English

Fujitsu Software BS2000

UDS/SQL V2.9
Creation and Restructuring

User Guide

Valid for:
UDS/SQL V2.9C

December 2025

Comments… Suggestions… Corrections…

The User Documentation Department would like to know your opinion on this manual. Your feedback helps us to optimize our documentation to suit your individual needs.

Feel free to send us your comments by e-mail to: bs2000.info@fujitsu.com.

Certified documentation according to DIN EN ISO 9001:2015

To ensure a consistently high quality standard and user-friendliness, this documentation was created to meet the regulations of a quality management system which complies with the requirements of the standard DIN EN ISO 9001:2015.

Copyright and Trademarks

Copyright © 2025 Fujitsu

All rights reserved.
Delivery subject to availability; right of technical modifications reserved.

All hardware and software names used are trademarks of their respective manufacturers.

Table of Contents

Creation and Restructuring

1 Preface

The **U**niversal **D**atabase **S**ystem UDS/SQL is a high-performance database system based on the structural concept of CODASYL. Its capabilities, however, go far beyond those of CODASYL as it also offers the features of the relational model. Both models can be used in coexistence with each other on the same data resources.

COBOL DML, CALL DML and (ISO standard) SQL are available for querying and updating data. COBOL DML statements are integrated in the COBOL language; SQL statements can be used in DRIVE programs.

To ensure confidentiality, integrity and availability, UDS/SQL provides effective but flexible protection mechanisms that control access to the database. These mechanisms are compatible with the openUTM transaction monitor.

The data security concept provided by UDS/SQL effectively protects data against corruption and loss. This concept combines UDS/SQL-specific mechanisms such as logging updated information with BS2000 functions such as DRV (Dual Recording by Volume).

If the add-on product UDS-D is used, it is also possible to process data resources in BS2000 computer networks. UDS/SQL ensures that the data remains consistent throughout the network. Distributed transaction processing in both BS2000 computer networks and networks of BS2000 and other operating systems can be implemented using UDS/SQL together with openUTM-D or openUTM (Unix/Linux/Windows). UDS/SQL can also be used as the database in client-server solutions.

The architecture of UDS/SQL (e.g. multitasking, multithreading, DB cache) and its structuring flexibility provide a very high level of throughput.

1.1 Structure of the UDS/SQL documentation

The “Guide through the manuals” section explains which manuals and which parts of the manuals contain the information required by the user. A glossary gives brief definitions of the technical terms used in the text.
In addition to using the table of contents, users can find answers to their queries either via the index or by referring to the running headers.

Guide through the manuals

The UDS/SQL database system is documented in five manuals:

- UDS/SQL Design and Definition
- UDS/SQL Application Programming
- UDS/SQL Creation and Restructuring
- UDS/SQL Database Operation
- UDS/SQL Recovery, Information and Reorganization

Further manuals describing additional UDS/SQL products and functions are listed on "Structure of the UDS/SQL documentation".

For a basic introduction the user should refer to chapters 2 and 3 of the "Design and Definition" manual; these chapters describe

- reasons for using databases
- the CODASYL database model
- the relational database model with regard to SQL
- the difference between the models
- the coexistence of the two database models in a UDS/SQL database
- the characteristic features of UDS/SQL

How the manuals are used depends on the user’s previous knowledge and tasks. Table 1 serves as a guide to help users find their way through the manuals.

Examples

A user whose task it is to write COBOL DML programs should look up the column “COBOL/CALL DML Programming” under “User task” in the second line of table 1.
There, the following chapters of the "Design and Definition" manual are recommended:

General information	B = Basic information
Schema DDL	D = Detailed information
SSL	D = Detailed information
Subschema DDL	L = Learning the functions

In the same column the user can also see which chapters of the other manual are of use.

Database administrators who are in charge of database administration and operation will find the appropriate information under the column “Administration and Operation”.

Contents of the five main manuals	User task							
	Design and definition	COBOL/ CALL DML programming	SQL program-ming	Creation and restructuring	Administration and operation	Working with openUTM	Working with IQS	Wo w UD
Manual UDS/SQL Design and Definition								
Preface	B	–	–	–	–	B	B	
General information	B	B	B	B	B	B	–	
Designing the database	B	–	–	–	–	–	–	
Schema DDL	L	D	–	L	L	–	–	
SSL	L	D	–	L	L	–	–	
Subschema DDL	L	L	–	L	L	–	–	
Relational schema	L	–	D	–	–	–	–	
Structure of pages	D	–	–	D	D	–	–	
Structure of records and tables	D	–	–	D	D	–	–	
Reference section	S	–	–	S	–	–	–	
Manual UDS/SQL Application Programming								
Preface	–	B	–	–	–	B	B	
Overview	–	B	–	–	–	–	–	
Transaction concept	–	L	–	L	L	D	D	
Currency table	–	L	–	L	L	–	–	
DML functions	D	L	–	L	–	–	–	
Using DML	–	L	–	D	–	–	–	
COBOL DML reference section	–	L	–	–	–	–	–	
CALL DML reference section	–	L	–	–	–	–	–	
Testing DML functions using DMLTEST	–	L	–	–	–	–	–	

Manual UDS/SQL Creation and Restructuring								
Preface	–	–	–	B	–	B	B	
Overview	–	–	–	B	B	–	–	
Database creation	–	–	–	L	–	–	–	
Defining access rights	–	–	–	L	–	–	–	
Storing and unloading data	D	–	–	L	–	D	–	
Restructuring the database	D	–	–	L	–	–	–	
Renaming database objects	D	–	–	L	–	–	–	
Database conversion	D	–	–	L	–	–	–	
Database conversion using BTRANS24	–	–	–	D	–	–	–	
Manual UDS/SQL Database Operation								
Preface	–	–	–	–	B	B	B	
The database handler	–	–	–	–	L	–	–	
DBH load parameters	–	–	–	–	L	–	–	
Administration	–	–	–	–	L	–	–	
High availability	–	–	–	–	B	–	–	
Resource extension and reorganisation during live operation	D	–	–	–	B	–	–	
Saving and recovering a database in the event of errors	D	–	–	D	L	D	–	

Optimizing performance	–	–	–	–	D	–	–	
Using BS2000 functionality	–	–	–	–	D	–	–	
The SQL conversation	–	–	–	–	L	–	–	
UDSMON	–	–	–	–	D	–	–	
General functions of the utility routines	–	–	–	–	D	–	–	
Using IQS	–	–	–	L	D	–	D	
Using UDS-D	D	D	–	D	D	D	–	
Function codes of DML statements	–	D	–	–	D	–	–	
ManualUDS/SQL Recovery, Information and Reorganization								
Preface	–	–	–	–	B	B	B	
Updating and reconstructing a database	D	–	–	D	L	D	–	
Checking the consistency of a database	–	–	–	–	L	–	–	
Output of database information	D	–	–	D	L	–	–	
Executing online services	D	–	–	D	L	–	–	
Database reorganization	D	–	–	D	L	–	–	
Controlling the reuse of deallocated database keys	D	–	–	D	L	–	–	
Additional Manuals								
UDS/SQL Messages	D	D	D	D	D	D	D	

UDS/SQL System Reference Guide	S	S	–	S	S	S	S	
IQS	–	–	–	D	D	–	L	
ADILOS	–	–	–	–	D	–	L	
KDBS	–	L [1]	–	D	–	–	–	
SQL for UDS/SQL Language Reference Manual	–	–	D	–	D	–	–	

Table 1: Guide through the manuals

[1] only for COBOL-DML

B	provides basic information for users with no experience of UDS/SQL
L	helps the user learn functions
D	provides detailed information
S	provides a reference to syntax rules for practical work with UDS/SQL

Additional notes on the manuals

References to other manuals appear in abbreviated form. For example:

(see the "Application Programming" manual, CONNECT)

advises the user to look up CONNECT in the "Application Programming" manual. The complete titles of the manuals can be found under “Related publications“ at the back of the manual.

UDS/SQL Messages

This manual contains all messages output by UDS/SQL. The messages are sorted in ascending numerical order, or in alphabetical order for some utility routines.

UDS/SQL System Reference Guide

The UDS/SQL System Reference Guide gives an overview of the UDS/SQL functions and formats.

SQL for UDS/SQL Language Reference Manual

This manual describes the SQL DML language elements of UDS/SQL.
In addition to UDS/SQL-specific extensions, the language elements described include dynamic SQL as an essential extension of the SQL standard.

1.2 Objectives and target groups of this manual

The manual is intended for the database (DB) administrator, i.e. the user responsible for creating the database, organizing database operation, and restructuring the database as needed to adapt it to changing operational requirements.

The database administrator should be familiar with all the steps involved in designing a database (including schema, subschema and SSL generation) and writing DB application programs.

In addition, he or she should have a comprehensive knowledge of BS2000.

1.3 Summary of contents

What does this manual contain?

This manual begins with an overview of the files required by the UDS/SQL Database System during database operation and of the UDS/SQL utility routines needed to create a UDS/SQL database.
It then goes on to describe all the stages involved in

- restructuring a database, and
- converting a database to a larger page format.

Readme file

The functional changes to the current product version and revisions to this manual are described in the product-specific Readme file.

Readme files are available to you online in addition to the product manuals under the various products at http://bs2manuals.ts.fujitsu.com. You will also find the Readme files on the Softbook DVD.

Information under BS2000

When a Readme file exists for a product version, you will find the following file on the BS2000 system:

`SYSRME.<product>.<version>.<lang>`

This file contains brief information on the Readme file in English or German (<lang>=E/D). You can view this information on screen using the `/SHOW-FILE` command or an editor.
The `/SHOW-INSTALLATION-PATH INSTALLATION-UNIT=<product>` command shows the user ID under which the product's files are stored.

Additional product information

Current information, version and hardware dependencies, and instructions for installing and using a product version are contained in the associated Release Notice.
These Release Notices are available online at http://bs2manuals.ts.fujitsu.com.

1.4 Changes since the last edition of the manuals

The main changes introduced in UDS/SQL V2.9 in comparison with Version V2.8 are listed in table 2 below together with the manuals and the sections in which the changes are described. If a specific topic has been dealt with in more than one manual, the manual in which a detailed description appears is listed first. The following codes are used in the “Manual” column for the individual manuals involved:

DES	Design and Definition	DBO	Database Operation
APP	Application Programming	RIR	Recovery, Information and Reorganization
CRE	Creation and Restructuring	MSG	Messages

Topic	Manual	Chapter
Changes in V2.9A		
FIND-/FETCH-7 with DESCENDING KEY: Suspension of the restriction		
The restriction for DESCENDING KEY is omitted	APP	7
Record references in COBOL programs:		
The new DDL-statement GENERATE-REC-REF generates a data field and condition names for the access to record references	CRE	3
Changing settings of ALOG files while the database is in use		
New DAL command DISPLAY ALOG shows ALOG settings	DBO	4
New DAL commands MODIFY ALOG/MODIFY ALOG-RES and MODIFY-ALOG-SIZE change ALOG settings	DBO	4
Change of the restrictions for the UDS Online Utility		
WAIT-FOR-TRANSACTION offers the possibility to wait until the locked source page is released by the locking transaction	RIR	8
With SET-RELOCATE-PARAMETERS a behaviour for the case that pages are locked can be specified also for *INDEX-LEVEL-TABLE-PAGES (CLASH-HANDLING)	RIR	8
BRENAME with after-image logging: The function “Renaming database objects (BRENAME, BALTER)” can also be executed when After Image Logging is activated. Thus logging gaps can be avoided		
New behaviour in a renaming cycle	CRE	7
After a renaming process a data base update can be executed	RIR	2
Specifying the size for DBTT extensions		
New operand EXT of the DAL command ACT DBTT-INCR	DBO	4

The BSTATUS output additionally contains the value of the EXT operand	RIR	6

New data type FIXED REAL BINARY 63		
Expansion of the syntax representation	DES	4, 9
Changes in messages to consider the new data type	MSG APP	2, 4 10
Changes in V2.9B		
BINILOAD of CSV files		
Explanation how BINILOAD handles CSV files	CRE	2, 5
Changes to consider new feature	MSG	3
BINILOAD with records of variable length		
Biniload can work with records with variable length	CRE	5
Changes to consider new feature	MSG	3
Default values for new fields in BALTER		
New BALTER statement to fill fields with specified values	CRE	6
Changes to consider new feature	MSG	3
Number of extents in DB-JV		
Number of extents of the ALOG file are added in Database Job Variable	DBO	9

Table 2: Changes in version V2.9 compared to V2.8

1.5 Notational conventions

This section provides an explanation of the symbols used for warnings and notes as well as the notational conventions used to describe syntax rules.

1.5.1 Warnings and notes

i	Points out particularly important information
! CAUTION!	Warnings

1.5.2 Non-SDF notational conventions

Language element	Explanation	Example
KEYWORD	Keywords are shown in underlined uppercase letters. You must specify at least the underlined parts of a keyword.	DATABASE-KEY MANUAL
OPTIONAL WORD	Optional words are shown in uppercase letters without underlining. Such words may be omitted without altering the meaning of a statement.	NAME IS ALLOWED PAGES
variable	Variables are shown in italic lowercase letters. In a format which contains variables, a current value must be entered in place of each variable.	*item-name* *literal-3* *integer*
{ either | or }	Exactly one of the expressions enclosed in braces must be specified. Indented lines belong to the preceding expression. The braces themselves must not be specified.	{ CALC | INDEX } { VALUE IS | VALUES ARE }
[optional]	The expression in square brackets can be omitted. UDS/SQL then uses the default value The brackets themselves must no be specified.	[IS *integer*] [WITHIN *realm-name*]
... or ,...	The immediately preceding expression can be repeated several times if required. The two language elements distinguish between repetitions which use blanks and those which use commas.	*item-name*,... {SEARCH KEY.....}...
..... or . .	Indicates where entries have been omitted for reasons of clarity. When the formats are used, these omissions are not allowed.	SEARCH KEY IS ..... RECORD NAME .. .
.	The period must be specified and must be followed by at least one blank. The underline must not be specified.	SET SECTION. 03 *item-name*..... .
Space	Means that at least one blank has to be specified.	USING CALC

Table 3: Notational conventions

All other characters such as ( ) , . ; “ = are not metacharacters;
they must be specified exactly as they appear in the formats.

