. Define the tables to be included in the view in the *Tables* field of the *Tables/Join conditions* tab page.

   Keep in mind that you can only include transparent tables in a database view.

3. Link the tables with join conditions [Page 97].

   If there are suitable foreign keys between the tables, you should copy the join conditions from these foreign keys (see Foreign Key Relationships and Join Conditions [Page 102]).

   Place the cursor on a table name and choose *Relationships*. All foreign keys to other tables defined for this table are displayed. Select the foreign keys and choose *Copy*. The join condition is now derived from the definitions in the foreign key.

   If you only want to see the foreign key relationship existing between two tables, you must first select these two tables (click on the first column of the input area *Tables*) and then choose *Relationships*.

4. On the *View fields* tab page, select the fields that you want to copy to the view.

   Choose *Table fields*. All the tables contained in the view are displayed in a dialog box. Select a table. All the fields contained in this table are displayed. You can copy fields by selecting them in the first column and choosing *Copy*.

   You can also include an entire table in the view (see Includes in Database Views [Page 108]).

5. On the *Selection conditions* tab page, you can (optionally) formulate restrictions for the data records to be displayed with the view (see Maintaining Selection Conditions for Views [Page 125]).

   The selection conditions define the data records that can be selected with the view.

6. With *Goto* → *Technical settings,* you can (optionally) maintain the technical settings of the database view.

   You can define whether and how the database view should be buffered here. Proceed as for the technical settings of a table (see Maintaining Technical Settings [Page 76]). Note that only the settings for buffering can be maintained for database views.

7. On the *Maintenance status* tab page, select the maintenance status [Page 103] of the database view.

   If the view contains more than one table, the maintenance status *read only* cannot be altered.

8. *Save* your entries. You are asked to assign the view a development class.

   You can change this development class later with *Goto* → *Object directory entry*.

9. Choose .

# Result

When a database view is activated, the corresponding view is also automatically created in the database if the base tables of the view were already created there.

At activation, a log is written; it can be displayed with *Utilities* → *Activation log*. If errors or warnings occurring when the view was activated, they are displayed directly in the activation log.

If the base tables are not yet created in the database, this is recorded in the activation log. The view is nevertheless activated in the ABAP Dictionary. In this case you can create the relevant view on the database later with the database utility.

## Other Options

- **Create documentation:** You can create information about using the view with *Goto* → *Documentation*. This documentation is also output for example when you print the view.
- **Change data element of a view field:** Select the *Mod* (Modify) column in the *View fields* tab page. The *Data element* field is now ready for input. You can enter a data element that refers to the same domain as the data element of the assigned table field here. Cancel the *Mod* flag if you want to use the data element of the assigned table field again.
- **Display view data:** With *Utilities* → *Contents* you can determine which data can be selected with the view.
- **Display create statement:** With *Extras* → *CREATE statement* you can display how the view was created in the database. The statement that was executed when the version of the view currently being edited was created in the database is displayed.
- **Check the definition of the view in the database:** With *Utilities* → *Database object* → *Check* you can determine whether the definition of the view in the database is consistent with the active version of the view. With *Utilities* → *Database object* → *Display* you can display the definition of the view in the database.
- **Check the runtime object of the view:** With *Utilities* → *Runtime object* → *Check* you can determine whether the definition of the view in the ABAP Dictionary maintenance screen is identical to the specifications in the runtime object of the view. With *Utilities* → *Runtime object* → *Display* you can display the runtime object of the view.

**See also:**

Database Views [Page 106]

# Creating Help Views

## Procedure

1. Enter an explanatory short text in the field *Short text*.

   You can for example find the view at a later time using this short text.

2. Enter the primary table of the view under *Tables* in the *Tables/Join conditions* tab page.

   Only tables that are linked with the primary table (indirectly) with a foreign key can be included in the view.

3. Save your entries.

   You are asked to assign the help view a development class. You can change this development class later with *Extras* → *Object directory entry*.

4. If required, include more tables in the view. In a help view you can only include tables that are linked to one another with foreign keys.

   Position the cursor on the primary table and choose *Relationships*. All existing foreign key relationships of the primary table are displayed. Select the foreign keys and choose *Copy*. The secondary table involved in such a foreign key is included in the view. The join conditions derived from the foreign keys (see Foreign Key Relationship and Join Condition [Page 102]) are displayed.

   You can also include tables that are linked with a foreign key to one of the secondary tables already included. To do this, place the cursor on the secondary table and choose *Relationships*. Then proceed as described above.

   For maintenance and help views, there are certain restrictions on the foreign keys with which the tables can be included in the view (see Restrictions for Maintenance and Help Views [Page 114]). The foreign keys violating these conditions are displayed at the end of the list under the header *Relationships with unsuitable cardinality*.

5. On the *View fields* tab page, select the fields that you want to copy to the view. The key fields of the primary table were automatically copied to the view as proposals.

   Choose *Table fields*. All the tables contained in the view are listed in a dialog box. Select a table. The fields of the table are now displayed in a dialog box. Select the required fields in the first column and choose *Copy*.

6. On the *Selection conditions* tab page, you can (optionally) formulate restrictions for the data records to be displayed with the view (see Maintaining Selection Conditions for Views [Page 125]).

   The selection conditions define the data records that can be selected with the view.

7. Choose .

## Result

The view is now activated. At activation, a log is written; it can be displayed with *Utilities* → *Activation log*. If errors or warnings occurring when the view was activated, they are displayed directly in the activation log.

## Other Options

- **Create documentation:** You can create information about using the view with *Goto* → *Documentation*. This documentation is output for example when you print the view.
- **Change data element of a view field:** Select column *Mod* (modification) for the view field. The *Data element* field is now ready for input. Enter the new data element there. You can enter a data element that refers to the same domain as the data element of the assigned table field here. Cancel the *Mod* flag if you want to use the data element of the assigned table field again.
- **Check functions:** With *Extras* → *Runtime object* → *Check* you can determine whether the definition of the view in the ABAP Dictionary maintenance screen is identical to the definitions in the runtime object of the view. With *Extras* → *Runtime object* → *Display* you can display the runtime object of the view.

**See also:**

Help Views [Page 111]

# Creating Projection Views

## Procedure

1. Enter an explanatory short text in the field *Short text*.

   You can for example find the view at a later time using this short text.

2. Enter a table name in the field *Base table*.

   A projection view always contains exactly one table.

3. Select the fields of the base table that you want to include in the view.

   Choose *Table fields*. The fields of the table are now displayed in a dialog box. You can copy fields by selecting them in the first column and choosing *Copy*.

4. *Save* your entries.

   You are asked to assign the view a development class. You can change this development class later with *Goto* → *Object directory entry*.

5. Choose .

## Result

The help view is activated. At activation, a log is written; it can be displayed with *Utilities* → *Activation log*. If errors or warnings occurring when the view was activated, they are displayed directly in the activation log.

## Other Options

- **Create documentation:** You can create information about using the view with *Goto* → *Documentation*. This documentation is output for example when you print the view.
- **Change data element of a view field:** Select column *Mod* (modification) for the particular view field. The *Data element* field is now ready for input. You can enter a data element that refers to the same domain as the data element of the assigned table field here. Cancel the *Mod* flag if you want to use the data element of the assigned table field again.
- **Change maintenance status:** The maintenance status [Page 103] determines how you can access the view data from ABAP programs (read only, read and change). Choose *Extras* → *Maintenance status*. A dialog box appears in which you can select the maintenance status of the view.
- **Check functions:** With *Utilities* → *Runtime object* → *Check* you can determine whether the definition of the view in the ABAP Dictionary maintenance screen is identical to the specifications in the runtime object of the view. With *Utilities* → *Runtime object* → *Display* you can display the runtime object of the view.

**See also:**

Projection Views [Page 110]

# Creating Maintenance Views

## Procedure

1. Enter an explanatory short text in the field *Short text*.

   You can for example find the view at a later time using this short text.

2. Enter the primary table of the view under *Tables* in the *Tables/Join conditions* tab page.

   Only those tables that are linked with the primary table (indirectly) with a foreign key can be included in the maintenance view.

3. If required, include more tables in the view. In a maintenance view you can only insert tables that are linked to one another with foreign keys.

   Place the cursor on the primary table and choose *Relationships*. All existing foreign key relationships of the primary table are displayed. Select the required foreign key and choose *Copy*. The secondary table used in such a foreign key is included in the view. The join conditions derived from the foreign keys (see Foreign Key Relationship and Join Condition [Page 102]) are displayed.

   You can also insert tables that are linked by foreign key with one of the secondary tables that was already inserted. To do this, place the cursor on the secondary table and choose *Relationships*. Then proceed as described above.

   For maintenance and help views, there are certain restrictions on the foreign keys with which the tables can be included in the view (see Restrictions for Maintenance and Help Views [Page 114]). The foreign keys violating these conditions are displayed at the end of the list under the header *Relationships with unsuitable cardinality*.

4. On the *View fields* tab page, select the fields that you want to copy to the view.

   Choose *Table fields*. All the tables contained in the view are displayed in a dialog box. Select a table. The fields of the table are now displayed in a dialog box. You can copy fields by selecting them in the first column and choosing *Copy*.

   All key fields of the primary table must be included in a maintenance view. In addition, all key fields of secondary tables that are not involved in the foreign key (that is, which are not linked via a join condition to a key field already included in the view) must be included in the view.

   This ensures that the records inserted with a maintenance view can be written correctly in the tables contained in the view.

5. On the *Selection conditions* tab page, you can (optionally) formulate restrictions for the data records that can be displayed with the view (see Maintaining Selection Conditions for Views [Page 125]).

   The selection conditions define the data records that can be selected with the view.

6. In the *Maintenance status* tab page, define the maintenance status [Page 103] of the view.

   The maintenance status defines how you can access the view data with the standard maintenance transaction (SM30).

7. Choose .

At activation, a log is written; it can be displayed with *Utilities* → *Activation log*. If errors or warnings occurring when the view was activated, the activation log is automatically displayed.

8. Go to Transaction SE54 with *Environment* → *Tab.maint.generator*.

   From the view definition you can generate maintenance modules and maintenance interfaces that distribute the data entered with the view to the base tables of the view. You can find more information in Creating a Maintenance Dialog [Ext.].

## Other Options

- **Create documentation:** You can create information about using the view with *Goto* → *Documentation*. This documentation is output for example when you print the view.
- **Change the data element of a view field:** Select column *Mod* (modification) for the view field. The *Data element* field is now ready for input. You can enter a data element that refers to the same domain as the data element of the assigned table field here. Cancel the *Mod* flag if you want to use the data element of the assigned table field again.
- **Define the delivery class of the view:** In the *Maintenance status* tab page, select the delivery class [Page 124] of the maintenance view.
- **Define the maintenance attribute of a view field** In the *View fields* tab page you can define the maintenance attributes [Page 123] of the view field in column *F*.
- **Check functions:** With *Extras* → *Runtime object* → *Check* you can determine whether the definition of the view in the ABAP Dictionary maintenance screen is identical to the definitions in the runtime object of the view. With *Extras* → *Runtime object* → *Display* you can display the runtime object of the view.

**See also:**

Maintenance Views [Page 112]

BC - Generating the Table Maintenance Dialog [Ext.]

# Maintenance Attribute of a View Field

You can control how to access a field of a maintenance view with the maintenance attribute. There are the following maintenance attributes:

- **R** : Only pure read accesses are allowed for such fields. You cannot maintain such fields with Transaction SM30.
- **S** : These fields are used to create subsets when maintaining view data. Only a subset of the data is displayed. This subset is defined by entering a suitable value in this field.
- **H** : These fields are hidden from the user in the maintenance dialog. The field does not appear in the maintenance screen. You have to ensure that such fields have the correct contents. By default they are left empty.
- : There are no restrictions for the field maintenance.

# Delivery Class of a Maintenance View

The delivery class of a maintenance view is used in the Extended Table Maintenance (SM30). If a maintenance interface is generated for the maintenance view, the following information is analyzed when view data is entered for this interface:

- For maintenance views having delivery class E or G, there is a check if the entered data satisfies the namespace defined in table TRESC for the view.
- There is a check if the transport connection built into the generated table maintenance makes sense. For example, there is no transport for maintenance views with delivery classes L and W.

The delivery class of the corresponding base table [Page 78] of the view alone defines how the data entered in a base table of the view is handled during an upgrade and during transport between customer systems.

# Maintaining Selection Conditions of Views

Once you have defined the base tables and fields of the view, you can restrict the set of data records that can be selected with the view by defining a selection condition (see Join, Projection and Selection [Page 97]).

## Procedure

1. You can enter a condition having the following form in each line of the *Selection conditions* tab page:

   **Table, Field name, Operator, Comparison value, AND/OR**

2. The entries have the following meaning:

   **Table:** Name of the base table from which the field was taken.

   **Field name:** Name of the field for which the selection condition was formulated.

   **Operator:** Operator for comparing the field contents and comparison value. You can find the valid operators with the F4 help.

   **Comparison value:** Constant value with which the field value is compared. Text literals, which must be enclosed in apostrophes, and numbers are permitted as comparison values, depending on the data type of the field.

   **AND/OR:** Link between two lines of the selection condition.

   You can enter the fields for which you want to define selection conditions directly or copy them with *Table fields*.

3. If you enter more than one selection condition, you must link them with AND or OR.

   OR operations are only possible between lines referring to the same field. Note that OR takes priority over AND. The condition <COND1> AND <COND2> OR <COND3> is therefore interpreted as <COND1> AND (<COND2> OR <COND3>).

4. Save your entries for the selection condition(s).

   You can formulate selection conditions for all the fields of the table contained in the view. It is not important whether or not these fields were included in the view.

# Deleting Views

## Prerequisites

You should only delete a view if it is no longer used in programs.

## Procedure

1. In the initial screen of the ABAP Dictionary, select object type *View* and enter the name of the view.

   With  , check if the view is still being used in programs.

2. Choose  .

   A dialog box appears in which you are asked to confirm the deletion request.

3. Confirm the deletion request.

## Result

The view is deleted in the ABAP Dictionary and in the database.

# Example for Views

Travel agencies often have to check what customer is booked on what flights (see the Flight Model [Page 295]). The corresponding data is distributed on several tables:

- **SCUSTOM:** Customer data such as the customer number, name, address.
- **SBOOK:** Booking data such as the carrier, flight number, passenger (customer number).
- **SPFLI:** Flight data such as the city of departure and destination.

You can get an overall view of all existing bookings by creating a view on tables SCUSTOM, SBOOK and SPFLI.

You can determine all the bookings in table SBOOK for a customer number (ID) for the customer number in table SCUSTOM. Using the carrier ID (CARRID) and the flight number (CONNID), the flight information can be read from table SPFLI for a booking that is found.

This results in the following join conditions for the view:

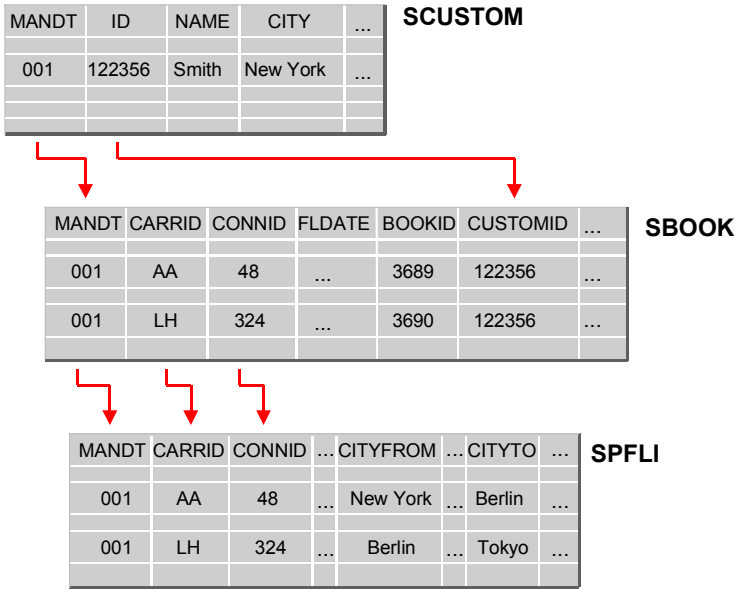
SBOOK-MANDT = SCUSTOM-MANDT

SBOOK-CUSTOMID = SCUSTOM-ID

SPFLI-MANDT = SBOOK-MANDT

SPFLI-CARRID = SBOOK-CARRID

SPFLI-CONNID = SBOOK-CONNID

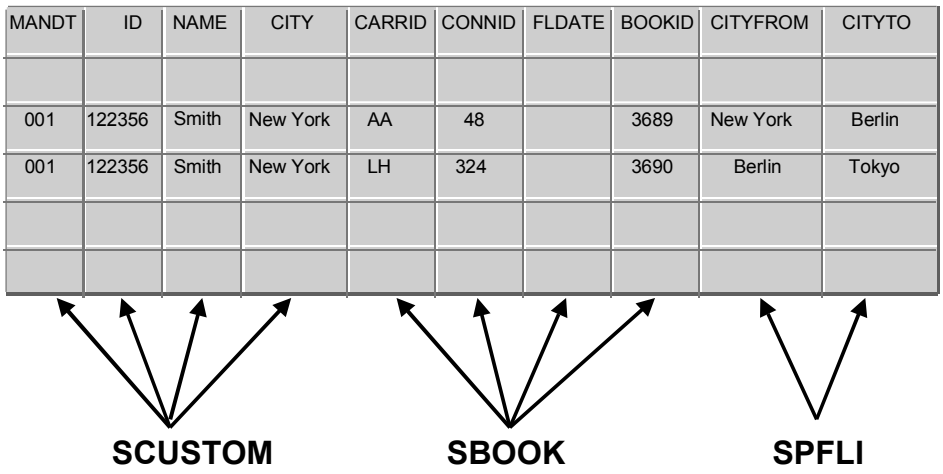



The join conditions can also be derived from the existing foreign key relationships between the tables of the view (see Foreign Key Relationship and Join Condition [Page 102]).

**Example for Views**

If you only want to display the customer bookings that were not canceled with the view, you can do this with the selection condition **SBOOK-CANCELLED <> 'X'**.

## Structure of View SCUS_BOOK for Customer Flight Bookings




## Data Selection with View SCUS_BOOK

A view can be used to select data in an ABAP program.