1.5.3 SDF syntax representation

This syntax description is based on SDF Version 4.7. The syntax of the SDF command/statement language is explained in the following three tables.

Table 4 : Metasyntax

Certain characters and representations are used in the statement formats; their meaning is explained in table 4.

Table 5 : Data types

Variable operand values are represented in SDF by data types. Each data type represents a specific value set. The number of data types is limited to those described in table 5.

The description of the data types is valid for all commands and statements. Therefore only deviations from table 5 are explained in the relevant operand descriptions.

Table 6 : Data type suffixes

The description of the “integer” data type in table 6 also contains a number of items in italics. The italics are not part of the syntax, but are used merely to make the table easier to read.

The description of the data type suffixes is valid for all commands and statements. Therefore only deviations from table 6 are explained in the relevant operand descriptions.

Representation	Meaning	Examples
UPPERCASE LETTERS	Uppercase letters denote keywords. Some keywords begin with *.	OPEN DATABASE COPY-NAME = *NONE
=	The equal sign connects an operand name with the associated operand values.	CONFIGURATION-NAME = <name 1..8>
< >	Angle brackets denote variables whose range of values is described by data types and their suffixes (Tables 5 and 6).	DATABASE = <dbname>
Underscoring	Underscoring denotes the default value of an operand.	SCHEMA-NAME = *STD
/	A slash separates alternative operand values.	CMD = *ALL / <dal-cmd>
(...)	Parentheses denote operand values that initiate a structure.	*KSET-FORMAT(...)
Indentation	Indentation indicates that the operand is dependent on a higher-ranking operand.	USER-GROUP-NAME = *KSET-FORMAT (...) *KSET-FORMAT(...) | HOST = <host>

| | |	A vertical bar identifies related operands within structure. Its length marks the beginning and end of a structure. A structure may contain further structures. The number of vertical preceding an operand corresponds to the depth of the structure.	USER-GROUP-NAME = *ALL-EXCEPT(...) *ALL-EXCEPT(...) | NAME = *KSET-FORMAT(...) | *KSET-FORMAT(...) | | HOST = <host> | | ...
,	A comma precedes further operands at the same structure level.	,SPACE = STD
list-poss(n):	`list-poss` signifies that the operand values following it may be entered as a list. If a value is specified for (n), the list may contain no more than that number of elements. A list of two or more elements must be enclosed in parentheses.	NAME = list-poss(30): <subschema-name>

Table 4: Metasyntax

Data type	Character set	Special rules
alog-seq-no	0..9	1..9 characters
appl	A..Z 0..9 $,#,@ Structure identifier: hyphen	1..8 characters String that can consist of a number of substrings separated by hyphens; first character A..Z or $, #, @ Strings of less than 8 characters are filled internally with underscore characters.
catid	A..Z 0..9	1..4 characters must not start with the string PUB
copyname	A..Z 0..9	1..7 characters, starting with A..Z
c-string	EBCDIC characters	1..4 characters Must be enclosed in single quotes; the letter C may be used as a prefix. Single quotes within c-string must be specified twice.
csv-filename	A..Z 0..9 Structure identifier: hyphen	1..30 characters Must be enclosed in single quotes

dal-cmd	A..Z 0..9 hyphen	1..64 characters
date	0..9 Structure identifier: hyphen	Date specification Input format: yyyy-mm-dd yyyy : year; may be 2 or 4 digits long mm : month dd : day
dbname	A..Z 0..9	1..17 characters, starting with A..Z
device	A..Z 0..9 $,#,@ Structure identifier: hyphen	5..8 characters, starting with A..Z or 0..9 String that can consist of a number of substrings separated by hyphens and and whicch corresponds to a device. In the dialog guidance, SDF shows the permissible operand values. Information as the possible devices can be found in the relevant operand description.
host	A..Z 0..9 $,#,@ Structure identifier: hyphen	1..8 characters String that can consist of a number of substrings separated by hyphens; first character A..Z or $, #, @ Strings of less than 8 characters are filled internally with underscore characters.
integer	0..9,+,-	+ or - may only be the first character.
kset	A..Z 0..9 $,#,@ Structure identifier: hyphen	1..8 characters String that can consist of a number of substrings separated by hyphens; first character A..Z or $, #, @ Strings of less than 8 characters are filled internally with underscore characters.
name	A..Z 0..9 $,#,@	1..8 characters Must not consist only of 0..9 and must not start with a digit
realm-name	A..Z 0..9 Structure identifier: hyphen	1..30 characters String that may consist of a number of substrings by hyphens; first character: A..Z
realmref	0..9	1..3 characters

record-name	A..Z 0..9 Structure identifier: hyphen	1..30 characters String that can consist of a number of substrings separated by hyphens; first character: A..Z In the case of record types with a search key it is recommendable to use names with no more than 26 characters, otherwise the set name created implicitly (SYS_...) will be truncated in accordance with the restriction on the name length for sets.
recordref	0..9	1..3 characters
schema-name	A..Z 0..9 Structure identifier: hyphen	1..30 characters String that can consist of a number of substrings separated by hyphens; first character: A..Z
set-name	A..Z 0..9 Structure identifier: hyphen	1..30 characters String that can consist of a number of substrings separated by hyphens; first character: A..Z
structured-name	A...Z 0...9 $, #, @ hyphen	Alphanumeric string which may comprise a number of substrings separated by a hyphen. First character: A...Z or $, #, @
subschema-name	A..Z 0..9 Structure identifier: hyphen	1..30 characters String that can consist of a number of substrings separated by hyphens; first character: A..Z
time	0..9 Structure identifier: colon	Time-of-day specification `{ hh:mm:ss | hh:mm | hh }` hh, mm, ss: Leading zeros may be omitted
userid	A..Z 0..9 $,#,@	1..8 characters, beginning with A..Z or $,#,@ BPRIVACY: Strings of less than 8 characters are filled internally with underscore characters.
volume	A..Z 0..9 $,#,@	1..6 characters starting with A..Z or 0..9

x-string	Hexadecimal: 00..FF	1..8 characters Must be enclosed in single quotes and prefixed with the letter X. There may be an odd number of characters

Table 5: Data types

Suffix	**Meaning**
x..y *unit*	For the “integer” data type: range specification. x Minimum value permitted for “integer”. x is an (optionally signed) integer. y Maximum value permitted for “integer”. y is an (optionally signed) integer. *unit* for “integer” only: additional units. The following units may be specified: *Mbyte, Kbyte, seconds*

Table 6: Data type suffixes

1.6 Sample databases

The examples used in this manual refer to a sample database configuration consisting of the following four databases:

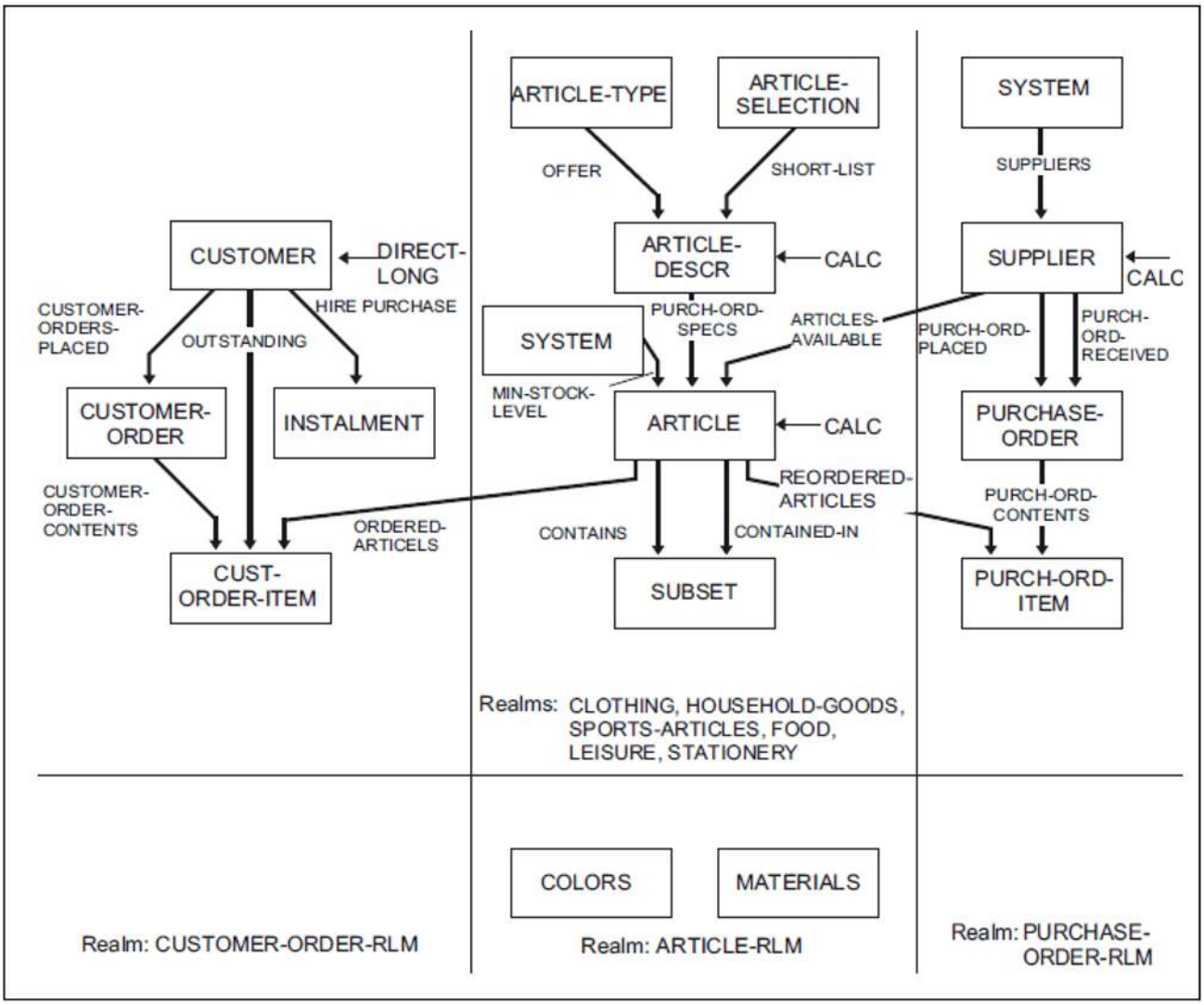


Figure 1: SHIPPING database with schema name MAIL-ORDERS

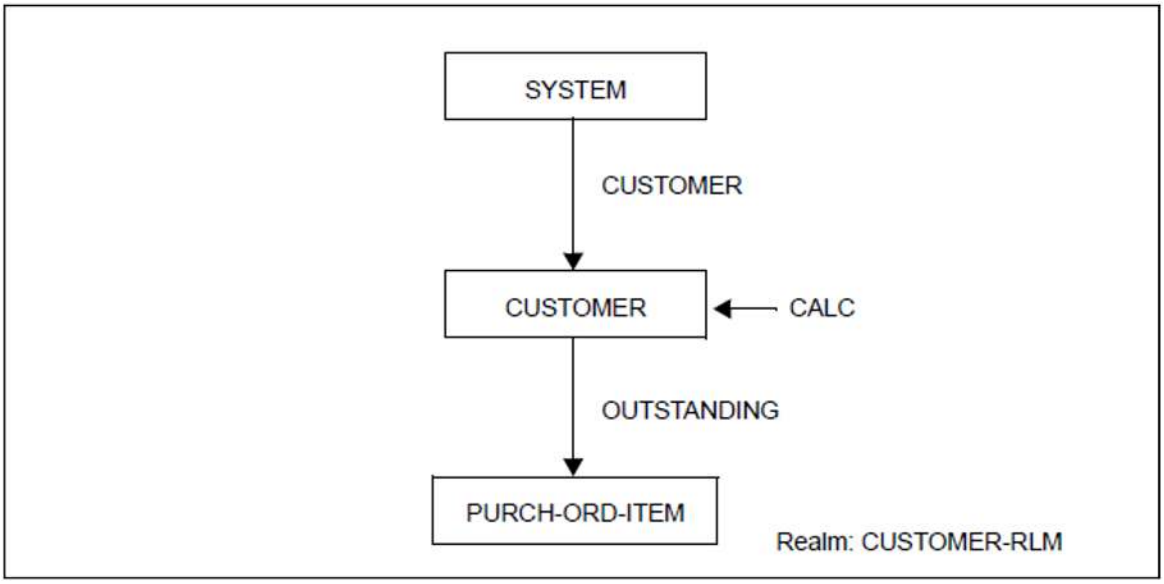


Figure 2: CUSTOMER database with schema name CUSTOMER-FILE

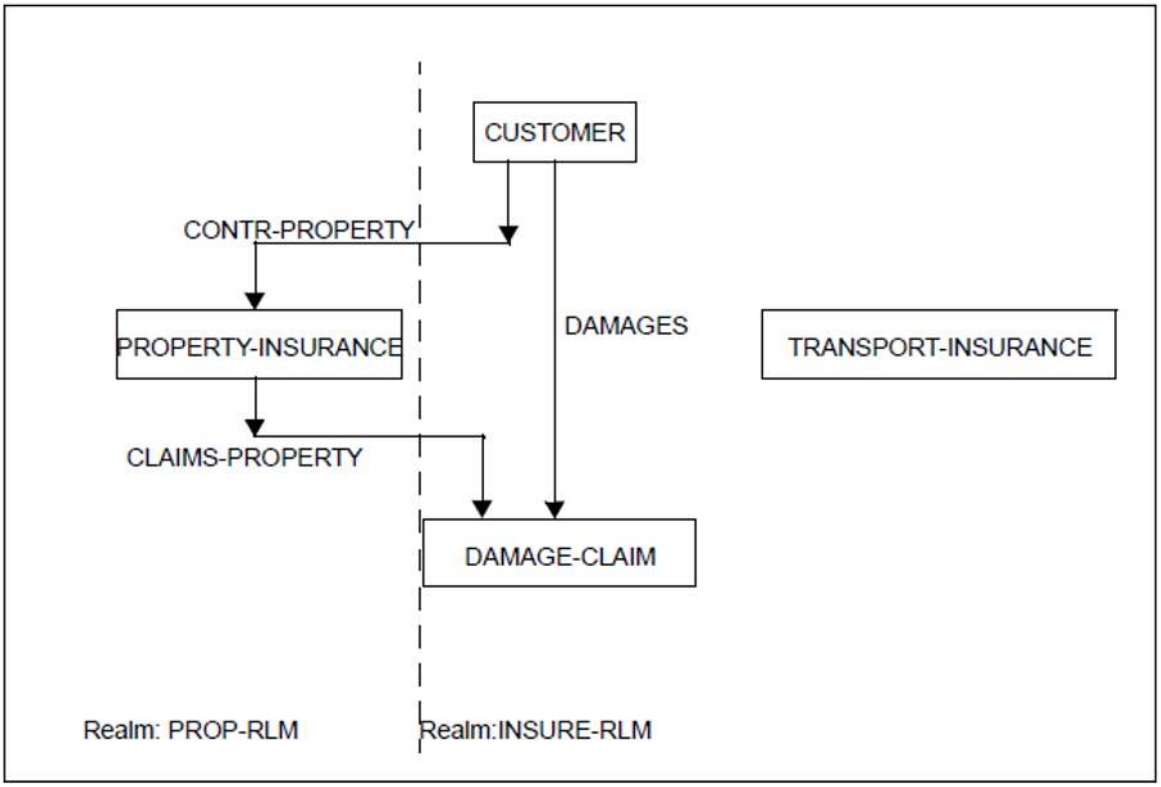


Figure 3: SHIPPINGDB database

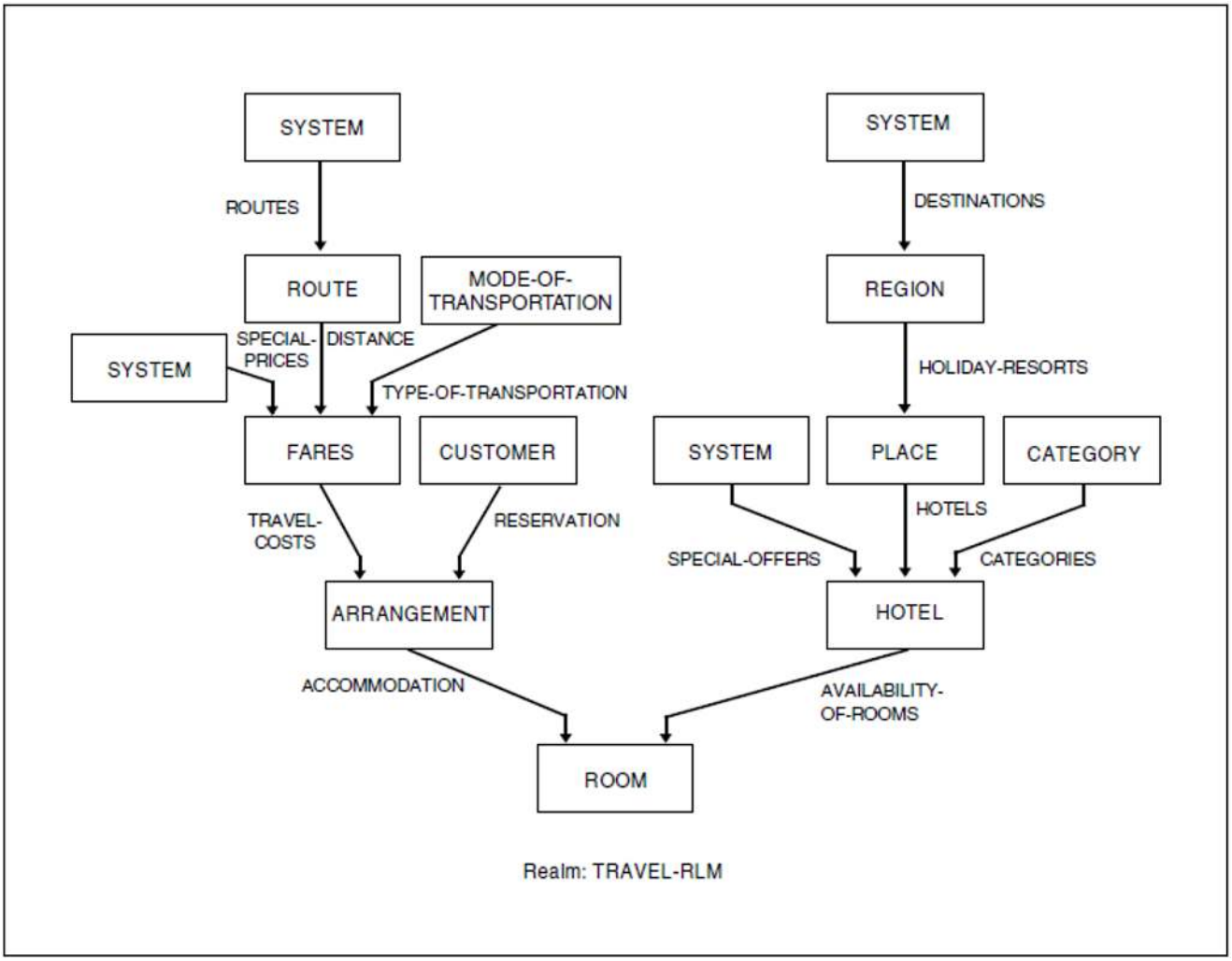


Figure 4: TRAVEL database with schema name TRAVEL-AGENCY

2 Overview of UDS/SQL

This chapter explains the basic concepts of UDS/SQL, provides an overview of the realms and files in a UDS/SQL database and introduces the programs of the UDS/SQL database system.

2.1 Basic concepts of the UDS/SQL database system

UDS/SQL database

A UDS/SQL database contains large amounts of interrelated data. The data in a database is stored in such a way that it is independent of programming functions and can be accessed to optimum effect by a range of different programs, while keeping redundancy to a minimum. The addition of new data and the retrieval, updating or deletion of existing data are closely controlled.

With UDS/SQL, multiple databases can be combined to form a multi-DB system that is processed as a single unit (see also “Database configuration”).

Database system

The database system is the sum of all the programs needed to create and maintain the data resources and to retrieve and store data.

Database handler (DBH)

The database handler (DBH), which controls access to the database, is the central component of UDS/SQL. It allows mono-DB operation or multi-DB operation.
The DBH is available in the following two versions:

- independent DBH
- linked-in DBH.

With the **independent DBH**, database operation is controlled by the following modules:

- UDSSQL
- UDSSUB
- UDSCT for UDS-D.

Each module executes as a separate task. Together, they form the task family of the independent DBH.

The modules have the following functions:

UDSSQL	Master task UDSSQL communicates with the database administrator and initiates, monitors and terminates the session.
UDSSUB	Server task (can be loaded more than once) UDSSUB receives the processing requests from the application programs and returns the results to the appropriate application program.
UDSCT	UDS-D task UDSCT handles the communication functions which are needed to process DML statements from remote application programs.

The **linked-in** DBH is not an independent program, but is linked into the application program concerned or loaded dynamically at runtime and runs as part of this program. An application program running under a linked-in DBH cannot update a database unless it is accessing it in EXCLUSIVE mode, although any number of linked-in DBHs can have RETRIEVAL mode access to a database. As the linked-in DBH, unlike the independent DBH, operates without task communication, it may produce improved runtimes.

Like the independent DBH, the linked-in DBH has multi-DB capability. The linked-in DBH cannot process SQL statements.

Session

A session is a period in which one or more users can work with the database(s). It begins with the loading of either DBH and ends with the message "`NORMAL SYSTEM TERMINATION`". The database configuration for the session is defined by operands entered by the database administrator when loading the DBH or in DAL commands.

Database configuration

When starting the session, the database administrator uses the DBH load parameters to specify which databases are to take part in the session. The databases selected for the session and the environment in which the session is to take place are known as the database configuration.

Every configuration is assigned a name by the administrator. The configuration data is stored in the session log file (SLF) for the duration of the session so that the current database configuration can be restored in the event of a restart.

Transaction

Every database application program must open a transaction with the DBH in order to communicate with a UDS/SQL database, regardless of the type of DBH used.

A transaction (TA) is a logically related sequence of DML statements that is either processed in its entirety or not at all. For example, a transaction in a COBOL-DML program begins with a READY statement and ends with FINISH. In mono-DB mode, every READY statement opens a transaction and also the database. Thereafter, the application program can access the opened database any number of times using DML statements. In multi-DB mode, by contrast, each database needs to be opened independently with a READY statement. Consequently, a transaction in multi-DB mode may include several READY statements, where the first such statement opens the transaction itself.
Once the COBOL-DML application program has issued the DML statement FINISH to terminate the transaction, it cannot access the database(s) again until the following applies:

- In mono-DB operation: a new transaction, and thus the database, is opened by the application program by means of another READY statement.
- In multi-DB operation: a new transaction, and thus a database, is opened by the application program by means of a new READY statement. Note that additional databases may have also been opened within the same transaction by means of further READY statements.

Even an SQL program also only access UDS/SQL databases from within a transaction In SQL programs, a transaction begins with the first SQL statement that differs from COMMIT WORK and ends with the SQL statement COMMIT WORK.

Conversation

In an SQL application, SQL-specific administrative data is maintained beyond transaction boundaries.
Each such administrative unit is called a conversation.

> In openUTM such an administrative unit is also called a service.

2.2 Files and realms of a UDS/SQL database

A UDS/SQL database consists of the user database and the compiler database and comprises a number of areas (known as realms).

The **user database** contains all the realms and files that are needed by the user to store data in the database and retrieve data from it. This includes:

- the user realms
- the database directory (DBDIR)
- the module library for hash routines (HASHLIB)

The **compiler database**, which is required by the DDL compiler and the COBOL compiler, contains the compiled schema and subschema descriptions. It includes:

- the database directory (DBDIR)
- the database compiler realm (DBCOM)
- the COBOL subschema directory (COSSD)

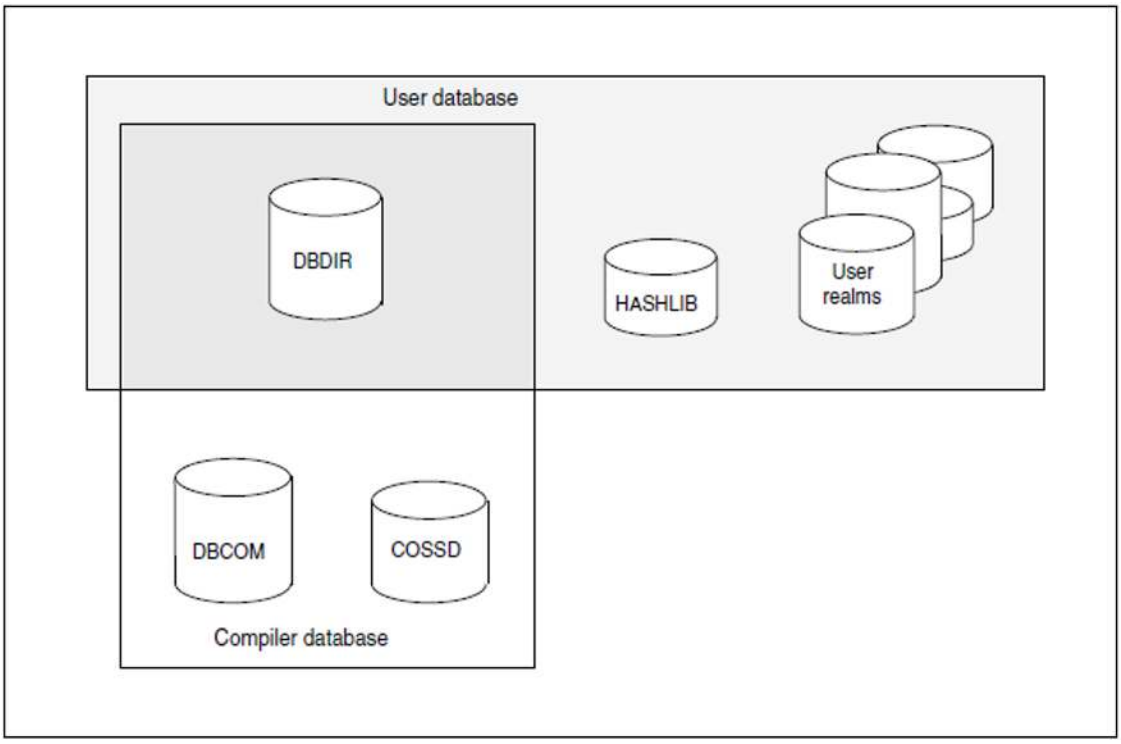


Figure 5: The UDS/SQL database

In addition to the files belonging to the user and compiler databases, certain other files are necessary for data security and for setting up the connection to the database.
The following is a comprehensive list of the files of a UDS/SQL database:

Database realms

dbname.realmname	original user realm(s)
dbname.DBDIR	database directory
dbname.DBCOM	database compiler realm

Files for database operation

dbname	link file to database for mono-DB operation
dbname.COSSD	COBOL subschema directory
dbname.HASHLIB	hash routine storage library
configuration-name	link file for database configuration
confname.DBSTAT	DB status file
confname.DBSTAT.SAVE	duplicate of the DB status file
confname.SLF	session log file (SLF)
confname.TEMPO.nnn	temporary user file
UDS.ENTER.tsn.ST0nn	ENTER files for starting server tasks

Files for ensuring the security of the database

Shadow database

- *dbname*.DBDIR.*copyname*
- *dbname*.DBCOM.*copyname*
- *dbname*.COSSD.*copyname*
- *dbname*.HASHLIB.*copyname*
- *dbname*.*realmname*.*copyname*

ALOG file

- *dbname*.A.*seqno*

RLOG files

- *confname*.RLOG.*rlogtimestamp*.1
- *confname*.RLOG.*rlogtimestamp*.2
- *confname*.RLOG.*rlogtimestamp*.1.SAVE
- *confname*.RLOG.*rlogtimestamp*.2.SAVE

Files of the database converted with BPGSIZE

- *dbname*.DBDIR.NEW

- *dbname*.DBCOM.NEW
- *dbname*.COSSD.NEW (if BPGSIZE is used to convert to a larger page format)
- *dbname*.*realmname*.NEW

Syntax rules

dbname

Database name; max. 17 characters in length. Used as a partial qualifier in almost all file names for database realms and files. It has to conform to the following rules:

- *dbname* must be the same for all the files in the database.
- *dbname* must not contain special characters or blanks, and its first character must be a letter.
- :*catid*:$*userid*.*dbname*.*realmname*.*copyname* may be at most 54 characters in length.