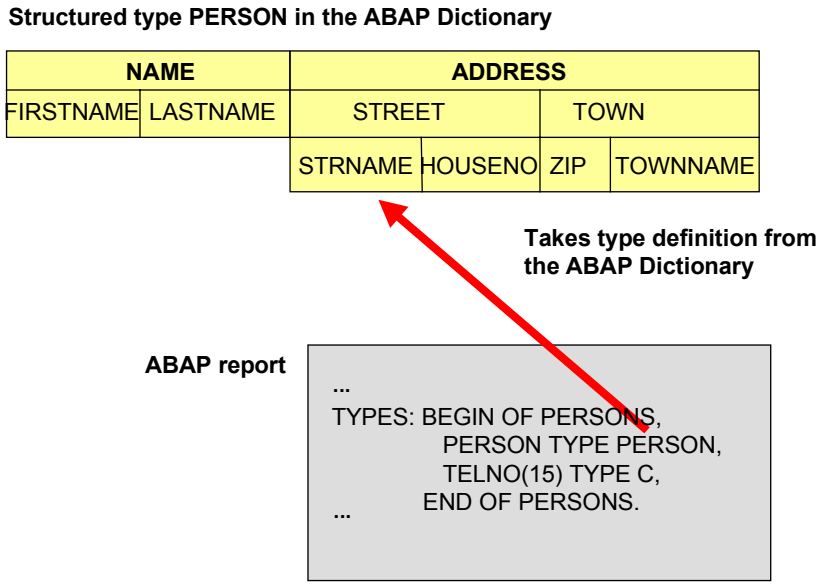
The following example program determines the existing flight bookings for a customer. The data is selected with view SCUS_BOOK.

```
REPORT CUSBOOK1.

PARAMETERS: CUSTOMID LIKE SBOOK-CUSTOMID.
TABLES: SCUS_BOOK.

WRITE: / 'Existing Bookings for Customer',
CUSTOMID,':'.

SELECT * FROM SCUS_BOOK WHERE CUSTOMID = CUSTOMID.
 WRITE: / 'CUSTOMER', SCUS_BOOK-NAME, 'booked on',
  SCUS_BOOK-CARRID,SCUS_BOOK-CONNID, 'from',
  SCUS_BOOK-CITYFROM, 'to',SCUS_BOOK-CITYTO,
  'on',SCUS_BOOK-FLDATE.
ENDSELECT.

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

# Types

User-defined data types can be stored for all programs in the ABAP Dictionary. These user-defined types provide the same functionality as the local types that can be defined in ABAP programs with TYPES (see Data Types and Data Objects [Ext.]).

The types defined globally in the ABAP Dictionary can be accessed in ABAP programs with TYPE. You can also refer to the types defined in the ABAP Dictionary when defining the type of a function module interface.




The central definition of types that are used more than once in the ABAP Dictionary permits them to be changed centrally. The active ABAP Dictionary ensures that such changes are made at all the relevant locations. ABAP programs, for example, are adjusted to the modified type definitions when they are recreated. If a type is changed, all the objects (e.g. types or tables) that use this type are determined at activation. The objects that are found are automatically adjusted to the change.

All types have a runtime object [Page 232]. This runtime object is generated the first time the type is activated and is adjusted to the current type definition at each further activation.

You can enter semantic information for a type in the type definition in the ABAP Dictionary. This includes for example text that is displayed for the F1 help, text for use in screens, search helps, and technical documentation.

There are three different type categories:

- Data elements [Page 132] (elementary types and reference types).
- Structures [Page 138] (structured types): A structure consists of components that also have a type, that is they refer to a type.

- Table types [Page 143]: A table type describes the structure and functional attributes of an internal table. A special case is the ranges table types [Page 151].

All the types lie in a common namespace. For example no structure or table type having the same name can be created for a data element.

The types defined locally in a program override the global types from the ABAP Dictionary having the same name.

# Data Elements

A data element describes either an *elementary type* or a *reference type*.

An elementary type is defined by the built-in data type, length and possibly the number of decimal places. These type attributes can either be defined directly in the data element or copied from a domain [Page 155].

A reference type defines the type of the reference variables containing pointers to objects or interfaces. A reference type is defined by specifying an existing class or existing interface. You can also define a generic reference to objects or data objects.

You can use a data element to define the type of a table field, structure component or the row type of a table type. A data element can also be referenced in ABAP programs with TYPE. As a result, variables that take on the attributes of a data element can be defined in an ABAP program.

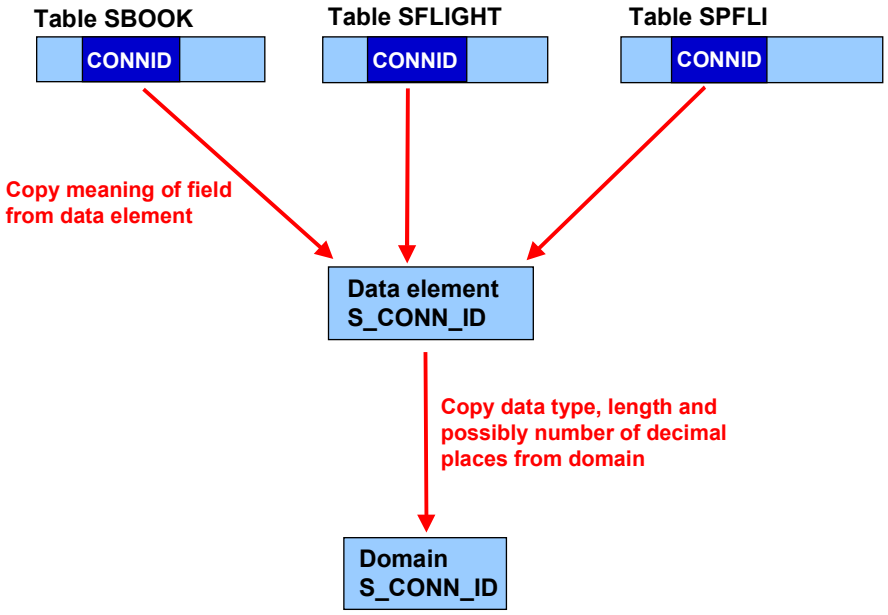
Information about the meaning of a table field or structure component and information about editing the corresponding screen field can be assigned to a data element. This information is automatically available to all screen fields that refer to the data element.

This information includes the display of the field in input templates using key word text, column headers for list output of the table contents (see Field Labels [Page 137] ), and output editing using parameter IDs.

This is also true for online field documentation. The text appearing in the field help (F1 help) in a field of an input template (see Documentation and Docu Status [Page 136]) comes from the corresponding data element.

The field CONNID (flight class) of table SBOOK refers to the data element S_CONN_ID. This data element gets its technical attributes (data type NUMC, field length 4) from domain S_CONN_ID. Data element S_CONN_ID describes the technical attributes and meaning (with an assigned long text and an explanatory short text) of field CONNID (and all other fields that refer to this data element).

A variable having the type of data element S_CONN_ID can be defined in an ABAP program with the statement DATA CONNID TYPE S_CONN_ID.




**See also:**

Creating Data Elements [Page 134]

# Creating Data Elements

## Procedure

1. In the initial screen of the ABAP Dictionary, select object type *Data type*, enter the data element name and choose *Create*.

   A dialog box appears.

2. Select the data element and choose .

   The maintenance screen for data elements appears.

3. Enter an explanatory short text in the field *Short text*.

   The short text appears as title in the F1 help for all the screen fields referring to this data element.

4. On the *Definition* tab page, define the data type [Page 235] , *number of places* and possibly the number of *decimal places* of the data element. You can define these attributes by specifying a domain or by direct type entry.

   If the data element should have the type attributes of a domain, you only have to select *Domain* and enter the domain name in the corresponding field. You can also define a new domain and create it by navigating to the domain maintenance screen by double-clicking (see Creating Domains [Page 162]).

   If you want to enter the type attributes directly, select *Direct type entry*. Entries are now possible in the fields *Data type*, *Number of places* and *Decimal places*.

   If the data element should be a reference to a class or an interface, select *Reference type*. Enter the name of the class or interface in the *Reference* field. You can also enter OBJECT or DATA if the data element should implement a generic reference to objects or data objects.

5. On the Field label [Page 137] tab page you can (optionally) maintain text information (short, medium, and long field labels and the title) for the data element.

   You can use this text information in input templates to represent fields that refer to this data element.

6. Save the data element.

   You are asked to assign the data element a development class.

7. Choose .

## Result

The data element is activated. You can find information about the activation flow in the activation log, which you can display with *Utilities* → *Activation log*. If errors occurred when activating the data element, the activation log is automatically displayed.

## Other Options

- **Create documentation:** With *Documentation*, create a text that describes the contents of the data element. This text is displayed for the F1 help in all screen fields that refer to this data element. You should therefore only leave out this step if the data element does

not appear on a screen. In this case you should set the documentation status [Page 136] appropriately.

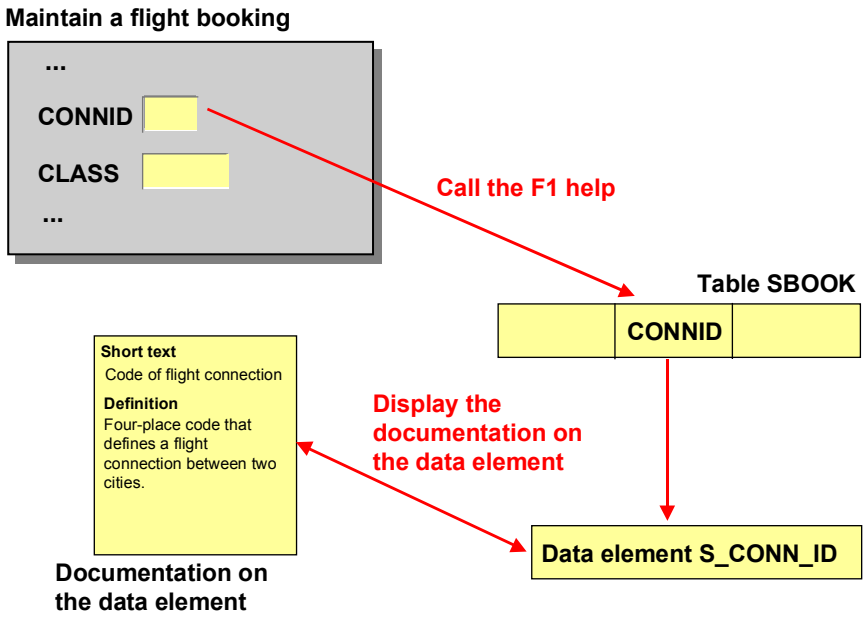
- **Assign a search help:** You can assign the data element a search help [Page 166]. This search help is offered on all the screen fields referring to this data element when the input help (F4 help) is called (see Attaching Search Helps to Data Elements [Page 177]). To assign the search help, you must specify its *Name* in the data element maintenance screen and an export parameter of the search help in the *Parameter* field.
- **Assign a parameter ID:** A field can be filled with default values from SAP memory using a parameter ID. A screen field is only filled automatically with the value stored under the parameter ID of the data element if this was explicitly permitted in the Screen Painter.

  For example, if a user only has authorization for company code 001, this company code can be stored in memory under the corresponding parameter ID at the beginning of a transaction. Fields referring to the data element for the company code are then automatically filled with the value 001 in all subsequent screen templates. In this case, the corresponding parameter ID must be only entered in the data element for the company code.
- **Assign a default component name:** You can store a proposal for the name of the table fields or structure components that refer to a data element. If possible, use an English-language default name. Always use this default name for components in BAPI structures (structures with a fixed interface). This results in a more unified assignment of field and component names.
- **Mark data element as relevant for change documents:** The data of a business-oriented object is often distributed on several tables. To be able to trace changes to this business object, these tables can be combined in a *Change document object*. Function modules that can be integrated in the corresponding application programs and that log the changes are generated from such a change document object. Changed field contents are only logged if the *Change document* flag is set for the data element of the field.

# Documentation and Docu Status

Documentation entered for a data element is displayed when you select *F1 Help* on each screen field that refers to the data element. If there is no documentation for the data element, only the short text of the data element appears for the F1 help. Data elements that are used in input templates should normally be documented.




The documentation status specifies whether documentation has already been written for a data element and whether documentation is in fact required. To display the docu status of a data element in the maintenance screen of the data element, choose *Goto* → *Documentation* → *Status*.

The following *status entries* are possible:

- *Object should be documented:* The standard setting. Documentation either already exists or should be written.
- *Object is not used in any screens:* The object is not used in any screen. Documentation does not exist and is not required.
- *Object is explained sufficiently by the short text:* The short text explains the object adequately. Documentation does not exist and is not required.
- *Documentation postponed:* This selection might be appropriate, for example, if the use of the data element has not yet been fully clarified. Documentation does not (yet) exist.

# Field Labels

You can assign text information to a data element with the *field labels*.

The field labels are used to display a screen field. You can copy structure components and table fields from the ABAP Dictionary to the input template when you define a screen in the *Screen Painter* (*Get from Dict.* function). You can also assign a field a field label that is derived from the data element of the field.

The text of the field label can be translated centrally with the translation tools. The text is then displayed in the input template in the user's logon language.

### Short, Medium and Long Field Labels

Keywords of different lengths for identifying screen fields. Since the space available for such texts depends on the input template, you can define the text information in three different lengths.

### Headers

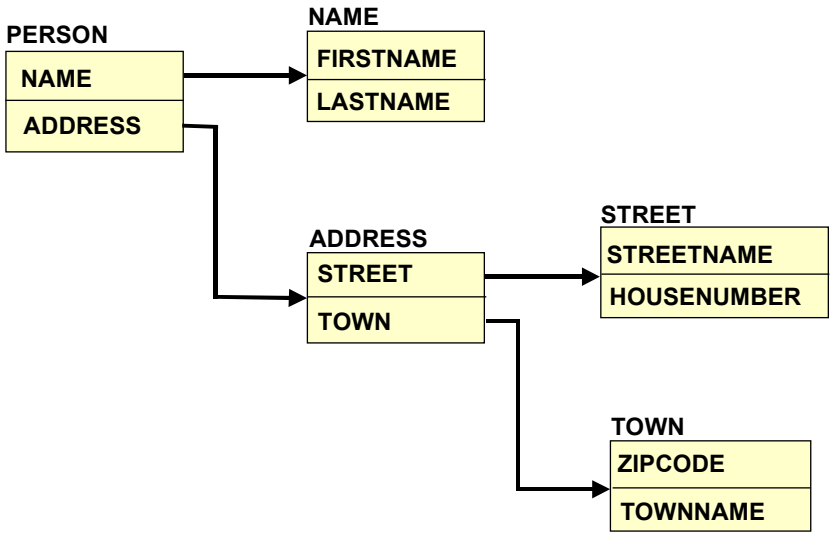
This text is used as a column header when you output the contents of fields as a list in the Data Browser. It can also be assigned to a field in the Screen Painter.

# Structures

A structure (structured type) comprises components (fields). Types are defined for the components A component can refer to an elementary type (via a data element or by directly specifying the data type and length in the structure definition), another structure or a table type. A structure can therefore be nested to any depth.

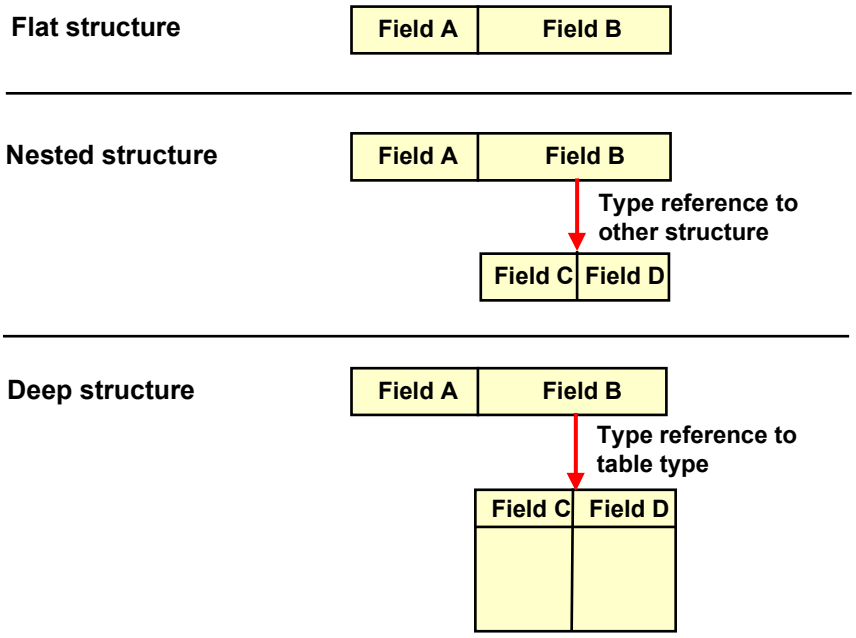
Structures are used to define the data at the interface of module pools and screens and to define the parameter types of function modules.

The data for managing the addresses of persons can be represented as a nested structure PERSON. The structure comprises the components (structures) NAME and ADDRESS. The structure NAME comprises the components (data elements) FIRSTNAME and LASTNAME. The structure ADDRESS comprises the components (structures) STREET and TOWN. The structure STREET comprises the components (data elements) STREETNAME and HOUSENO. The structure TOWN comprises the components (data elements) ZIP and TOWNNAME.




The central definition of structures that are used more than once makes it possible for them to be changed centrally. The active ABAP Dictionary then makes this change wherever required. ABAP programs or screen templates that use a structure are automatically adjusted when the structure is changed (see Runtime Objects [Page 232]). This ensures the greatest possible consistency of the data definition, also for complex programs.

There are *Flat*, *nested* and *deep* structures. A flag structure only references elementary types. A nested structure references at least one further structure, but not a table type. A deep structure references at least one table type.




In a database table, you can only include flat structures as substructures [Page 83].

**See also:**

Creating Structures [Page 140]

# Creating Structures

## Procedure

1. In the initial screen of the ABAP Dictionary, enter the structure name in field *Data type* and choose *Create*.

   A dialog box appears in which you must select the type category.

2. Select *Structure* and choose .

   The maintenance screen for structures appears.

3. Enter an explanatory short text in field *Short text*.

   You can for example find the structure at a later time using this short text.

   Carry out the following steps for all the components you want to include in the structure.

   You can also include structures, tables or views instead of individual components. The procedure here is described in Inserting an Include [Page 83].

4. Enter a name in column *Components*.

   If you want to use the structure as an include [Page 16] in a transparent table at a later time, the component names may not be longer than 16 characters.

5. Enter the name of the type whose attributes should be used in the component in field *Component name*. You can enter any type (data element, structure or table type) here.

   You can also add components by directly specifying the data type and length. Choose *Built-in type*. You can now enter values for fields *DTyp*, *Length, Dec.places* and *Short description*. With *Component type* you can switch back to the screen for entering references to existing types.

   You can combine components with direct type definition and components that are defined by referencing an existing type as you like.

6. The reference field and reference table [Page 15] must be specified for components of type CURR (currency) and QUAN (quantity).

   You can make these entries on the *Currency/quantity fields* tab page.

7. Now maintain the foreign keys of the structure.

   Proceed as when maintaining the foreign keys of a table (see Creating Foreign Keys [Page 74]).

8. Save your entries once you have defined all the components of the structure.

   You are asked to assign the structure a development class. You can change the development class later with *Goto* → *Change object directory entry*.

9. Choose .

## Result

The structure is now activated. The runtime object of the structure is created. At activation, a log is written; it can be displayed with *Utilities* → *Activation log*. If errors occurred when the structure was activated, the activation log is automatically displayed.

## Other Options

- **Create documentation:** You can enter information about using the structure with *Goto* → *Documentation*. This documentation is also output when the structure is printed.
- **Assign a search help:** You can assign the structure a search help. This search help is offered on all the screen fields referring to this structure field when the input help (F4 help) is called (see Attaching Search Helps to Table Fields [Page 181]). Proceed as follows:

1. Choose *Goto* → *Search help*.
2. In the next dialog box, enter the search help name and choose ✔
3. The system automatically makes a proposal for the assignment of the search help parameters to structure fields. An attempt is made to assign a structure field with the same domain to a search help parameter. If there is no such field, the assignment is open. The assignment is displayed in a dialog box.
4. You can change the system proposal (except for the search field, that is the field to which the search help was assigned) by assigning a structure field with the same data type and same field length to a search help parameter.
5. Choose ✔*Copy*.

- **Define the activation type:** The activation type [Page 80] defines whether the structure can be activated directly from the ABAP Dictionary, or whether the runtime object of the structure must first be generated with a C program. It is only important to define an activation type for structures of the runtime environment. You can define the activation type with *Extras* → *Activation type*.

# Named Includes

If an include [Page 16] is used to define a database table or structure, a name can be assigned to the included substructure. The group of fields in the include can be addressed as a whole in ABAP programs with this name.

In ABAP programs, you can either access the fields directly with *<Table/structure name>-<Field name>* or analogously with *<Table/structure name>-<Group name>-<Field name>*. You can access the fields of the group as a whole with *<Table/structure name>-<Group name>*.

Structure PERSON includes structure ADDRESS with the name ADR. ADDRESS has a field CITY. With PERSON-ADR you can address all the fields in structure ADDRESS. The included field CITY can also be addressed with PERSON-CITY or PERSON-ADR-CITY.

You can include a structure more than once (e.g. in a period group). Since direct access by field name should be permitted here, the included field names must be renamed to ensure that they are unique.

A suffix can be assigned to each group, extending the names of the group fields. The fields can then be addressed in ABAP programs with *<Table/structure name>-<Field name (with suffix)>* or *<Table/structure name>-<Group name>-<Field name (with suffix)>*.

Structure PERSON includes structure ADDRESS twice. An address is the private address with suffix H and name ADRH. The other address is the business address with suffix W and name ADRW. You can access field CITY in the private address with PERSON-CITYH or PERSON-ADRH-CITY.

The functionality of the named includes in the ABAP Dictionary corresponds to the ABAP construction INCLUDE TYPE ... AS ... RENAMING ... .

# Table Types

A table type describes the structure and functional attributes of an internal table in ABAP. In ABAP programs you can reference a table type *TTYP* defined in the ABAP Dictionary with the command DATA <*inttab*> TYPE *TTYP*. An internal table <*inttab*> is created in the program with the attributes defined for *TTYP* in the ABAP Dictionary.

A table type is defined by:

- its line type, that defines the structure and data type attributes of a line of the internal table
- the options for managing and accessing the data (access mode [Page 148]) in the internal table
- the key (key definition [Page 149] and key category [Page 147]) of the internal table

The row type is defined by directly entering the *data type*, *length* and *number of decimal places* or by referencing a data element [Page 132], structured type (structure [Page 138], table [Page 13] or view [Page 95]) or other table type.

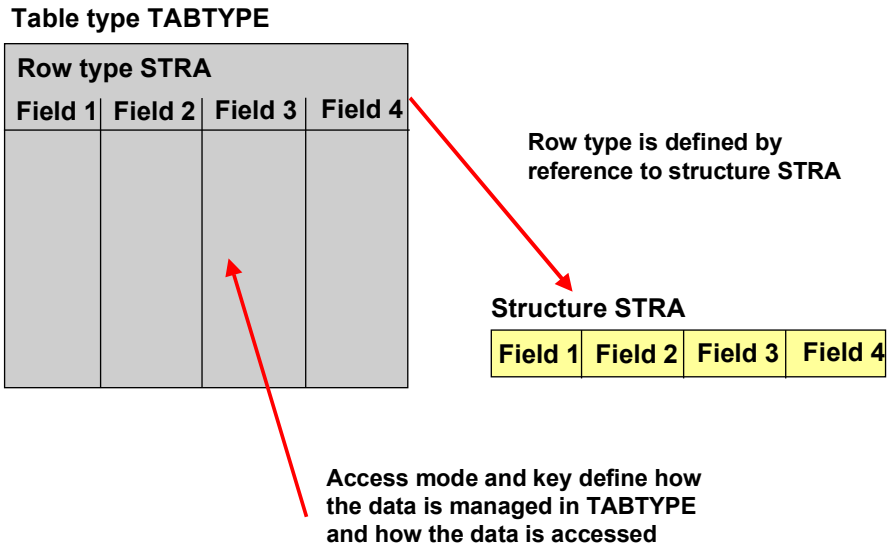



Table type TABTYPE shown in the graphic can be used with the statement DATA <*name*> TYPE TABTYPE in ABAP programs to define an internal table <*name*>.

A local type <*name*> that takes on the attributes of type TABTYPE can be defined in the program with TYPES <*name*> TYPE TABTYPE.

A special case of table types are the ranges table types [Page 151].

## Table Types

**See also:**

Creating Table Types [Page 145]

# Creating Table Types

## Procedure

1. In the initial screen of the ABAP Dictionary, enter the table type name in field *Data type* and choose *Create*.

   A dialog box appears in which you must define the type category.

2. Select *Table type* and choose *Choose*.

   The maintenance screen for table types appears.

3. Enter an explanatory short text in the field *Short text*.

   You can for example find the table type at a later time using this short text.

   Now define the structure of a row of the table type (row type), the options for accessing data in the defined internal table (access mode) and the key of the table type.

4. Enter the row type of the table type on the *Row type and Access* tab page. You can refer to an existing type (data element, structure, table, view, table type) or directly enter the row type.

   If you want to refer to an existing type, you must set the *Row type* flag and enter the name of the type in the field below it.

   If you want to enter the *data type*, *field length* and possibly the *number of decimal places* directly, you must set the *Built-in type* flag. Entries are now possible in the fields *Data type*, *Number of places* and *Decimal places*.

5. Define the access mode [Page 148] of the table type.

   This defines how to access the data in an internal table defined by the table type in ABAP programs.

6. Define the key of the table type on the *Key* tab page.

   The key of a table type is defined by the key definition [Page 147] and the key category [Page 149].

   If you select *Key components*, you can define the key of the table type directly in the input area with the same name. This option is only possible if the row type of the table type is a structure, table or view.

   You can display all the components of the row type with *Select components*. Select the components you want to copy to the key and choose  .

7. Save the table type.

   You are asked to assign the table type a development class. You can change this development class later with *Goto* → *Object directory entry*.

8. Choose .

## Result

The table type is now activated. The runtime object [Page 232] of the type is created. At activation, a log is written; it can be displayed with *Utilities* → *Activation log*. If errors occurred during activation, the activation log is automatically displayed.

## Other Options

- You can enter an explanatory text for the table type with *Goto* → *Documentation*. This is purely technical documentation that is not displayed in the online help system.
- You can display the runtime object of the table type with *Utilities* → *Runtime object* → *Display*. With *Utilities* → *Runtime object* → *Check* you can check if the runtime object of the table type is consistent with its definition in the maintenance screen of the ABAP Dictionary.

# Key Definition of a Table Type

When a table type is defined, the key to be used for the table type must be specified.

You have the following options:

- *Standard key*: The structure of the key depends on the row type category. In a structured row type, the standard key consists of all the character-like components of the table row. In an elementary row type or reference type as row type, the standard key consists of the entire table row. If the row type is a table type, the standard key is empty. Note that an empty key is only allowed for access mode *Standard table*.
- *Row type*: The key consists of all the fields of the row type.
- *Key components:* The key is explicitly defined by selecting components (fields) of the row type. This is only possible if a structure, table or view was selected as row type.
- *Key not specified*: The key is not specified. This defines a generic table type [Page 150].

# Access Mode

The access mode defines how to access the data in the internal table defined by the table type when performing key operations (READ TABLE, INSERT TABLE, MODIFY TABLE, COLLECT). In particular, it defines whether key accesses (see Keys of Table Types [Page 147]) to the internal table are allowed.

Possible access modes are:

- **Standard table:**

  The key access to a standard table uses a sequential search. The time required for an access is linearly dependent on the number of entries in the internal table.

  You should usually access a standard table with index operations.

- **Sorted table:**

  The table is always stored internally sorted by its key. Key access to a sorted table can therefore use a binary search. If the key is not unique, the entry with the lowest index is accessed. The time required for an access is logarithmically dependent on the number of entries in the internal table.

  Index accesses to sorted tables are also allowed. You should usually access a sorted table using its key.

- **Hash table:**

  The table is internally managed with a hash procedure. All the entries must have a unique key. The time required for a key access is constant, that is it does not depend on the number of entries in the internal table.

  You cannot access a hash table with an index. Accesses must use generic key operations (SORT, LOOP, etc.).

- **Index table:**

  The table can be a standard table or a sorted table.

  Index access is allowed to such an index table. Index tables can be used to define the type of generic parameters of a FORM (subroutine) or a function module.

- **not specified:**

  The table can be a standard table, a sorted table or a hash table. The set of valid operations on such a table is the intersection of the valid operations for these three access modes.

  You cannot access tables of this type with index operations.

  If *Index table* or *Not specified* is selected for the access mode, a generic table type [Page 150] that cannot be used to define data objects or types in programs is created.

# Key Category

The key category defines whether the internal table defined by the table type may only contain keys with a unique key or whether the key may have duplicates.

The following key categories may be defined:

- *unique:* This table type may only contain records with a unique key.
- *non-unique:* A table with this table type may also contain records that do not have different keys for the table type.
- *not specified:* The key category is unique or non-unique. This defines a generic table type [Page 150].

Only certain combinations of access mode [Page 148] and key category are allowed. These are listed in the following table.

| Access Mode | Key category |
|---|---|
| Not specified | *Not specified* |
| Index table | *Not specified* |
| Standard table | *Non-unique* |
| Sorted table | *Unique*, *non-unique* or *not specified* |
| Hash table | *Unique* |

There is one exception: If the key of a standard table or hash table is not specified, you have to select key category *not specified*.

# Generic Table Types

A generic table type does not define all the attributes of an internal table in the ABAP Dictionary; it leaves some of these attributes undefined.

A table type is generic in the following cases:

- *Index table* or *not specified* is selected for the access type,
- *Not specified* is selected for the key definition,
- *Not specified* is selected for the key category.

Generic table types are used to define the types of generic table parameters in function modules and forms.

If a generic table type with the *index table* access type is used as the parameter of a function module, you can pass either a sorted table or a standard table in the call.

Generic table types offer a degree of freedom in the arguments passed in the corresponding calls.

Since generic table types do not define all the necessary attributes of an internal table, they cannot be used to define data objects (with DATA) or types (with TYPE).

# Ranges Table Types

A ranges table type is a special case of a table type. A ranges table type describes the structure of an internal table for administering complex areas, i.e. the type of an internal ranges table in the ABAP program.

Ranges tables can be used for example in logical conditions (IN operators) in the SELECT, IF, WHILE and CHECK statements or to pass data to selection tables.