With UDS-D:

- *dbname* must be unique network-wide.

configuration-name

Freely selectable name of the database configuration; in mono-DB operation this name can be identical to *dbname*. It has to conform to the following rules:

- *configuration-name* may be at most 41 characters in length.
- The first seven characters of all the *configuration-names* in a BS2000 session must be unique network-wide.
- The first eight characters of *configuration-name* must not contain special characters.

confname

the first eight characters of the database configuration defined by the database administrator at the start of the session

With UDS-D:

- *confname* must be unique in the first seven characters.

nnn number of the transaction to which the file has been allocated

nn number of the server task

pool name of the common memory pool

tsn task sequence number of the master task

realm-name

name of a realm of the database;

for user realms: realm name as defined in the AREA clause of the Schema DDL

for database directory: DBDIR

for database compiler realm: DBCOM

copyname

suffix for the shadow database; *copyname* consists of up to seven characters.

seqno

nine-digit sequence number allocated to each ALOG file

rlogtimestamp

time at which the corresponding RLOG file was opened

The following is a short description of all the files and realms in a UDS/SQL database:

Database realms

- User realms (*dbname*.*realmname*)

  Before loading the realms with data, they have to be defined in the AREA clause of the Schema DDL.

- Database directory (*dbname*.DBDIR)

  The database directory (DBDIR) includes the full schema description, all subschema descriptions, and information on access rights.
  It also contains information as to whether AFIM logging is switched on or off and whether realms are added or dropped or have been deleted during restructuring. The database handler needs this information in order to handle the user's database access requests within the area of the used subschema.

- Database compiler realm (*dbname*.DBCOM)

  The database compiler realm (DBCOM) stores information on the realms, records and sets that the user has defined in the Schema DDL and the Subschema DDL. The DBCOM is required only for compiling the Schema DDL and the Subschema DDL and for creating the DBDIR and the COSSD.

Files required for database operation

- Link file (*dbname*)

  Empty link file for the database in mono-DB operation.

- COBOL subschema directory (*dbname*.COSSD)

  On compiling the subschema, the DDL compiler writes information on it into the COBOL subschema directory (COSSD). This information is required by the COBOL compiler to compile DB application programs. The COSSD provides the COBOL compiler with the data structure of the subschema and with a table for checking the validity of the DML commands.

- Module library for hash routines (*dbname*.HASHLIB)

  The module library *dbname*.HASHLIB stores the hash routines for the database.

- Link file (*configuration-name*)

  Empty link file for the DB configuration in multi-DB operation.

- DB status file
  (*confname*.DBSTAT)
  (*confname*.DBSTAT.SAVE)

  The DB status file is required by openUTM for a restart; it contains information on the transaction that was most recently rolled back in each UDS/SQL/openUTM application. The DB status file is duplicated for data security reasons.

  With UDS-D please note:

  In distributed processing with UDS-D, this file may also hold information stored when the transaction is committed.

- Session log file (*confname*.SLF)

  The session log file (SLF) is required by the DBH for restarts, as it contains information on the databases attached to the configuration and the current values of the DBH load parameters.

- Temporary user file (*confname*.TEMPO.*nnn)*

  If a temporary realm has been declared for (at least) one of the databases in the configuration, the DBH creates a temporary file for each main reference (parallel open transaction). This file stores temporary information.

  nnn number of main reference

- ENTER files (UDS.ENTER.*tsn*.ST0*nn*)

  The master task generates one or more ENTER files for server tasks (UDS.ENTER.*tsn*.ST0*nn*).

  These ENTER tasks are started by the master task using the ENTER commands. If the session is terminated normally, the master task deletes all ENTER files.

Files for ensuring data security

- Shadow database

  (*dbname*.DBDIR.*copyname*)
  (*dbname*.DBCOM.*copyname*)
  (*dbname*.COSSD.*copyname*)
  (*dbname*.HASHLIB.*copyname*)
  (*dbname*.*realmname*.*copyname*)

  You can use the COPY-FILE command to copy the realms and files of the database.
  The MODIFY-FILE-ATTRIBUTES commands allow you to rename the database. You can also save the database using the BS2000 utility ARCHIVE.

- ALOG files (*dbname*.A.*seqno*)

  The DBH or any updating utility routine logs every update to a page, i.e. the status of the page *after* the update (after-image) in ALOG files. These ALOG files can thus be used to apply updates to the database.
  The shadow database has no ALOG files; however, all changes can be incorporated into the shadow database by using the ALOG files of the original database.

- RLOG files

  (*confname*.RLOG.*rlogtimestamp*.1)
  (*confname*.RLOG.*rlogtimestamp*.2)
  (*confname*.RLOG.*rlogtimestamp*.1.SAVE)
  (*confname*.RLOG.*rlogtimestamp*.2.SAVE)

  The RLOG files are used by the DBH to log information (data) being changed both **before** the change (before-image, or BFIM) and **after** the change (after-image, or AFIM) as needed for use in any rollbacks or warm starts that may be required.

Maximum size of UDS/SQL files

UDS/SQL can administer a maximum of 16777214 database pages in a realm. This results in the following limit values for the maximum file sizes:

Database page size	Maximum file size in PAM pages
2 Kbyte	16777214
4 Kbyte	33554428
8 Kbyte	67108856

Table 7: Limit values for file sizes

UDS/SQL supports of utility routines realm files, temporary files, logging files and work files as LARGE FILEs. Files which, because of their nature, can only be of a limited size are not processed with the LARGE FILE property by UDS/SQL (DBSTAT, COSSD, HASHLIB, parameter files).
The monitor's output files also do not have the LARGE FILE property.

Files set up as auxiliary files by the UDS/SQL utility routines are not created with BLKCNTRL=PAMKEY so that it is not made implicitly impossible to use them as LARGE FILEs.

The following prerequisites must be fulfilled in the system if the LARGE FILE property is to be used:

- Large files may only be used in pubsets that have the property LARGE-FILES-ALLOWED.
- Large files cannot be used in the HOME pubset.
- Large files cannot be used with BLKCTRL=PAMKEY.

Passwords for UDS/SQL files

UDS/SQL protects the automatically generated files with the default password: C'UDS'BLANK''.The RLOG file is an exception. The password for the RLOG file, which is made up of parts of the RLOG time stamp, is assigned automatically. The password can only be deleted without password protection in the system ID ($TSOS).

2.3 Overview of UDS/SQL programs

The UDS/SQL (BS2000) system in its entirety incorporates a series of programs required for creating, maintaining and communicating with the database

The functions of these programs are described in brief below:

Creating the database	**Preparing the program run**	**Loading or unloading the database**	**Monitoring the session**	**Working with the database**	**Testing DML functions**
BCREATE DDL SSL BGSIA BFORMAT BGSSIA BPRIVACY OPRIVACY	BCALLSI	BINILOAD BOUTLOAD	UDSMON	IQS (not part of the UDS/SQL delivery package)	DMLTEST
Database maintenance					
Information output	**Reorganizing the database**	**Restructuring the database/Renaming database objects**	**Recovering the database**	**Checking the database**	**Database conversion**
BPSIA BPSQLSIA BSTATUS BPRECORD	BREORG BMODTT ONLUTIL	BCHANGE BRENAME BALTER	BMEND	BCHECK	BPGSIZE BTRANS24
Database operation					
		Administer UDS/SQL			
		UDSADM			

Table 8: Program overview

Creating the database

BCREATE	formats the DBDIR and the DBCOM.
DDL	DDL compiler compiles the Schema DDL and the Subschema DDL, and sets up DBCOM and COSSD.
SSL	SSL compiler compiles the SSL and modifies data in DBCOM.
BGSIA	sets up the schema information area (SIA) and stores it in the DBDIR.

BFORMAT	formats the user realms of the database and modifies the SIA.
BGSSIA	sets up the subschema information area (SSIA) and stores it in the DBDIR
BPRIVACY or OPRIVACY	enters the user access rights in the DBDIR.

Preparing for the program run

BCALLSI	only needed in conjunction with CALL DML. BCALLSI makes subschema information available to CALL DML users.

Loading the database

BINILOAD	rapidly loads large volumes of data of the same record type into the database.

Unloading the database

BOUTLOAD	copies, deletes or unloads record types from a database.

Monitoring the session

UDSMON	outputs the UDS/SQL operating values during database operation.

Testing DML functions

DMLTEST	tests individual DML functions in interactive mode and in procedures.

Outputting information on the database

BPSIA	prints a summary of the chief information from the schema or a given subschema of the database
BPSQLSIA	prints the relational schema information of an existing UDS/SQL subschema defined in accordance with the CODASYL model. The relational schema information serves as a programming aid for the SQL user.
BSTATUS	generates statistics on storage occupancy in the database realms.
BPRECORD	outputs the contents of database realms.

Reorganizing the database

BREORG	increases and reduces the size of database realms, increases and reduces the permissible number of records of a record type and reorganizes tables and hash areas
BMODTT	controls the reuse of database key values that have been released and the search for free space by the DBH.
ONLUTIL	relocates records in a database and modifies settings to a database.

Restructuring/renaming the database

BCHANGE prepares DBDIR, DBCOM and COSSD for restructuring

BRENAME prepares DBDIR, DBCOM and COSSD for renaming.

BALTER executes the restructuring/renaming of the existing database in accordance with the new schema description.

Recovering the database

BMEND creates ALOG files and offers functions for recovering a destroyed database and outputting information on the status of realms to be updated and ALOG files.

Checking the database

BCHECK checks whether the physical structures of a database are correct; can be used in conjunction with data security so that inconsistencies in the database can be detected and eliminated at an early stage.

Converting the database

BPGSIZE defines a new page format for the database (database conversion).
During conversion, BPGSIZE can optionally create

- a copy of the database with the larger page length.
- a copy of the database with the unaltered page length. Note that the realms of the converted database usually require less storage space.

BTRANS24 converts databases of UDS/SQL V2.0 to V2.3 for use in UDS/SQL V2.4 and higher.

UDS/SQL administration

UDSADM The UDSADM program can be used to administer a UDS/SQL configuration.

Some utility programs that access UDS/SQL databases have to be run in conjunction with the DBH: such programs load the linked-in DBH using default values for the load parameters, and they all work with only one database.

The following table shows which programs load the linked-in DBH dynamically and which require files from the compiler database during the program run:

UDS/SQL program	Loads linked-in DBH dynamically	Access to		
		DBDIR	DBCOM	COSSD
BALTER	-	X	X	-
BCALLSI	-	-	-	X
BCHANGE	-	X	X	X
BCHECK	-	X	-	-

BCREATE	-	X	X	-
BFORMAT	-	X	-	-
BGSIA	X	X	X	-
BGSSIA	X	X	X	-
BINILOAD	-	X	X[1]	-
BMEND	-	X	X	-
BMODTT	-	X	-	-

BOUTLOAD	-	X	X[2]	-
BPGSIZE	-	X	X	X
BPRECORD	-	X	-	-
BPRIVACY	X	X	X[3]	-
BPSIA	-	X	-	-
BPSQLSIA	-	X	-	X
BRENAME	-	X	X	X
BREORG	-	X	-	-
BSTATUS	-	X	-	-
BTRANS24	-	X	X	-
COBOL compiler	-	-	-	X
DDL compiler	X	X	X	X
DMLTEST	X[4]	X	-	-
ONLINE-PRIVACY	-	X	X[3]	-
ONLINE-UTILITY	X[4]	X	-	-
SSL compiler	X	X	X	-
UDSADM	-	-	-	-
UDSMON	-	-	-	-

Table 9: Overview of programs in the UDS/SQL database system

[1] DBCOM is used when input file is in CSV format or record type contains a variable field

[2] DBCOM is used when CSV-OUTPUT = *YES is specified.