The row type of a ranges table type has a fixed structure. The row type consists of the 4 components SIGN (sign), OPTION (comparison operator), LOW (lower limit) and HIGH (upper limit) in this order.

The type of components LOW and HIGH is defined by an elementary associated type. It can be defined by specifying a data element or by directly defining the data type, number of places and if necessary the number of decimal places.

A ranges table type always has *Standard table* access mode and a *standard key* that is *non-unique*.

**See also:**

Creating Ranges Table Types [Page 152]

# Creating a Ranges Table Type

1. In the initial screen of the ABAP Dictionary, enter the name of the ranges table type [Page 151] in field *Data type* and choose *Create*.

   A dialog box appears in which you must select the type category.

2. Select *Table type* and choose ✔.

   The maintenance screen for table types appears.

3. Enter an explanatory short text in the field *Short text*.

   You can for example find the ranges table type at a later time using this short text.

4. Choose *Edit* → *Define as ranges table type*.

   You go to the maintenance screen for ranges table types. The access mode [Page 148], key definition [Page 147] and key category [Page 149] are defined in advance for a ranges table type, so you do not need to define them.

   To define the ranges table type, you only have to specify an elementary type for the types of the LOW and HIGH components of the corresponding row type.

   If you want to define the types of the LOW and HIGH components using an existing data element, you must select *Data element* and enter the name of the data element in the subsequent field. You can also enter the name of a data element that does not yet exist and go directly to the data element maintenance screen by double-clicking to create it.

   You can also define the elementary type of the LOW and HIGH components directly. Select *Built-in type*. You can now enter values for fields *Data type*, *Number of places* and *Decimal places.*

5. Now enter a name for the row type of the ranges table type in field *Structured row type*.

   A ranges table type must have a row type, just like any other table type. This row type always has a fixed structure for the special case of ranges table types. You can therefore generate the row type from the maintenance screen of the ranges table type.

6. *Save* your entries.

   You are asked to assign the ranges table type a development class. If needed, you can change this development class later with *Goto* → *Object directory entry*.

7. Choose *Create.* You can generate the row type of the ranges table type with this function.

   The maintenance screen for structures is displayed. The components of the row type are already assigned based on your definition of the associated type for the LOW and HIGH components. You only have to enter a short text for the row type and *Activate* it. Then choose *Back*.

8. Choose .

## Result

The ranges table type is now activated. The runtime object [Page 232] of the type is created. At activation, a log is written; it can be displayed with *Utilities* → *Activation log*. If errors occurred during activation, the activation log is displayed immediately.

# Deleting Types

## Prerequisites

Note that you can only delete a type (data element, structure, table type) in the ABAP Dictionary when it is no longer used in other objects (for example in tables, structures or programs).

## Procedure

1. In the initial screen of the ABAP Dictionary, select the name of the type (data element, structure, table type) in field *Data type*.

   With  you can get an overview of where the type is still in use. Make sure that the type is no longer being used in other objects.

2. Choose  .

   A dialog box appears in which you are asked to confirm the deletion request.

3. Confirm the deletion request.

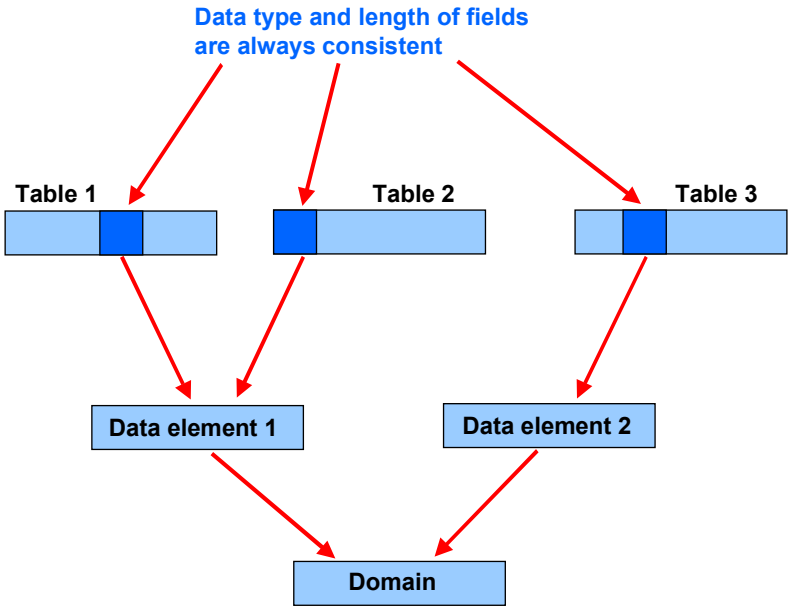
## Result

The type is deleted if it is no longer being used in other objects.

# Domains

A domain defines a value range. A domain is assigned to a data element. All table fields or structure components that use this data element then have the value range defined by the domain. The relationship between the field or component and the domain is thus defined by the data element of the field or component.

Fields or components that refer to the same domain (with the assigned data elements) are also changed when the domain is changed. This ensures that the value ranges of these fields or components are consistent. Fields or components that are technically the same can thus be combined with a reference to the same domain.




The value range of a domain is defined by specifying a data type and length (and number of decimal places for numeric data types).

A personnel number is defined by the data format NUMC and by specifying the number of places for this personnel number.

The value range of a domain can be restricted by defining fixed values [Page 157]. If all the fields or components that refer to the domain should be checked against a certain table, this table can be defined as the value table [Page 158] of the domain.

Output attributes can also be defined for all the fields or components that refer to the domain (see Creating Domains [Page 162]). A conversion routine [Page 159] can be assigned to a domain. This conversion routine converts values from display format to internal format for the fields or components that refer to this domain.

## Domains

**See also:**

Creating Domains [Page 162]

Changing Domains [Page 164]

Deleting Domains [Page 165]

# Fixed Values

The value range of a domain can be further restricted by defining fixed values. If fixed values are defined for a domain, these are used in the input check in screen templates. If no other means of help is defined for a field (search help [Page 166], foreign key [Page 19]), the fixed values are also offered in the input (F4) help.

Domain S_CLASS (data type CHAR, length 1) in the flight model [Page 295] describes the possible classes of a flight booking. The value range of domain S_CLASS is defined by the fixed values C (business class), F (first class) and Y (economy class). Only the values C, F and Y may be entered in screen templates for all the fields that refer to this domain.

You can define *fixed value intervals* either by entering upper and lower limits or by specifying *single values*. Value ranges and single values can be combined as required. You can enter an explanatory text for every single value or interval; it is displayed in the input help.

It is only possible to define fixed values for domains of data types CHAR, NUMC, DEC, INT1, INT2 and INT4. There is only an input check of the template for data types CHAR and NUMC.

# Value Table

In some cases you can see when you define a domain that all the table fields or structure components referring to this domain should be checked against a certain table. This information can be stored in the domain by entering a value table.

The system proposes the value table as check table when you try to define a foreign key [Page 19] for the field or component. This proposal can be overridden.

Domain S_CARR_ID (data type CHAR, length 3) in the flight model [Page 295] describes the three-place code of the airlines. All the airlines are listed together with their codes in table SCARR. It is generally advisable to check fields referring to domain S_CARR_ID against table SCARR. SCARR is therefore entered as value table for domain S_CARR_ID. If you want to define a foreign key for a field referring to S_CARR_ID, SCARR is proposed as the check table.

A check is not implemented by simply entering a value table! The check against the value table only takes effect when a foreign key has been defined.

# Input and Output Conversions

Depending on the data type of the field, there is a conversion when the contents of a screen field are converted from display format to SAP-internal format and vice versa. If this standard conversion is not suitable, it can be overridden by defining a conversion routine in the underlying domain.

Conversion routines are identified by a five-place name and are stored as a group of two function modules. The function modules have a fixed naming convention. The following function modules are assigned to conversion routine xxxxx:

- CONVERSION_EXIT_xxxxx_INPUT
- CONVERSION_EXIT_xxxxx_OUTPUT

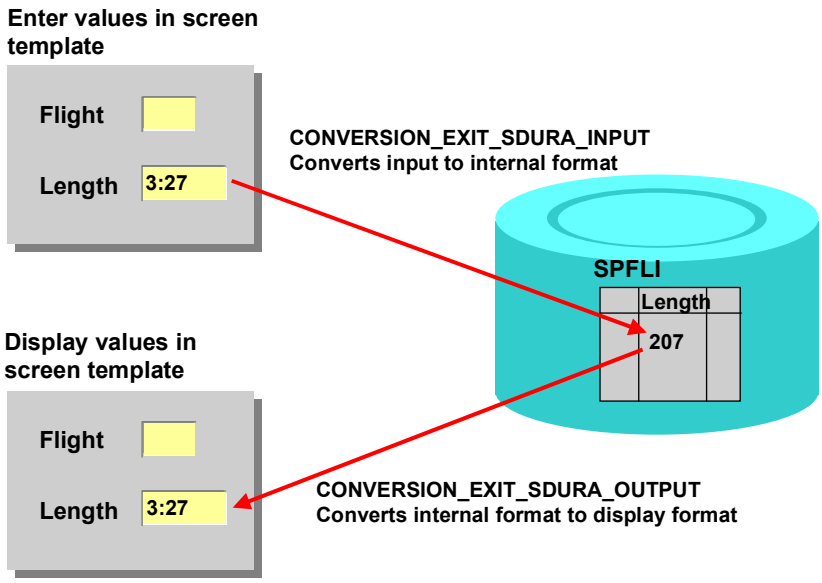
The INPUT module converts from display format to internal format, and the OUTPUT module converts from internal format to display format.

## When is a Conversion Routine Executed?

If a screen field refers to a domain with a conversion routine, this conversion routine is executed automatically when entries are saved in this screen field or when values are displayed in this screen field. The conversion routine of the domain is also triggered when the field contents are output with the WRITE statement.

Table SPFLI in the flight model [Page 295] contains information about the flights offered by the carriers. The time for each flight is recorded in field FLTIME. Enter and display the time of the flight in input templates in the form HHH:MM (hours:minutes). Store the flight time entered in the database as an integer number (number of minutes of the flight). An entry 3:27 is therefore stored in the database as 207 (3 hours, 27 minutes = 207 minutes).

Field FLTIME refers to domain S_DURA, to which conversion routine SDURA is assigned. The value is thus converted by the two function modules CONVERSION_EXIT_SDURA_INPUT and CONVERSION_EXIT_SDURA_OUTPUT.




A conversion routine can also be triggered by specifying its five-place name in the attributes of a field in the *Screen Painter* or with the addition *USING EDIT MASK <Name of conversion routine>* in the WRITE command in the program. With the *USING NO EDIT MASK* addition in the WRITE statement, you can skip a conversion routine defined for a domain when outputting.

## Parameters

The two function modules of a conversion routine must have precisely two parameters with the names INPUT and OUTPUT for the value to be converted and the converted value.

The INPUT parameter in the INPUT conversion and the OUTPUT parameter in the OUTPUT conversion should not have any reference fields because the value passed in the call could have a different length than that expected.

## Programming Conversion Routines

ABAP statements that result in an interruption of processing (such as CALL SCREEN, CALL DIALOG, CALL TRANSACTION, SUBMIT, COMMIT WORK, ROLLBACK WORK, MESSAGE I, MESSAGE W) are **not allowed** in conversion routines.

Only A messages are meaningful in output conversion, but A, E and S messages can be triggered in input conversion (although S messages are not very meaningful here). E messages result in an error dialog. Exceptions are not intercepted in the call.

The output conversion is also triggered with WRITE and WRITE TO. The conversion routine may therefore occur very frequently with lists. The output conversion should therefore be programmed as efficiently as possible.

No external performs should be used in conversion routines. Programs that are called externally use the table work areas of the first calling main program. In conversion routines this can result in

errors that cannot be easily analyzed since they are sometimes called at unpredictable times in the program flow.

# Creating Domains

Before creating a new domain, check whether a domain that defines the same value range already exists. In this case you should use the existing domain if possible.

## Procedure

1. Select object type *Domains* in the initial screen of the ABAP Dictionary, enter the name of the domain and choose *Create*.

   The maintenance screen for domains appears.

2. Enter an explanatory short text in the field *Short text*.

   You can for example find the domain at a later time using this short text.

3. On the *Data type* tab page, choose the data type [Page 235], *number of places* (valid positions without editing characters such as comma or period) and number of *decimal places* (only needed for data types DEC, FLTP, QUAN and CURR).

   Note that some data types have a fixed length. For example, the data type CLNT (client) always has 3 places. If you enter an invalid number of places for such a data type, the system corrects this automatically after issuing a warning.

4. If only certain input values are valid for the domain, you can enter them in the *Value range* tab page as fixed values [Page 157].

   You can also define a value table [Page 158] as proposed value for foreign key checks on this tab page.

5. Save the domain.

   You are asked to assign the domain a development class.

6. Choose .

## Result

The domain is activated. You can find information about the activation flow in the activation log, which you can call with *Utilities* → *Activation log*. If errors occurred when the domain was activated, the activation log is automatically displayed.

## Other Options

- **Create documentation:** You can create technical documentation about the domain with *Goto* → *Documentation*.
- **Restrict output length:** The value for the output length (maximum field length including editing characters such as comma and period) is automatically computed from the definitions for the data type, number of places and number of decimal places. If fields that refer to this domain should be output on the screen or in lists with only a certain length, you can reduce this size.
- **Distinguish uppercase/lowercase:** If you want to distinguish uppercase and lowercase for the fields referring to this domain, you have to select *Lowercase*. Otherwise, all the entered letters are converted to uppercase in the database. Entries can only made for this field in data types CHAR and LCHR.

- **Assign conversion routine:** If field values have to be converted during input or output, you can define a conversion routine [Page 159].
- **Output with sign:** If fields that refer to this domain can contain negative values, the *Sign* flag must be set. When the field contents are output to the screen, the first output location is reserved for a sign. If the flag is not set but the field contains negative values, problems could occur when outputting to the screen. Entries can only made in this field for data types DEC, FLTP, QUAN and CURR.

# Changing Domains

## Procedure

In the initial screen of the ABAP Dictionary select object type *Domain*, enter the domain name and choose *Change*.

All the tables and structures in which a field or component refers to the particular domain are affected by changes made to the domain.

A conversion of these tables could be necessary (see Adjusting Database Structures [Page 219]). This could be very time-consuming for tables containing a large number of records. Foreign keys might also become inconsistent in such tables and structures. You should therefore find out what effects the change will have using the where-used list before you change a domain.

### Changing the Data Type, Number of Places or Number of Decimal Places

You can change the data type and number of places or decimal places simply by overwriting the relevant data. Note that a change of this type can cause all tables containing a field referring to the changed domain to be converted.

You should therefore find out all the database tables in which a field refers to the domain before making such a change. To do this, choose and then *Indirect use* in the next dialog box. A list appears. Select *DB tables* and choose . All the database tables in which a field refers to the domain are displayed.

### Changing the Output Attributes

Changing the output attributes of a domain has an effect on the dialog behavior in all the screens in which a field refers to this domain.

You should therefore find out which screens are affected before making such a change. To do this, choose and then *Indirect use* in the next dialog box. A list appears. Select *Screens* and choose . All the screens in which a field refers to the domain are listed.

### Changing the Value Table

The value table [Page 158] is used as proposal value for foreign key definitions. If you change the value table, use the where-used list to check if foreign keys that were already created also have to be changed. You can find out which foreign keys are affected by finding all the fields that refer to the domain and checking them against the changed value table.

# Deleting Domains

## Prerequisites

You can only delete a domain if it is no longer being used by a data element.

## Procedure

1. In the initial screen of the ABAP Dictionary, select object type *Domain* and enter the domain name.

   Choose  and check if the domain is still being used in data elements.

2. Choose  .

   A dialog box appears in which you are asked to confirm the deletion request.

3. Confirm the deletion request.

## Result

The domain is now deleted if it is not being used in any data elements.

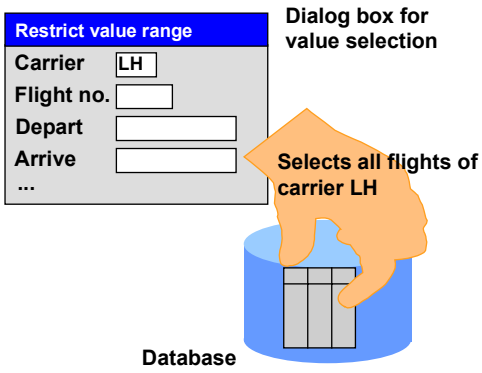
# Search Helps

The input help (F4 help) is a standard function of the R/3 System. The user can display the list of all possible input values for a screen field with the input help. The possible input values can be enhanced with further information. This is meaningful especially when the field requires the input of a formal key.

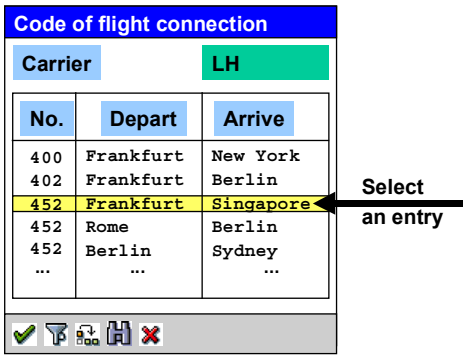
## Standard Input Help Process

A user calls an input help with the following steps (some steps can be omitted, depending on the definition of the input help):

1. The user starts the input help to display the possible input values for a field (search field) in a screen template.
2. The system offers the user a number of possible search paths. The user selects one of these search paths. Each search path offers a number of restrictions to limit the number of possible input values. These values are offered in a *Dialog box for value restriction* when the search path is selected.
3. The user enters restrictions if required and then starts the search.




4. The system determines the values that satisfy the entered restrictions (hits) and displays them as a list (hit list).




5. The user selects the most suitable line from the hit list by double-clicking.

6. The value of the search field is returned to the screen template (possibly together with other values).

Steps 2 and 3 are omitted if there is only a single search path available. In this case the dialog box for the value selection is offered immediately. You can also output the hit list directly after starting the input help. Steps 2 to 4 are omitted in this case.

## Function of a Search Help

This standard process can be completely defined by creating a search help in the ABAP Dictionary. This search help only has to be assigned to the screen fields in which they should be available (see Attaching Search Helps to Screen Fields [Page 176]).

There are two types of search help:

- Elementary search helps [Page 168] describe a search path. The elementary search help must define where the data of the hit list should be read from (selection method), how the exchange of values between the screen template and selection method is implemented (interface of the search help) and how the online input help should be defined (online behavior of the search help).
- Collective search helps [Page 172] combine several elementary search helps. A collective search help thus can offer several alternative search paths.

**See also:**

Example for Search Helps [Page 198]

Creating Elementary Search Helps [Page 187]

Creating Collective Search Helps [Page 193]

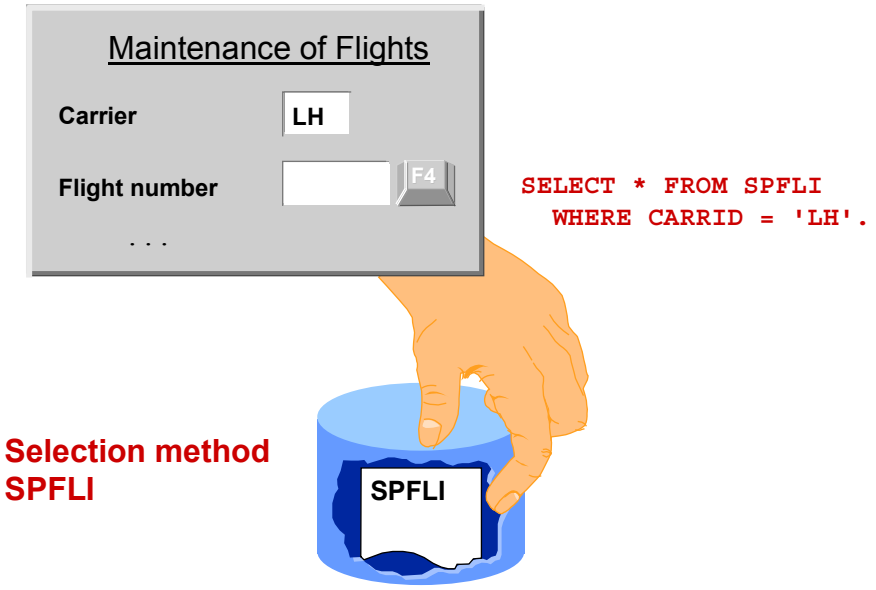
# Structure of an Elementary Search Help

An elementary search help defines the standard flow of an input help. You can define the following components of this flow in the search help:

- where does the data displayed in the hit list come from (selection method)
- what information should be displayed in the dialog box for value selection and in the hit list (search help parameters)
- what field contents can be taken into account for hit list selections and which values in the hit list can be returned to the screen fields (search help parameters)
- what dialog steps should be executed in the input help (dialog behavior)

## Selection Method

The possible input values displayed for a field in the hit list are determined at runtime by database selection.




If all the data required in the hit list comes from one single table, you only have to select this table (or a projection view on this table) as selection method. If there is a text table [Page 27] for the table, its fields are also available in the input help. A table entry is linked with the corresponding text by the existing foreign key.

If the data needed in the hit list comes from more than one table, you must link these tables with a view (database view or help view). This view must be defined as the selection method.

If the underlying tables are client-specific, the client field must be contained in the view. Otherwise selection for the input help would be for all clients.

## Search Help Parameters

A search help has an interface consisting of parameters. These parameters define the fields of the selection method that should be used in the input help.

A parameter of the search help must correspond to each field in the dialog box for value selection and to each field of the hit list. The parameters are copied from the corresponding selection method, that is they always have the same name as the corresponding field of the selection method.

If the search is restricted with a parameter of the search help, this is used in the data selection for formulating a WHERE condition for the field of the selection method with the same name. Vice versa, the parameters of the search help are assigned the contents of the fields of the selection method having the same name.

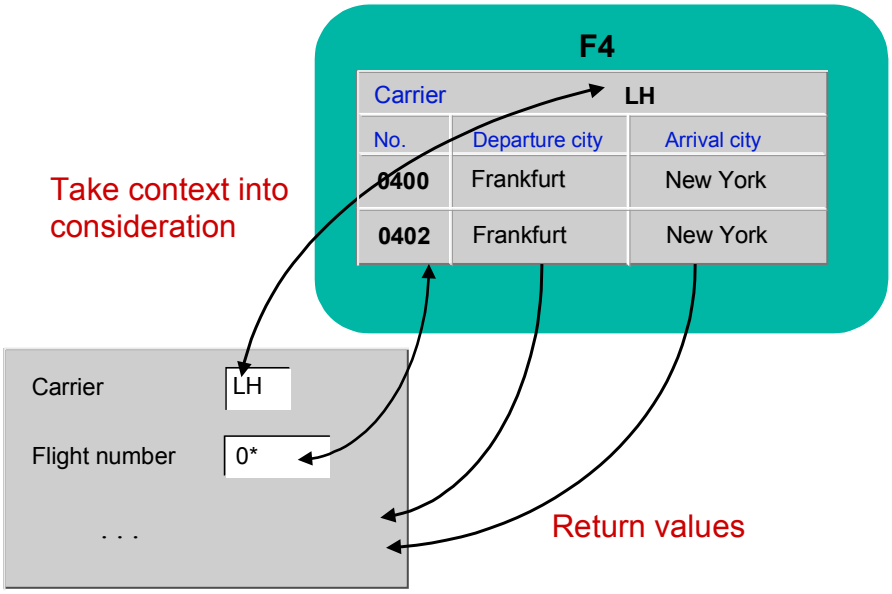
The search help should not contain any parameters for the clients. In the input help, selection is automatically in the logon client of the user.

A data element must be assigned to each search help parameter, that is a type is always defined for the search help parameters.

A search help can contain further parameters that do not correspond to any field of the selection method. This is normally only necessary if the standard flow of the input help described by the search help still has to be modified by with a search help exit [Page 196].

## Import and Export Parameters

When an input help is called, the entries that the user already made in the input template are taken into consideration. For example, if a user calls the input help for the flight number and already specified the carrier, of course only the numbers of flights of this carrier should be offered.
On the other hand, if the user selects one row of the hit list, more than one field of the input template might have to be filled with data from the selected row of the hit list. For example, if the flight number is obtained from the hit list, the city of departure and the destination should also be returned in the screen template.




The interface of a search help defines the context data that can be used in the input help and the data that can be returned in the input template.

A parameter of a search help can be classified as:

- **Import parameters:** Parameters with which context information from the processed input template (screen) may be copied to the help process.
- **Export parameters:** Parameters with which values from the hit list may be returned to the input template.

A parameter can simultaneously be an input and an export parameter. A search help can also contain parameters that are neither import nor export parameters. Such parameters could be required for the internal input help process, for example.

When you attach a search help [Page 176], you must define where the import parameters of the search help get their values from and the fields in which the contents of the export parameters are returned. See also Value Transport for Input Helps [Page 185].

## Description of the Online Behavior

The online behavior defines the steps executed in the input help process and the structure of the hit list and dialog box for value selection.

The dialog type [Page 190] defines whether or not the dialog box for value selection should be displayed. If you want to skip the dialog box for value selection, the hit list is displayed directly after calling the input help.

When you define an elementary search help, you can define how the dialog box for value selection and the hit list should look. For example, you can define the position of a parameter in the dialog box for value selection here. The column position in which the values of a parameter are displayed in the hit list can also be defined here.

**See also:**

Creating Elementary Search Helps [Page 187]

Example for Search Helps [Page 198]

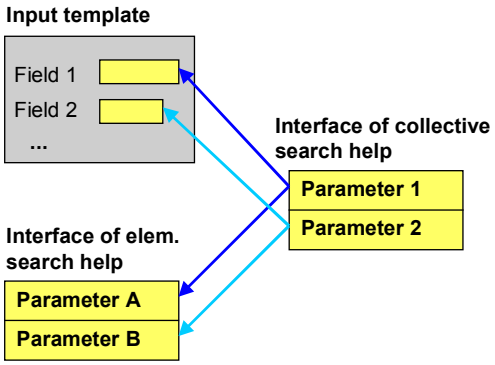
# Structure of a Collective Search Help

A collective search help combines several elementary search helps. The user can thus choose one of several alternative search paths with a collective search help.

When you define a collective search help, you only have to specify the search helps that are to be combined in the collective search help. In the input help, the values are transported between the elementary search help selected by the user and the input template using the collective search help. This is why a collective search help also has an interface for transporting the values.

## Interface of the Collective Search Help

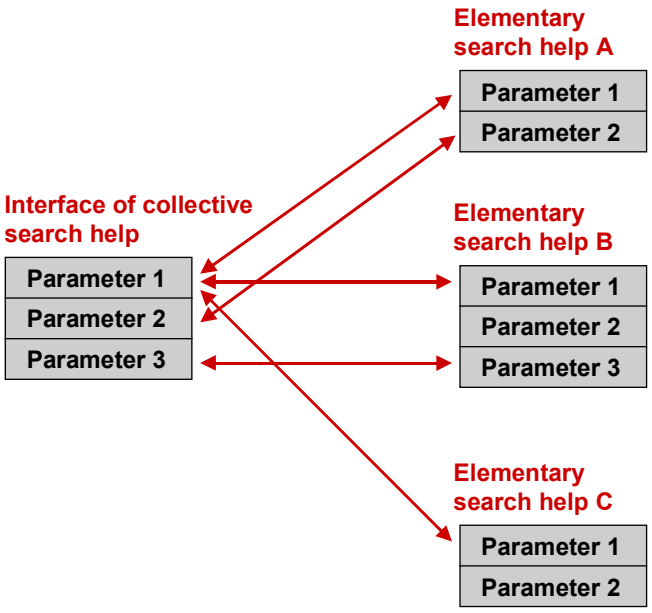
Like an elementary search help [Page 168], a collective search help has an interface of import and export parameters. The data is exchanged between the screen template and the parameters of the assigned elementary search helps using this interface.




## Assigned Search Helps

A collective search help comprises several elementary search helps. It combines all the search paths that are meaningful for a field.

The interface parameters (import and export parameters) of the included search helps must be assigned to the parameters of the collective search help. Not all of the parameters need to be assigned, that is the assignment can be open for some of the parameters.




Both elementary search helps and other search helps can be included in a collective search help. If other collective search helps are contained in a collective search help, they are expanded to the level of the elementary search helps when the input help is called.

**See also:**

Creating Collective Search Helps [Page 193]

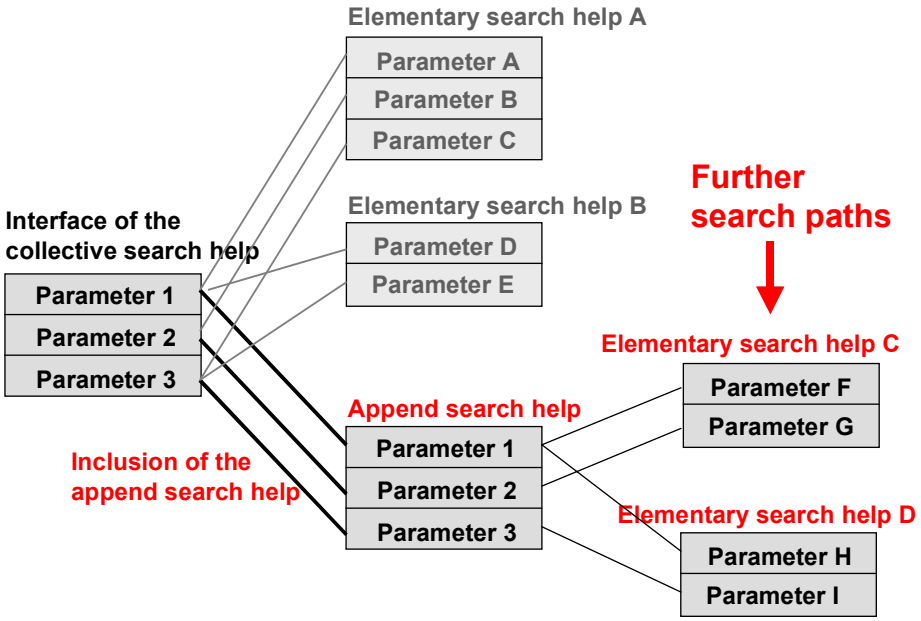
Example for Search Helps [Page 198]

# Append Search Helps

An append search help is used to enhance a collective search help (that is not the original in the current system) by further search paths (elementary search helps) without modifications. This technique can be used for example by special developments, country versions, SAP partners and SAP customers to add further search paths to a collective search help in the SAP standard version.

An append search help has a fixed assignment to a collective search help [Page 172] (of your appending object). This appending object is enhanced with an append search help. The structure of an append search help corresponds to the structure of a collective search help.

The append search help takes on the parameters of your appending object. An append search help is automatically included in your appending object. The parameters of both search helps having the same name are assigned to one another.




Append search helps can also be used themselves to describe an input help. In this case they are used like collective search helps. A search help exit cannot be assigned to an append search help.

Elementary search helps without modifications can be hidden in an SAP collective search help with an append search help. You have to include the search help to be hidden in the append search help as well and hide the inclusion there. The search path defined with this search help is no longer offered in the appending search help.

If the parameters of the appending object change, this change is not automatically made in the append search help. Instead, you are informed that you should adjust

the parameters of the append search help. In this case you should check if you want to change the assignments between the parameters of the append search help and the search helps included in them.

It would not make sense to automatically change the interface of the appending objects since if the interface of the collective search help changes, some of the contained elementary search helps normally have to be adjusted to this change.

**See also:**

Creating an Append Search Help [Page 195]

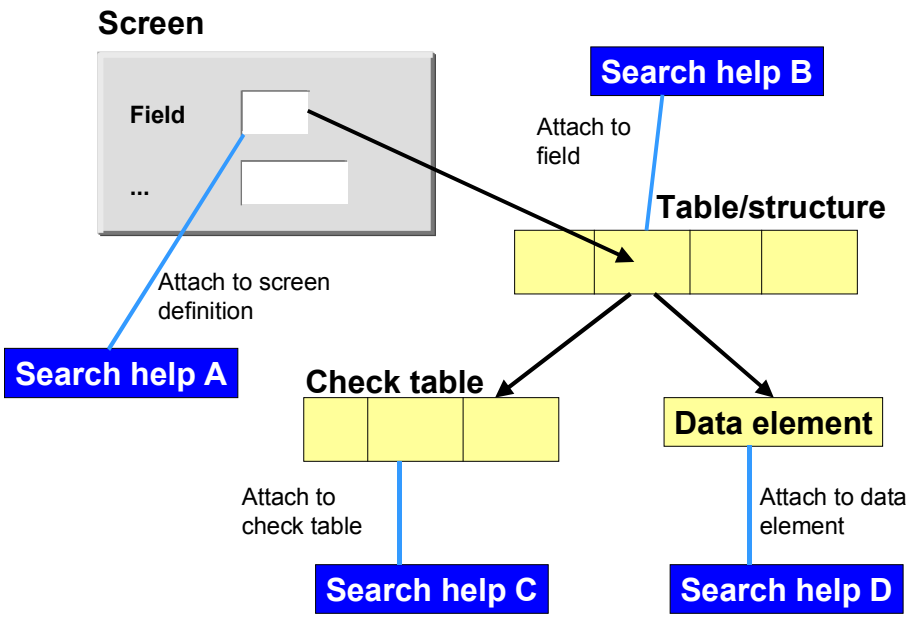
# Attaching Search Helps to Screen Fields

A search help can influence the behavior of a field when the input help is called. The search help must be assigned to the field in order to do this. You have the following options for this assignment:

- Attach the search help to a data element [Page 177]
- Attach the search help to a check table [Page 179]
- Attach the search help to a table field [Page 181]
- Attach the search help to a screen field [Page 183]

More than one search help can thus be assigned to a field. Conflicts when calling the input help are resolved with a hierarchy of search help attachments [Page 184].

Search help A is assigned directly to a field in the screen. Search help B is attached to the corresponding table field. There is a check table for the field to which search help C is attached. Search help D is attached to the data element of the field.



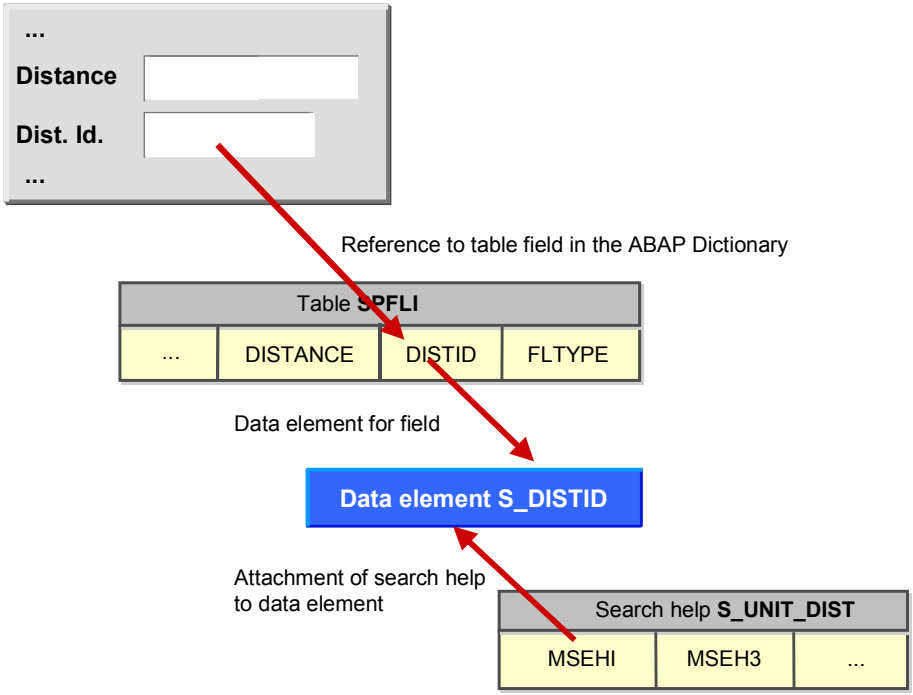
# Attaching to Data Elements

The search help can be used by all screen fields that refer to the data element [Page 132]. All the fields with the same meaning therefore have an identical input help.

When the search help is attached to a data element, an export parameter of the search help must be assigned to the data element. If the user selects a line of the hit list in the input help, the contents of this parameter are returned to the corresponding screen field. It is not possible to return several values when the search help is attached to a data element.

Table SPFLI (flight schedule) contains the field DISTID for the unit (kilometers, miles) of the flight distance specified in the field DISTANCE. The field DISTID refers to the data element S_DISTID. The elementary search help S_UNIT_DIST (search for distance units) is attached to this data element . The export parameter MSEHI of the search help is assigned to the data element.

The search help S_UNIT_DIST is therefore available for the field DISTID when flight data is maintained in table SPFLI. If the user selects a line of the hit list in the input help, the contents of parameter MSEHI are returned to the screen field.




If the parameter assigned to the data element is also an import parameter of the search help, the contents of the search field are used for the value selection if they contain a pattern. Other import parameters of the search field cannot be taken into consideration when the search help is attached to a data element.

### Attaching to Data Elements

The attachment of the search help to the data element is part of the definition of the data element. You can find out how to attach a search help to a data element in Creating Data Elements [Page 134].

# Attaching to Check Tables

If a field has a check table, the set of all possible entries to the field is defined by the contents of this check table (see Foreign Keys [Page 19]). The contents of the key fields of the check table are therefore offered automatically in the input help. If a text table [Page 27] is defined for the check table, the contents of the first text field of this text table is also displayed in the user's logon language.

This input help coming from the check table can be further edited by assigning a search help to the check table. The assigned search help can then be used by all screen fields that are checked against the table.

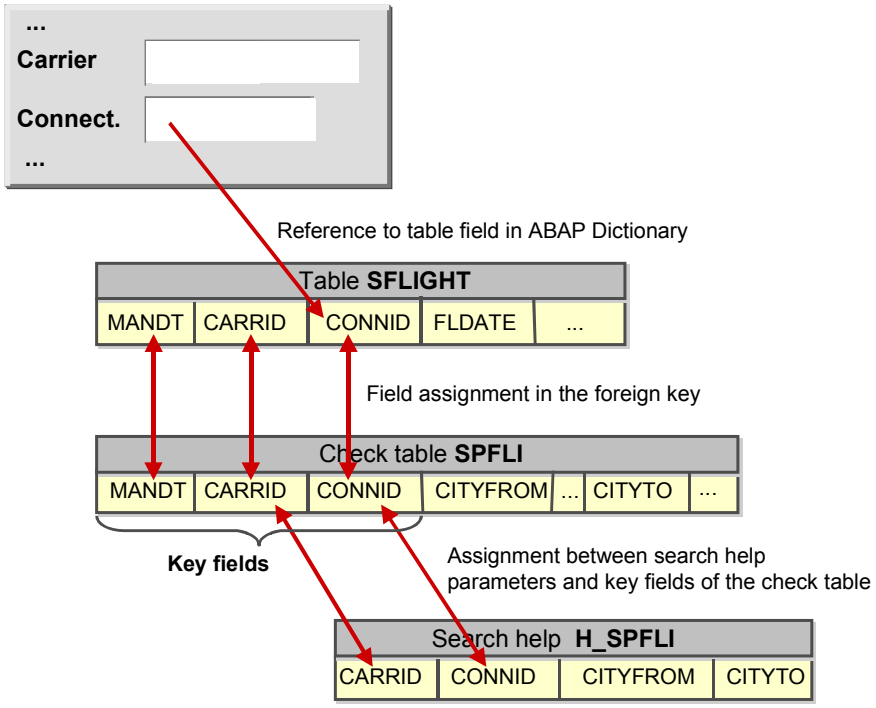
If a search help is attached to a check table, it should display the data contained in this check table. In this case, the check table itself or a view on this check table must be given as the selection method of the search help.

When a search help is assigned to a check table, there must be an assignment between the key fields of the check table and the parameters of the search help. If an export parameter of the search help is assigned to a key field of the check table, the contents of this parameter are returned to the corresponding screen field as soon as the user has selected a line of the hit list in the input help. If an import parameter of the search help is assigned to a key field of the check table, the field contents are used for the value selection (see Value Transport for the Input Help [Page 185]).

The field CONNID (number of the flight connection) of table SFLIGHT (flights) is checked against table SPFLI (flight schedule). The elementary search help H_SPFLI (search for flights by cities of departure and arrival) is allocated to table SPFLI. This search help is available for the field CONNID when data is entered in table SFLIGHT.

Search help H_SPFLI does not contain a client field since for the input help the selection is always made in the logon client of the user.

## Attaching to Check Tables




The search help attachment is part of the definition of the table/structure to which the search help is attached. You can find out how to attach a search help to a table in Creating Tables [Page 71].

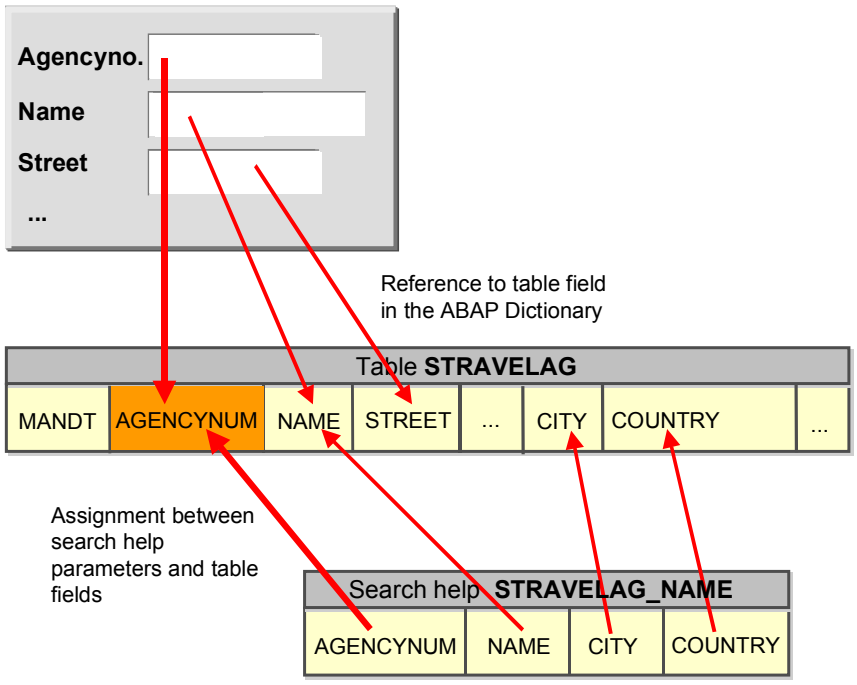
# Attaching to a Table Field or Structure Field

The search help can be used by all screen fields that refer to the table field or structure field. The search help parameters and the fields of the table or structure must be assigned to one other in this type of attachment.

If an export parameter of the search help is assigned to a table field, the contents of this parameter are returned to the corresponding screen field as soon as the user has selected a line of the hit list in the input help. If an import parameter of the search help is assigned to a table field, the field contents are used for the value selection (see Value Transport for the Input Help [Page 185]).

Normally some parameters of the search help cannot be assigned to a table field. The assignment is thus left open for some search help parameters. A constant or any other field that is searched for in the module pool when the input help is called can also be assigned to a search help parameter.

The search help STRAVELAG_NAME permits you to search for the number of a travel agency using supplementary information about the agency (name, city, country). This search help should be available to all screen fields that reference field AGENCYNUM (number of the agency) in table STRAVELAG. The search help is therefore directly attached to the field AGENCYNUM of table STRAVELAG. All the parameters of the search help can be allocated to corresponding table fields.




The search help attachment is part of the definition of the table/structure to which the search help is attached. You can find out how to attach a search help to a table field in Creating Tables [Page 71].

# Attaching to Screen Fields

A search help can be directly assigned to a screen field. In this case, the search help is only available for this screen. If the same field is used on several screens, the search help should generally be attached to the referenced table field or structure field (see Attaching to Table Fields [Page 181]).

You can attach a search help directly to a screen in the following ways.

- The name of the search help must be entered in the *Screen Painter* in the *Attributes for the field* in the field *Search help*.
- The name of the search help can be defined for selection screens in ABAP reports in the PARAMETERS or SELECT-OPTIONS statement directly following the supplement AS SEARCH PATTERN.

In both cases this assigns the first parameter of the search help to the screen field. As a result, only a value from the hit list can be returned to the screen template.

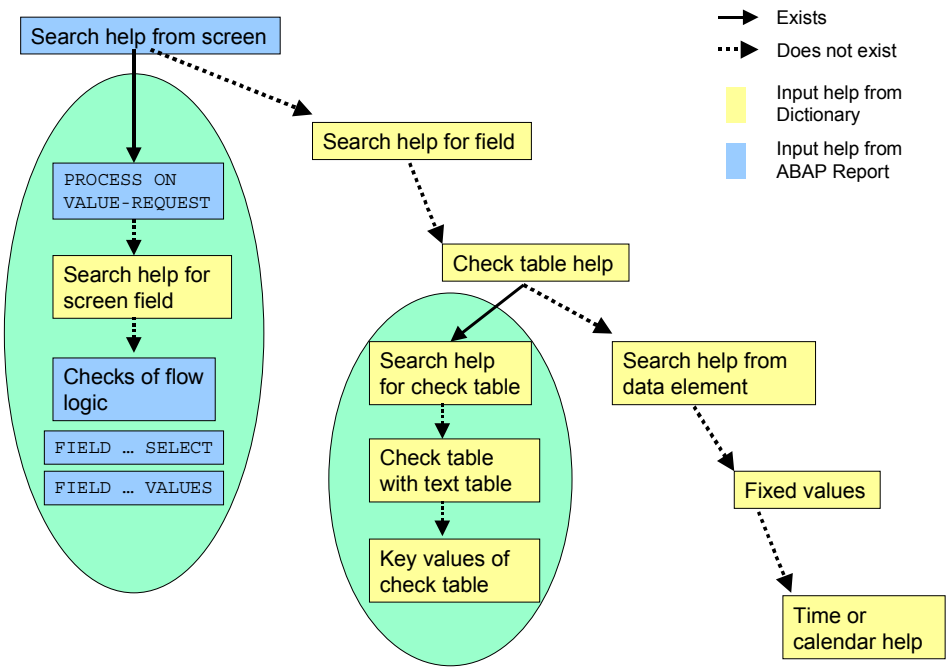
# Hierarchy of the Search Help Call

Search helps can be attached [Page 176] to screen fields in different ways. Several search helps can thus be assigned to one screen field. Conflict situations when calling the input help are resolved with a hierarchy of different types of search help attachment.

When you call the input help for a screen field, there is first a check if an input help is defined for the field on the screen. There are three cases here. If there is a programmed help with `PROCESS ON VALUE-REQUEST`, it is executed. If there is no programmed help, the system tries to call the search help assigned to the field on the screen. If no search help is assigned to the field on the screen, the help defined with `FIELD SELECT` or `FIELD VALUES` is offered.

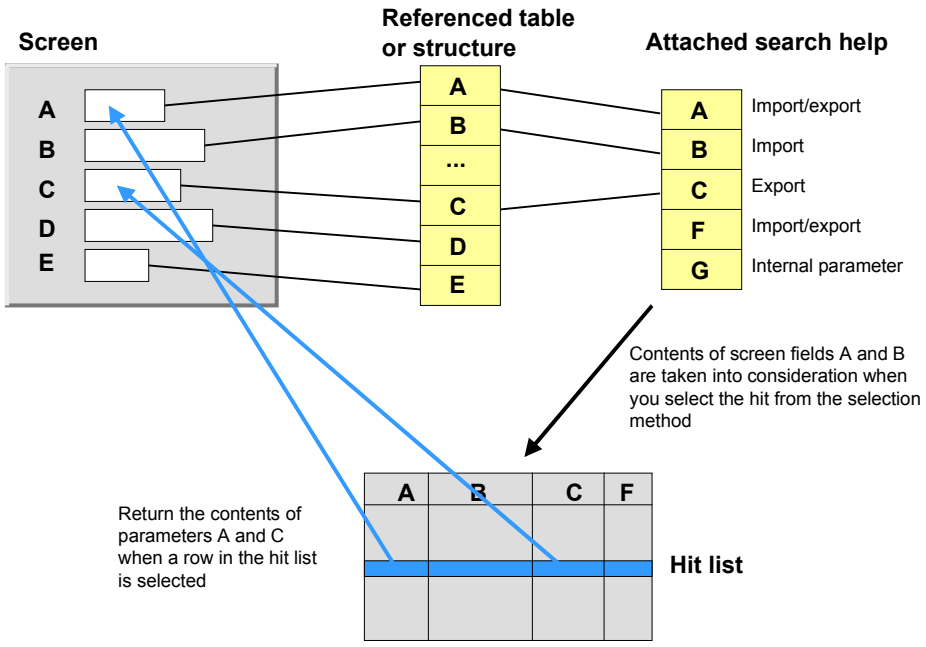
If no input help is defined for the field on the screen, the system tries to call the search help attached to the table field. If there is no search help here, it tries to display the check table help. There are two cases here. If a search help is attached to the check table, it is displayed. Otherwise only the key values of the check table are displayed. If there is a text table [Page 27] for the check table, the text for the key value is added in the user's logon language.

If there is no check table for the field, the system tries to call the search help from the data element. If there is no search help for the data element either, existing domain fixed values are displayed. The calendar help and time help are automatically provided for fields with data type DATS and TIMS.



# Value Transport for Input Helps

There is a value transport between the field contents on the screen and the interface of the search help when the input help is called and when a line of the hit list is selected. The values already entered on the screen can then be used as restrictions for a selection from the hit list. Only those hits that are consistent with the values already entered are displayed.




In the above example, screen fields A, B and C are linked with parameters of the search help. As a result, values can only be transported between the screen and the search help for these three fields. Existing contents of screen fields A and B can be used for selecting the hit list since they are linked with an import parameter of the search help. The values of parameters A and C can be returned to the screen from the hit list since these parameters are declared as export parameters of the search help.

## Parametrizing the Import Parameters of the Search Help

If the search help is attached to the screen field with a data element (Attaching to Data Elements [Page 177]) or directly in the screen (Attaching to Screen Fields [Page 183]), only one search help parameter is assigned to this field. The value transport can only take place between the screen field and this parameter.

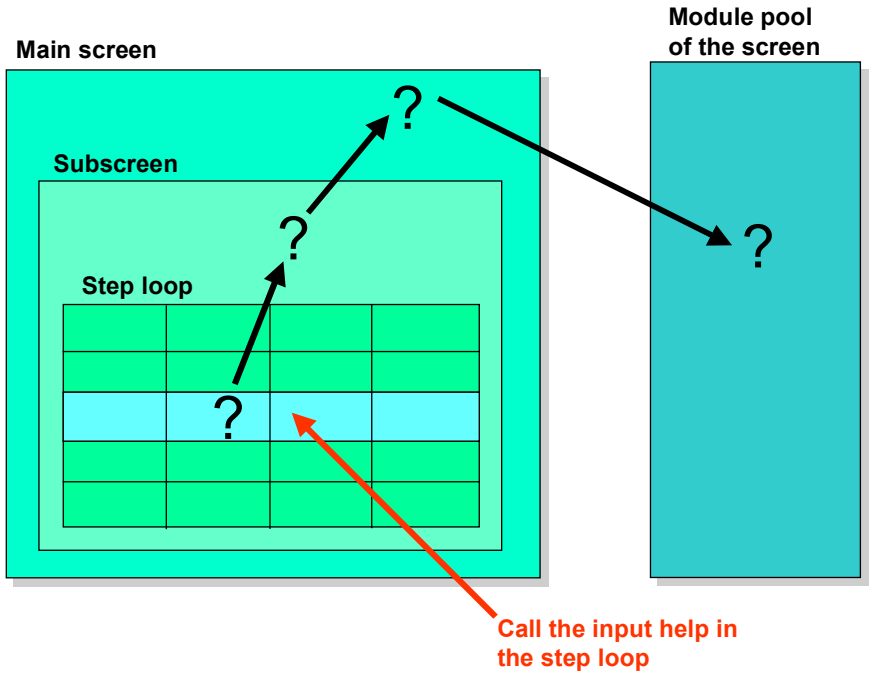
If the search help is attached to the table field (Attaching to Table Fields [Page 181]) or to the check table of the field (Attaching to Tables [Page 179]), a value transport can take place for all the screen fields that are linked with a parameter of the search help.

When the input help is called, the system tries to find a field **with the same name** on the screen for each import parameter of the search help that is attached to a table or structure field. If such a field is found, the contents of the screen field are copied to the search help parameter.

There are at most four steps when searching for the screen field with the same name:

**Value Transport for Input Helps**

- If the input help is called in a step loop, a field with the same name is searched for in this step loop.
- If the step loop does not contain a field with the same name, the corresponding subscreen is searched.
- If the subscreen does not have a field with the same name, the main screen is searched.
- If the main screen does not contain a field with the same name either, the module pool of the corresponding screen is searched.




## Returning the Values from the Hit List

Values are only returned at the level where the input help was called. If the input help is called for example within a step loop, the values are only returned from the hit list in fields of the corresponding line of this step loop. In particular, values are never returned from the hit list to the module pool of the screen.

The values of the hit list are only returned in input fields and in fields that are only linked with an export parameter of the search help (used to parametrize pure text fields that are not used in the selection).

# Creating Elementary Search Helps

## Procedure

1. In the initial screen of the ABAP Dictionary, select object class *Search help*, enter the name of the search help and choose *Create*.

   A dialog box appears in which you must select the type of search help.

2. Select *Elementary search help* and choose .

   The maintenance screen for elementary search helps appears.

3. Enter an explanatory text in the field *Short text*.

   You can for example find the search help at a later time using this short text.

4. In the *Definition* tab page enter the selection method of the search help.

   You can enter the name of a table or a view (database view, projection view or help view) here. If you enter a table that has a text table [Page 27], the name of the text table is automatically entered in the corresponding field.

5. Using the input help (F4 help), select fields of the selection method as parameter in the *Search help parameter* area. Select the fields that should be used in the dialog box for value selection or in the hit list.

   If the selection method is a table that has a text table, both the fields of the table and the fields of the text table are offered in the input help.

   The data element of the parameter is automatically copied from the selection method. The data element defines the output attributes and the F1 help of the parameter in the hit list and in the dialog box for value selection.

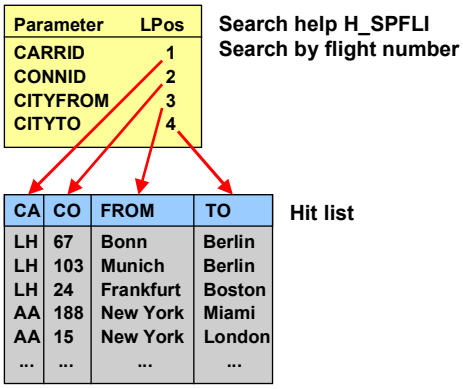
   You can assign the parameter another data element. To do so, select the *Mod* flag. The *Data element* field is now ready for input. Then select a data element with the input help (F4 help). Only data elements whose data type, length and number of decimal places is the same as those of the previous data element can be assigned.

   This removes the link between the data element of the search help parameter and the data element of the selection method field having the same name. If you cancel the *Mod* flag, the data element of the assigned table field is used again.

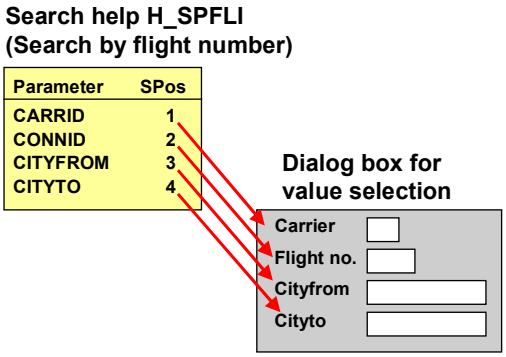
6. Define the attributes of the search help parameters.

   Select the *IMP* flag if it is an import parameter. Select the *EXP* flag if it is an export parameter.

   You can define the dialog for the input help with the fields *LPos*, *SPos* and *SDis*. Enter the parameter position in the hit list in *LPos*. If you enter nothing or the value 0 here, the parameter is not displayed in the hit list.




Enter the parameter position in the dialog box for value selection in *SPos*. If you enter nothing or the value 0 here, the parameter is not displayed in the dialog box for value selection.




Set the *SDis* flag if the parameter should be a pure display field in the dialog box for value selection. The user is thus informed that the contents of the parameter restrict the value, but he cannot change this restriction. This makes sense for example when the parameter is an import parameter or if it has a default value.

You can assign the parameter a default value [Page 192] in the *Default value* field.

7. Select the dialog type [Page 190] of the search help.

   The dialog type defines how the hit list is displayed in the input help.

8. Save your entries.

   A dialog box appears in which you have to assign the search help a development class.

9. Choose .

Do not forget to link the search help to a screen field [Page 176]. The search help attachment is not part of the search help definition; it is part of the object definition to which the search help is attached.

## Result

The search help is activated. You can find information about the activation flow in the activation log, which you can display with *Utilities* → *Activation log*. If errors occurred during activation, the activation log is automatically displayed.

## Other Options

- **Assign a hot key:** If the search help is to be accessed with a hot key [Page 191], you must enter a one-place ID in the *Hot key* field. All the elementary search helps contained in a collective search help should have different short cuts.
- **Assign a search help exit:** In exceptions, you might have to change the standard flow defined by the search help with a search help exit [Page 196]. In this case enter the name of the search help exit in the corresponding field.
- **Test the search help:** You can test the flow of an input help defined by the elementary search help with  . A dialog box appears in which you can simulate the behavior of the search help under different conditions. You can obtain information about the options provided in this window with  .

**See also:**

Structure of an Elementary Search Help [Page 168]

# Dialog Types

The dialog type of an elementary search help defines how the hit list is displayed when the input help is called.

The following dialog types are possible:

- **Immediate value display:** The hit list is immediately displayed when the input help is called. This is only meaningful if the hit list usually only contains a few entries.
- **Complex dialog with value restriction:** The dialog window for restricting values is offered immediately. Choose this option if the list of possible entries is usually very large. If the user limits the amount of data to be processed, the hit list will become more comprehensible and the system load during value selection will be reduced.
- **Dialog depending on number of values:** If the hit list contains less than 100 entries, it is displayed immediately. If the hit list contains more than 100 entries, the dialog box for restricting values is displayed.

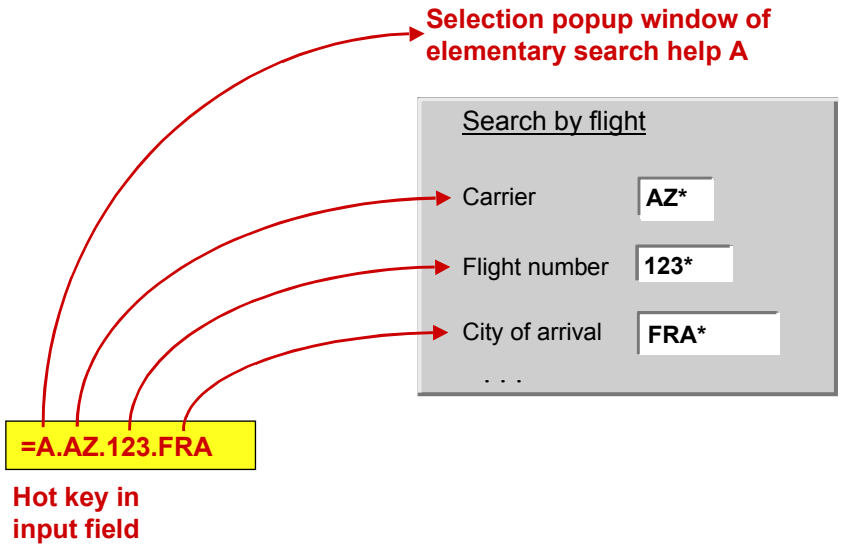
# Hot Keys

The hot key is used to select the elementary search help from the collective search help and to enter the restrictions in the dialog box for restricting values directly from the entry field. If the user often searches for values using the same search help, this procedure can save time.

The short cut must be entered in the input field according to the following convention:

*=<Hot key>.S1.S2.S3....*

*S1*, *S2*, *S3*, etc. are the restrictions that would be entered in the corresponding dialog box in this order. Each entry is interpreted generically.




If the elementary search help stored as standard search help is to be used, the hot key need not be specified. If *S1* contains more than one character, the first separator '.' can also be omitted.

If the hot key is defined without restrictions ( *=<Hot key>* ), the dialog box for restricting values is called. If restrictions are defined, the dialog box for restricting values is skipped and the hit list is displayed immediately. If exactly one hit is found, the hit list is not displayed. The values of the hits found are immediately returned to the screen template in this case.

# Default Values for Search Help Parameters

A default value can be assigned to a parameter of a search help.

The following default values are allowed:

- Constants, which are enclosed in apostrophes when they are defined in the maintenance screen. The constant must be defined in internal representation for parameters whose data type has an editing template (such as date and time). For example, the date 01.03.1998 must be defined as ‘19980301’.
- System fields, i.e. fields of structure SYST, where the prefix SY- can be used in place of the prefix SYST-.
- Get parameter IDs, with which the parameter is assigned values from SAP memory.

The parameter is assigned the default value in the input help process in the following cases:

- If the parameter is not an import parameter.
- If nothing was assigned to the parameter in the search help attachment with which the search help was attached to the screen field.
- If a field that does not exist in either the screen or the flow logic in the input help process was assigned to the parameter in the search help attachment.
- If a search help is included in a collective search help and the parameter is not linked to any parameter of this collective search help.

# Creating Collective Search Helps

## Procedure

1. In the initial screen of the ABAP Dictionary, select object class *Search help*, enter the name of the search help and choose *Create*.

   A dialog box appears in which you must select the type of search help.

2. Select *Collective search help* and choose .

   The maintenance screen for collective search helps is displayed.

3. Enter an explanatory text in the field *Short text*.

   You can for example find the search help at a later time using this short text.

4. In the *Definition* tab page enter the parameters of the collective search help.

   Select the *Imp* flag if it is an import parameter. Select the *Exp* flag if it is an export parameter.

   Define the types for the parameters of a collective search help by assigning a data element. Enter the name of the data element that describes the contents of the search help parameter in the *Data element* field.

   You can assign the parameter a default value [Page 192] in the *Default value* field.

5. In exceptions it could be necessary to change the standard process defined by the search help. You can implement the deviation from the standard using a search help exit [Page 196].

   In this case enter the name of the search help exit in the corresponding field.

6. On the *Included search helps* tab page, define the search helps that you want to include in the collective search help.

   You can include elementary search helps and collective search helps.

   Use the *Hide* flag to control whether an included search help should appear in the dialog box for selecting the elementary search help. If the flag is set, the search help is not offered.

   It makes sense to hide search help inclusions if one or more search paths in the standard system should not be used in a concrete R/3 System. Similarly, search help inclusions can also be already hidden in the standard system because they only can be used meaningfully in a few R/3 Systems. You have to cancel the flag in this case.

7. Position the cursor one after the other on each allocated search help and choose *Parameter assignment*.

   In the next screen, enter the parameter names of the elementary search helps to which the corresponding parameters of the collective search help should be assigned in the field *Reference parameter*.

   You can select the parameters contained in the included search help using the input help. Create a proposal for the assignment with *Proposal*.

8. Save your entries.

A dialog box appears in which you have to assign a development class to the search help.

9. Choose .

## Result

The collective search help is activated. You can find information about the activation flow in the activation log, which you can display with *Utilities* → *Activation log*. If errors occurred when the collective search help was activated, the activation log is automatically displayed.

Do not forget to link the search help to a screen field [Page 176]. The search help attachment is not part of the search help definition; it is part of the object definition to which the search help is attached.

## Other Options

You can test the flow of an input help defined by the collective search help with . A dialog box appears in which you can simulate the behavior of the search help under different conditions. You can obtain information about the options provided in this window with .

**See also:**

Structure of Collective Search Helps [Page 172]

# Creating an Append Search Help

1. In the initial screen of the ABAP Dictionary, enter the name of the collective search help to which you want to add the append search help Choose *Display*.

   The maintenance screen of the collective search help appears in display mode.

2. Choose *Goto* → *Append search helps*.

   A dialog box appears in which you must enter the name of the append search help. This name should lie in the customer namespace (or in the namespace of the partner or special development).

3. Choose .

The maintenance screen of the append search help appears. You can now continue as when creating a collective search help [Page 193].

Note the following:

- The collective search help always uses the interface of its appending objects. This is why you cannot change the entries in the *Definition* tab page.
- If you want to hide a search help included in the appending object with the append search help, you have to include this search help in the append search help. Enter the name of the search help to be hidden in the *Included search helps* tab page and select *Hidden*.
- The definition of the appending search help is automatically adjusted when the append search help is activated. The append search help is automatically included in the appending object.

**See also:**

Append Search Helps [Page 174]

# Search Help Exit

A search help describes the standard input help process. In exceptions it could be necessary to deviate in some points from this standard. Such a deviation from the standard can also be implemented with a search help exit.

The input help process should look as much the same as possible throughout the entire system. Search help exits should therefore only be used for exceptions.

A search help exit is a function module that has a predefined interface. A search help exit is called at certain times by the help processor. The administrative data of the help processor are passed to the search help exit using the interface.

You can store your own program logic that manipulates this administrative data in the search help exit. Individual steps of the input help process can be skipped with a search help exit.

Search help exit F4UT_OPTIMIZE_COLWIDTH adjusts the column width in the hit list to the contents of the column. It makes sense to use this search help exit when the columns of the hit list have to be made very wide for extreme cases (e.g. for name fields), but normally they will contain much smaller values.

Each search help exit must have the same interface as function module F4IF_SHLP_EXIT_EXAMPLE (is used as pattern for all the search help exits to be created). You can find more information about the interface in the documentation for this function module.

## Calling the Search Help Exit

If a search help exit is assigned to a search help, the help processor calls it at the following times:

### Before Displaying the Dialog Box for Selecting the Required Search Path.

It is only called for collective search helps. Using the search help exit, the set of elementary search helps available can for example be restricted depending on the context.

### Before Displaying the Dialog Box for Entering Search Conditions.

You can either influence the dialog for entering search conditions or skip it altogether here. You can also influence how the selection screen looks.

### Before Selecting Data.

The data selection can be partly or completely copied from the search help exit. This can become necessary if the data selection cannot be implemented with a SELECT statement for a table or a view.

### Before Displaying the Hit List.

You can influence the display of the hit list in this step with the search help exit. You can reduce the number of values displayed here. For example, you can display only values for which the person calling the input help has authorization. You can also copy the complete hit list from the search help exit.

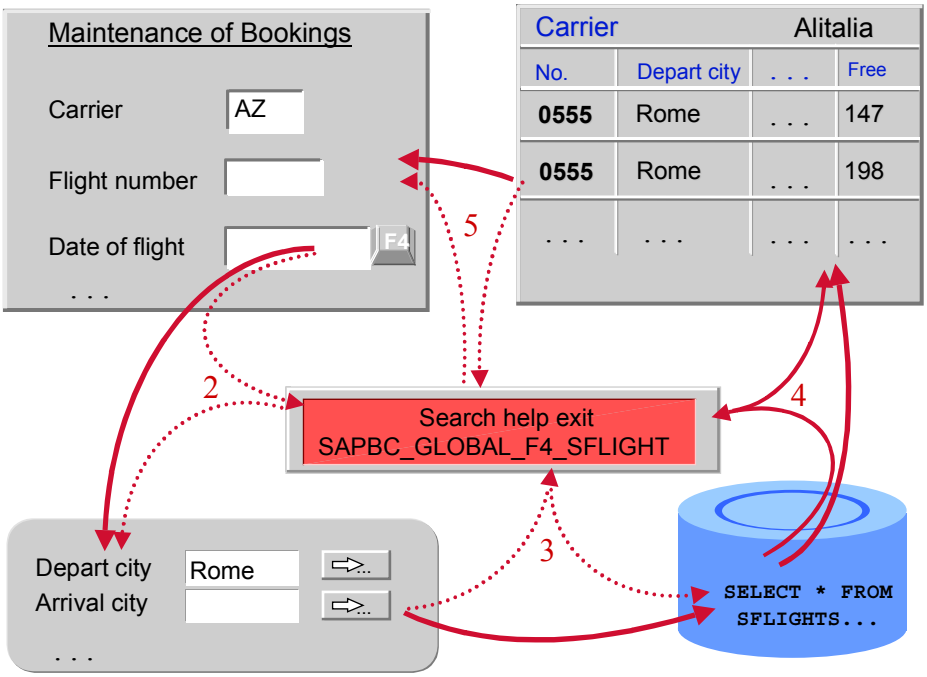
### Before Returning the Values Selected by the User to the Input Template.

It could be advisable to intervene at this time if control of the further transaction flow should depend on the value selected. A typical example is setting set/get parameters.