[3] The DBCOM is not read, but it must be present

[4] Optional

2.3.1 START commands for the UDS/SQL programs

program run the system sets the job variable to the followingThe following table shows the START commands (and their aliases) that you can use to call the specified UDS/SQL programs

The following prerequisites must be fulfilled:

- UDS/SQL must be installed with IMON and
- the SDF system syntax file must be activated.

Program	START command	Alias
UDSSQL	START-UDS-DBH	UDS, SYSINT
BALTER	START-UDS-BALTER	BALTER
BCALLSI	START-UDS-BCALLSI	BCALLSI
BCHANGE	START-UDS-BCHANGE	BCHANGE
BRENAME	START-UDS-BRENAME	BRENAME
BCHECK	START-UDS-BCHECK	BCHECK
BCREATE	START-UDS-BCREATE	BCREATE
BFORMAT	START-UDS-BFORMAT	BFORMAT
BGSIA	START-UDS-BGSIA	BGSIA
BGSSIA	START-UDS-BGSSIA	BGSSIA
BINILOAD	START-UDS-BINILOAD	BINILOAD
BMEND	START-UDS-BMEND	BMEND, START-UDS-REPAIR
BMODTT	START-UDS-BMODTT	BMODTT
BOUTLOAD	START-UDS-BOUTLOAD	BOUTLOAD, START-UDS-OUTLOAD
BPGSIZE	START-UDS-BPGSIZE	BPGSIZE, START-UDS-PAGE-RESIZING
BPRECORD	START-UDS-BPRECORD	BPRECORD
BPRIVACY	START-UDS-BPRIVACY	BPRIVACY, START-UDS-AUTHORIZATION
BPSIA	START-UDS-BPSIA	BPSIA
BPSQLSIA	START-UDS-BPSQLSIA	BPSQLSIA, START-UDS-PRINT-SQLSIA

BREORG	START-UDS-BREORG	BREORG, START-UDS-REORGANIZATION
BSTATUS	START-UDS-BSTATUS	BSTATUS
DDL	START-UDS-DDL	DDL
DMLTEST	START-UDS-DMLTEST	DMLTEST
SSL	START-UDS-SSL	SSL
ONLINE-PRIVACY	START-UDS-ONLINE-PRIVACY	OPRIVACY
UDSADM	START-UDS-ADM	UDSADM, START-UDS-ADMINISTRATION
UDSMON	START-UDS-UDSMON	UDSMON
UDS online utility	START-UDS-ONLINE-UTILITY	ONLUTIL

Table 10: Calling UDS/SQL programs using START commands

Syntax of the START-UDS-... commands

START-UDS-...

VERSION = *STD / <product-version>

,MONJV = *NONE / <filename 1..54 without-gen-vers>

,CPU-LIMIT = *JOB-REST / <integer 1..32767 seconds>

,RESIDENT-PAGES = [*PARAMETERS](...) *Only for DBH*

[***PAR**AMETERS](...)

| **MIN**IMUM = ***STD** / <integer 0..32767 4Kbyte>

| ,**MAX**IMUM = ***STD** / <integer 0..32767 4Kbyte>

VERSION =
Product version of the program which is to be started.

VERSION = *STD
No explicit specification of the product version. The product version is selected as follows:

1. The version predefined with the /SELECT-PRODUCT-VERSION command.
2. The highest version installed with IMON.

VERSION = <product-version>
Explicit specification of the product version in the form mm.n[a[kk]].

You are recommended always to specify the version in full, e.g. 02.9C00, in order to facilitate migration in the event of correction packages.

MONJV =
Specifies a monitor job variable to monitor the program run.

MONJV = *NONE
No monitor job variable is used.

MONJV = <filename 1..54 without-gen-vers>
Name of the job variable to be used.

During the program run the system sets the job variable to the following values:

$R Program running

$T Program successfully terminated

$A Program terminated with error

CPU-LIMIT =
Maximum CPU time in seconds which the program may take to execute.

CPU-LIMIT = *JOB-REST
The remaining CPU time for the BS2000 job is to be used for the task.

CPU-LIMIT = <integer 1..32767 *seconds* >
Only the time specified should be used.

RESIDENT-PAGES = *PARAMETERS(...)
This operand is only permitted for the DBH.

Number of resident memory pages which are required for the DBH run. This operand must be specified if pages are to be made resident in the program by means of a CSTAT macro (see the "Executive Macros" manual). The permissible number of resident memory pages can be influenced by the operator.
If the operand is missing (this corresponds to MIN=*STD, MAX=*STD), the memory requests are taken from the first record in the program. The file must be open to do this.

MINIMUM = *STD / <integer 0..32767 *4Kbyte* >
Minimum number of resident memory pages required.

MAXIMUM = *STD / <integer 0..32767 *4Kbyte* >
Maximum number of resident memory pages required.

UDS/SQL programs not installed with IMON

If UDS/SQL has not been installed with IMON, you must enter the following commands to start the UDS/SQL programs:

```
[/MODIFY-SDF-OPTIONS SYNTAX-FILE=$userid.SYSSDF.UDS-SQL.029.USER]
  /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname
  /ADD-FILE-LINK LINK-NAME=$UDSLIB,FILE-NAME=$userid.SYSLNK.UDS-SQL.029
  /START-EXECUTABLE-PROGRAM FROM-FILE=(LIB=$userid.SYSLNK.UDS-SQL.029
,ELEMENT=uds-utility)
```

You only need to specify the USER syntax file with the MODIFY-SDF-OPTIONS command if the system syntax file SYSSDF.UDS-SQL.029 is not active and if UDS/SQL programs are used with the SDF command interface, i.e. for

- BMEND
- BOUTLOAD
- BPGSIZE
- BPRIVACY and OPRIVACY
- BPSQLSIA
- BREORG
- UDSADM
- BTRANS24

2.4 Tools for UDS/SQL

A number of tools that are not an integral part of the UDS/SQL product scope are supplied as an additional service. Descriptions of these tools can be found in the information files included in the delivery package.

The tools are not subject to any maintenance obligation and can be modified or withdrawn by Fujitsu Technology Solutions without prior announcement.

3 Database creation (BCREATE, BFORMAT, DDL- and SSL- Compiler, BGSIA, BGSSIA, BCALLSI)

The creation of a database requires a number of steps, which are listed in figure 6. The diagram gives you the exact sequence of the preparations you have to make and the programs you have to run.

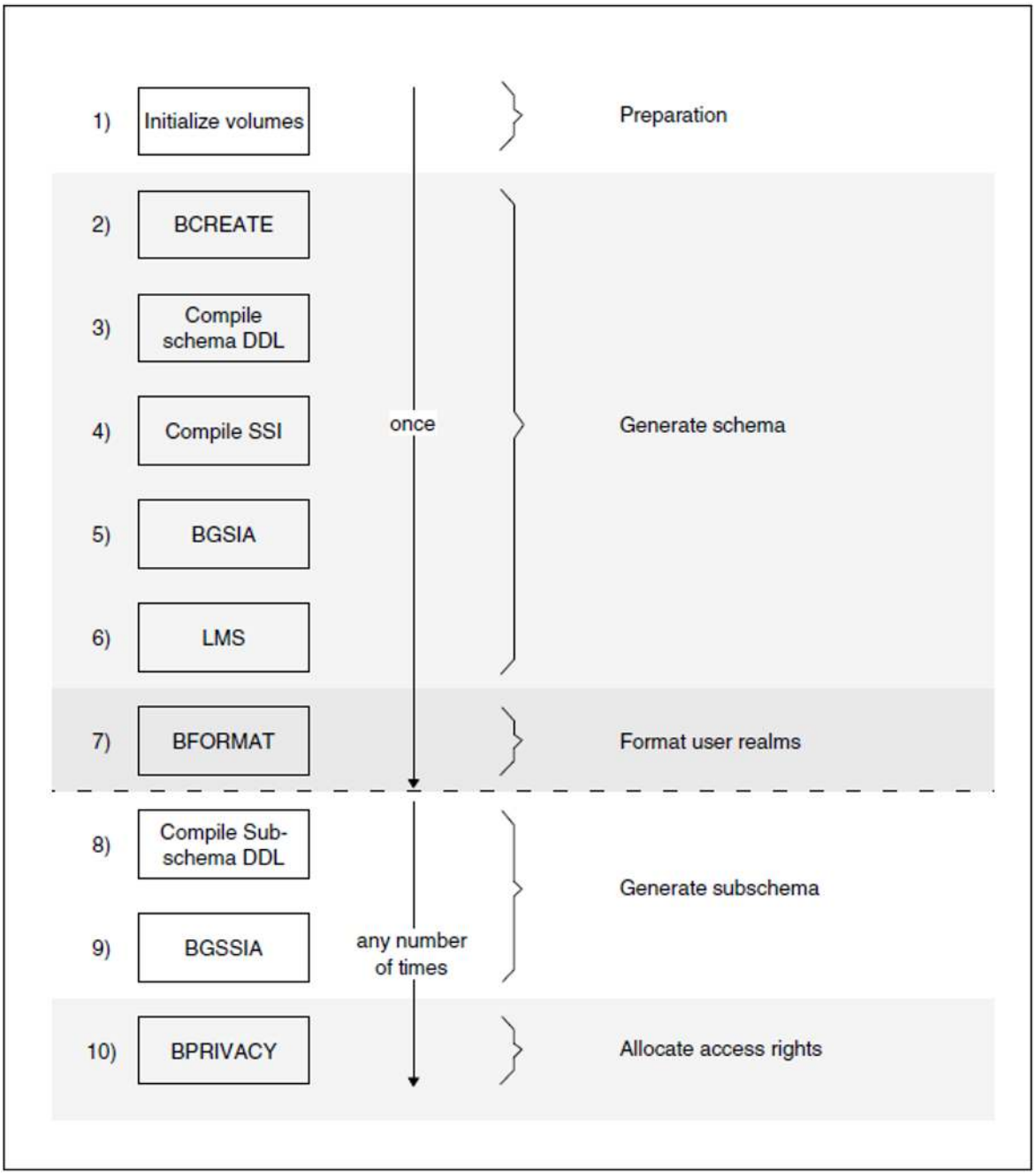


Figure 6: Stages in the creation of a database

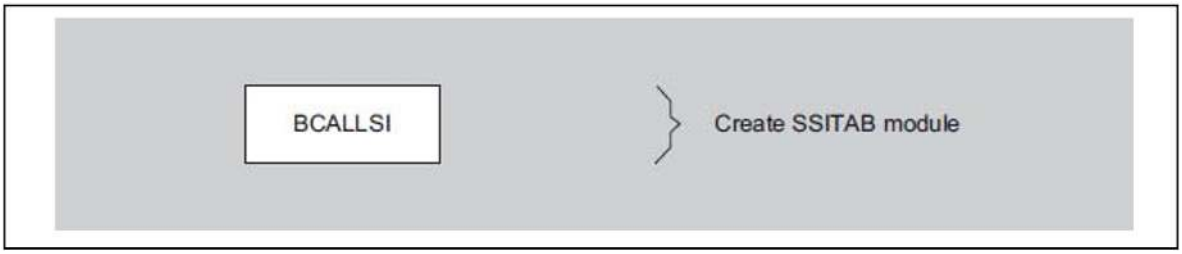


Figure 7: Additional database creation measures

Preparatory stage

1. The CREATE-FILE command is used to allocate storage space for the realms and files of the compiler database: DBDIR, DBCOM, COSSD and DBSTAT.

Generating the schema

2. The BCREATE utility routine formats the realms and files of the compiler database:
   DBDIR and DBCOM.

3. The DDL compiler compiles the Schema DDL and stores the compiled result in the DBCOM. It also stores information in the COSSD file which it has set up previously.

4. SSL compilation is optional, depending on whether the default values for the storage structure are to apply or whether the physical structure of the database is to be defined with SSL. If a storage structure definition (SSL) has already been generated, it has to be compiled at this point.
   The SSL compiler redefines the record and set entries in DBCOM accordingly.

5. The BGSIA utility routine sets up the schema information area (SIA) on the basis of the entries in the DBCOM and stores it in the DBDIR. In addition, BGSIA generates the UDSHASH module and stores it in the EAM file.

6. The UDSHASH module generated by BGSIA has to be entered in the modulelibrary *dbname*.HASHLIB using the BS2000 utility routine LMS (see the "LMS (BS2000)" manual).

Formatting user realms

7. The BFORMAT utility routine formats the user realms of the database on the basis of the information stored in the DBDIR. The data is then located in the schema information area (SIA).

Generating the subschema

8. The DDL compiler compiles the Subschema DDL and stores it in the DBCOM and COSSD.

9. The BGSSIA utility routine sets up the subschema information area (SSIA) and stores it in the DBDIR.

Defining access rights

10. The ONLINE-PRIVACY or BPRIVACY utility routine is used to assign access rights (see the chapter “Specifying access authorizations (ONLINE-PRIVACY, BPRIVACY)”).

Generating the SSITAB module

The BCALLSI utility routine is required by CALL DML users only. BCALLSI generates the SSITAB module with the subschema information needed by the CALL DML application program.

3.1 Preparing database creation

Preparations for database creation comprise:

- setting up the compiler database
- setting up the user database.

When using MPVS to determine the location of storage on public disks, please refer to the instructions in the "UDS/SQL Database Operation" manual (see Using MPVS in UDS/SQL in the "Database Operation" manual).

3.1.1 Setting up the compiler database

The compiler database (see section “Files and realms of a UDS/SQL database”) consists of the following realms:

DBDIR database directory

DBCOM database compiler realm

the file:

COSSD COBOL subschema directory

These files and realms have to be set up and their size specified using the BS2000 command CREATE-FILE.