Search help SFLIGHT is used to find flight data. When they look for flights, the staff of travel agencies normally need information about whether there are still seats available on the flight. The selection method of the search help (view on tables SCARR, SFLIGHT and SPFLI) does not directly contain this information. The selection method only contains information about the number of seats available on the flight and how many seats are already reserved.

From this information, search help exit SAPBC_GLOBAL_F4_SFLIGHT computes the number of seats still available and returns the results in a parameter of the search help. The number of seats still available can thus be displayed in the hit list.

You therefore only have to program one action in the search help exit for the call before displaying the hit list.



# Example for Search Helps

Each customer of a carrier (see Flight Model [Page 295]) or of a travel agency has a customer number. You want to find a search option for this customer number.

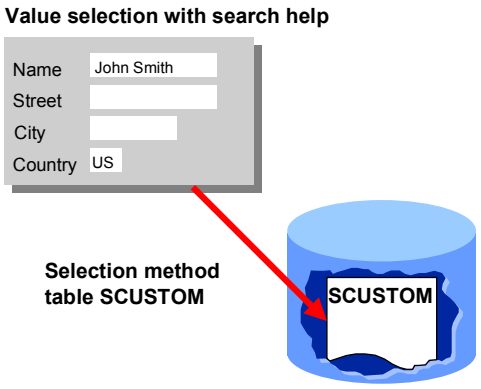
The user must be offered two different search paths.

- The user should be able to search for the customer number using the customer data, such as the name and address.
- The user should be able to search for the customer number using existing customer bookings.

You can provide the required search option by creating a collective search help [Page 172] SCUSTOM. Two elementary search helps [Page 168] SCUSTOM_NAME (for searching with the customer data) and SCUSTOM_BOOK (for searching with the existing bookings) are created for the actual search paths. These elementary search helps are included in the collective search help.

## Elementary Search Help SCUSTOM_NAME

This elementary search help should enable you to search for the customer number using the name and address (street, city, country). All this data is contained in table SCUSTOM. Table SCUSTOM must therefore be selected as the selection method of the elementary search help.




You now have to decide which fields of the selection method are needed for the input help process. These are the fields that should appear either in the dialog box for restricting values or in the hit list.

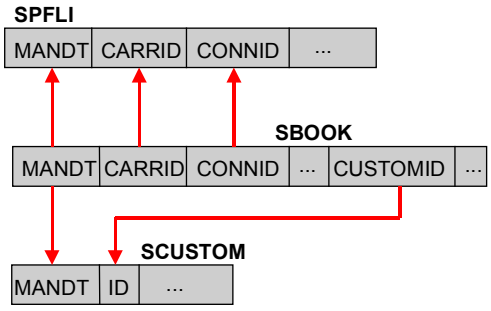
In the dialog box for restricting values, the user should be able to restrict values with the customer's name and address, i.e. the fields for the street, city and country. These fields as well as the customer's number (the information to be found must always be in the hit list) should appear in the hit list. The fields ID, NAME, STREET, CITY and COUNTRY of table SCUSTOM must be included in the search help as parameters.

The parameter ID is declared to be an import parameter. A pattern entered in the corresponding field of a screen template can therefore be used directly for the value selection. Restrictions for the other parameters of the search help must be entered in the dialog box for value selection.

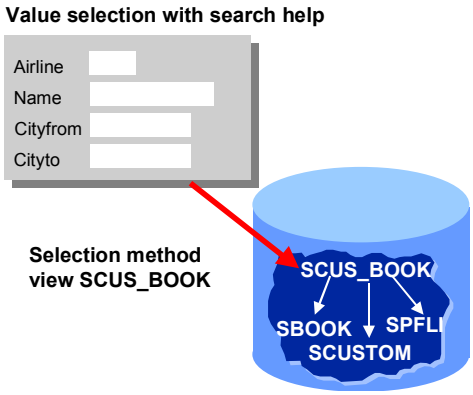
All the parameters of the search help are declared to be export parameters. As a result, all the parameters of the hit list can be returned to the screen template if the corresponding fields are available there.

# Elementary Search Help SCUSTOM_BOOK

This elementary search help should enable you to search for the customer number using existing customer bookings. The flight data for the booking (flight number, date of flight, city of departure, city of arrival) and the name of the customer should be used for the search here. This data is distributed on the tables SBOOK (bookings), SCUSTOM (name) and SPFLI (cities of departure and arrival). The following graphic shows the relationship between the relevant tables, that is the existing foreign key relationships.




In this case a database view SCUS_BOOK must be created on these three tables (see Example for Views [Page 127]) as selection method. The tables in the view (join) are linked as defined by the existing foreign key relationships (see Foreign Key Relationship and Join Condition [Page 102]).



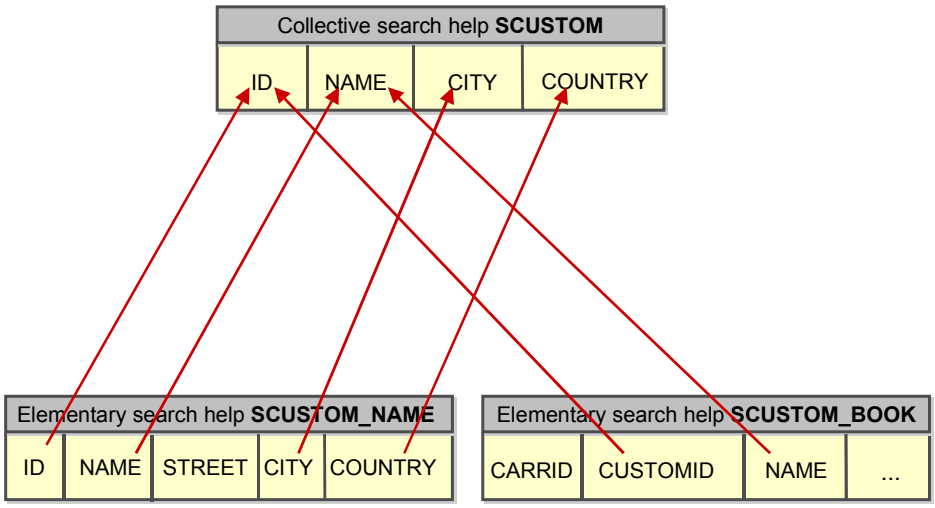

In the dialog box for restricting values, the user should be able to restrict the search for booking data with the carrier ID, customer name, city of departure and city of arrival The flight date and of course the customer number should also be displayed in the hit list. Fields CARRID, FLDATE, CUSTOMID, NAME, CITYFROM and CITYTO of view SCUS_BOOK must be included in the elementary search help as parameters of the search help.

The parameter CUSTOMID is declared to be an import parameter. All the parameters of the search help are export parameters.

Example for Search Helps

# Collective Search Help SCUSTOM

The two elementary search helps are now included in the collective search help. You must now allocate the parameters of the elementary search helps to the parameters of the collective search help.




The parameter ID of the collective search help is marked as an import parameter. All the parameters are export parameters. The values can thus be copied from the hit list to the screen template.

# Attaching the Search Help

In order to be able to use the search help SCUSTOM in screen templates, the attachment of the search help (see Attaching Search Helps with Screen Fields [Page 176]) must be defined.

### Attaching to the Check Table SCUSTOM

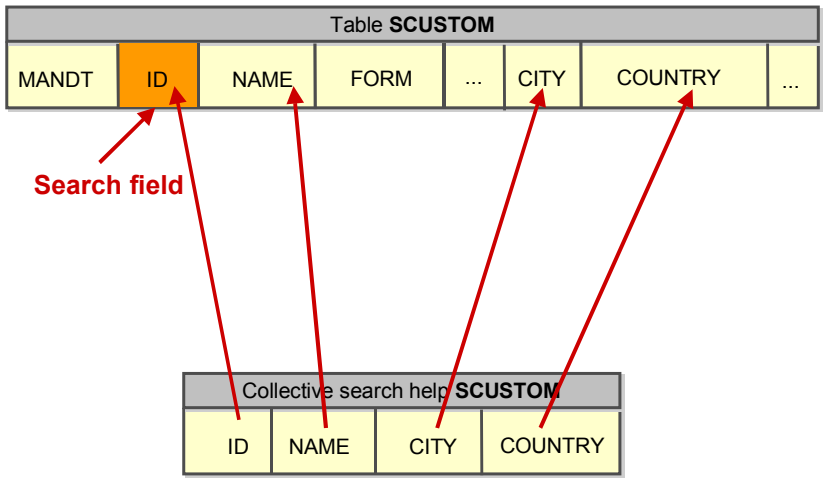
The search help should be available for all the fields that are checked against table SCUSTOM. The search help therefore must be attached to table SCUSTOM. The search help parameters must therefore be assigned to the key fields of table SCUSTOM.

The parameter ID of search help SCUSTOM is here assigned to the field ID of table SCUSTOM in this field assignment. No assignment is possible for all other parameters of the search help (NAME, CITY and COUNTRY) since table SCUSTOM does not contain this information as key fields.

### Attaching to a Field of Table SCUSTOM

In order that the search help is available when the field SCUSTOM-ID is directly copied to the input template, you have to attach the search help to this field.

With this type of attachment, all the parameters of the search help can be assigned to the corresponding fields of the table.

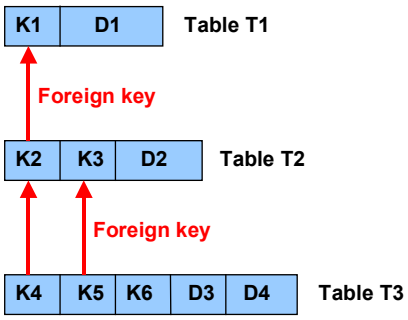


# Lock Objects

The R/3 System synchronizes simultaneous access of several users to the same data records with a lock mechanism [Page 210]. When interactive transactions are programmed, locks are set and released by calling function modules (see Function Modules for Lock Requests [Page 206]). These function modules are automatically generated from the definition of lock objects in the ABAP Dictionary.

## Structure of a Lock Object

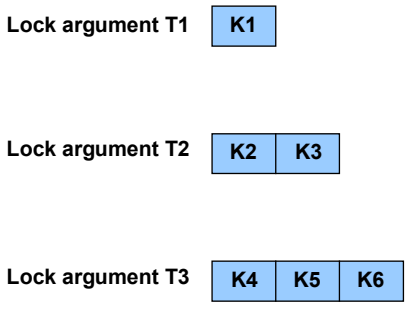
The tables in which data records should be locked with a lock request are defined in a lock object together with their key fields. When tables are selected, one table (the primary table) is first selected. Further tables (secondary tables) can also be added using foreign key relationships (see also Conditions for Foreign Keys [Page 209]).




## Lock Arguments

The lock argument of a table in the lock object consists of the key fields of the table.

The lock argument fields of a lock object are used as input parameters in the function modules for setting and removing locks generated from the lock object definition. When these function modules are called, the table rows to be locked or unlocked are specified by defining certain values in these fields. These values can also be generic. The lock argument fields therefore define which subset of the table rows should be locked.



The simplest case of a lock object consists of exactly one table and the lock argument of the table is the primary key of this table. Several tables can also be included in a lock object. A lock request therefore can lock an entire logical object, and not only a record of a table. Such a logical object can be for example a document comprising an entry in a header table and N entries in a position table.

**Call the lock function module with K1=2 and K3=1 (K6 unspecified)**

**Table T1**

| K1 | D1 |
|---|---|
| 1 | ... |
| 2 | ... |
| 3 | ... |
| ... | |

**Table T2**

| K2 | K3 | D2 |
|---|---|---|
| 1 | 1 | ... |
| 2 | 1 | ... |
| 2 | 2 | ... |
| 3 | 1 | ... |
| 3 | 3 | ... |
| ... | | |

**Table T3**

| K4 | K5 | K6 | D3 | D4 |
|---|---|---|---|---|
| 1 | 1 | A | ... | ... |
| 2 | 1 | A | ... | ... |
| 2 | 1 | B | ... | ... |
| 2 | 1 | C | ... | ... |
| 2 | 3 | A | ... | ... |
| 2 | 3 | B | ... | ... |
| 3 | 1 | A | ... | ... |
| ... | | | | |

**Red entries are locked**

Locks can also be set from programs in other systems with the corresponding interfaces if the lock object was defined with RFC authorization.

A lock mode [Page 205] can be assigned for each table in the lock object. This mode defines how other users can access a locked record of the table.

Table SFLIGHT in the flight model [Page 295] contains all the scheduled flights of a carrier. Field SEATSMAX contains the number of seats available. Field SEATSOCC contains the number of seats already booked. If a booking is made for a customer (by a travel agency or sales desk), you must check whether there are enough seats available. The number of seats booked is incremented when the booking is made.

This mechanism must ensure that two sales desks do not make the same booking at the same time and that the flight is not overbooked.

This can be done by creating lock object ESFLIGHT. Only the table SFLIGHT must be included in this lock object. The flight can then be locked (with the function modules generated from the lock object) when booking. If another sales desk also wants to book seats for this flight, the lock will prevent the flight from being overbooked.

## Lock Objects

**See also:**

Example for Lock Objects [Page 216]

Creating Lock Objects [Page 213]

Deleting Lock Objects [Page 215]

# Lock Mode

Sperrtabelle [Ext.]

Kollisionen von Sperren [Ext.]

The lock mode controls whether several users can access data records at the same time. The lock mode can be assigned separately for each table in the lock object. When the lock is set, the corresponding lock entry is stored in the lock table of the system for each table.

Access by more than one user can be synchronized in the following ways:

- **Exclusive lock:** The locked data can only be displayed or edited by a single user. A request for another exclusive lock or for a shared lock is rejected.
- **Shared lock:** More than one user can access the locked data at the same time in display mode. A request for another shared lock is accepted, even if it comes from another user. An exclusive lock is rejected.
- **Exclusive but not cumulative:** Exclusive locks can be requested several times from the same transaction and are processed successively. In contrast, exclusive but not cumulative locks can be called only once from the same transaction. All other lock requests are rejected.

**See also:**

Lock Collisions

# Function Modules for Lock Requests

Das Eigentümerkonzept von Sperren [Ext.]

Sperrkonflikts [Ext.]

Activating a lock object in the ABAP Dictionary automatically creates function modules for setting (*ENQUEUE_<lock object name>*) and releasing (*DEQUEUE_<lock object name>*) locks.

The generated function modules are automatically assigned to function groups. You should not change these function modules and their assignment to function groups since the function modules are generated again each time the lock object is activated.

Never transport the function groups, which contain the automatically generated function modules. The generated function modules of a lock object could reside in a different function group in the target system. Always transport the lock objects. When a lock object is activated in the target system, the function modules are generated again and correctly assigned to function groups.

## Parameters of the Function Modules

### Field Names of the Lock Object

The keys to be locked must be passed here.

A further parameter *X_<field>* that defines the lock behavior when the initial value is passed exists for every lock field *<field>*. If the initial value is assigned to *<field>* and *X_<field>,* then a generic lock is initialized with respect to *<field>*. If *<field>* is assigned the initial value and *X_<field>* is defined as *X*, the lock is set with exactly the initial value of *<field>.*

### Parameters for Passing Locks to the Update Program

A lock is generally removed at the end of the transaction or when the corresponding DEQUEUE function module is called. However, this is not the case if the transaction has called update routines. In this case a parameter must check that the lock has been removed.

Parameter *_SCOPE* controls how the lock or lock release is passed to the update program (see The Owner Concept for Locks). You have the following options:

- _SCOPE = 1: Locks and lock releases are not passed to the update program. The lock is removed when the transaction is ended.
- _SCOPE = 2: The lock or lock release is passed to the update program. The update program is responsible for removing the lock. The interactive program with which the lock was requested no longer has an influence on the lock behavior. This is the standard setting for the ENQUEUE function module.
- _SCOPE = 3: The lock or lock release is also passed to the update program. The lock must be removed in both the interactive program and in the update program. This is the standard setting for the DEQUEUE function module.

### Parameters for Lock Mode

A parameter MODE_<*TAB*> exists for each base table *TAB* of the lock object. The lock mode [Page 205] for this base table can be set dynamically with this parameter. Valid values for this parameter are *S* (shared), *E* (exclusive) and *X* (exclusive but not cumulative).

The lock mode specified when the lock object for the table is created is the default value for this parameter. This default value can however be overridden as required when the function module is called.

If a lock set with a lock mode is to be removed by calling the DEQUEUE function module, this call must have the same value for the parameter MODE_<*TAB*>.

### Controlling Lock Transmission

Parameter _COLLECT controls whether the lock request or lock release should be performed directly or whether it should first be written to the local lock container [Page 212]. This parameter can have the following values:

- *Initial value*: The lock request or lock release is sent directly to the lock server.
- *X* : The lock request or lock release is placed in the local lock container. The lock requests and lock releases collected in this lock container can then be sent to the lock server at a later time as a group by calling the function module FLUSH_ENQUEUE.

### Behavior for Lock Conflicts (ENQUEUE only)

The ENQUEUE function module also has the parameter _WAIT. This parameter determines the lock behavior when there is a lock conflict.

You have the following options:

- *Initial value*: If a lock attempt fails because there is a competing lock, the exception FOREIGN_LOCK is triggered.
- *X* : If a lock attempt fails because there is a competing lock, the lock attempt is repeated after waiting for a certain time. The exception FOREIGN_LOCK is triggered only if a certain time limit has elapsed since the first lock attempt. The waiting time and the time limit are defined by profile parameters.

### Controlling Deletion of the Lock Entry (DEQUEUE only)

The DEQUEUE function module also has the parameter _SYNCHRON.

If *X* is passed, the DEQUEUE function waits until the entry has been removed from the lock table. Otherwise it is deleted asynchronously, that is, if the lock table of the system is read directly after the lock is removed, the entry in the lock table may still exist.

### Exceptions of the ENQUEUE Function Module

- FOREIGN_LOCK: A competing lock already exists. You can find out the name of the user holding the lock by looking at system variable SY-MSGV1.
- SYSTEM_FAILURE: This exception is triggered when the lock server reports that a problem occurred while setting the lock. In this case, the lock could not be set.

If the exceptions are not processed by the calling program itself, appropriate messages are issued for all exceptions.

## Function Modules for Lock Requests

### Reference Fields for RFC-Enabled Lock Objects

The type of an RFC-enabled function module must be completely defined. The parameters of the generated function module therefore have the following reference fields for RFC-enabled lock objects:

| Parameters | Reference fields |
|---|---|
| X_<field name> | DDENQ_LIKE-XPARFLAG |
| _WAIT | DDENQ_LIKE-WAITFLAG |
| _SCOPE | DDENQ_LIKE-SCOPE |
| _SYNCHRON | DDENQ_LIKE-SYNCHRON |

**See also:**

Example for Lock Objects [Page 216]

# Conditions Required of Foreign Keys

All the tables that can be included in a lock object must be linked with foreign keys [Page 19]. There are a number of restrictions to the valid relationships.

1. The foreign key relationships of the tables of the lock object must form a tree. The tables are the nodes of the tree. The links of the tree mean *is the check table of*.
2. The foreign key fields must be key fields of the foreign key table.
3. The foreign key relationships defined between the base tables of the lock objects may not have any field that is checked against more than one other field. A field therefore may not occur twice as foreign key field in a relationship and may not be checked against two different fields in two different foreign key relationships.
4. You must keep one restriction in mind for multi-structured foreign keys [Page 29]. If a field is assigned to a field that is outside the check table, the table containing this field must be in a sub-tree that contains the check table of this foreign key relationship as a root.

These conditions are always satisfied if the key of the foreign key table is an extension of the key of the check table.

Conditions 2, 3 and 4 are meaningless if the particular foreign key field was excluded from the assignment to the key fields of the check table by marking it as generic or setting it to a constant [Page 22]. This is also true for multi-structured foreign keys if the foreign key field refers to a table that is not contained in the lock object.

# Lock Mechanism

Sperrtabelle [Ext.]

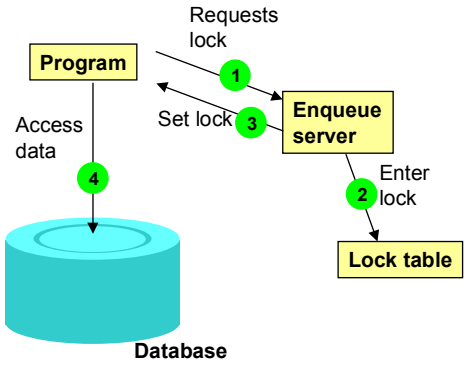
Kollisionen von Sperren [Ext.]

Das R/3 Sperrkonzept [Ext.]

You can synchronize access by several programs to the same data with a logical lock mechanism. This lock mechanism fulfills two main functions:

- A program can tell other programs which data records it is just reading or changing.
- A program can prevent itself from reading data that is just being changed by another program.

The data records of a table to be locked are defined by a logical condition. When a lock is set, this logical condition is entered in a lock table. This entry is retained until it is removed by the program or the program comes to an end. All the locks set by a program are thus removed at the end of the program.

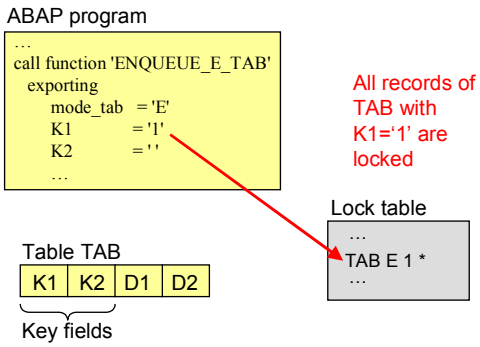



When accessing data records, the records just being edited by other programs can be identified by the entry in the lock table. Such an entry for the lock must define a number of fully specified key fields, that is either a value is passed for the key field or this field is locked generically.

To set locks, a lock object [Page 202] must be defined in the ABAP Dictionary. When this lock object is activated, two function modules (see Function Modules for Lock Requests [Page 206]) are generated with the names *ENQUEUE_<lockobjectname>* and *DEQUEUE_<lockobjectname>*.

If data records are to be locked, you must call function module *ENQUEUE_<lockobjectname>*. The values of the key fields that specify the records to be locked are passed for all the tables contained in the lock object when the function module is called. There is a generic lock if a value is not passed for all the key fields. The function module writes the appropriate lock entry (see Example for Lock Objects [Page 216]). If another program also requests a lock, it will be accepted or rejected depending on the lock mode [Page 205] (see Lock Collisions). The program can then react to this situation.

Locked data records can be unlocked by calling function module *DEQUEUE_<lockobjectname>*. The key values and the lock mode used to set the lock must be passed to the function module.




This lock procedure requires that all programs involved cooperate. Inconsistencies can occur if a program reads or changes data without having previously locked it. When a lock is set, the data records are only protected against changes by another program if this program also requests a lock before accessing the data.

Instead of writing lock requests or lock releases directly in the lock table, it is also possible to collect them first in a local lock container [Page 212]. The collected locks can be sent at a later time as a group. A parameter of the relevant function module controls whether a lock request or lock release is sent directly.

You can find further information about the lock concept and how lock management works in the documentation The R/3 Lock Concept.

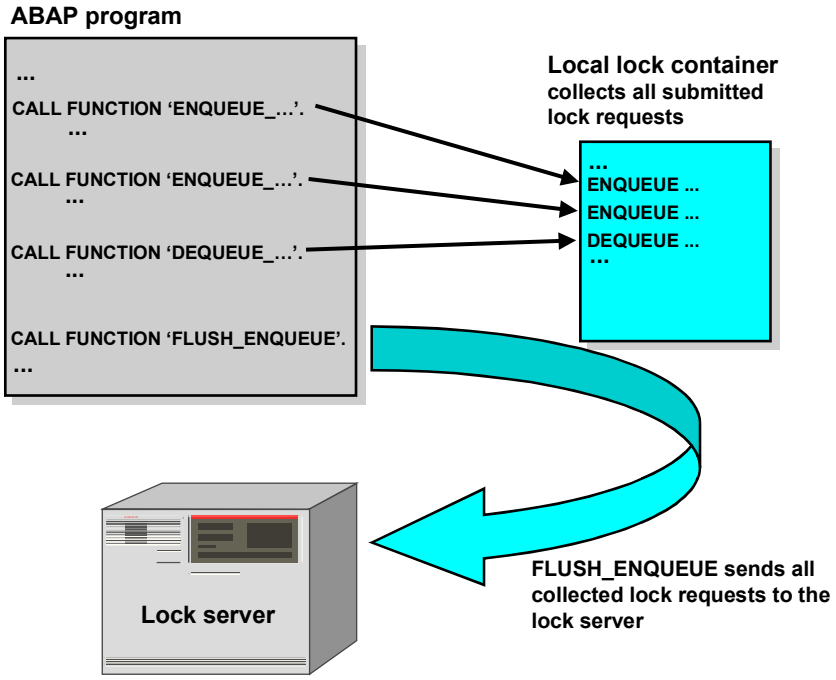
# Local Lock Containers

Lock requests and lock releases can be collected in a local lock container and sent together by calling the function module FLUSH_ENQUEUE.

This has two advantages over sending the lock requests directly:

- Communications with the lock server are minimized if the lock requests are sent as a group.
- The collected lock requests are handled as a group, which means that they are only written to the lock table if this is possible for all the individual requests.

The local lock container is emptied if all the collected lock requests can be executed; otherwise its contents remain unchanged.




The local lock container can be emptied by calling the function module RESET_ENQUEUE. All the collected lock requests or lock releases are deleted. The local lock container is automatically emptied when the corresponding internal session is ended.

Lock requests and lock releases are managed together in the local lock container. When the collected requests are sent, the lock requests are sent first. The lock releases are sent once all the requested locks have been assigned.

The lock requests and lock releases are not adjusted in the local lock container. The order in which the individual requests are written in the local lock container is of no importance.

# Creating Lock Objects

## Procedure

1. Select object type *Lock object* in the initial screen of the ABAP Dictionary, enter an object name and choose *Create*. The name of a lock object should begin with an E (Enqueue).

   The maintenance screen for lock objects is displayed.

2. Enter an explanatory short text in the field *Short text*.

   You can then use the short text to find the lock object at a later time, for example with the R/3 Repository Information System.

3. Enter the name of the primary table of the lock object.

   All other tables in the lock object must be linked with the primary table using foreign keys [Page 19]. There are also some restrictions on the valid foreign key relationships [Page 209].

4. Select the lock mode [Page 205] of the primary table in the field below it.

   The lock mode is used as the default value for the corresponding parameters of the function modules [Page 206] generated from the lock object.

5. Choose *Add* if you want to lock records in more than one table with the lock object.

   A list of all the tables linked with the primary table using valid foreign keys [Page 209] is displayed. Select the appropriate table. The lock mode of the primary table is copied as lock mode. You can change this setting as required, for example you can assign the lock mode separately for each table.

   Similarly, you can add a table linked with the secondary table just added with foreign keys. To do this, place the cursor on the name of the secondary table and choose *Add*.

   If no lock mode is assigned to a table, no lock is set for the entries in this table when the generated function modules are called. You should not assign a lock mode if a secondary table was only used to define a path between the primary table and another secondary table with foreign keys.

6. Save your entries.

   A dialog box appears in which you have to assign the lock object a development class.

7. You can (optionally) exclude lock parameters (see lock objects [Page 202]) from the function module generation on the *Lock parameter* tab page. This makes sense for example if you always want to lock a parameter generically.

   To do this, simply deselect the *Weight* flag for the parameter. The parameter is not taken into consideration in the generated function modules. This parameter is then always locked generically.

   The name of a lock parameter is usually the name of the corresponding table field. If two fields with the same name are used as lock parameters in the lock object from different tables, you must choose a new name for one of the fields in field *Lock parameter.*

8. You can define whether the function modules generated from the lock object should be RFC-enabled on the *Attributes* tab page.

   If you set the *Allow RFC* flag, the generated function modules can be called from within another system with *Remote Function Call*.

   If you permit Remote Function Calls for an existing lock object, you must ensure that the generated function modules are called from within an ABAP program with parameters appropriate for the type. You should therefore check all programs that use the associated function modules before activating the lock object with the new option.

9. Choose .

## Result

When you activate the lock object, the two function modules ENQUEUE_<*lockobjectname*> and DEQUEUE_<*lockobjectname*> are generated from its definition to set and release locks.

You can find information about the activation flow in the activation log, which you can display with *Utilities* → *Activation log*. If errors occurred during activation, the activation log is displayed immediately.

# Deleting Lock Objects

## Prerequisites

When you delete a lock object, the function modules [Page 206] generated when you activated the lock object are automatically deleted as well. These generated function modules might still be in use in programs or classes.

Therefore, before deleting a lock object, find all programs or classes that contain these function modules and remove the calls to the function modules.

## Procedure

1. In the initial screen of the ABAP Dictionary, select object type *Lock object* and enter the lock object name.

   Choose  to find all the programs or classes that are still using the lock object. Remove the lock module calls in the objects you found.

2. *Choose*  .

   A dialog box appears in which you must confirm the deletion request. If the function modules belonging to the lock object are still in use in programs or classes, a corresponding warning appears. In this case you must adjust the programs or classes affected before deleting the lock object.

3. Confirm the deletion request.

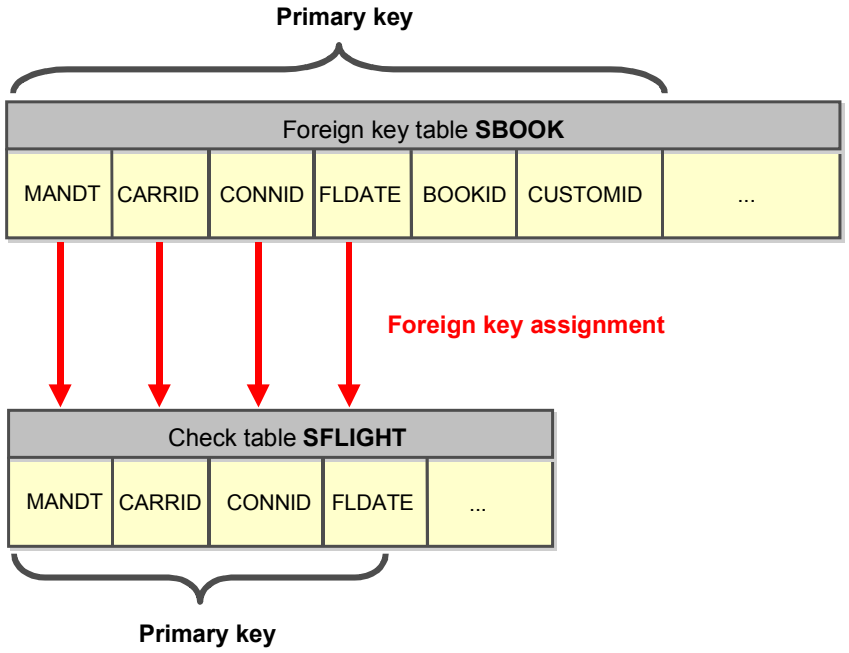
## Result

The lock object is deleted together with the function modules generated from this lock object.

# Example for Lock Objects

When booking flights (see Flight Model [Page 295]) it is important to prevent flights from being overbooked. For this reason, you have to lock the particular flight as well as all the bookings existing for this flight during processing. You can do this with lock object E_BOOKING.

The flights are recorded in table SFLIGHT and the bookings for the flights in table SBOOK. The two tables are linked with a foreign key. Lock object E_BOOKING must therefore contain table SFLIGHT as primary table and table SBOOK as further table.




The lock argument of table SFLIGHT thus contains the fields MANDT, CARRID, CONNID, and FLDATE. The lock argument of table SBOOK thus contains the fields MANDT, CARRID, CONNID, FLDATE, BOOKID and CUSTOMID.

Select exclusive lock mode, that is the locked data can only be displayed and edited by one user.

When the lock object is activated, the following function modules are generated from its definition:

- ENQUEUE_ E_BOOKING (set locks)
- ENQUEUE_ E_BOOKING (release locks)

These function modules can now be linked to ABAP programs.

The following example shows how function module ENQUEUE_ E_BOOKING is called.