Setting up DBDIR and DBCOM

```
/CREATE-FILE FILE-NAME=dbname.DBDIR
    ,SUPPORT=*PUBLIC-DISK(SPACE=*RELATIVE(PRIMARY-ALLOCATION=primary
    ,SECONDARY-ALLOCATION=secondary))
   [,SUPPORT=*PRIVATE-DISK(VOLUME=priv-vsn,DEVICE-TYPE=device,SPACE=...)]

/CREATE-FILE FILE-NAME=dbname.DBCOM
    ,SUPPORT=*PUBLIC-DISK(SPACE=*RELATIVE(PRIMARY-ALLOCATION=primary
    ,SECONDARY-ALLOCATION=secondary))
   [,SUPPORT=*PRIVATE-DISK(VOLUME=priv-vsn,DEVICE-TYPE=device,SPACE=...)]

/ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR
```

dbname

Database name; max. 17 characters in length. *dbname* is used as a partial qualifier in almost all file names for database files and has to conform to the following rules:

- *dbname* must be the same for all files in the database.
- *dbname* must not contain special characters or blanks, and its first character must be a letter.
- :*catid*:$*userid*.*dbname*.*realmname*.*copyname* may be at most 54 characters in length.

With UDS-D:

- *dbname* must be unique network-wide.

SPACE

Specification of storage space (see section “Maximum size of UDS/SQL files" in chapter "Files and realms of a UDS/SQL database").

PRIMARY-ALLOCATION=*primary*

Primary allocation;
In order to format the DBCOM and DBDIR with the BCREATE utility routine, the following minimum values are required for the primary allocation (see section “Formatting the compiler database with BCREATE”):

	DBDIR	DBCOM
2-Kbyte format	52	100
4-Kbyte format	79	424
8-Kbyte format	127	607

In the case of databases with large schemas, a correspondingly higher allocation is required, or automatic extensibility must be enable by a secondary allocation > 0.
The BREORG utility routine can be used later to reduce the amount of unused space (see the "Recovery, Information and Reorganization" manual, BREORG).

SECONDARY-ALLOCATION=*secondary*

Secondary allocation.

secondary=0

This setting suppresses the option of automatic realm extensibility and of online realm extensibility.

secondary>0

Prerequisite for automatic realm extensibility and online realm extensibility and prerequisite for activating online extensibility of the DBDIR using the ACT INCR command.
In the case of secondary>0, online realm extensibility is already activated for the realm concerned when the database is created.
When online realm extensibility is activated in this way, the default values NR-PAGES=64 and MIN-PAGES=0 are entered for NR-PAGES and MIN-PAGES (see Activating online extensibility when creating databases and DAL command ACT INCR in the "Database Operation" manual).

VOLUME
DEVICE-TYPE

if DBDIR and DBCOM are stored on private disk (PRIVATE VOLUME), the following must be specified:

priv-vsn

Volume serial number

device

Device type of private disk

LINK-NAME

The database directory should be linked to the database via the file link name DATABASE.

Setting up COSSD

```
/CREATE-FILE FILE-NAME=dbname.COSSD
     ,SUPPORT=*PUBLIC-DISK(SPACE=*RELATIVE(PRIMARY-ALLOCATION=primary
     ,SECONDARY-ALLOCATION=secondary))
    [,SUPPORT=*PRIVATE-DISK(VOLUME=priv-vsn,DEVICE-TYPE=device,SPACE=...)]
```

SPACE

Specification of storage space (see section “Maximum size of UDS/SQL files in chapter "Files and realms of a UDS/SQL database").

PRIMARY-ALLOCATION=*primary*

Primary allocation;
the storage space requirement of the COSSD is directly dependent on the size of the compiled subschemas. In order to ensure that additional compiled subschemas can be added too the COSSD at a later stage if required, it is advisable to create the COSSD with a primary and secondary allocation (see below) of 100 2K units (BS2000 half pages) each.

SECONDARY-ALLOCATION=*secondary*

Secondary allocation;
depending on how the operating system has been generated, there may be limits on the extent to which COSSD can be dynamically extended.
A value of 100 2K units (BS2000 half pages) is recommended.

VOLUME
DEVICE-TYPE

If you store COSSD on private disk (PRIVATE VOLUME), the following must be specified:

priv-vsn

Volume serial number

device

Device type of private disk.

3.1.2 Setting up the user realms

Like the realms and files of the compiler database, the user realms have to be set up with the CREATE-FILE command.

If it is not possible to estimate the size of the user realms before the Schema DDL and SSL are compiled, setting up the user realms after BGSIA and before BFORMAT will suffice.
BGSIA prints out the ESTIMATE-REPORT listing the sizes of individual user realms as estimated by UDS/SQL (see “Description of the ESTIMATE-REPORT” in chapter “Setting up the Schema Information Area (SIA) with BGSIA”).

```
/CREATE-FILE FILE-NAME=dbname.realm-name
     ,SUPPORT=*PUBLIC-DISK(SPACE=*RELATIVE(PRIMARY-ALLOCATION=primary
     ,SECONDARY-ALLOCATION=secondary))
    [,SUPPORT=*PRIVATE-DISK(VOLUME=priv-vsn,DEVICE-TYPE=device,SPACE=...)]
```

dbname

Database name

realmname

Name of the user realm defined in the Schema DDL

SPACE

Specification of storage space (see section "Maximum size of UDS/SQL files" in chapter "Files and realms of a UDS/SQL database").

PRIMARY-ALLOCATION=*primary*

Primary allocation for user realm

SECONDARY-ALLOCATION=*secondary*

Secondary allocation;

secondary=0

Suppresses automatic realm extensibility and online realm extensibility.

secondary>0

Prerequisite for automatic realm extensibility and online realm extensibility.In the case of secondary>0, online realm extensibility is already activated for the realm concerned when the database is created.
When online realm extensibility is activated in this way, the default values NR-PAGES=64 and MIN-PAGES=0 are entered for NR-PAGES and MIN-PAGES (see Activating online extensibility when creating databases and DAL command ACT INCR in the "Database Operation" manual).

VOLUME
DEVICE-TYPE

If the user realm is stored on private disk (PRIVATE VOLUME), the following must be specified:

priv-vsn

Volume serial number

device

Device type of private disk

3.2 Generating the schema

In order to generate the schema, the following programs must execute one after the other:

BCREATE	formats the compiler database
DDL compiler	compiles the Schema DDL
SSL compiler	compiles the SSL
BGSIA	sets up the Schema Information Area (SIA)

3.2.1 Formatting the compiler database with BCREATE

The BCREATE utility routine is used to format the DBDIR and DBCOM realms of the compiler database. BCREATE assigns to the DBDIR and DBCOM an Act-key-0 page (security information, creation date, etc.) and at least one FPA page (free place administration).

When required, BCREATE automatically extends the realms of the database being processed. For details, please refer to the "Database Operation" manual, Automatic realm extension by means of utility routines).

At startup BCREATE takes into account any assigned UDS/SQL pubset declaration (see the "Database Operation" manual, Pubset declaration job variable). Faulty assignment leads to the program aborting.

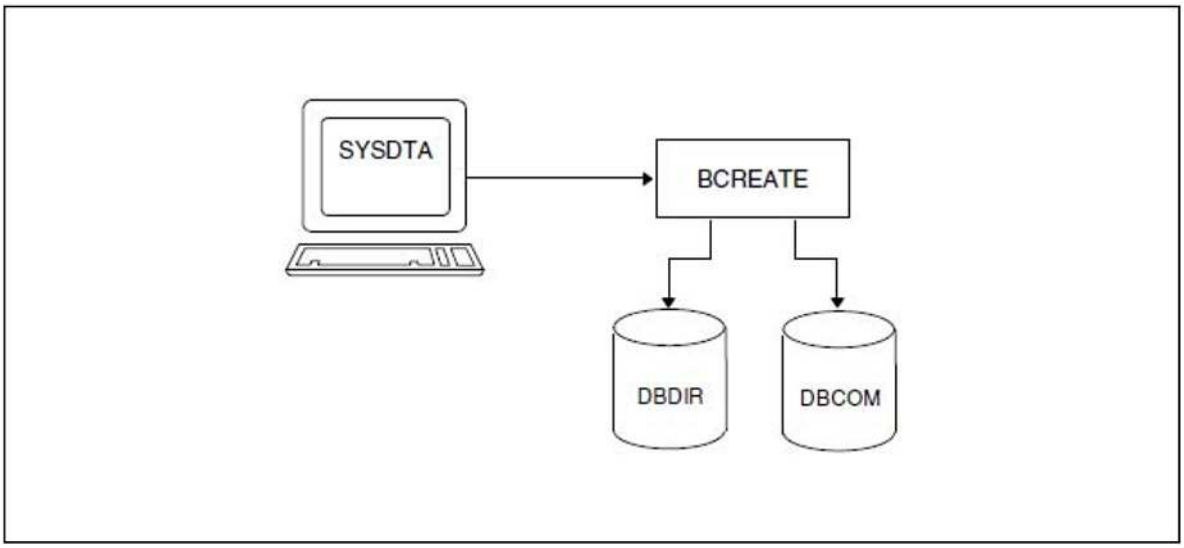


Figure 8: System environment for BCREATE

A UDS/SQL database can be optionally created with a 2-Kbyte, 4-Kbyte, or 8-Kbyte page format. In the 4-Kbyte and 8-Kbyte page formats, every database page is embedded in a page container (see the "Design and Definition" manual). This results in the following values for the database page length:

- 2048 bytes for databases with a 2-Kbyte page format
- 4000 bytes for databases with a 4-Kbyte page format (the page container is 4096 bytes)
- 8096 bytes for databases with a 8-Kbyte page format (the page container is 8192 bytes)

Statements for BCREATE

Statement	Default value	Meaning
DATABASE-PAGE-LENGTH IS { 2KB | 4KB | 8KB }	4KB	Optional; defines the page length for a new database: • 2KB The database is created with a 2-Kbyte page format • 4KB The database is created with a 4-Kbyte page format • 8KB The database is created with an 8-Kbyte page format
END	-	mandatory; terminates the statement input

Table 11: Statements for BCREATE

Command sequence to start BCREATE

It is assumed for the command sequence described here that UDS/SQL was installed with IMON (see section “START commands for the UDS/SQL programs”).

```
01 /CREATE-FILE FILE-NAME=dbname.DBDIR ...
02 /CREATE-FILE FILE-NAME=dbname.DBCOM ...
03 /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR
04 /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,
SCOPE=*TASK
05 /START-UDS-BCREATE
06 [DATABASE-PAGE-LENGTH IS {2/4/8}KB]
07 END
```

01/02 See section “Setting up DBDIR and DBCOM” in chapter “Setting up the compiler database”.

04 The specified version of BCREATE is selected.
It is generally recommended that you specify the version since it is possible for several UDS/SQL versions to be installed in parallel.

05 The UDS/SQL utility routine can also be started using the alias BCREATE.

06 The DATABASE-PAGE-LENGTH statement may be dropped only if the database is to be created with a 4-Kbyte page format.

07 The END statement is mandatory.

Example

```
/CREATE-FILE FILE-NAME=TRAVEL.DBDIR,SUPPORT=PUBLIC-DISK(SPACE= -
/      RELATIVE(PRIMARY-ALLOCATION=150,SECONDARY-ALLOCATION=50))
/CREATE-FILE FILE-NAME=TRAVEL.DBCOM,SUPPORT=PUBLIC-DISK(SPACE=RELATIVE -
/      (PRIMARY-ALLOCATION=550,SECONDARY-ALLOCATION=50))
/ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=TRAVEL.DBDIR
/SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,    VERSION=02.9C00
/START-UDS-BCREATE
***** START        BCREATE      (UDS/SQL  V2.9  1801  )     2019-01-29   09:26:53
 +++++ WARNING: 1917 BLOCKLENGTH SET TO 4KB
 * SCHEMAS AND SUBSCHEMAS WRITTEN TO DBDIR
 * VERSION-RECORDS WRITTEN TO DBDIR
 * DBCOM SUCCESSFULLY FORMATTED
 * DBDIR SUCCESSFULLY FORMATTED

 ***** DIAGNOSTIC SUMMARY OF BCREATE

 +++++         1 WARNINGS
              NO ERRORS
              NO SYSTEM-ERRORS

 ***** END OF DIAGNOSTIC SUMMARY
 ***** NR OF DATABASE ACCESSES  :           69
 ***** NORMAL END   BCREATE      (UDS/SQL  V2.9 1801  )     2019-01-29   09:26:53
```

3.2.2 Compiling the Schema DDL

The Schema DDL is compiled with the aid of the DDL compiler; it has to be assigned to the compiler as an input file.

On completing the compilation, the DDL compiler stores the schema description in the DBCOM (database compiler realm).

On the basis of this information, the subsequent BGSIA utility routine creates the SIA and stores it in the DBDIR (database directory).

Once an SSL description has been created, the compiled schema description in the DBCOM forms the basis for further processing by the SSL compiler and the BGSIA utility routine.

If no SSL description has been created, the schema description in the DBCOM is complete.

The DDL compiler also creates the COBOL subschema directory (COSSD), which stores information for the COBOL compiler. This information is needed for generation of the DB application programs. The actual contents of the COSSD are not generated until the Subschema DDL is compiled.

When required, the DDL compiler automatically extends the DBDIR and DBCOM of the database being processed or the DBTTs of the record types in the DBDIR and DBCOM. For details,
please refer to the "Database Operation" manual, Automatic realm extension by means of utility routines.

At startup the DDL compiler takes into account any assigned UDS/SQL pubset declaration (see the "Database Operation" manual, Pubset declaration job variable). Faulty assignment leads to the program aborting.

During execution the DDL compiler uses the linked-in DBH.