```
CALL FUNCTION 'ENQUEUE_E_BOOKING'
   exporting
        mode_sflight     = 'E'
        mode_sbook       = 'E'
        mandt            = sy-mandt
        carrid           = 'LH '
        connid           = 400
        fldate           = '19981129'
        bookid           = 0
        customid         = 0
        x_carrid         = ' '
        x_connid         = ' '
        x_fldate         = ' '
        x_bookid         = ' '
        x_customid       = ' '
        _scope           = '2'
        _wait            = 'X'
        _collect         = ' '
    exceptions
         foreign_lock    = 1
         system_failure  = 2
         others          = 3.
```

Lock modes

Lock parameters

Lock behavior when copying initial value

Pass lock to update program

Behavior in conflict situations

Lock container

With this call, flight LH 400 on Nov. 29,1998 is exclusively (lock mode E) locked in table SFLIGHT together with all the bookings entered in table SBOOK for this flight (since the initial value 0 is transferred for BOOKID and CUSTOMID). The lock is sent to the update program (_SCOPE = ‘2’). If there is a lock conflict, another attempt is made to set the lock after a certain time (_WAIT = ‘X’).

The set locks can be removed by calling the function module DEQUEUE_E_BOOKING as follows:

## Example for Lock Objects

```
CALL FUNCTION 'DEQUEUE_E_BOOKING'
    exporting
         mode_sflight = 'E'                    } Lock modes
         mode_sbook   = 'E'
         mandt        = sy-mandt               }
         carrid       = 'LH'                   }
         connid       = 400                    } Lock parameters
         fldate       = '19981129'             }
         bookid       = 0                      }
         customid     = 0                      }
         x_carrid     = ' '                    }
         x_connid     = ' '                    }
         x_fldate     = ' '                    } Lock behavior when
         x_bookid     = ' '                    } copying initial value
         x_customid   = ' '                    }
         _scope       = '3'                    } Pass lock to update program
         _synchron    = ' '                    } Synchronous deletion of lock entry
         _collect     = ' '.                   } Lock container
```

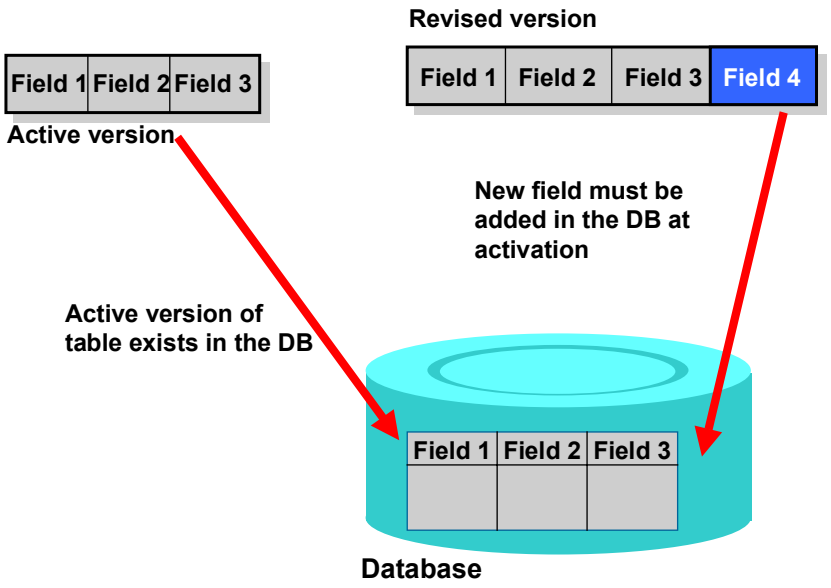
The existing exclusive lock entries for flight LH 400 are deleted in table SFLIGHT and the bookings for this flight are deleted in table SBOOK. The request to delete the lock entries is passed on to the update program (_SCOPE = ‘3’).

**See also:**

Function Modules for Lock Requests [Page 206]

# Adjusting Database Structures

To enable ABAP programs to access database tables correctly, the runtime object of a table [Page 232] must correspond to the structure of the table in the database. If the table is changed in the ABAP Dictionary, you must ensure that the database structure of the table is adjusted to the change in the ABAP Dictionary during activation (when the runtime object is rewritten).




The database structure does not need to be changed for certain changes in the ABAP Dictionary. For example, the database table does not need to be adjusted to a change in the field sequence (except for key fields) in the ABAP Dictionary, since the field sequence in the ABAP Dictionary does not have to correspond to the field sequence in the database. In this case, the changed structure is simply activated in the ABAP Dictionary and the database structure remains unchanged.

The database structure of a table can be adjusted to its changed ABAP Dictionary definition in three ways:

- By **deleting and recreating** the database table The table is deleted in the database. The revised version of the table is then activated in the ABAP Dictionary and created again in the database. Data in the table is lost during this process.
- By **changing the database catalog** (ALTER TABLE) Only the definition of the table is changed in the database. Data in the table is retained. The indexes on the table might have to be rebuilt.
- By **converting the table** (see Conversion Process [Page 221]). The database table is renamed and serves as a temporary buffer for the data. The revised version of the table is activated in the ABAP Dictionary and created in the database. The data is reloaded

**Adjusting Database Structures**

from the temporary buffer to the new database table (with MOVE-CORRESPONDING) and the indexes on the table are built.

The procedure actually used by the system in a particular case depends on the following:

- type of structural change
- database system used
- whether data already exists in the table

If the table does not contain any data, the existing table is deleted in the database and recreated. If there is data in the table, the system tries to change the structure using ALTER TABLE. If the database system used cannot execute the structural change with ALTER TABLE, the table is converted.

Conversion is usually the most resource-intensive method of adjusting structures. But structural changes involving changes to the database catalog can also result in costly internal data reorganizations in some database systems. For details on the processes occurring in the database for structural changes with ALTER TABLE, refer to the documentation of your database system.

Normally you should not adjust the database structure during production. At least all the applications that access the table should be deactivated during the structural adjustment. Since the table data is not consistent during the structural adjustment (in particular during conversion), programs could behave incorrectly when they access this data.

**See also:**

Conversion Problems [Page 226]

# Conversion Process

The following example shows the steps necessary for a conversion

## Initial Situation

Table TAB was changed in the ABAP Dictionary. At this time the length of a field (*Field 3*) was reduced from 60 to 30 places. The ABAP Dictionary therefore contains an active version (where the field has a length of 60 places) and a revised version (where the field only has a length of 30 places) of the table.

The active version of the table is created in the database, that is *Field 3* currently has 60 places in the database. Two secondary indexes that were also created in the database are defined in the ABAP Dictionary for the table. The table already contains data.

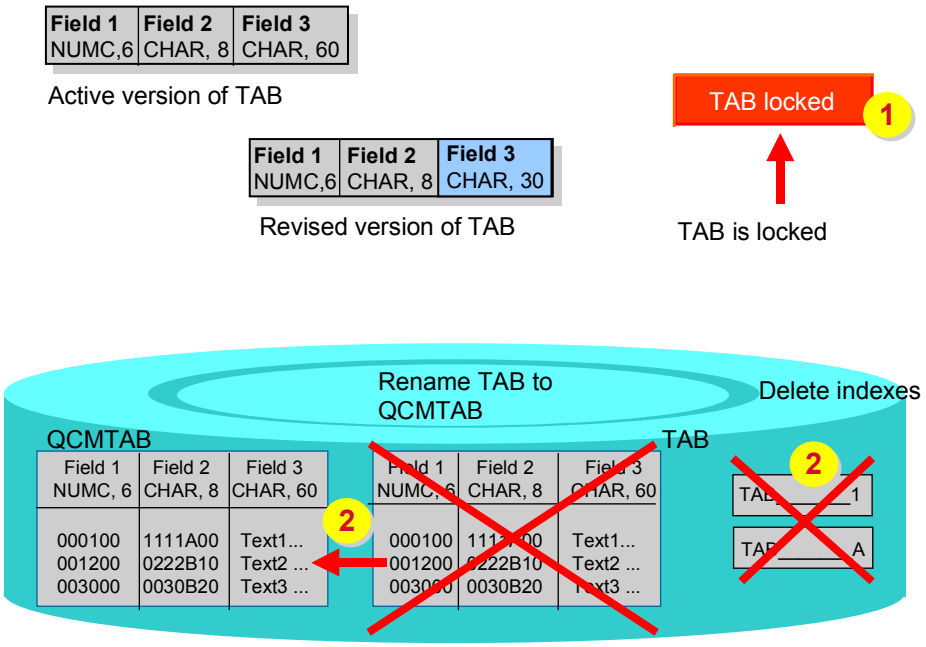
Reducing the length of a field results in a table conversion. The table is converted in a series of 7 steps.

## Step 1: Set Lock

The table is locked against further structural changes. This lock mechanism prevents a new structural change from being made before the conversion has finished correctly. If the conversion terminates after executing Step 2 and before completing Step 4, another structural change could result in a data loss (for example by attempting another conversion).

## Step 2: Rename Table

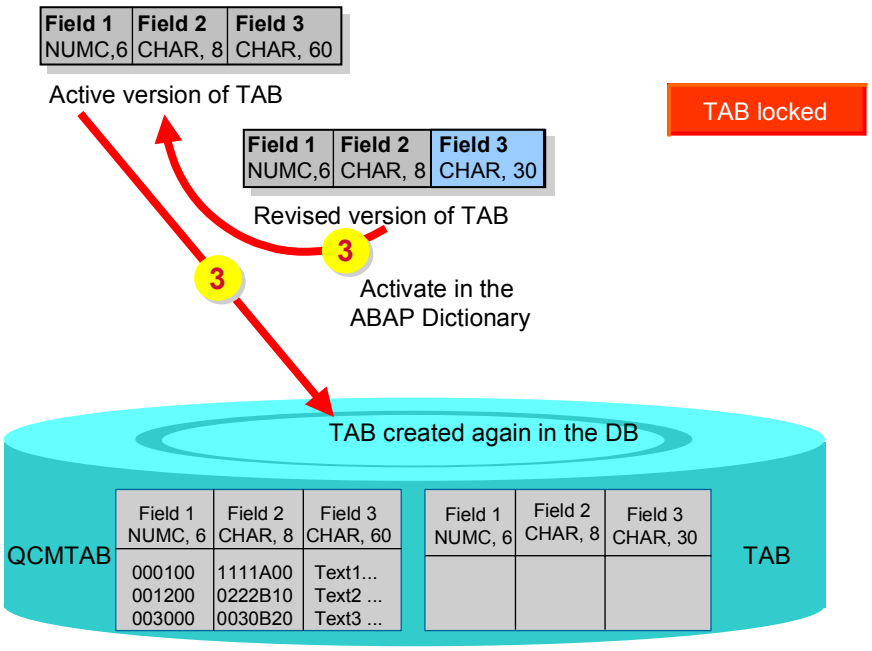
The table is renamed in the database. All indexes on the table are deleted. The name of the new table consists of the prefix QCM and the table name. The name of the temporary table for table TAB is thus QCMTAB.




## Step 3: Activating Tables

The revised version of the table is activated in the ABAP Dictionary. The table is created in the database with its new structure and primary index. The structure of the database table thus corresponds to the structure of the table in the ABAP Dictionary after this step. The database table does not, however, contain any data.

A database lock is also set for the table to be converted. During the conversion, it is not possible to write to the table to be converted from within application programs.

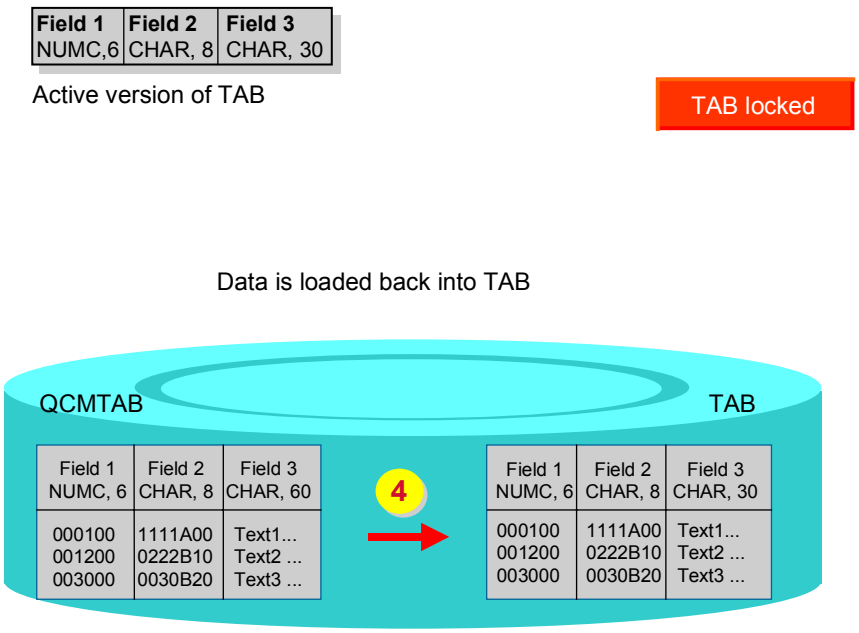



## Step 4: Reload Data

The data is reloaded from the temporary table (QCM table) to the new table (with the ABAP command MOVE-CORRESPONDING). After this step, the data is present in the original table and the temporary table. You can find information about how existing entries are handled in a field when the field type changes in the ABAP documentation about the MOVE-CORRESPONDING statement.

Since the data is present in both the original table and the temporary table during the conversion, more space is required when the conversion is performed. Before converting large tables, you should therefore check whether there is enough space in the tablespace concerned.

When you copy data from the temporary table to the original table, a database commit is stored after 16 MB. The conversion process therefore needs 16 MB of resources in the rollback segment.




When the length of a field is reduced, the extra places are truncated when the field is reloaded. When the length of a key is reduced and there are several records whose new key cannot be distinguished, only one of these records can be reloaded. In general it is not possible to determine in advance which record this will be. In such a case, you should clean up the data of the table before conversion.

The table data is only consistent again once Step 4 has been completed. For this reason, programs must not access the table while the conversion is running. Otherwise a program could behave incorrectly when reading the table since some of the records were not yet copied back from the temporary table. **For this reason conversions may not be running during production.** At least all the applications that use the table to be converted must be deactivated.

## Step 5: Create New Secondary Indexes for the Table

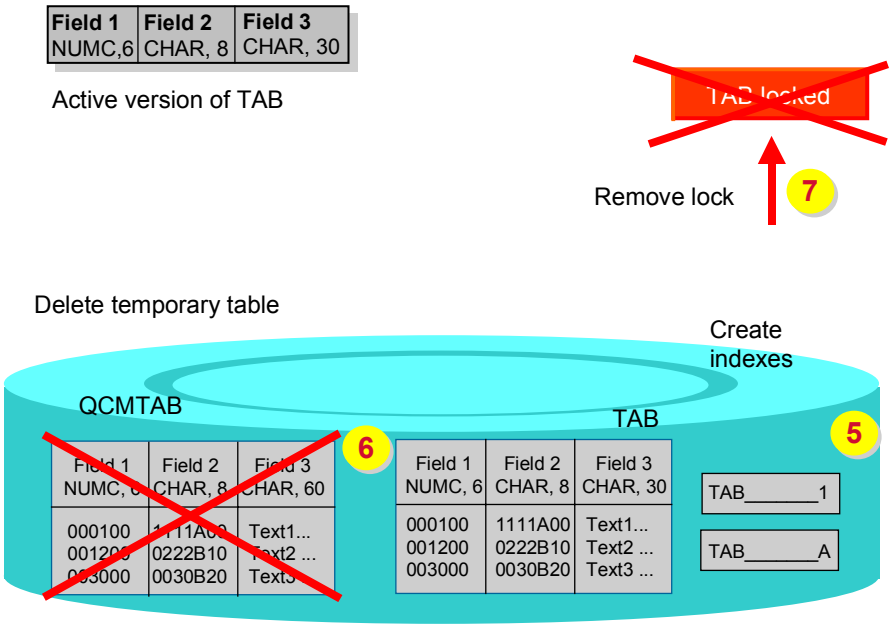
The secondary indexes on the table defined in the ABAP Dictionary are newly created in the database.

## Step 6: Delete the QCM Table

The data in the temporary table (QCM table) is no longer required at the end of the conversion. The temporary table is therefore deleted.

## Step 7: Remove the Lock

The lock set at the beginning of the conversion is removed.




If the conversion terminates, the table stays locked is a reset log is written. Each successful step of the conversion is recorded in this reset log. The restart log is deleted together with the lock when the conversion has been completed. If the conversion terminates (see Conversion Problems [Page 226]), you can find the place where the conversion terminated in the reset log (last step to be executed).

**You must clean up terminated conversions!** Programs that access tables might otherwise not execute correctly. You must find out why the conversion terminated (for example overflow of the corresponding tablespace) and correct it. You must then continue the terminated conversion (see Continuing Terminated Conversions [Page 228]).

The conversion process described is valid only for transparent tables. For pooled and cluster tables [Page 256] , a database table with the name *QCM<table_name>*, corresponding to the structure of the pooled/cluster table, is created and the data is copied to this table. Then the data is deleted from the physical table pool/table cluster. Finally, the data is copied from the QCM table back to the physical table pool/table cluster according to the new table definition.

**See also:**

Conversion Problems [Page 226]

Finding Terminated Conversions [Page 229]

# Conversion Problems

Some of the problems that occur occasionally during conversions are listed here. See also Conversion Process [Page 221].

## Termination due to Tablespace Overflow

Since the data is present in both the original table and the temporary table (QCM table) during the conversion, more space is required when the conversion is performed. Before converting large tables, you should therefore check whether there is enough space in the tablespace concerned.

If there is a tablespace overflow when data is reloaded from the temporary table into the original table, the conversion terminates at this point. If this happens, you must extend the tablespace and resume the conversion in the database utility with *Continue adjustment* (see Continuing Terminated Conversions [Page 228]). The system then continues the conversion at the point at which it terminated.

## Deletion of a Client Field

If the client field is removed from a client-specific table, it may no longer be possible to distinguish the new key of records from different clients.

If deleting a client field resulted in records having the same key, only one of these records can be reloaded into the table. Which record this is depends on the order in which the records are reloaded to the table. You therefore cannot be sure that the reloaded records only come from one client. In such a case, you should clean up the table **before** conversion.

## Data Loss when Keys are Shortened

If a table key is shortened (for example, by removing or shortening the field length of key fields), the new key of existing table records can no longer be distinguished. When data is reloaded from the temporary table, only one of these records can be reloaded into the table (since the database does not allow duplicate records).

It is not possible to say in advance which of the records can be reloaded. If there are particular records you want to reload, you must clean up the table **before** the conversion!

## Type Conversion not Possible

During a conversion, the data is copied from the temporary table back to the database table using the ABAP command MOVE-CORRESPONDING. A type can only be changed during a conversion if this can be executed with MOVE-CORRESPONDING. If a field type cannot be changed using MOVE-CORRESPONDING, the conversion terminates when the data is reloaded into the original table.

If the conversion terminates because a particular type change is not supported, you must restore the old state before the conversion.

To do this, you need to delete the database table in the first step with database means and rename the QCM table to its original name. In the second step, you have to remove the lock in the database utility with *Unlock table*. In the third step you must reconstruct the runtime object for the table using the database utility (see Editing Database Tables and Indexes [Page 242]) and return the table definition in the ABAP Dictionary to the state it had before the conversion. The last step is to activate the table in the ABAP Dictionary.

Note that problems occurring during type conversions depend on the contents of the field concerned.

If the accuracy of a DECIMAL field (the number of places before the period) is reduced and there are entries in the field in which all positions before the period are used, a termination occurs when the records are reloaded into the table. If there are no such entries, the records can be reloaded.

## Termination during Conversion of Pooled or Cluster Tables

When a pooled or cluster table is converted (see Pooled and Cluster Tables [Page 256]), the QCM table is created in the database as a transparent table and the data from the pooled or cluster table is copied to it. There is an upper limit for the number of fields in a database table depending on the database system used.

It is therefore not possible to convert pooled and cluster tables if the number of fields in the pooled or cluster table exceeds the maximum number of fields possible in a database table.

**See also:**

Continuing Terminated Conversions [Page 228]

Finding Terminated Conversions [Page 229]

The Database Utility [Page 240]

# Continuing Terminated Conversions

## Prerequisites

If a conversion terminates, the lock entry for the table set in the first step is retained (see Conversion Process [Page 221]). The table therefore cannot be edited with the ABAP Dictionary maintenance tools (Transaction SE11).

If a table conversion terminates, you must correct it. Applications that access this table might not otherwise be able to read any data and therefore malfunction or fail to run.

## Procedure

1. In the initial screen of the ABAP Dictionary, choose *Utilities* → *Database utility* (Transaction SE14).

   The initial screen for the database utility appears.

2. Select object type *Tables*, enter the table name and choose *Edit*.

   You will go to the maintenance screen for database tables.

3. Choose *Analyze adjustment*.

   The next screen lists the conversion steps that were processed correctly. It also shows the status of the tables involved in the conversion (table with old structure, QCM table and table with new structure). It can also be used to display the existing syslog entries or short dumps.

4. Choose *Object log*.

   You can usually find out the exact cause of the termination from the *object log*. If the object log does not provide any information about the error, you have to analyze the system log or the short dumps.

5. Correct the errors found in the object log or syslog or in the short dumps (see Conversion Problems [Page 226]).

   The conversion can usually only be continued once the error causing the termination has been corrected.

6. Go back to the previous maintenance screen and continue the conversion there with *Continue adjustment*.

## Result

The system now tries to continue the conversion from the termination point.

Besides *Continue adjustment*, there is also the option *Unlock table*. This option only deletes the lock entry that exists for the table. Never choose *Unlock table* for a terminated conversion if the data only exists in the temporary table, that is the conversion terminated after Step 2 was completed and before Step 4 was completed.

# Finding Terminated Conversions

## Procedure

1. In the initial screen of the ABAP Dictionary, choose *Utilities* → *Database utility* (Transaction SE14).

   The initial screen of the database utility [Page 240] appears.

2. Choose *DB requests* → *Terminated*.

   The terminated conversions in the system are listed. You can find the table name in the first column of the list.

3. Double click on the corresponding line to go to the maintenance screen of the database utility for the particular table.

   You can analyze the reason for the termination there and continue the terminated conversion. See Continuing Terminated Conversions [Page 228].

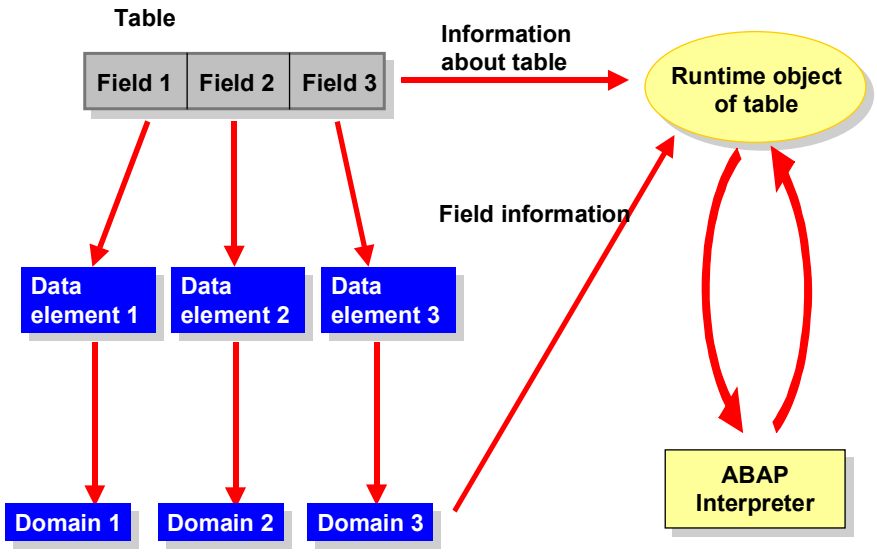
   

   If a table is locked due to a terminated conversion, this must be corrected. Applications that access this table might not otherwise be able to read any data and therefore malfunction or fail to run.

# Activation

When tables, types (data elements, structures, table types) and views are activated, they are placed at the disposal of the runtime environment in the form of runtime objects. These runtime objects contain the information about the object in a form that is optimal for access by ABAP programs and screens. The runtime objects are buffered so that ABAP programs and screens can access the information relevant to them quickly.

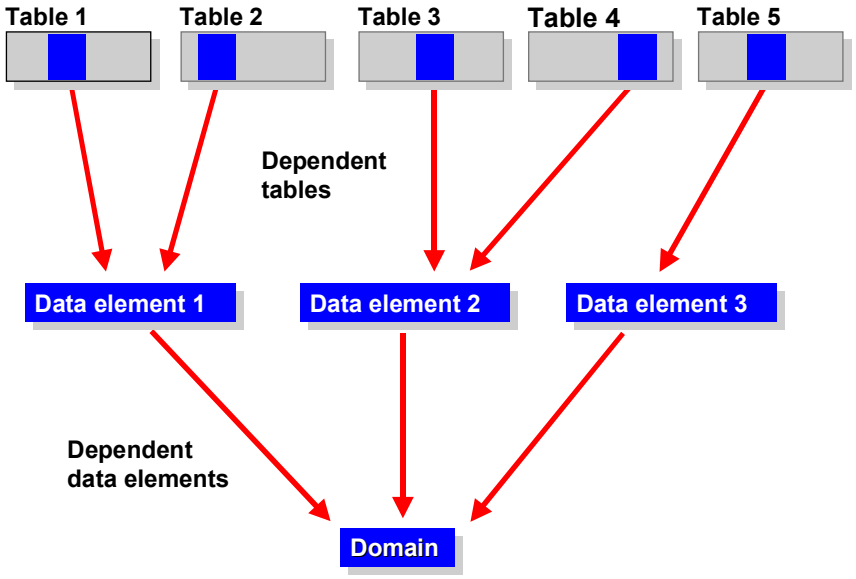
The information about a table is divided amongst domains, data elements, field definitions, and the table definitions in the ABAP Dictionary. The runtime object of the table contains this information in an optimized form.




ABAP programs and screens get the information they require about ABAP Dictionary objects from the runtime objects. Changes to ABAP Dictionary objects are determined by comparing the time stamps (see Runtime Objects [Page 232]). Changes to ABAP Dictionary objects therefore become effective in all system components when the objects are activated.

When an object is activated, all the objects dependent on it are also reactivated.

After changing a domain, for example changing the data type or the length, all tables in which a field refers to this domain must be reactivated. This ensures that all these tables are adjusted to the changed technical field information.




Activating an object can therefore affect numerous dependent objects. Prior to activating a modified object, you should therefore find out what the effect of this action will be. You can display all the objects that are dependent on an object with the *Where-used list* in the maintenance screen of the object.

You can also activate a large number of objects simultaneously using the mass activation program [Page 233].

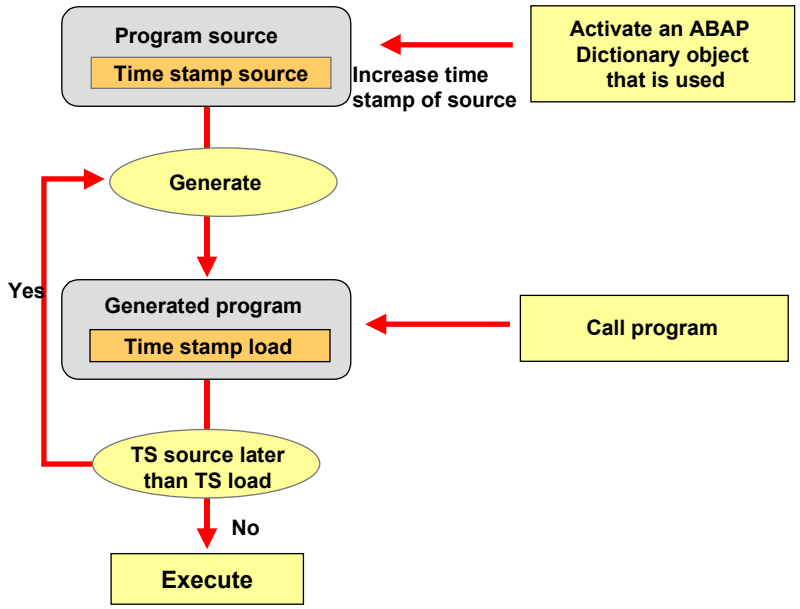
If a long runtime is expected in an activation because of a large number of dependent objects (for example, if a domain that is used in many tables is to be activated), you should choose Activating Objects in the Background [Page 234] for the object.

# Runtime Objects

Tables, views and types (data elements, structures and table types) have runtime objects that contain the information about the object that is relevant for ABAP programs or screens. The runtime object (nametab) is generated the first time an object is activated and subsequently adjusted to the most recent version of the object each time it is activated thereafter. The runtime objects are buffered so that ABAP programs and screens can access the information relevant to them quickly.

Time stamps are used to ensure that ABAP programs and screens always access the most up-to-date information. When an object is activated in the ABAP Dictionary, the time stamp of the runtime object and the time stamp of all programs and screens that use this object are adjusted. However, the time stamp of dependent programs and screens is only adjusted if there was a change relevant for programs or screens during activation.

The next time the program or screen is called, the time stamps can be compared to determine whether the program or screen can be executed directly or must be regenerated. This avoids unnecessary generating, and several consecutive changes to an object can be made in a single generating step.




In the maintenance screen of a table, view or type (data element, structure, table type), you can display the corresponding runtime object with *Utilities* → *Runtime object* → *Display*. Choose  for explanations about the displayed information.

# Mass Activation

If you want to activate a large set of objects simultaneously, for example, after importing data, you can use mass activation with the program RADMASG0. This program is called the mass activation program in the following text.

The mass activation program is automatically called when a transport request is imported into a system. The mass activation program must get a list of ABAP Dictionary objects. All objects in the list are then activated in one action.

The mass activation program has two advantages over activating objects one at a time:

- If dependent tables are affected by changes to different domains or data elements, they only have to be reactivated once.
- Related objects, for example, a domain and its associated value table, can be activated together. If you are activating objects individually, however, you need to activate each of the objects separately and in the correct sequence. For example, before activating a table, you must activate all domains and data elements which are referred to by the table fields.

You can pass the ABAP Dictionary objects to be activated to the mass activation program as follows:

- **Transport request:** All the Dictionary objects contained in the transport request are activated.
- **External table:** The external table must be a pooled table from the ATAB pool and must have the same structure as table TACOB. You can also define table TACOB itself. All the objects entered in this table are activated (see Activating in the Background [Page 234]).
- **Direct objects:** The objects to be activated can also be entered directly with an input template when the mass activation program is called.

You must call the mass activation program from Reporting (SE38). The mass activation program has several input parameters. For an explanation of them and possible entries, use the F1 and F4 help.

# Activating Objects in the Background

## Prerequisites

It is particularly important to activate objects in the background if the activation of a large number of objects is likely to result in long runtimes. In this case you should select a start time that ensures that the objects are activated at a time when the system is not so busy.

## Procedure

1. Choose *System* → *Services* → *Table maintenance* → *Extended table maintenance*.

   The initial screen of *Extended Table Maintenance* appears.

2. Enter table name *TACOB* and choose *Maintain*.

   Specify the object to be activated in the next screen with the object type and object name.

3. Enter the object types and object names.

   There is an input help for the *Otyp* field that displays the possible object types.

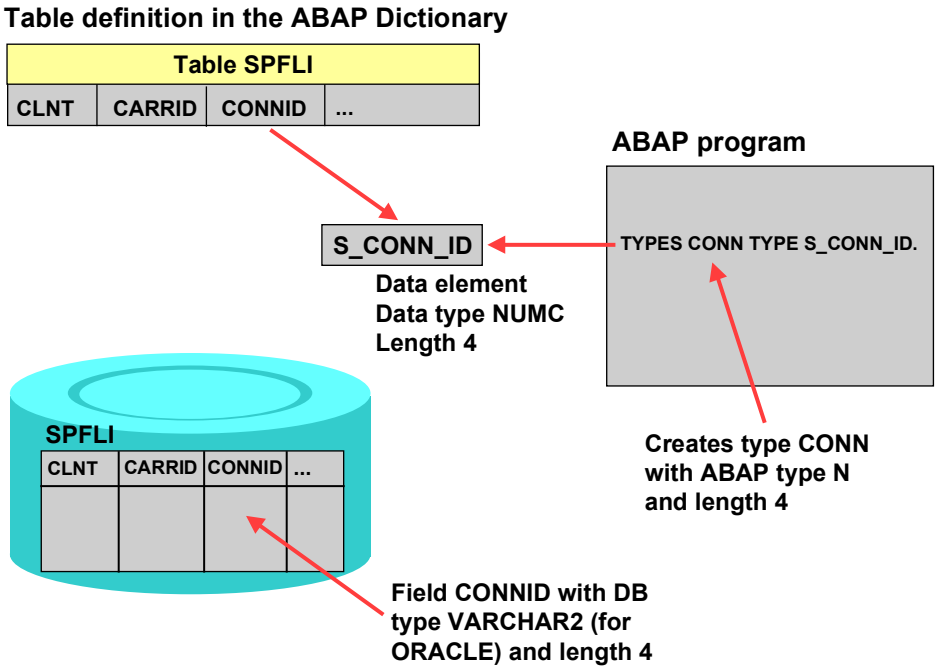
4. Save your entries and exit *Extended table maintenance*.

   With the mass activation program [Page 233], now schedule a background job for activating the entries in table *TACOB* (call Report RADMASG0).

# Data Types in the ABAP Dictionary

The data type in the ABAP Dictionary is the user's view on the data, that is, the data format at the user interface. This data format depends on the database system used. When a table is defined in the ABAP Dictionary, the data types defined in the ABAP Dictionary are copied to the data types of the database system used.

If an ABAP Dictionary object (data element, structure, table type, table, view) is used in an ABAP program, the Dictionary data types of the object fields are converted to the corresponding ABAP data types.




Note that some data types have a predefined length and set templates for output.

## Existing Data Types

**ACCP**: Posting period. The length is set to 6 places for this data type. The format is YYYYMM. In input and output, a point is inserted between the year and month, so the template of this data type has the form '____.__'.

**CHAR**: Character string. Fields of type CHAR may have a maximum length of only 255 in tables. If longer character fields are to be used in tables, you must choose data type LCHR. There are no restrictions on the length of such fields in structures.

**CLNT**: Client. Client fields always have three places.

**CUKY**: Currency key. Fields of this type are referenced by fields of type CURR. The length is set to 5 places for this data type.

**CURR**: Currency field. Equivalent to an amount field DEC. A field of this type must refer to a field of type CUKY (reference field). The maximum length for this data type is 31 places.

**Data Types in the ABAP Dictionary**

**DATS**: Date. The length is set to 8 places for this data type. The output template can be defined with the user profile.

**DEC**: Counter or amount field with decimal point, sign, and commas separating thousands. A DEC field has a maximum length of 31 places.

**FLTP**: Floating point number. The length (including decimal places) is set to 16 places for this data type.

**INT1**: 1-byte integer between 0 and 255. The length of this data type is set to 3 places.

**INT2**: 2-byte integer between -32767 and 32767. Fields of this type should only be used for length fields. These long fields are positioned immediately in front of a long field (type LCHR, LRAW). With INSERT or UPDATE on the long field, the database interface enters the length which was actually used in the length field. The length is set to 5 places for this data type.

**INT4**: 4-byte integer between -2177483647 and 2177483647. The length is set to 10 places for this data type.

**LANG**: Language key. Has its own field format for special functions. This data type always has length 1. The language key is displayed at the user interface with 2 places, but is only stored with 1 place in the database. The conversion exit ISOLA converts the display at the user interface for the database and vice versa. This conversion exit is automatically allocated to a domain with data type LANG at activation.

**LCHR**: Character string of arbitrary length, but with a minimum of 256 characters. Fields of this type must be located at the end of transparent tables and must be preceded by a length field of type INT2. If there is an INSERT or UPDATE in ABAP programs, this length field must be filled with the length actually required.

**LRAW**: Uninterpreted byte string of arbitrary length, but with a minimum length of 256. Fields of this type must be located at the end of transparent tables and must be preceded by a length field of type INT2. If there is an INSERT or UPDATE in ABAP programs, this length field must be filled with the length actually required.

**NUMC**: Long character field in which only numbers can be entered. The length of this field is limited to a maximum of 255 places.

**PREC**: Accuracy of a QUAN field. The length is set to 2 places for this data type.

**QUAN**: Quantity. Equivalent to an amount field DEC. A field of this type must always refer to a units field with UNIT format (reference field). The maximum length for this data type is 31 places.

**RAW**: Uninterpreted sequence of bytes. Fields of type RAW may have only a maximum length of 255 in tables. If longer raw fields are required in tables, you should select data type LRAW.

**RAWSTRING**: Uninterpreted byte string of variable length This type can only be used in types (data elements, structures, table types) and domains. It cannot be used in database tables. In ABAP, this type is implemented as a reference to a storage area of variable size.

**STRING**: Character string with variable length This type can only be used in types (data elements, structures, table types) and domains. It cannot be used in database tables. In ABAP, this type is implemented as a reference to a storage area of variable size.

**TIMS**: Time. The length is set to 6 places for this data type. The format is hhmmss. The template for input and output has the form '__.__.__'.

**UNIT**: Units key. Fields of this type are referenced by fields of type QUAN. The length of this data type is set to 2 or 3 places.

**VARC**: Character field of variable length. Creation of new fields of this data type is not supported as of Release 3.0. However, existing fields with this data type can still be used.

When using the numeric data types CURR, DEC, FLPT, INT2, INT4 and QUAN, you can choose whether or not a sign should be output on screens.

With the data types CURR, DEC and QUAN, commas separating thousands and decimal points are set by the system. The output length (number of places plus the number of necessary editing characters such as commas separating thousands and decimal points) is therefore greater than the specified length.

The system omits editing characters if the output length specified in the domain maintenance is too small.

**See also:**

Mapping of the ABAP Data Types [Page 238]

# Mapping of the ABAP Data Types

The ABAP processor uses ABAP data types in the work areas for data.

Possible ABAP data types:

**C**: Character

**D**: Date, format YYYYMMDD

**F**: Floating point number in DOUBLE PRECISION (8 bytes)

**I:** Integer

**N**: Numeric character string of arbitrary length

**P**: Amount or counter field (packed; implementation depends on hardware platform)

**S**: Time stamp YYYYMMDDHHMMSS

**T**: Time of day HHMMSS

**V**: Character string of variable length, length is given in the first two bytes

**X**: Hexadecimal (binary) storage

**STRING**: Character string of variable length

**XSTRING**: Uninterpreted byte string of variable length

If a data element or a field of an ABAP Dictionary object (structure, table type, table, view) is used in an ABAP program, the Dictionary data type is converted to the corresponding ABAP data type.

## Mapping of the ABAP Dictionary and ABAP Processor Data Types

| ABAP Dictionary type | ABAP type |
|---|---|
| ACCP | N(6) |
| CHAR n | C(n) |
| CLNT | C(3) |
| CUKY | C(5) |
| CURR n,m | P((n+1)/2) DECIMAL m |
| DEC n,m | P((n+1)/2) DECIMAL m |
| DATS | D(8) |
| FLTP | F(8) |
| INT1 | X(1) |
| INT2 | X(2) |
| INT4 | X(4) |
| LANG | C(1) |

| | |
|---|---|
| NUMC n | N(n) |
| PREC | X(2) |
| QUAN n,m | P((n+1)/2) DECIMAL m |
| RAW n | X(n) |
| TIMS | T(6) |
| UNIT | C(n) |
| VARC n | C(n) |
| LRAW | X(n) |
| LCHR | C(n) |
| STRING | STRING |
| RAWSTRING | XSTRING |

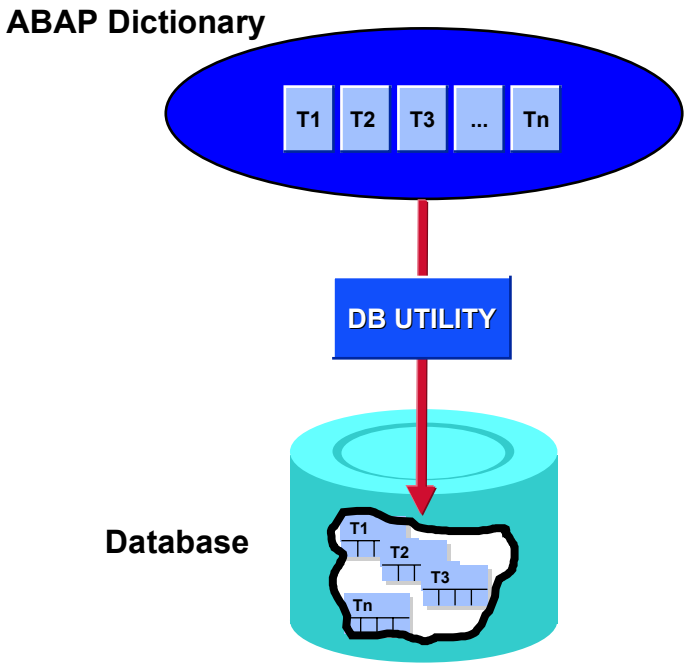
The characters used in the table mean:

**n**: number of places of the field in the ABAP Dictionary

**m**: number of decimal places of the field in the ABAP Dictionary

# The Database Utility

The database utility is the interface between the ABAP Dictionary and the relational database underlying the R/3 System. The database utility allows you to edit (create, delete and adjust to changes to their definition in the ABAP Dictionary) database objects derived from objects of the ABAP Dictionary.




## Editing Database Objects

You can call the database utility from the initial screen of the ABAP Dictionary with *Utilities* → *Database utility* (Transaction SE14). The initial screen for the database utility appears.

You can use the database utility to edit all the database objects that are generated from objects of the ABAP Dictionary. These are database tables that are generated from transparent tables or physical table pools or table clusters, indexes, database views and matchcode pooled tables or matchcode views.

Since different functions are required for different object types, a separate maintenance screen is displayed for each object type. You can go to the corresponding maintenance screen by entering the *object name* and selecting the *object type* in the initial screen of the database utility. You go directly to this screen from the maintenance screen of the object in the ABAP Dictionary with *Utilities* → *Database utility*.

If you want to use the database utility, you need authorization for authorization object S_DDIC_OBJ, e. g. S_DDIC_ALL.

You can find information about the functions provided in:

- Editing Tables and Indexes in the Database [Page 242]
- Editing Views in the Database [Page 244]
- Editing Matchcodes in the Database [Page 245]
- Editing Table Pools and Table Clusters in the Database [Page 247]

## Analyzing and Continuing Terminated Conversions

If a conversion [Page 221] terminates, you can find the reason using the database utility. After correcting the error you can continue the conversion with the database utility. See:

- Conversion Problems [Page 226]
- Continuing Terminated Conversions [Page 228]
- Finding Terminated Conversions [Page 229]

## Administration of Requests for Database Modifications

The database utility provides a number of options for administering and monitoring requests for database modifications. You can perform these functions directly in the initial screen of the database utility.

You can:

- Schedule jobs for mass processing [Page 253]
- Display requests for mass processing [Page 251]
- Display logs for mass processing [Page 254]
- Display temporary tables without restart logs [Page 255]

**See also:**

Adjusting Database Structures [Page 219]

Activation [Page 230]

# Editing Tables and Indexes in the Database

## Basic Functions

The functions *Create database table*, *Delete database table* and *Activate and adjust database* are provided for transparent tables. You can execute such a function by selecting the processing type [Page 248] and pressing the corresponding button.

- **Create database table:** The table is created in the database with its active version and the primary index. Active secondary indexes are also created in the database if this was not explicitly excluded when the index was defined (see Creating Secondary Indexes [Page 77]).
- **Delete database table** The table and all its indexes are deleted from the database.
- **Activate and adjust database** The revised version of the table is activated in the database table is adjusted [Page 219] to this modified table definition. You can select *Keep data* or *Delete data*. For *Delete data,* the table is deleted in the database and created again with the new definition. Data in the table is lost. With *Keep data,* there is an attempt to perform an adjustment with an ALTER TABLE. If this is not possible, a table conversion [Page 221] is triggered.

  Since pooled and cluster tables [Page 256] are not separate tables in the database, the functions *Create database table* and *Delete database table* are not applicable for these table categories. The *Delete data* function is offered instead. It deletes the table data from the corresponding physical table pool or table cluster.

## Functions for Indexes

You go to the maintenance screen for indexes with *Goto* → *Indexes*. A list of all the table indexes that exist in the ABAP Dictionary is displayed. Select the required index by double-clicking. The following functions are offered in the next screen:

- **Create database index:** Create a secondary index or the primary index of a transparent table in the database.
- **Delete database index:** Delete a secondary index of a transparent table in the database. The primary index of a transparent table created in the database cannot be deleted as long as the table still exists in the database.
- **Activate and adjust database** The index is deleted from the database. The revised version of the index is activated. The index is then created and built again in the database.

## Other Functions

The database utility also offers a number of check and repair functions for tables.

- **Maintain storage parameters:** Storage parameters [Page 249] that influence the database settings (such as extent sizes) for the table can be maintained for transparent tables. You can display the corresponding maintenance screen with *Storage parameters*.

- **Check consistency:** You can compare the table definition in the database with the runtime object of the table [Page 232] with *Extras → Database object → Check.* The indexes on the table in the ABAP Dictionary are compared with the indexes on the database. You can compare the runtime object of the table with the information entered in the ABAP Dictionary maintenance screen with *Extras → Runtime object → Check*. Both definitions are displayed. The differences found are highlighted. In both cases you can switch between a *delta* (only differences) or *full* (all information) display of the check results.
- **Check the Existence of Data:** With *Table → Data exists?* you can check if the table contains data. Data is selected for all clients. This function is of use for example to check whether a table is empty prior to a conversion. You can display the data in the logon client with *Table → Table contents*.
- **Display the Runtime Object and Database Table:** You can list the structure of the table in the database and the indexes defined for the table with *Extras → Database object → Display*. With *Extras → Runtime object → Display* you can display the runtime object of the table. You can get more information about interpreting the displayed data with the  button.
- **Force conversion:** You can trigger a table conversion [Page 221] with *Extras → Force conversion*. This function is of use for example if only the storage parameters [Page 249] or technical settings [Page 30] of the table were modified, but the table structure was not changed. Some of the modified settings can only take effect in the database once the table has been converted. A table conversion cannot be triggered with the *Activate and adjust database* function in this case because the table structure has not changed.
- **Reconstruct:** You can create a suitable runtime object [Page 232] for the database table with *Table → Reconstruct*. This runtime object only contains the information about the table that is available in the database, such as the field name and data types. It does not contain other information about the table that only exists in the ABAP Dictionary, such as specifications for buffering. Only user DDIC has the authorization to execute this function. Only use this function to temporarily correct inconsistencies (database status and runtime object differ) and be sure to adjust the table in the usual way in the ABAP Dictionary and activate it there at a later time.

# Editing Views in the Database

## Basic Functions

The database utility provides the functions *Create database view*, *Delete database view* and *Activate and adjust database* for editing database views [Page 106]. You can execute such a function by choosing a processing type [Page 248] and pressing the corresponding button.

- **Create database view:** A database view defined in the ABAP Dictionary is physically created in the database.
- **Delete database view:** The database view is deleted from the database.
- **Activate and adjust database:** The view is deleted from the database. The revised version of the view is activated and this version of it is created again in the database.

## Other Functions

The database utility also offers a number of check functions for database views:

- **Check consistency:** You can compare the view definition in the database with the runtime object [Page 232] of the view with *Extras → Database object → Check.* You can compare the runtime object of the view with the information given in the ABAP Dictionary maintenance screen with *Extras → Runtime object → Check*. Both definitions are displayed. The differences found are highlighted. In both cases you can choose either a delta (only differences) or full (all information) display of the check results.
- **Display the view definition in the database:** You can list the definition of the view in the database with *Extras → Database object → Display.*
- **Display the Runtime Object:** You can list the information in the runtime object for the view with *Extras → Runtime object → Display*. You can get more information about interpreting the displayed data with the  button.

**See also:**

Views [Page 95]

# Editing Matchcodes in the Database

## Functions for Matchcode Objects

The database utility provides the functions *Create database table* and *Delete database table* for editing matchcode objects.

You can execute such a function by choosing a processing type [Page 248] and then pressing the corresponding button.

- **Create database table:** The physical matchcode pool *M_<name of MC object>* is created in the database.
- **Delete database table** The physical matchcode pool *M_<name of MC object>* is deleted from the database.

## Functions for Matchcode IDs

If you wish to edit individual matchcode IDs, you can display the matchcode IDs defined for the matchcode object with *Goto → Matchcode ID*. A list of all matchcode IDs for the matchcode object is displayed from which you can select the IDs you require.

This function distinguishes between transparent and physically implemented matchcode IDs.

### Functions for Physically Implemented Matchcode IDs

The following functions are supported for physically implemented matchcode IDs (update types A, S, P):

- **Build data:** The data for the matchcode ID is built in the matchcode pool.
- **Delete data:** The data for the matchcode ID is deleted from the matchcode pool.
- **Activate and adjust database** You have the following options for physically implemented matchcode IDs:

*With data construction*: The data for the matchcode ID is deleted from the matchcode pool. The matchcode ID is activated in the ABAP Dictionary. The data for the matchcode ID is then rebuilt with the new structure.

*Without data construction*: The data for the matchcode ID is deleted from the matchcode pool. The matchcode ID is activated in the ABAP Dictionary. The matchcode pooled table therefore contains no data for the matchcode ID.

### Functions for Transparent Matchcode IDs

The following functions are supported for transparent matchcode Ids (update type I):

- **Create database view:** The view *M_<MC object><ID>* for the matchcode search is created in the database. This function is normally executed when a transparent matchcode ID is activated in the ABAP Dictionary.
- **Delete database view** The view *M_<MC object><ID>* created for the matchcode search is deleted from the database.
- **Activate and adjust database** The view *M_<MC object><ID>* created for the matchcode search is deleted from the database. The matchcode ID is activated in the

ABAP Dictionary and the view is created in the database again with the new active version of the matchcode ID.

# Editing Pools and Clusters in the Database

## Basic Functions

The database utility provides the functions *Create database table,* and *Delete database table* for editing pooled and cluster tables [Page 256]. You can execute such a function by choosing a processing type [Page 248] and pressing the corresponding button.

- **Create database table:** Create the physical pool or cluster in the database.
- **Delete database table** Delete the physical pool or cluster from the database.

## Other Functions

The database utility also offers a number of check functions for pooled and cluster tables:

- **Maintain storage parameters:** Storage parameters [Page 249] for pooled and cluster tables that affect the database settings of the table (such as extent sizes) can be maintained with *Goto* → *Storage parameters*.
- **Check consistency:** You can compare the definition of the pool or cluster in the database with the runtime object [Page 232] of the pool or cluster with *Extras* → *Database object* → *Check.* You can compare the runtime object of the pool or cluster with the information entered in the ABAP Dictionary maintenance screen with *Extras* → *Runtime object* → *Check*. In both cases you can choose either a *delta* (only differences) or *full* (all information) display of the check results.
- **Display the runtime object and define it in the database** You can list the structure of the pool or cluster in the database with *Extras* → *Database object* → *Display* You can list the information contained in the runtime object with *Extras* → *Runtime object* → *Display*. You can get more information about interpreting the displayed data with the  button.

**See also:**

Pooled and Cluster Tables [Page 256]

# Processing Types

The database utility can be used to make modifications to database objects with various processing types.

### Direct

The required database modifications are carried out immediately.

This processing type is not suitable for converting large tables. First of all, the conversion places a great load on the system. Secondly, the conversion can terminate due to a runtime restriction. You should therefore choose the processing type *Background* when converting large tables.

### Background

A background job is scheduled for the required database modifications.

You can specify the time at which background processing is to begin. If you wish, you can choose to start background processing immediately. The advantage of selecting this processing type rather than *Direct* is that there is no danger of the operation terminating as the result of a runtime restriction.

### Entering for mass processing

If you choose this processing type, entries are created with the corresponding function in a system table (TBATG). The requests collected in this table can then be processed in the background at fixed times (generally overnight). The background process for editing the requests must be scheduled explicitly as a job (see Scheduling Jobs for Mass Processing [Page 253]).

The objects already scheduled for mass processing can be displayed from the initial screen of the database utility with *DB requests* → *Mass processing*. The interpretation of the displayed list is described in Displaying Requests for Mass Processing [Page 251].

The background program writes messages in a log file. You can display these day logs starting with the initial screen of the database utility with *Extras* → *Logs*. The interpretation of the displayed list is described in Displaying Requests for Mass Processing [Page 254].

# Storage Parameters

Storage parameters can be set for database tables (transparent tables, table pools, table clusters). These parameters affect the way in which tables are handled in the database. For example, the storage parameter defines the tablespace in which the table should be created in the database and the size to be selected for the table extents.

The existing storage parameters and the particular settings that are allowed depend on the database system in use. To find out which parameters can be set in your database system, please consult the system documentation for your database system.

## Which Storage Parameters are Effective for a Table?

When a table is activated for the first time, the storage parameters for the table are computed from the technical settings of the table (if there are no technical settings, the default values of the database system are used) and the table is created in the database with these storage parameters.

If the table is deleted from the database and created again at a later time (for example due to a table conversion [Page 221]), the storage parameters to be used are defined in a fixed hierarchy.

If there are user-defined storage parameters, these are used. If there are no user-defined storage parameters, the valid parameters used previously in the database are used. If there are no such parameters either (for example if the table was previously a pooled or cluster table that was converted to a transparent table by conversion), the storage parameters are computed from the technical settings. If there are no technical settings for the table either, the default values of the database system are used.

If the technical settings of a table that already exists in the database are changed and activated, the valid storage parameters for the table in the database are not changed. In this case you have to change the valid storage parameters manually with the database utility.

## Displaying and Maintaining the Storage Parameters

In the database utility, go to the table maintenance screen. You can display the storage parameters that are currently valid for the table and its indexes with *Storage parameters*.

The parameters displayed here depend on the database system used. You can find information about the meaning of the displayed parameters by selecting the parameter and pressing F1.

You can change some storage parameters directly. Choose  . You can now enter values for the changeable parameters. Make your changes and choose  *Apply*. The changed settings are now active in the database.

For example, if you change the extent size directly, the next table extent [Page 32] is created with the new size. This change has no effect on existing extents.

The tablespace in which the table lies cannot be changed directly. Such a change requires that you delete the table in the old tablespace and create a new one in the new tablespace.

**Storage Parameters**

With *Techn. setting* you can display which storage parameters would be computed from the current values of the technical settings.

You can also explicitly define which storage parameters should be used next time the table or an index of the table is created. For example, this can be necessary if you changed the technical settings of the table and want to make sure that these changed settings are used when the table is converted [Page 221].

Proceed as follows:

1. Choose *For new creation*.
2. You can create a template for the parameters in the next screen with . You can decide here if you want to use the storage parameters currently valid in the database, the storage parameters computed from the technical settings, the default values of the database system or the parameters of another table as template.
3. Have the system create the template. You can now maintain all the parameters, so that you can enter the required values.
4. Then save your entries.

If you create the table again in the database (e.g. during a conversion), the parameters you entered will be used.

**See also:**

Technical Settings [Page 30]

# Displaying Requests for Mass Processing

## Procedure for Displaying the Requests

Choose *DB Requests* → *Mass processing* in the initial screen of the database utility to obtain a list of all entries for mass processing that you yourself scheduled and which have not yet been processed. Only requests for which no job has yet been scheduled for processing and those which are still running or have terminated are displayed here.

You can obtain the list of all scheduled and not yet processed requests in the system by choosing *All requests*. You can return to the display of the requests you scheduled with *Own requests*.

You can obtain a list of all entries for mass processing which arose from an import of objects from another system and which have not yet been processed successfully with *DB Requests* → *Created with import.*

## Information in the Lists

These lists contain the following information:

- *Object:* Object type.
- *Object name:* Name of the ABAP Dictionary object.
- *ID:* If a matchcode ID is being processed, its name is given here. If an index is being processed, its name is given here.
- *Function:* Function to be carried out for the object in the database. Possible entries are *CRE* (create), *DEL* (delete), *DNA* (delete the object in the database and delete the runtime object), *MDF* (delete, create again) and *CNV* (convert).
- *User:* User who entered the request in TBATG.
- *Job status:* Status of the job scheduled for the conversion.
- *Start date, Start time:* Date and time when the job should be started.
- *Job name:* Name of the scheduled job. This name always has the form TBATG-<*Date job is scheduled*>.
- *Job count:* Identifier with which several jobs scheduled on the same day can be distinguished.
- *Date of creation:* Date when the request was entered in TBATG.
- *Order:* If several requests exist for an object, the sequence is given here.

## Scheduling jobs

A corresponding job must be scheduled so that the requests for mass processing can be processed. Requests for which no job has yet been entered in the field *Job name* are therefore not processed.

**Displaying Requests for Mass Processing**

You can schedule a job for the displayed request directly by selecting the required entries in the first column and choosing *Schedule selected.* A dialog box then appears in which you can specify the time when the job is to be started.

## Deleting Scheduled Requests

You can delete scheduled requests by selecting the corresponding line and choosing  *Marked*.

# Scheduling Jobs for Mass Processing

## Prerequisites

If you choose the processing type *Enter for mass processing*, the action is not executed immediately. Instead, entries with the corresponding function are generated in a system table (TBATG). The requests collected in this table are processed in the background at fixed times. The background process must be scheduled explicitly as a job.

## Procedure

1. You can schedule the job in the initial screen of the database utility (Transaction SE14). To do this choose *Extras* → *Schedule job*.

   A dialog box is displayed in which you can schedule the job. You can enter the date and time when the job starts here.

2. The job can also be started depending on a certain event, certain operating mode or when another job has ended.

   To do this, choose the relevant pushbuttons and make the necessary entries.

3. If you want to schedule the job periodically, you must set the flag *Execute job periodically*.

   In this case you also need to choose the period values. Choose *Period values*. The possible values are displayed in a dialog box. Select the value you require and choose *Save*.

4. Choose *Save* to store the start time.

   A dialog box appears in which you can enter further restrictions for the requests to be processed by the job. The job then only processes the TABTG entries that satisfy the specified restrictions. The F1 help for the input fields tells you what the individual parameters of this screen mean.

5. Choose *Schedule* → *Set in job*.

   The job is scheduled. The job processes all requests scheduled for mass processing (TBATG entries) that match the selection conditions.

**See also:**

Processing Types [Page 248]

# Displaying Logs for Mass Processing

## Procedure

Select *Extras* → *Logs* in the initial screen of the database utility to obtain the logs of all the requests for mass processing that have been processed in the background.

## Information in the List

The list contains the following information:

- *Log name:* Name of the background job.
- *Severity:* Maximum error severity (‘ ’ = no error, W = warning, E = error) for a processing step within the job.
- *Length:* Number of lines in the mass processing log.
- *User:* Name of the user who scheduled the background job.
- *Date, Time:* Date and time when the job was started.

## Actions in the List

You can perform the following operations from within this list:

- **Display a log:** Select the log name and choose  .
- **Change the log name:** You can assign the log another name with *Edit* → *Rename* or you can copy the log to a log with another name with *Edit* → *Copy*.
- **Delete a log:** Select the log to be deleted and choose  .

# Displaying Temporary Tables without Restart Logs

## Procedure

With *Extras* → *Invalid temp. table* you can display the temporary tables in your system (QCM tables) for which there is no restart log. Tables for which there is a restart log because the conversion terminated are not displayed here.

These temporary tables are created during the conversion (see Conversion Process [Page 221]). In such a conversion the data from the table is saved in a temporary table. After the table has been created in the database with its new structure, the data is reloaded to the original table. If the conversion runs without any errors, the temporary table is no longer required.

As of Release 3.0, the temporary table is deleted after the conversion has been completed successfully. Prior to Release 3.0 these tables were retained even after the conversion had been completed successfully. Such tables can therefore be deleted.

When deleting the temporary tables displayed, you should make sure that no terminated conversion was unlocked by mistake. In this case the existing restart log would be deleted. If the conversion terminated at a time when the data was present only in the temporary table, there is a danger of data loss.

If you are not sure whether this is the case, you should check whether the original table exists in the database and contains the expected volume of data before you delete the temporary table.

# Pooled and Cluster Tables

Table pools (pools) and table clusters (clusters) are special table types in the ABAP Dictionary. The data from several different tables can be stored together in a table pool or table cluster. Tables assigned to a table pool or table cluster are referred to as pooled tables or cluster tables.

A table pool or table cluster should be used exclusively for storing internal control information (screen sequences, program parameters, temporary data, continuous texts such as documentation). All data of commercial relevance is stored **exclusively** in transparent tables!

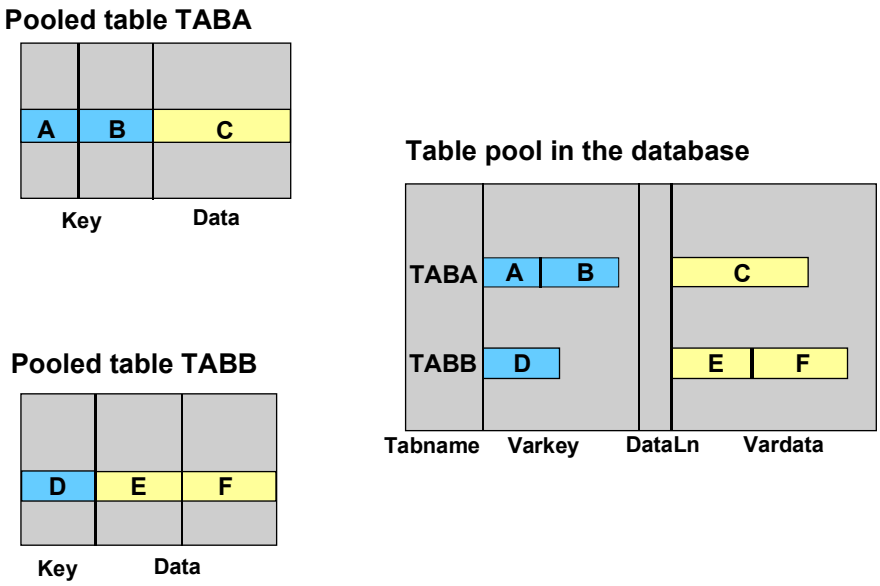
## Table Pools

A table in the database in which all records from the pooled tables assigned to the table pool are stored corresponds to a table pool.

The definition of a pool consists essentially of two key fields (*Tabname* and *Varkey*) and a long argument field (*Vardata*). A pool has the following structure:

| Field | Data type | Meaning |
|---|---|---|
| Tabname | CHAR(10) | Name of pooled table |
| Varkey | CHAR (n) | Contains the entries from all key fields of the pooled table record as a string, max. length for n is 110 |
| Dataln | INT2(5) | Length of the string in Vardata |
| Vardata | RAW (n) | Contains the entries from all data fields of the pooled table record as a string, max. length n depends on the database system used |

If a pooled table record is saved, it is stored in the table pool assigned. The name of the pooled table is written to the field *Tabname*. The contents of all key fields of the pooled table are written as a string to field *Varkey* and the contents of all data fields as a string to field *Vardata*. The length of the string stored in *Vardata* is entered in field *Dataln* by the database interface.




Due to the structure of a table pool, there are certain restrictions on the pooled tables assigned to it. The name of a pooled table may not exceed 10 characters. Since *Varkey* is a character field, all key fields of a pooled table must have character data types (for example, CHAR, NUMC, CLNT). The total length of all key fields or all data fields of a pooled table must not exceed the length of the *Varkey* or *Vardata* field of the assigned pool.

## Table Clusters

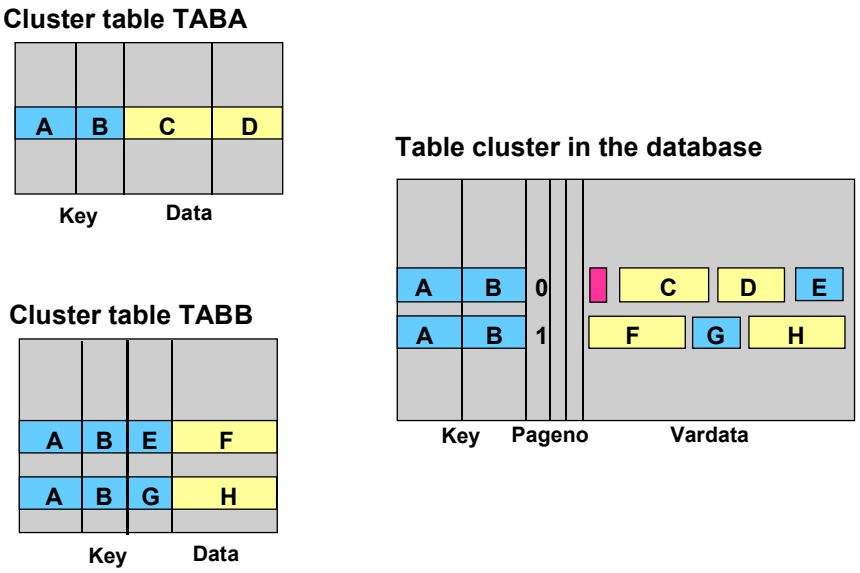
Several logical data records from different cluster tables can be stored together in one physical record in a table cluster.

A cluster key consists of a series of freely definable key fields and a field (*Pageno*) for distinguishing continuation records. A cluster also contains a long field (*Vardata*) that contains the contents of the data fields of the cluster tables for this key. If the data does not fit into the long field, continuation records are created. Control information on the structure of the data string is still written at the beginning of the *Vardata* field. A table cluster has the following structure:

| Field | Data type | Meaning |
|---|---|---|
| CLKEY1 | * | First key field |
| CLKEY2 | * | Second key field |
| ... | ... | ... |
| CLKEYn | * | nth key field |
| Pageno | INT2(5) | Number of the continuation record |
| Timestamp | CHAR(14) | Time stamps |

| | | |
|---|---|---|
| Pagelg | INT2(5) | Length of the string in Vardata |
| Vardata | RAW (n) | Contains the entries from the data fields of the assigned cluster tables as a string, max. length n depends on the database system used |

The records of all cluster tables with the same key are stored under one key in the assigned table cluster. The values of the key fields are stored in the corresponding key fields of the table cluster. The values of all data fields of the assigned cluster tables are written as a string to the *Vardata* field of the table cluster. Besides the actual data values, the data string contains information on the structure of the data and which table it comes from. If the string exceeds the maximum length of the *Vardata* field, a continuation record is written with the same key values. The continuation records for a key are distinguished by their value in field *Pageno*. The actual length of the string in the *Vardata* field is stored in the *Pagelg* field by the database interface.




You need the structural information stored in the ABAP Dictionary to read the data from a pooled table or cluster table correctly. These tables can therefore only be processed using Open SQL with the cluster interface, and not with Native SQL directly in the database.

**See also:**

Creating Table Pools/Table Clusters [Page 259]

Deleting Table Pools/Table Clusters [Page 260]

Creating Pooled Tables/Cluster Tables [Page 261]

Changing Pooled Tables/Cluster Tables [Page 262]

# Creating Table Pools/Table Clusters

## Procedure

1. In the initial screen of the ABAP Dictionary, choose *Utilities* → Further Dictionary Objects.

   A dialog box appears.

2. Select the object type *Table pool/cluster* and enter the object name. Choose  .

   A dialog box appears in which you must specify if it is a table pool or a table cluster.

   Select the required object type and choose  .

3. The maintenance screen for table pools/clusters appears.

   The necessary entries will have been made automatically for the fields for table pools since a table pool has a fixed structure. You should not change these standard settings if you can avoid it.

   The structure of a table cluster is also mostly fixed. Certain fields are therefore proposed when the table cluster is created. You can adjust this proposal to your requirements, for example by inserting further key fields. However, make sure you conform to the structure necessary for a table cluster.

4. Enter an explanatory text in the field *Short text*.

   If necessary, select the activation type [Page 80] of the table pool/cluster with *Utilities* → *Activation type*.

5. Create documentation about the table pool/cluster with *Goto* → *Documentation*.

   This documentation should describe what the table pool/cluster is used for. The documentation is also output when the table pool/cluster is printed.

6. Go to the maintenance screen for the technical settings by choosing *Goto* → *Technical settings.*

   In contrast to the table maintenance screen, you can only define the size category [Page 32] here. All other attributes of the technical settings are preset.

7. Activate the table pool/cluster with  .

## Result

The table pool/cluster is activated. You can look at the log of the activation with *Utilities* → *Activation log*. If errors occurred during activation, the activation log is automatically displayed.

After the table pool/cluster has been activated, you need to create it in the database. To do this, use the database utility (*Utilities* → *Database utility*).

Once a table pool contains data, it can no longer be changed.

# Deleting Table Pools/Table Clusters

## Prerequisites

Please note that you cannot delete a table pool or table cluster if it still contains pooled tables or cluster tables.

## Procedure:

1. In the initial screen of the ABAP Dictionary, choose *Utilities* → *Further Dictionary Objects*.

   A dialog box appears.

2. Select object type *Table pool/cluster*.

   Enter the name of the table pool or table cluster.

3. Choose  .

   You are informed if the table pool or table cluster still contains data. If it does not contain any data, a dialog box appears in which you are requested to confirm the deletion request.

   If pooled or cluster tables are still assigned to the table pool or table cluster, you cannot delete it.

4. Confirm the deletion request.

## Result

The pool or cluster is deleted in the ABAP Dictionary and in the database.

# Creating Pooled Tables/Cluster Tables

## Procedure

1. In the initial screen of the ABAP Dictionary select object type *Table*, enter a table name and choose *Create*.

   The field maintenance screen for the table is displayed. Table type *Transparent table* is set as default.

2. Make the necessary entries in the *Short description* and *Delivery class* fields on the *Attributes* tab page. Then define the fields of the table.

   Proceed as when creating a transparent table [Page 71]. Save your entries.

3. Choose *Extras* → *Change table category*.

   A dialog box appears in which you have to select the table type *Pooled table* or *Cluster table*.

4. Choose *Select*.

   You return to the field maintenance screen for the table. Field *Pool/cluster name* is displayed on the *Attributes* tab page in addition to the standard fields.

5. Enter the name of the table pool or table cluster [Page 256] to which you want to assign the pooled table or cluster table in field *Pool/cluster name*

   Note that the total key length of a pooled table may not exceed the key length of the associated table pool. The key of a cluster table must correspond to the key of the associated table cluster.

6. Proceed as when creating a transparent table (see Creating Tables [Page 71]). Remember that you cannot create indexes for pooled or cluster tables.

   All the attributes of the technical settings [Page 30] can be maintained for pooled tables and cluster tables. Before you can access these attributes, however, you must convert the table to a transparent table.

# Changing Pooled Tables/Cluster Tables

## Procedure

To change a pooled or cluster table, proceed as described for transparent tables (see Changing Tables [Page 81]).

## Effect of Changes

However, there are certain differences between pooled/cluster tables and transparent tables with regard to the effect of such changes.

If new fields are appended to a pooled or cluster table, it is not necessary to convert the table. The new fields are defined automatically as NOT NULL (see Initial Values [Page 86]). Therefore, when the modified table is activated, the complete table is scanned and the new field is filled with the initial value. This can take a considerable time in the case of pooled or cluster tables containing a large number of records.

Unlike for transparent tables, inserting new fields in a pooled or cluster table and making changes to the field sequence always causes the table to be converted [Page 221].

If a pooled or cluster table is assigned to another physical table pool or table cluster, this results in the table being converted.

Pooled and cluster tables with more than 250 fields cannot be converted. You may not make changes to such tables that would require a table conversion (for example, inserting fields, changing the field sequence, deleting fields). New fields must always be appended at the end of the table.

# Matchcodes

Matchcodes were replaced with search helps [Page 166] starting with Release 4.0. Use search helps to assign an input help to a field.

Existing matchcodes were automatically converted to search helps. A matchcode object is hereby converted to a collective search help [Page 172] with the same name. Each matchcode ID of the matchcode object is converted to an elementary search help [Page 168] with the same name and assigned to the collective search help created from the matchcode object.

A matchcode is a means of finding data records stored in the system. The matchcode is defined in the ABAP Dictionary in two steps:

- You first define the relevant tables and fields for the search in a matchcode object. A matchcode object describes the set of all possible search paths for a search string.
- You then create one or more matchcode IDs for a matchcode object. A matchcode ID describes a special search path for a search string. The matchcode ID defines the fields or field combinations to be used in the search.

A material number must be entered in a screen field. Since the user cannot be expected to know this number, it must be possible to search for this number using the attributes of the corresponding material.

Several search paths are possible for this search. For example, you can search for the material number with the material name, the material class or the material manufacturer.

The corresponding matchcode object then comprises the fields for the material number, material name, material class and manufacturer. One matchcode ID corresponds to each search path. For example, ID A could describe the search for the material number by manufacturer. This ID only contains the fields for the material number and manufacturer.

## Matchcode Objects

The tables relevant for the search are included in a matchcode object. The table selection is based on one primary table. Further secondary tables can also be included, which are linked with the primary table by foreign keys. The fields of the matchcode object can then be selected from the base tables.

A matchcode object is not stored physically. It only describes a complete logical view on one or more tables.

## Matchcode IDs

Several matchcode IDs can be created for one matchcode object. The matchcode IDs are derived from the matchcode object by projection (field selection) and selection (definition of a selection condition).

## Matchcodes

A matchcode ID must be identified within a matchcode object with one letter or digit. This means that a maximum of 36 matchcode IDs (26 letters and 10 digits) can be defined for each matchcode object.

**Selection conditions** can be defined for each matchcode ID. These are used as a filter for the matchcode to be built. These selection conditions can define restrictions for all the fields from the base tables of the relevant matchcode ID.

You must also specify an update type [Page 265]. The update type defines how the data should be accessed and how the matchcode adjusts to data changes in the tables contained in the matchcode.

**See also:**

Creating Matchcodes [Page 268]

Changing Matchcodes [Page 285]

# Update Types

The update type of a matchcode ID defines how the matchcode is built and how it is adjusted to changes in the data records of the base tables. IDs with different update types can be defined for a matchcode object.

## Building Matchcodes

There are two ways to build a matchcode:

### Logically

The matchcode data is built temporarily when the matchcode is accessed. In this case it is implemented with a database view. This is called transparent storage of matchcodes.

### Physically

The matchcode data is stored redundantly as a separate table in the system. You must define here how the data in this redundant table should be adjusted to changes in the base tables of the matchcode object.

All the physically stored matchcode IDs of a matchcode object are stored in a table pool (called *M_<Matchcode_object>*). This matchcode pool is automatically created in the ABAP Dictionary and in the database when the first physically stored ID of the matchcode object is activated. A pooled table (called *M_<Matchcode_object><Matchcode_ID>*) that is assigned to the table pool of the matchcode object is created in the ABAP Dictionary for each matchcode ID when a pooled table is activated.

## Update Types

#### Update Type A

The matchcode data is stored redundantly as a separate table in the system. The matchcode data is only built independently of data changes at certain times using the matchcode utility. Since it could be quite time-consuming to build the matchcode data, you should do so at times when the system load is low, for example during the night.

When searching for a matchcode, it might not always be possible to access the most current data. This inaccuracy is usually acceptable when using the online search help.

#### Update Type S

The matchcode data is stored redundantly as a separate table in the system. The matchcode data is adjusted to each modification operation performed on the base tables of the ID with ABAP Open SQL (INSERT, DELETE or UPDATE).

The matchcode data affected by the table change is therefore updated automatically without this being explicitly requested by the application. Automatic updating, however, results in a loss of performance.

#### Update Type P

The matchcode data is stored redundantly as a separate table in the system. The matchcode data is updated by the application program itself, so that this application program and other programs always have access to current matchcode data.

**Update Types**

To do this, a function module MC_UPDATE_<matchcode_object_name> is generated when the matchcode ID is activated. The matchcode data is updated when the application program calls this function module.

### Update Type I

The matchcode data is implemented logically as a database view. Only transparent tables may be used in transparent matchcodes. When a transparent matchcode ID is activated, a view definition is automatically created in the ABAP Dictionary. This view definition contains the fields defined in the matchcode ID (in exactly the same order) and any defined matchcode selection condition as view selection condition. A corresponding database index should be created in user-defined matchcode IDs to support view access.

The transparent matchcodes are much more efficient than for example synchronously maintained matchcodes, especially for modification operations, since there are no extra database accesses, thus reducing the costs for communications with a remote database server in a client/server environment. The matchcode selection itself can be compared with the selection behavior of the physically stored matchcodes if there are suitable indexes in the database.

### Update Type K

Classification matchcodes are user-specific search paths that can be accessed with the F4 help like all other matchcodes. However no matchcode data can be recorded for them (logical structure). The data is obtained from a function module. This module must be specified in the matchcode ID definition and is called in the matchcode selection.

# Special Features of Program-Driven Matchcodes

Program-driven matchcodes (update type P) are stored physically, that is the matchcode data is stored redundantly as a separate table in the system. The matchcode data is updated by the application program itself, so that this application program and other programs always have access to current matchcode data.

To do this, a function module MC_UPDATE_<matchcode_object_name> is generated when the matchcode ID is activated. The matchcode data is updated when the application program calls this function module.

Converting a program-driven matchcode to transparent storage therefore requires that you change the application programs using this function module. You only have to turn the function module call into a comment line.

Adding tables to a matchcode object that contains a program-driven matchcode ID can lead to problems. Only one function module exists for all program-driven IDs of the object. This function module takes its information directly from the matchcode object. Consequently, if you change the base tables of the object, you also have to change the interface to this function module, even if the change does not directly affect the program-driven ID.

# Creating Matchcodes

## Procedure

1. You can display the matchcode maintenance screen from the initial screen of the ABAP Dictionary. Choose *Utilities* → *Further Dictionary objects*.

   A dialog box appears.

2. Select the object type *Matchcode object* and enter a four-place name. Choose *Create.*

   The maintenance screen for matchcode objects appears. All other steps are described based on this maintenance screen.

To create a matchcode, you must first create a matchcode object and then create one or more matchcode Ids for this matchcode object.

## Procedure for Creating Matchcode Objects

1. Define the attributes of matchcode objects [Page 269]
2. Select the secondary tables of matchcode objects [Page 270]
3. Select the fields of matchcode objects [Page 271]
4. Activate the matchcode objects [Page 272]

## Procedure for Creating Matchcode IDs

1. Define the attributes of matchcode IDs [Page 273]
2. Select the secondary tables of matchcode IDs [Page 275]
3. Select the fields of matchcode IDs [Page 276]
4. Define the selection conditions of matchcode IDs [Page 278] (optional)
5. Activate the matchcode IDs [Page 279]

Depending on the selected update type of the matchcode IDs of a matchcode object, you have to define further structures or carry out certain actions:

- Update types A, S or P: build matchcode data [Page 280]
- Update Type I: create matchcode indexes [Page 282]
- Update type K: Create a function module for a matchcode ID [Page 284]

**See also:**

Update Types [Page 265]

# Defining Attributes of Matchcode Objects

## Procedure

1. Go to the maintenance screen for matchcode objects.
2. Enter an explanatory short text in the field *Short text*.
3. Choose the primary table of the matchcode object. Enter the name of this table in the field *Primary table*.
4. Save your entries.
5. A dialog box appears in which you have to assign a development class to the matchcode object.

## Additional Information

The following additional information is displayed in the screen *Maintain Matchcode Object (Attributes)*:

### Secondary tables

Tables linked to the primary table by foreign keys.

### Matchcode pools

The name automatically defined for the table pool (see Pooled and Cluster Tables [Page 256]) belonging to the matchcode object is displayed after you activate the first physically stored ID (see Update Types [Page 265]) of the matchcode object.

The table pool is automatically defined in the ABAP Dictionary and in the database when the first physically stored ID of the matchcode object is activated. The name of the table pool contains the prefix *M_* and the name of the matchcode object. The name of the table pool for the matchcode object TEST would be for example M_TEST.

### Next Action when Creating a Matchcode:

Select the secondary tables of matchcode objects [Page 270]

# Selecting Secondary Tables of Matchcode Objects

## Procedure

1. In the maintenance screen for the attributes of a matchcode object, choose *Goto* → *Tables*.

   The maintenance screen for the table selection of the matchcode object is displayed. All the tables already included in the object are listed in the upper part of the screen.

   If secondary tables were already defined, the foreign keys [Page 19] linking the tables are listed in tabular form in the lower part of the screen. You will find information there on the referenced tables, dependent tables and check field used.

   You can display the definition for the foreign key by positioning the cursor on the relevant line and choosing *Edit* → *Foreign key.*

2. If you want to include additional secondary tables, position the cursor on a table name (primary table or secondary table already included) and choose *Edit* → *Choose sec. tab*.

   A list of all the tables linked to the selected table by foreign keys is displayed in a dialog box.

3. Position the cursor on a table that you want to include as a secondary table and choose *Choose*. The table is highlighted. Repeat this for all the tables you want to include in the matchcode object. Choose *Copy*.

   You return to the maintenance screen. The selected tables are included as secondary tables and the appropriate foreign key is displayed.

   You can delete incorrect secondary tables by positioning the cursor on the table and choosing *Edit -> Delete SecTab*.