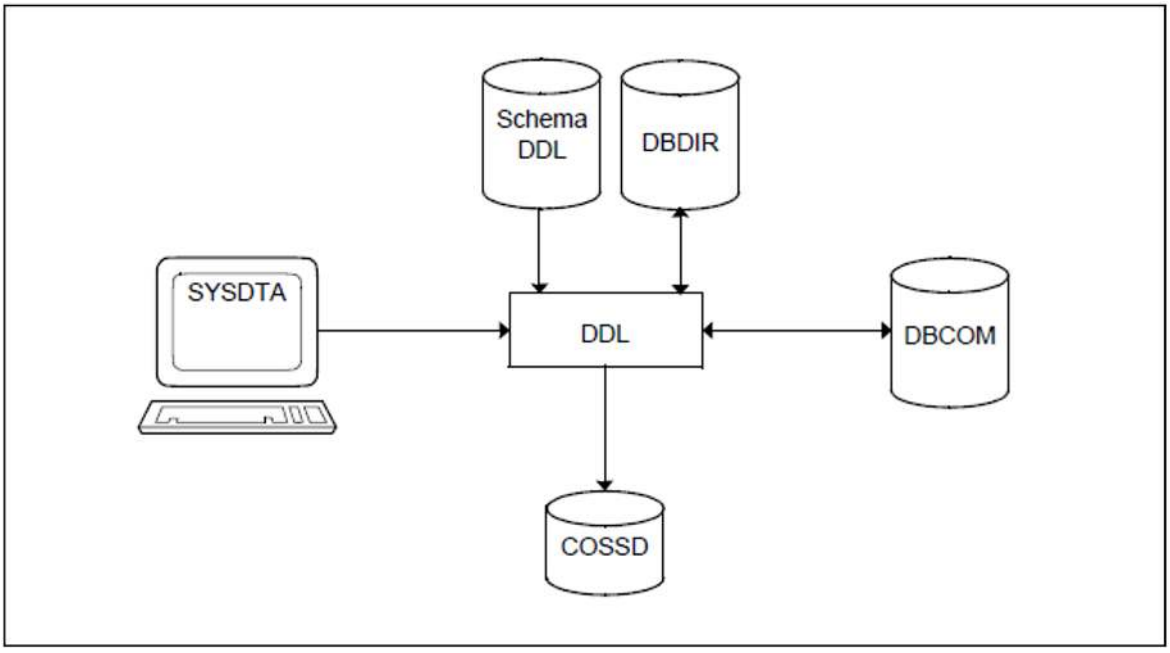


Figure 9: System environment for Schema DDL compilation

Compiler statements:

The DDL compiler is used to compile both the Schema DDL and the Subschema DDL; the SSL, on the other hand, is compiled by the SSL compiler.
The following table is a list of compiler statements for:

- the Schema DDL, marked with DDL
- the Subschema DDL, marked with SDDL
- the SSL, marked with SSL.

Statement	Compiler	Default value	Meaning
`PARLIST IS { YES | NO }`	DDL SDDL SSL	NO	optional; YES all statements are listed on SYSLST NO statements are not listed
`SORCLIST IS { YES | NO }`	DDL SDDL SSL	YES	optional; YES a listing is printed out on SYSLST, possibly containing error messages NO no listing is printed

`SOURCE IS`
`{ '`*`file-name`*`' |`
`'`*`lib`*`(`*`element`*`)' }`
	DDL
	SDDL
	SSL
		-	not required for inputs in interactive mode or if SYSDTA is assigned as the input file - in this case, it should be noted that all the statements, (at least END) must be entered first followed by the actual source.
			
			assigns the compiler the file containing the Schema DDL/Subschema DDL/SSL.
			Instead of '*file-name*' it is also possible to specify an element of a program library
			(see "Program libraries" in the "LMS (BS2000)" manual).
			
			lib: name of program library
			element: name of element
			
			SYSDTA is switched to the input file. It is reset to SYSCMD upon completion of the compiler run.
			The statements “SOURCE IS“ and “DELETE SCHEMA“ or “DELETE SUBSCHEMA“ may not be used within the same DDL compiler run.

SUBSCHEMA FORM IS OLD	SDDL	-	optional; this statement is only required for subschemas which are used by KDBS applications; it is permissible only in conjunction with the "SOURCE IS *filename*" statement and is ignored when compiling schemas. The "SUBSCHEMA FORM IS OLD" statement causes the transformed subschema and the check table (CHECK-TABLE) to be entered in the COSSD in an internal format which was the standard format up to and including UDS/SQL V1.2 ("old" form; all reference numbers are 1 byte long). A subschema can be compiled to a format compatible with UDS/SQL V1.2 only if the following conditions are satisfied: • No item of the subschema is of type DATABASE-KEY-LONG. • No item of the subschema is of the type NATIONAL. • No record type of the subschema is longer than 2020 bytes. • All record references and set numbers of the schema are <= 254. Otherwise, the DDL compiler aborts with syntax errors, and the subschema is not entered in the DBCOM and COSSD.

GENERATE-REC-REF IS { YES | NO }	SDDL	NO	optional; YES record references are generated In the IMPLICITLY-DEFINED-DATA-NAMES structure a field REC-REF PIC S9(4) BINARY is defined. For each record reference a condition name (level number 88) is assigned to this field, which matches the following pattern: REF-*record_name*. As the maximum length of a name is 30 characters, *record_name*, is truncated to 26 characters if necessary. In this case *record_name* must be unique in the first 26 characters. The record reference can be used in a COBOL program as follows: SET REF-*record_name* IN REC-REF TO TRUE. MOVE REC-REF TO *dbkey*. NO record references are not generated This statement is effective only for the generating of subschemas. When generating schemas the statement is ignored.
DELETE SCHEMA '*schema-name*'	DDL	-	optional; deletes the specified schema; useful after restructuring with BALTER if the DDL executes correctly and the SSL compilation reports errors actually attributable to the DDL *schema-name*: name of schema The “SOURCE IS“ and “DELETE SCHEMA“ statements must not be used within the same DDL compiler run.

DELETE [ONLY] SUBSCHEMA '*subschema-name*' {OF | :} SCHEMA '*schema-name*'	SDDL	-	optional; deletes the specified subschema. The subschema being compiled may have the same name as the subschema named in the DELETE statement, as it is deleted before the compiler run. ONLY if the parameter is omitted, a SOURCE statement must follow the DELETE statement. If the parameter is specified, any SOURCE statement is ignored. *subschema-name*: name of subschema *schema-name*: name of schema Both names have to be given in single quotes
DISPLAY IS { YES | NO }	DDL SDDL SSL	NO	optional; YES information held in DBCOM relating to record types, sets, etc. is output in unencoded form. NO values in DBCOM are not output
CREATE COSSD '*schema-name*'	DDL SDDL	-	Retroactive creation of COSSD. If this was forgotten during schema compilation or the DDL compiler was terminated abnormally owing to an error when the COSSD was being configured, this can be carried out in a separate run before the first subschema is compiled; *schema-name*: has to be given in single quotes. The COSSD has to be created with a CREATE-FILE command prior to the compiler run. N.B.: If the SOURCE IS ... parameter is specified at the same time, compilation will be suppressed.
COMPARE SUBSCHEMAS	SDDL	-	admissible only after restructuring with BALTER. The subschemas of the old schema are checked for compatibility with the new schema; for this purpose the DDL compiler reads the subschemas from the old COSSD after the BALTER run. If an old subschema is compatible with the new schema, it is entered in the new DBCOM and in the new COSSD.

DIAGNOSTIC IS { YES | NO }	SDDL	NO	only meaningful in conjunction with COMPARE YES diagnoses incompatibilities between old subschemas and the new schema and lists them in the form of error messages NO no error messages are output
QUOTE IS { SINGLE | DOUBLE }	DDL SDDL	DOUBLE	either; SINGLE literals in the Schema DDL/ Subschema DDL are given in single quotes DOUBLE literals in the Schema DDL/ Subschema DDL are given in double quotes
END	DDL SDDL SSL	-	mandatory; terminates statement input

Table 12: Compiler statements for the Schema DDL/Subschema DDL/SSL

Command sequence for compiling the Schema DDL

It is assumed for the command sequence described here that UDS/SQL was installed with IMON (see section “START commands for the UDS/SQL programs”).

```
01 /CREATE-FILE FILE-NAME=dbname.COSSD ...
02 /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR
03 /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,SCOPE=*TASK
04 /CREATE-FILE FILE-NAME=dbname.DBSTAT,SUPPRESS-ERRORS=*FILE-EXISTING
   /CREATE-FILE FILE-NAME=dbname.DBSTAT.SAVE,SUPPRESS-ERRORS=*FILE-EXISTING
05 /START-UDS-DDL
06 ddl-compiler-statements
07 END
```

01 See section “Setting up COSSD” in chapter "Setting up the compiler database".

03 The version-dependent module of the linked-in DBH of the relevant version is loaded dynamically (see section "Compiling, linking and loading UDS/SQL-TIAM application programs" in the "Application Programming" manual).

04 The DBH requires the DB status files.

If the database name contains more than 8 characters, only the first 8 characters of the database name may be specified for d*bname*.

05 The UDS/SQL utility routine can also be started with the alias DDL.

06 The individual statements can be entered in one line if they are separated by commas or blanks.

Example

```
/CREATE-FILE FILE-NAME=TRAVEL.COSSD,SUPPORT=PUBLIC-DISK(SPACE=RELATIVE -
/      (PRIMARY-ALLOCATION=30,SECONDARY-ALLOCATION=10))
/CREATE-FILE FILE-NAME=TRAVEL.DBSTAT,SUPPORT=PUBLIC-DISK(SPACE=RELATIVE -
/      (PRIMARY-ALLOCATION=24,SECONDARY-ALLOCATION=48))
/CREATE-FILE FILE-NAME=TRAVEL.DBSTAT.SAVE,SUPPORT=PUBLIC-DISK(SPACE=RELATIVE -
/      (PRIMARY-ALLOCATION=24,SECONDARY-ALLOCATION=48))
/ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=TRAVEL.DBDIR
***** START        DDLCOMP      (UDS/SQL  V2.9 1801  )     2019-01-29   09:26:53
*  DDLCOMP: INPUT SYSTEMPARAMETERS
SOURCE IS 'S.TRAVEL.DDL'
END
*  DDLCOMP: READ SCHEMA/SUBSCHEMA
%  UDS0215 UDS STARTING UDS/SQL V2.9 (LINKED-IN), DATE=2019-01-29 (ILL2038,09:26:53/4TE7)
%  UDS0746 UDS PUBSET DECLARATION (CURRENT) FOLLOWS (ILL1746,09:26:53/4TE7)
4TE7: UDS-PUBSET-JV:  :IUDS:$XXXXXXXX.PUBSDECL.PUBS
4TE7: PUBSETS:        IUDS
4TE7: DEFAULT PUBSET: IUDS
4TE7: ----------------------------------------
*  DDLCOMP: START SCHEMA-PHASE
*  DDLCOMP: CHECK SCHEMA RULES
*  DDLCOMP: CHECK DATA ALLOCATION
*  DDLCOMP: SEMANTIC TEST
*  DDLCOMP: CYCLUS TESTS
*  DDLCOMP: ERROR DIAGNOSTIC
*  DDLCOMP: NO ERRORS IN SCHEMA-PHASE
*  DDLCOMP: CREATE FILE COSSD
*  DDLCOMP: NO ERRORS DETECTED
%  UDS0758 NUMBER OF DML-STATEMENTS AND I/O COUNTERS PER DATABASE (ILL1758,09:26:54/4TE7)
4TE7: DATABASE NAME         DMLS    LOG READ  PHYS READ  LOG WRITE PHYS WRITE
4TE7: ------------------------------------------------------------------------
4TE7: TRAVEL                1394        4046         67       1760         45
%  UDS0213 UDS NORMAL SYSTEM TERMINATION WITH ************1394 DML-STATEMENTS 2019-01-29
(ILLY033,09:26:54/4TE7)

***** DIAGNOSTIC SUMMARY FOR DDL-SCHEMA TRAVEL-AGENCY

              NO ERRORS
+++++           8 WARNINGS

***** END OF DIAGNOSTIC SUMMARY
***** NORMAL END   DDLCOMP      (UDS/SQL  V2.9 1801 )     2019-01-29   09:26:54
```

3.2.3 Compiling the SSL

Compilation of the storage structure description is optional; if SSL is not used, UDS/SQL assumes default values. If an SSL description has been written, it can be compiled by the SSL compiler.

The SSL compiler analyzes the storage structure description and modifies the entries in the DBCOM to match the SSL.

When required, the SSL compiler automatically extends the DBDIR and DBCOM of the database being processed or the DBTTs of the record types in the DBDIR and DBCOM. For details,
please refer to the "Database Operation" manual, Automatic realm extension by means of utility routines.

At startup the SSL compiler takes into account any assigned UDS/SQL pubset declaration (see the "Database Operation" manual, Pubset declaration job variable). Faulty assignment leads to the program aborting.

During execution the SSL compiler uses the linked-in DBH.