4. Save the table selection when you have included all the tables you require.

**Next Action when Creating a Matchcode:**

Select the fields of a matchcode object [Page 271]

# Selecting Fields of Matchcode Objects

## Procedure

1. In the maintenance screen for the attributes of the matchcode object or the maintenance screen for the table selection choose *Goto* → *Fields*.

   The maintenance screen for the field selection for matchcode objects is displayed. The key fields of all the tables that are relevant for the matchcode object are automatically copied. These fields are indicated by an appropriate entry in the *Key* column.

2. If you want to include more fields in the matchcode object, position the cursor on the name of the table you require and choose *Edit* → *Choose fields*.

   The fields of the corresponding table are displayed in a dialog box.

3. You can include a field in the matchcode object by clicking on the corresponding entry and selecting *Choose*.

   The entry is highlighted. You can remove fields that were wrongly selected by positioning the cursor on the field and choosing *Delete selection*.

4. Press *Copy* after marking all the required fields.

   The selected fields are included in the matchcode object.

   The order of the fields in the matchcode object is irrelevant. If two fields of different tables have the same name, a new name must be entered in the column *MC field* for at least one field.

5. Link the screen field and the matchcode object field.

   In order to use matchcodes as an online help, there must be a link between a field on the screen and a matchcode field. This link is established with the *search field,* which is normally assigned to the first key field of the primary table.

   In a matchcode selection within the online help, the value of the search field is copied to the screen. If you want to select another search field, you must simply mark it in the corresponding column.

6. Save the field selection.

   All the fields in matchcodes must have character data types (CHAR, NUMC, ACCP, CLNT, CUKY, DATS, TIMS, LANG, UNIT).

**Next Action when Creating a Matchcode:**

Activate the matchcode objects [Page 272]

# Activating Matchcode Objects

Before you can create matchcode IDs, you must activate the matchcode object. In the maintenance screen for the attributes of a matchcode object choose *Matchcode object* → *Activate*.

Information on the activation process can be found in the activation log. You can display the activation log with *Utilities* → *Display log* → *Activate*. If errors occurred during activation, the activation log is automatically displayed.

**Next Action when Creating a Matchcode:**

Create the matchcode IDs [Ext.]

# Defining Attributes of Matchcode IDs

## Procedure

1. *Goto* → *Matchcode IDs*.

   A list of all existing IDs for the object is displayed.

2. Choose *Create* to define a new ID.

   A dialog box appears in which you can enter any alphanumeric character as identifier for the matchcode ID. The digits 0 to 9 are reserved for customers.

3. Choose *Continue*.

   The maintenance screen for the attributes of a matchcode ID appears.

4. Enter an explanatory short text in the field *Short text*.

5. Choose an update type [Page 265] for the Matchcode ID.

   Update type I (transparent ID) is entered as default value. If you want to select another update type, simply overwrite this entry. However, in general you should create all IDs as transparent IDs.

6. Save your entries if you do not want to make any of the optional settings described below.

### Other Options

#### System Matchcode

The SAP software uses this matchcode if this indicator is set. The customer may not change it.

#### Auth. check

If this indicator is set, authorization checks are made, which may be linked to individual matchcode records. In the matchcode display, a user exit in which an application-specific authorization check can be programmed is called before a matchcode record which has been read is output.

The user exit is called *CHECK_AUTHORITY_<mcid>* and is located in the include *MC_<obj>&,* where *<obj>* is the name of the matchcode object.

## Additional Information Depending on the Update Type

#### Update Types A, S and P (Physical Matchcodes):

- *MC pooled table*: If the matchcode ID was already activated, the corresponding pooled table is listed here. The name of the pooled table contains the prefix *M_*, the name of the matchcode object and the matchcode ID. The corresponding name is therefore M_TEST1 for ID 1 of matchcode object TEST.
- *Deletion flag:* Set this flag if the relevant matchcode records are to be assigned a deletion flag but are not actually to be physically deleted when a record of a base table is deleted. All the records flagged in this way are physically deleted at a later time during a mass deletion process. Records assigned a deletion flag still appear in the matchcode display until the end of this deletion process.

**Defining Attributes of Matchcode IDs**

The deletion flag cannot be set for matchcode IDs of update type P.

**Update Type I:**

- *MC view name*: If the matchcode ID has already been activated, the corresponding view name is listed here. The view name contains the prefix *M_*, the name of the matchcode object and the matchcode ID. The corresponding name is therefore M_TEST1 for ID 1 of matchcode object TEST.
- *Table name, Index ID*: You can define the table and index ID here for the index you created to support view selection. These fields have a purely documentary function at present.

**Update Type K:**

- Function: For matchcode IDs of update type K the search is via a function module that is triggered when the ID is selected. This entry shows the name of the function module used if such a module was already created.

**Next Action when Creating a Matchcode:**

Select the secondary tables of matchcode IDs [Page 275]

# Selecting Secondary Tables of Matchcode IDs

## Procedure

1. In the maintenance screen of the matchcode ID, position the cursor on one of the base tables of the ID and select *Edit* → *Choose sec. tables*.

   A dialog box appears listing the tables of the matchcode object linked to the table by foreign keys.

2. Select the secondary tables required in the dialog box. You can remove tables that were wrongly selected by selecting them again.

   Only transparent tables may be selected for transparent matchcodes (update type I).

   For performance reasons, some restrictions apply to table selection in matchcodes of update type S (synchronous updating).

   Only key extensions are permitted for synchronous matchcode IDs in dependent tables. The selected tables must, therefore, be related hierarchically.

   Table T2 is dependent on table T1, while a further table T3 is dependent on T2. The tables are said to have a hierarchical relationship if the key of T2 contains the key of T1 in its first position and the key of T3 contains the key of T2 in its first position. Only with this type of relationship can T2 and T3 be included as secondary tables for primary table T1 for a matchcode to be updated synchronously.

   If two tables T2 and T4 are dependent on a table T1, only one of the tables may be included in a synchronous matchcode ID in addition to table T1. An exception to this rule are text tables. If T2 or T4 is a text table [Page 27] of T1, then both can be included in the matchcode ID.

3. Choose *Continue*.

   You return to the maintenance screen. The selected secondary tables are included in the ID.

4. Save the selection of the secondary tables.

**Next Action when Creating a Matchcode:**

Select fields of a matchcode ID [Page 276]

# Selecting Fields of Matchcode IDs

## Procedure

1. In the maintenance screen for the attributes of the matchcode ID, choose *Goto* → *Fields*.

   The maintenance screen for the matchcode ID appears.

2. Position the cursor on a table name and choose *Edit* → *Choose fields*.

   The fields of the corresponding table are displayed in a dialog box.

3. You can include a field in the matchcode ID by clicking on the corresponding entry and choosing *Choose*. The entry is highlighted. You can remove a wrongly selected field by positioning the cursor on the field in question and choosing *Delete field*.

   Press *Copy* after selecting all the required fields. The selected fields are copied to the matchcode ID.

   The order of the selection exactly corresponds to the order in which the fields are listed in the matchcode ID. The order of the fields in the matchcode ID is of paramount importance for later accessing behavior, for example, in connection with the possible entries help *F4*. The fields that are most frequently accessed should be placed at the start.

4. Save your field selections if you do not wish to make any of the optional settings described below.

## Constraints

For reasons of consistency, matchcode IDs may only contain fields of tables that are linked with a foreign key. For example it is not possible to include fields from two tables that are only linked with a third table that is not contained in the matchcode ID. In such a case this linking table must be included in the matchcode ID with a key field.

The data of a matchcode ID of update types A, S and P are stored in pooled tables. The system tries to store all the fields of the matchcode ID in the key of the corresponding pooled table. The key of a pooled table, however, may not be of arbitrary length. It is therefore possible that some of the fields of the matchcode ID cannot be stored in the key of the pooled table. In order to ensure that the records of a matchcode ID are unique, however, at least all the key fields of the matchcode ID (that is all the fields necessary to uniquely identify a matchcode record) must lie within the first 110 bytes.

The fields of the ID must uniquely identify each record in matchcode IDs of update type S. However, not all of the key fields of the tables used in the ID need be included. A further constraint is that the ID must contain at least one key field of a table included in the ID, even if the remaining fields already uniquely identify each record.

## Other Options

### Storage

Here you can define how the matchcode records should be saved.

- *Lngth, Offs:* You can define a field as a sub-field by specifying its offset and length. You have to define a suitable data element (the type and length of the referenced domain must agree

with that of the sub-field) in the column *Data elem*. Key fields may not be sub-fields. No sub-fields may be defined for transparent matchcode IDs (update type I).

- *G/K:* This flag defines whether to distinguish uppercase and lowercase notation when building the matchcode records. If nothing is specified here, the matchcode records are built and output in uppercase for matchcode IDs of update type [Page 265] A, S or P. The search is not case-sensitive. This field is irrelevant for matchcode IDs of update type I because no separate matchcode records are written (search is via a view).

### Display

You can influence how the selected matchcode records are output. The following settings are possible:

- *Ln:* Line in which a particular field is to appear in the matchcode output. A maximum of three lines can be output.
- *Cl:* Column in which a particular field is to appear in the matchcode output. Column + field length for the matchcode output may not exceed the assumed page width of 80 characters.
- *Mod:* This flag is selected automatically if you change the entries in columns *Ln* and *Cl*.
- *Ns:* If this flag is set, the corresponding field is not used for matchcode selection. It does not appear in the field input list for the possible entries help *F4* and is not evaluated in the matchcode search string.

### Parameters

Here you define whether values from matchcode records that were selected using the online help function should be stored in global SAP memory or whether values for the matchcode selection should be obtained from memory.

- *PID:* Parameter ID for defining default values for a field in the user master record.
- *SP:* Select the flag *SP* if the field value read from the selected matchcode record should be stored in memory.
- *GP:* Select the flag *GP* if the field value for matchcode analysis should be obtained from memory. The advantage of this method is that the user no longer has to enter a field value obtained from memory on the screen by hand.

### Changing Data Elements

The system assigns the matchcode field the data element of the corresponding table field. You can change this data element. This may be necessary, for example, if you want to allocate other documentation to the matchcode field.

Simply overwrite the name of the data element in the field *Data elem*. The new data element must refer to a domain with the same data type and the same length as the original data element.

### Next Action when Creating a Matchcode (Optional):

Define the selection conditions of a matchcode ID [Page 278]

# Defining Selection Conditions of Matchcode IDs

## Prerequisites

Once you have defined the structure of a matchcode ID and specified the fields it is to contain, you can restrict the number of data records found by a matchcode search by specifying a selection condition. When the matchcode records are built, only those entries satisfying this condition are included.

## Procedure

1. In the field maintenance screen for the matchcode ID, choose *Goto* → *Selection condition*.

   The maintenance screen for the selection conditions of the matchcode ID appears.

2. You can enter a selection condition of the form *NOT MC Field Lngth Offs Op Constant AND/OR* in each line.

3. A list of all the possible fields is displayed by choosing *Edit* → *Choose fields.*

   You can choose fields from this list and copy them for use in the definition of the field conditions by positioning the cursor on the field name and selecting *Choose*.

4. Save your entries for the selection condition(s).

The entries in the selection conditions are as follows:

- *NOT:* Entering NOT negates the corresponding line of the selection condition.
- *MC field:* Name of the matchcode field for which the condition is formulated.
- *Lngth, Offs:* If the field is defined as a sub-field with entries having been made for offset and length, the corresponding values are entered here.
- *Op.:* Comparison operator for the selection condition. The following operators are permitted: LK (like), GT (greater than), GE (greater than / equal to), NE (not equal to), EQ (equal to), LT (less than) and LE (less than / equal to).
- *Constant:* Constant value with which the field value is compared. Text literals enclosed in apostrophes and numbers are allowed as comparison values.
- *AND/OR:* Link between two lines of the selection condition. Note that OR takes priority over AND. The condition <COND1> AND <COND2> OR <COND3> is therefore interpreted as <COND1> AND (<COND2> OR <COND3>). OR operations are only possible between lines referring to the same field.

  The field conditions need not refer only to the fields belonging to the matchcode ID. They can be defined for **any** fields of the matchcode object. You can therefore include any fields of the matchcode object in the selection condition without using them in the ID as fields.

**Next Action when Creating a Matchcode:**

Activate the matchcode IDs [Page 279]

# Activating Matchcode IDs

You can activate the matchcode ID from the maintenance screen for the attributes of the matchcode ID with *Matchcode ID* → *Activate*.

The corresponding database view is created on the database during activation for matchcode IDs with update type I. During activation, there is also a check whether there is a corresponding index on the database for supporting view selection. If this is not the case, a warning is given.

Information on the activation process can be found in the activation log. You can display the activation log with *Utilities* → *Display log* → *Activate*. If errors or warnings occurring during activation, the activation log is displayed.

### Next Action when Creating a Matchcode:

For IDs of update types A, S, P: build matchcode data [Page 280]

For IDs of update type I: create a matchcode index [Page 282]

For IDs of update type K: create a function module for a matchcode ID [Page 284]

# Building Matchcode Data

## Prerequisites

In physical matchcodes (update types A, S or P), the matchcode data is stored redundantly in separate tables in the system. You have to build these tables after you have created and activated the matchcode object and the matchcode IDs. The tables are built and deleted using the matchcode utility (report SAPMACO). The procedure used to build the tables is the same for all physical update types (A, S or P).

The matchcode utility operates globally across all clients. A parameter controls whether matchcodes are to be built or deleted for a specific client or for all clients. The utility also permits matchcodes to be built for all matchcode objects at one time.

## Procedure

1. Choose *Utilities* → *Matchcode data* → *Build* in the maintenance screen of the matchcode object. You can also call report *SAPMACO* directly.
2. Follow the instructions in the F1 help. It describes how to fill in the input template. Execute the report.
3. Check whether the matchcode data was built correctly. You can see this in Displaying the Built Matchcode Data [Page 281].

Transparent matchcode IDs (update type I) and classification matchcode IDs (update type K) are not stored physically. Therefore, there is no reason to build such IDs with the matchcode utility.

# Displaying the Built Matchcode Data

## Procedure

1. Go to the maintenance screen for the attributes of the matchcode object.
2. Choose *Utilities* → *Matchcode data* → *Display.*

   A dialog box appears in which all the IDs of the matchcode object are listed.
3. Select one of the IDs by clicking on the corresponding line and selecting *Choose*.

   The dialog box *Restrict value ranges* appears for you to limit the value range of the records to be output by entering selection conditions for the fields of the ID.
4. Choose *Continue*.

   All the records which satisfy the specified condition are output.

   You can also display the matchcode data of an ID directly from the maintenance screen of the ID. Select *Utilities* → *Display MC data*. The dialog box *Restrict value ranges* appears. Proceed as described above.

# Creating Matchcode Indexes

When you activate a transparent matchcode ID, a check is made to see whether a suitable database index exists for this ID. Such an index is generally necessary to support matchcode selection. If no such index exists, considerable performance problems could result during matchcode selection.

The system assumes that the first field of the matchcode definition (the first field after the client field in cross-client matchcodes) is the relevant search field for this matchcode, that is, that the user limits this field during the matchcode search by entering a selection value. An index is considered to be suitable if it contains the relevant matchcode search field (possibly after a client field).

If there is no such suitable database index, a warning is given when the matchcode ID is activated. Two cases are distinguished here:

- If the matchcode view on the database (taking into consideration the selection condition) contains fewer than 1000 data records, you need not create an index.
- If the matchcode view on the database contains considerably more than 1000 records, you should create an index.

The first position of the index for supporting the matchcode selection should contain the fields to be searched for with equality (client, language, or, more generally, all fields for which the Get Parameter flag, that is, the GP flag in the screen *Maintain Matchcode ID (Fields),* is set).

The index should have the following structure:

- client field
- fields for which the Get Parameter flag is set
- field to be found

This index structure, however, does not always ensure that the index is used by the underlying database system for data selection. The database system optimizer determines which index is actually used. Therefore, you must ensure that the secondary index you create is better than the primary index of the corresponding table created by the system.

You can check whether the index you created is used to support the matchcode selection as follows:

1. In the maintenance screen for the attributes of the matchcode ID, choose *Utilities* → *Explain plan*.

   A dialog box appears in which you can enter a search string for this ID.

2. Choose *Continue*. The search string is committed.

   The explain plan for the search string is now displayed. The explain plan shows the structure of the SELECT statement used for the search string and how the SELECT statement is processed in the database. In particular, it displays the indexes used for reading the data from the base tables of the matchcode ID.

3. Analyze the explain plan and, if necessary, create indexes for the tables that did not have a suitable index for the access.

The structure of the explain plan depends on the database system used. Further information about the explain plan can be found in the documentation on your database system.

# Function Modules for Matchcode IDs

You can search for matchcodes of update type K with the F4 help using a function module that has to be specified in the matchcode ID definition and that is called in the matchcode selection. This function module must have the following interface:

| | |
|---|---|
| CALL FUNCTION fname | |
| EXPORTING MCONAME = mcname | "Name of the matchcode object" |
| SELSTR = 'M.....' | "Matchcode entry" |

The VALUES table has the Dictionary structure RSMVA with the fields:

| | |
|---|---|
| FLDID(3) TYPE C = | field ID of the field to be transported to the screen |
| FLDLG(6) TYPE N = | length of the field to be transported |
| VALUE(79) TYPE C = | value to be defined |

In the current version, only one line of the VALUES table is analyzed after returning from the function module. Only the search field (at which the matchcode selection was triggered) specified in the definition of the matchcode object is transported to the screen. The value that is returned must already have been converted to the external format.

# Changing Matchcodes

## Procedure

1. You can display the matchcode maintenance screen from the initial screen of the ABAP Dictionary. Choose *Utilities* → *Further Dictionary objects*.

   A dialog box appears.

2. Select the object type *Matchcode object* and enter a four-place name. Choose *Change.*

Here you can:

- change the matchcode objects [Page 286]
- change the matchcode IDs [Page 288]
- convert to transparent matchcodes [Page 290]
- determine the effect of conversion on transparent matchcodes [Page 291]
- deactivate the matchcode IDs [Page 292]
- delete the matchcode objects [Page 294]
- delete the matchcode IDs [Page 293]

# Changing Matchcode Objects

This section describes what you should be aware of when changing an existing matchcode object.

## Changing the Primary Table

To change the primary table, simply enter a new table name in the field *Primary table*.

A warning is issued if you already selected secondary tables or fields. If you confirm the change to the primary table by pressing *ENTER,* all dependent objects (secondary tables and fields) are deleted.

If IDs already exist for the matchcode object, you cannot change the primary table. In this case you must first delete the existing IDs.

## Removing Secondary Tables

To remove a secondary table from the matchcode object, position the cursor on the table and choose *Edit -> Delete SecTab*.

Deleting a table from the matchcode object also removes all the tables that are linked with the primary table via the deleted table.

## Including Additional Secondary Tables

If you want to include additional secondary tables, position the cursor on the table name and choose *Edit* → *Choose sec. tab.* A list is displayed of all tables linked to the selected table with a foreign key.

Position the cursor on a table that you want to include as a secondary table and choose *Choose*. The table is highlighted. Repeat this for all the tables you want to include in the matchcode object. Choose *Copy*.

You return to the maintenance screen. The selected tables are included as secondary tables and the appropriate foreign key is displayed. Save the table selection when you have included all the tables you require.

## Deleting fields

To delete a field, position the cursor on the relevant field and choose *Edit* → *Delete field*. Key fields cannot be deleted.

Before you delete a field from a matchcode object, you must make sure that this field is not used in any of the matchcode IDs for this object. Otherwise an error occurs when you try to activate the matchcode object! You can correct this error by deleting the relevant field from all the IDs of the matchcode object or by simply including the field again in the matchcode object.

## Inserting Fields

To insert a field, position the cursor on a field that was already entered and choose *Edit* → *Insert field*. A new line is opened directly above the line containing the cursor. You can make the necessary entries in this line.

## Changing the Search Field

In order to use matchcodes as an online help, there must be a link between a field on the screen and a matchcode field. This link is established with the *search field,* which is normally assigned to the first key field of the primary table.

In a matchcode selection within the online help, the value of the search field is copied to the screen. If you want to select another search field, you must simply mark it in the corresponding column.

Only one field may be selected as search field!

# Changing Matchcode IDs

This section describes what you should be aware of when changing an existing matchcode ID.

## Changing the Update Type

You can change the update type [Page 265] by overwriting the existing entry in the field *Update type*. If you wish to convert physically stored matchcode IDs to transparent matchcode IDs, read the information in Converting to Transparent Matchcodes [Page 290].

You have to delete all dependent objects of the existing ID before changing the update type. If you change the update type from I to A, for example, you must first delete the matchcode view created in the database.

## Removing Secondary Tables

You can remove a table already selected for the matchcode ID by selecting the relevant table again.

Position the cursor on the table with which the table to be removed is linked to the primary table. Choose *Edit* → *Choose sec.tab.* A dialog box appears listing the tables linked with the table via foreign key. In the dialog box, cancel the selection of the secondary tables to be removed. Choose *Continue*. You return to the maintenance screen. The selected secondary tables are deleted from the ID. Save the selection of the secondary tables.

The fields of this table already included in the matchcode ID are **not** removed automatically. They must be deleted in the field maintenance of the matchcode ID! Removing a secondary table from the ID also removes all the tables linked to the primary table with this table from the matchcode ID.

## Including Secondary Tables

Position the cursor on a base table. Choose *Edit* → *Choose sec.tab.* A dialog box appears listing the tables linked with the table via foreign key. Select the secondary tables required in the dialog box. Choose *Continue*. You return to the maintenance screen. The selected secondary tables are included in the ID. Save the selection of the secondary tables.

Only transparent tables may be selected for transparent IDs. For performance reasons, only hierarchical relationships are allowed for the dependent tables of synchronous matchcodes (update type S).

## Deleting Fields

To delete a field, position the cursor on the relevant field and choose *Edit* → *Delete field*.

For synchronous matchcode IDs (update type S), key fields should only be deleted if the remaining fields still uniquely identify the record. Otherwise the synchronous matchcode maintenance no longer functions correctly. In this case deleted records can no longer be removed from the matchcode data and the original version is kept in addition to the revised version for modified records. In this case you must build the matchcode records again using the matchcode utility.

## Inserting Fields

To insert a field, position the cursor on a field that was already entered and choose *Edit* → *Insert field*. A new line is opened directly above the line containing the cursor. You can make the necessary entries in this line.

If you change the fields of a matchcode ID of update types A, S or P, you have to convert the matchcode data. The existing matchcode data is deleted and then rebuilt. If the fields of a matchcode ID of update type I are changed, the matchcode view on the database is automatically deleted and built again with the new structure when the ID is activated.

If you inserted fields in a matchcode ID delivered by SAP, this change is kept throughout the upgrade. After the upgrade, the matchcode ID contains all the fields of the revised version as well as all the fields of the new version, that is the join of the sets of fields of both versions of the ID is created during the upgrade.

## Changing Data Elements

The system assigns the matchcode field the data element of the corresponding table field. You can change this data element. This may be necessary, for example, if you want to allocate other documentation to the matchcode field.

Simply overwrite the name of the data element in the field *Data elem*. The new data element must refer to a domain with the same data type and the same length as the original data element.

## Changing the Selection Condition

If you change the selection condition of a matchcode ID of update types A, S or P, you must build the matchcode data again using the matchcode utility.

# Converting to Transparent Matchcodes

## Prerequisites

The following requirements must be satisfied so that a physically stored ID (update types A, S or P) can be converted to transparent storage (update type I).

- All base tables of the ID must be transparent tables.
- No sub-fields may be defined.

When converting a program-driven ID, it might be necessary to adjust application programs (see Special Features of Program-Driven Matchcodes [Page 267]).

## Procedure

1. First delete the matchcode data stored in the database.

   If you wish to convert all physically stored matchcode IDs of a matchcode object to transparent matchcode IDs, you should delete the matchcode pool from the database. To do so, go to the database utility from the matchcode object maintenance screen by selecting *Utilities* → *Database utility*. Then choose *Database table* → *Delete*. The matchcode pool is deleted from the database.

   If you wish to convert only one physically stored matchcode ID of a matchcode object to a transparent matchcode ID, you must delete any matchcode data existing for this ID from the database. To do this, call the matchcode utility with the function *DEL*. Select *Utilities* → *Matchcode data* → *Delete* in the screen *Maintain Matchcode Object (Attributes).* The *Matchcode Utility* screen appears. Choose *Program* → *Execute*. A list of all matchcode IDs for the matchcode object is displayed. Select the ID you require from this list and choose *Execute*. The matchcode data is deleted.

2. Go to the *Maintain Matchcode ID (Attributes)* screen.
3. In the field *Update type*, change the entry to I (transparent storage).
4. Activate the matchcode ID. The matchcode view is created automatically in the database. The pooled table is then deleted automatically.
5. To support matchcode selection, it may be advisable to create an index. To find out when an index is required and how to create an index, refer to Creating Matchcode Indexes [Page 282].

**See also:**

Effect of Conversion on Transparent Matchcodes [Page 291]

# Effect of Conversion on Transparent IDs

The following changes occur when a physically stored matchcode ID is converted to transparent storage:

### Search is case-sensitive

Searching via transparent IDs is case-sensitive. When entering a search condition, a distinction is made between uppercase and lowercase notation in text fields.

### Size of the hit list can be reduced

The hit list displayed as the result of searching with a transparent ID can be a genuine subset of the hit list found with an equivalent search via a physically stored ID. The reason for this is that the access with transparent IDs is implemented using an inner join, while, with physically stored IDs, an outer join is formed.

When you search with a transparent ID, records of the primary table for which there is no corresponding entry in the foreign key fields of a secondary table are not found.

An ID is used to search for an employee's personnel number on the basis of the employee's name and department. The base table contains data on the number and name. The secondary table contains the departments and their employees. There is no entry in the secondary table for employees who are not assigned to a department. These employees cannot be found using a transparent ID with the fields for the number, name, and department. However, if a physically stored ID with the same structure is used, these employees will be found.

### Adjustment of component fields

If component fields were defined for the ID, these definitions must be canceled when an ID is converted to transparent storage. You can return the output list to its previous format by changing the layout.

# Deactivating Matchcode IDs

In applications, many matchcode IDs are created automatically, but are rarely used in certain areas. These superfluous IDs could affect system response times.

The *Deactivate* function in the menu *Matchcode ID* in the maintenance screen for the attributes of the matchcode ID allows you to alleviate this problem. Unlike when a matchcode ID is deleted, the definition of the ID is kept in the ABAP Dictionary when an ID is deactivated. If the ID is needed at a later time, it can be activated again.

The *Deactivate* function is only possible for active matchcode IDs. Before a matchcode ID can be deactivated, the objects in the database belonging to the ID (such as a view or index for IDs of update type I) are deleted.

Note that the matchcode records of a deactive ID are not automatically adjusted to data changes for matchcode IDs of update type S. After reactivation, the matchcode records must be rebuilt using the matchcode utility.

Inactive matchcode IDs are not proposed for selection when you use the F4 help.

Deactivation is a pure customizing function, that is it is not an attribute that can be transported. If a deactivated matchcode ID is included as part of an upgrade, it will be active again after the upgrade.

# Deleting Matchcode IDs

## Prerequisites

You can only delete a matchcode ID once all the corresponding database objects for this ID have been deleted. You must do the following before you can delete a matchcode ID:

- **For matchcode IDs of update type I:** Delete the matchcode view on the database. In the screen *Maintain Matchcode ID (Attributes),* call the database utility with *Utilities* → *Database utility*.
- **For matchcode IDs of update types A, S and P:** Delete the matchcode data built using the matchcode utility. It is sufficient to delete the matchcode pool from the database using the database utility.

## Procedure

1. Go to the maintenance screen for the attributes of the matchcode ID.

   Choose *Matchcode ID* → *Delete.*

2. A dialog box appears for you to confirm your deletion request.

   Confirm the deletion request.

3. If no more dependent objects exist for the matchcode ID, it will deleted.

   Information on the deletion process and on any errors that may have occurred is recorded in the deletion log. You can display the deletion log with *Utilities* → *Display log* → *Delete*.

   It is often better to only deactivate a matchcode ID. Unlike when a matchcode ID is deleted, the definition of the ID is kept in the ABAP Dictionary when an ID is deactivated [Page 292]. If the ID is needed at a later time, it can be activated again.

# Deleting Matchcode Objects

## Prerequisites

When you delete a matchcode object, all the IDs for this object are also deleted. Therefore, you can only delete a matchcode object if all the IDs for the object can also be deleted, that is, no more database objects exist for these IDs. For example, no more matchcode views may exist in the database for transparent IDs of the object.

## Procedure

1. In the initial screen of the ABAP Dictionary, choose *Utilities* → *Further Dictionary Objects*.

   A dialog box appears.

2. Select the object type *Matchcode object* and enter the name of the matchcode object.

   Check whether database objects still exist for the matchcode IDs of the object before deleting it.

3. Choose *Delete*.

   The matchcode object and the relevant matchcode IDs are deleted if no more dependent objects exist for the matchcode IDs belonging to the matchcode object.

# Flight Model

The flight model is the basis of all the examples in this documentation. All the tables mentioned in the examples exist in your system, so you can reproduce the examples directly in the system.

The flight model is based on data model BC_TRAVEL, which you can look at in your system using the *Data Modeler*.

## Procedure for Displaying the Flight Model

1. Go to the initial screen of the ABAP Workbench and choose *Development* → *Data Modeler*.

   The initial screen of the Data Modeler is displayed.

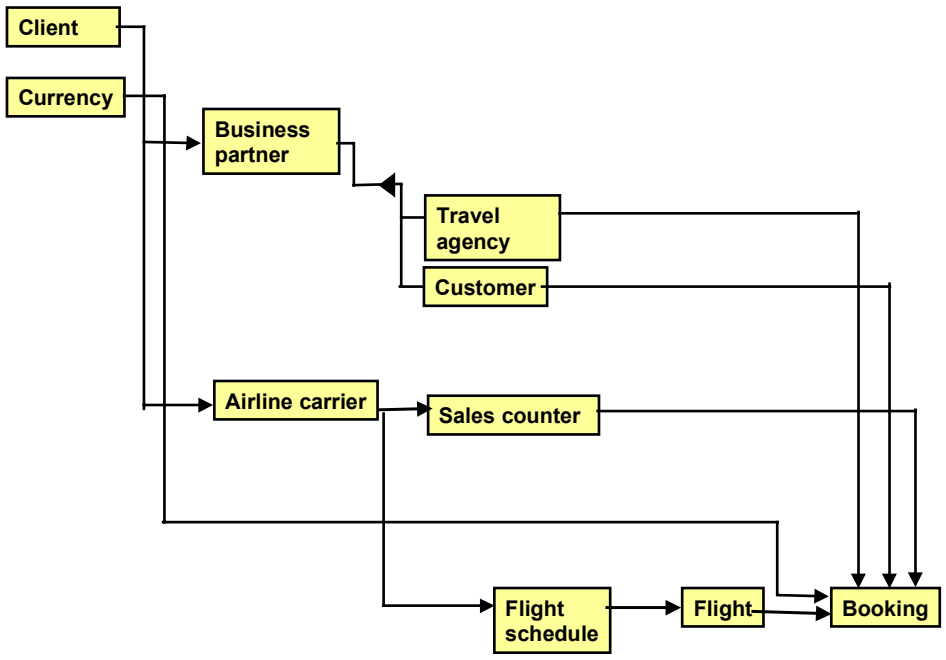
2. Enter BC_TRAVEL in field *Modeling object* and choose *Display*.

   The description of the flight model is displayed.

3. You can display the data model as a graphic with *Utilities* → *Graphic*.

You can find further information on data modeling and on using the Data Modeler in the documentation on the Data Modeler [Ext.].

## Flight Model (Simplified Representation)




The flight model gives a simple description of seat bookings in passenger airplanes by flight customers. The booking can be made either at the sales desk of a carrier or at a travel agency.

A flight booking is based on the individual flight connections described in the flight schedule. There are concrete flights for every flight connection.

## Assigned Tables in the ABAP Dictionary

There is a table in the ABAP Dictionary containing the data of the corresponding entities (versions of the entity type) for each entity type.

The most important tables of the flight model are:

- **T000**: Client table
- **SCURX**: Currencies (key: currency key)
- **SBUSPART**: Business partner (key: client, partner number)
- **STRAVELAG**: Travel agencies (key: client, travel agency number)
- **SCUSTOM**: Customers (key: client, customer number)
- **SCARR**: Carriers (key: client, carrier ID)
- **SCOUNTER**: Sales counters (key: client, carrier ID, sales counter number)
- **SPFLI**: Flight schedule (key: client, carrier ID, connection number)
- **SFLIGHT**: Flights (key: client, carrier ID, connection number, date of flight)
- **SBOOK**: Flight bookings (key: client, carrier ID, connection number, date of flight, booking number, customer number)

## Relationships between the Tables

Table SBUSPART contains all the business partners of a carrier. A business partner is identified by his number in this table. The data of the contact person for the business partner is also stored. A business partner can be a travel agency or a customer (e.g. company that frequently books flights directly with the carrier). Different data is required for these two types of business partner. The data for a travel agency is stored in table STRAVELAG and the data for a customer in table SCUSTOM. There is therefore an entry with the same key in either table STRAVELAG or table SCUSTOM for each entry in table SBUSPART.

Table SCARR contains the IDs and names of the carriers. Each carrier has a number of connections. These flight connections are stored in table SPFLI. Table SFLIGHT contains the concrete flight data for each connection Bookings can be made for each flight in table SFLIGHT. The bookings made for each flight are entered in table SBOOK.

The carriers have sales counters in the airports. These sales counters are entered in table SCOUNTER. The customer number or agency number for which the booking was made is stored in table SBOOK for each booking. If the customer books his flight directly at a counter, the counter number is also entered in the booking data in table SBOOK