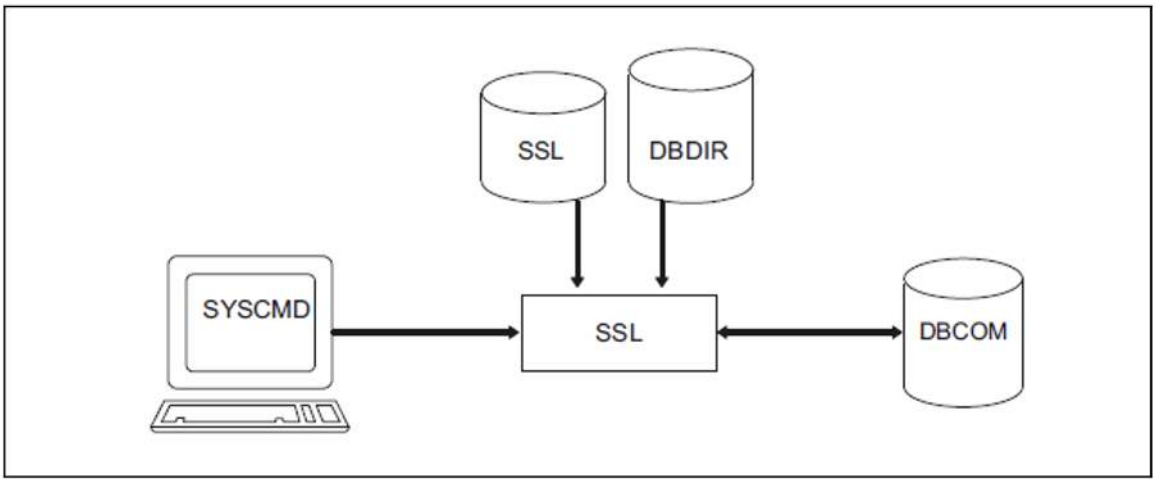


Figure 10: System environment for SSL compilation

SSL compiler statements

The statements for the SSL compiler are given in the Table of compiler statements(see table 12 of a chapter "Compiling the Schema DDL").

Command sequence for compiling the SSL

It is assumed for the command sequence described here that UDS/SQL was installed with IMON (see section “START commands for the UDS/SQL programs”).

```
01 /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR
02 /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,SCOPE=*TASK
03 /START-UDS-SSL
04 ssl-compiler-statements
05 END
```

02 The version-dependent module of the linked-in DBH of the relevant version is loaded dynamically (see the section entitled "Compiling, linking and loading UDS/SQL-TIAM appliction programs" in the "Application Programming" manual).

03 The UDS/SQL utility routine can also be started with the alias SSL.

04 The individual statements can be entered in one line if they are separated by commas or blanks.

Example

```
/ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=TRAVEL.DBDIR
/SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,    VERSION=02.9C00
/START-UDS-SSL
***** START        SSLCOMP      (UDS/SQL  V2.9  1801 )    2019-01-29   09:26:54
*  SSLCOMP: INPUT SYSTEMPARAMETERS
SORCLIST IS YES
SOURCE IS 'S.TRAVEL.SSL'
END
*  SSLCOMP: READ SSL-SCHEMA
%  UDS0215 UDS STARTING UDS/SQL V2.9 (LINKED-IN), DATE=2019-01-29 (ILL2038,09:26:54/4TE7)
%  UDS0746 UDS PUBSET DECLARATION (CURRENT) FOLLOWS (ILL1746,09:26:54/4TE7)
4TE7:UDS-PUBSET-JV: :IUDS:$XXXXXXXX.PUBSDECL.PUBS
4TE7:PUBSETS:        IUDS
4TE7:DEFAULT PUBSET: IUDS
4TE7: ----------------------------------------
*  SSLCOMP: START SSL-PHASE
*  SSLCOMP: CHECK SSL RULES
*  SSLCOMP: SEMANTIC TEST
*  SSLCOMP: ERROR DIAGNOSTIC
*  SSLCOMP: NO ERRORS DETECTED
%  UDS0758 NUMBER OF DML-STATEMENTS AND I/O COUNTERS PER DATABASE (ILL1758,09:26:55/4TE7)
4TE7: DATABASE NAME      DMLS   LOG READ  PHYS READ  LOG WRITE PHYS WRITE
4TE7: ------------------------------------------------------------------
4TE7: TRAVEL              354        563         62         57         27
%  UDS0213 UDS NORMAL SYSTEM TERMINATION WITH **********354 DML-STATEMENTS 2019-01-29
(ILLY033,09:26:55/4TE7)

***** DIAGNOSTIC SUMMARY FOR SSL - SCHEMA

              NO ERRORS
              NO WARNINGS

***** END OF DIAGNOSTIC SUMMARY
***** NORMAL END   SSLCOMP      (UDS/SQL  V2.9 1801 )     2019-01-29   09:26:55
```

3.2.4 Setting up the Schema Information Area (SIA) with BGSIA

The schema information area (SIA) has to be set up in the DBDIR (database directory) using the BGSIA utility routine.

For this purpose BGSIA requires the information which has been stored in the DBCOM during compilation of the Schema DDL and the SSL. The SIA then contains information in table form on the database schema and its storage structure.

The DBH and other utility routines need the SIA when data has to be stored, retrieved or updated in the user realms.

BGSIA assigns reference numbers to the names of the realms, record types, sets and keys, and it will print them out at the end of its run if requested to do so in a DISPLAY statement. This report is analogous to the one produced by the BPSIA utility routine (see the "Recovery, Information and Reorganization" manual, SIA PRINT REPORT).

BGSIA also generates the UDSHASH module and stores it in the EAM file. This module contains a table with the names of all the hash routines defined in the Schema DDL. After the BGSIA run the UDSHASH module has to be transferred with the attributes RMODE=ANY and AMODE=ANY to a module library with the name dbname. HASHLIB; this also applies if no hash routines are used.

If you have programmed your own hash routines (see the "Design and Definition" manual, Direct access), you must also enter these modules in the HASHLIB.

When required, BGSIA automatically extends the DBDIR of the database being processed. For details, please refer to the “Database Operation” manual, Automatic realm extension by means of utility routines.

At startup BGSIA takes into account any assigned UDS/SQL pubset declaration (see the “Database Operation” manual, Pubset declaration job variable). Faulty assignment leads to the program aborting.

During execution BGSIA requires the linked-in DBH.

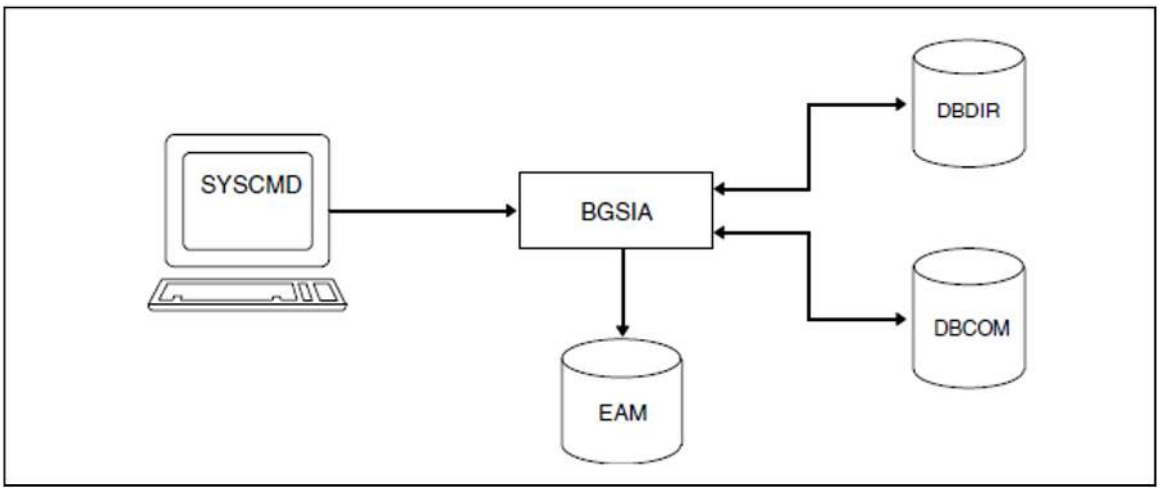


Figure 11: System environment for BGSIA

Statements for BGSIA

Statement	Default value	Meaning
GENERATE SCHEMA *schema-name*	-	Mandatory; Checks and generates the SIA. *schema-name* name of schema as specified in Schema DDL
RENAME { AREA | RECORD | SET } {*'name-old'* TO *'name-new'*} [, ...] .		May only be specified in the renaming cycle; changes the names of record types, sets and user realms *name-old*: name which is to be changed *name-new* new name The renaming of and changes to items in record types cannot be specified here.
DISPLAY [SCHEMA *schema-name*]	-	Optional; Prints the SIA generated by BGSIA *schema-name* name of schema as specified in GENERATE statement It is sufficient to specify DISPLAY.
END	-	mandatory; terminates statement input

Table 13: Statements for BGSIA

Command sequence for starting BGSIA

It is assumed for the command sequence described here that UDS/SQL was installed with IMON (see section “START commands for the UDS/SQL programs”).

```
01 /DELETE-SYSTEM-FILE FILE-NAME=*OMF
02 /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR
03 /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,SCOPE=*TASK
04 /START-UDS-BGSIA
05 bgsia-statements
06 END
```

03 The version-independent module of the linked-in DBH of the relevant version is loaded dynamically (see the section entitled "Compiling, linking and loading UDS/SQL-TIAM appliction programs" in the "Application Programming" manual).

04 The UDS/SQL utility routine can also be started with the alias BGSIA.

Entering the UDSHASH module in the HASHLIB

```
01 /START-LMS
02 //OPEN-LIB LIB=dbname.HASHLIB,MODE=*UPDATE(STATE=*NEW)
03 //ADD-ELEMENT FROM-FILE=*OMF,TO-ELEMENT=*LIBRARY-ELEMENT(TYPE=R)
04 //END
```

Example

```
/DELETE-SYSTEM-FILE FILE-NAME=*OMF
/ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=TRAVEL.DBDIR
/SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,    VERSION=02.9C00
/START-UDS-BGSIA
***** START        BGSIA        (UDS/SQL  V2.9  1801 )     2019-01-29   09:26:55
GENERATE TRAVEL-AGENCY
DISPLAY
END
%  UDS0215 UDS STARTING UDS/SQL V2.9(LINKED-IN), DATE=2019-01-29 (ILL2038,09:26:55/4TE7)
%  UDS0746 UDS PUBSET DECLARATION (CURRENT) FOLLOWS (ILL1746,09:26:55/4TE7)
4TE7: UDS-PUBSET-JV:  :4TE7:$XXXXXXXX.PUBSDECL.PUBS
4TE7: PUBSETS:        4TE7
4TE7: DEFAULT PUBSET: 4TE7
4TE7: ----------------------------------------
ESTIMATE-REPORT
********** FOR USER-REALM     3 NAME IS : TRAVEL-RLM
      A SIZE OF        147 BLOCKS WAS ESTIMATED
END OF ESTIMATE-REPORT
% UDS0758 NUMBER OF DML-STATEMENTS AND I/O COUNTERS PER DATABASE  (ILL1758,09:26:55/4TE7)
 4TE7: DATABASE NAME      DMLS   LOG READ  PHYS READ  LOG WRITE PHYS WRITE
 4TE7: ------------------------------------------------------------------
 4TE7: TRAVEL             1179       1310         61        276         35
%  UDS0213 UDS NORMAL SYSTEM TERMINATION WITH ************1179 DML-STATEMENTS 2019-01-29
(ILLY033,09:26:56/4TE7)

***** DIAGNOSTIC SUMMARY OF BGSIA
              NO WARNINGS
              NO ERRORS
              NO SYSTEM-ERRORS

***** END OF DIAGNOSTIC SUMMARY
***** NORMAL END   BGSIA       (UDS/SQL  V2.9 1801 )     2019-01-29   09:26:56
/ASSIGN-SYSDTA TO-FILE=*SYSCMD
/MODIFY-JOB-SWITCHES ON=(1,4)
/START-LMS
//MODIFY-LOGGING-PARAMETERS LOG=*MAX
//OPEN-LIBRARY LIB=TRAVEL.HASHLIB,MODE=*UPDATE
 LIBRARY IS CLEARED AND PREPARED
//ADD-ELEMENT FROM-FILE=*OMF,TO-ELEM=*LIB-ELEM(TYPE=R),WRITE-MODE=*ANY

INPUT  OMF
OUTPUT LIBRARY= :IUDS:$XXXXXXXX.TRAVEL.HASHLIB
        ADD UDSHASH AS (R)UDSHASH/@(0001)/2019-01-29
//SHOW-ELEM-ATTR

INPUT  LIBRARY= :IUDS:$XXXXXXXX.TRAVEL.HASHLIB
TYP NAME    VER (VAR#) DATE
(R) UDSHASH @   (0001) 2019-01-29
      1 (R)-ELEMENT(S) IN THIS TABLE OF CONTENTS
//END
```

SIA report

The SIA report printed by DISPLAY is almost identical to the report printed by the BPSIA utility routine. At some points it does not contain its definitive values, since certain values are entered at a later stage by BFORMAT. The report is described in detail in "SIA PRINT REPORT" in the "Recovery, Information and Reorganization" manual.

Description of the ESTIMATE-REPORT

In the BGSIA run listing the start message is followed by the ESTIMATE REPORT, which serves to estimate the size of the user realms.

This is necessary because, for example, when a user realm is not large enough, BFORMAT does not perform formatting if the realm concerned is not automatically extendable. For details, please see the “Database Operation” manual, Automatic realm extension by means of utility routines.

The ESTIMATE-REPORT always outputs the following information:

- realm number
- realm name
- realm size (in database pages)
  The value that is output for the realm size by ESTIMATE-REPORT must be interpreted differently, depending on the database page format (2-Kbyte, 4-Kbyte or 8-Kbyte), and cannot be directly used in the CREATE-FILE command, for example. In the case of the CREATE-FILE command, the value for “SPACE=“ is specified in units of 2K (BS2000 half pages), whereas the ESTIMATE-REPORT returns size specifications in units of a “database page”. Consequently, when converting to 2K units, the sizes given in the ESTIMATE-REPORT for a 4-Kbyte database must be multiplied by a factor of 2, and those for an 8-Kbyte database must be multiplied by a factor of 4.

ESTIMATE-REPORT outputs additional information in the following cases:

- The realm contains records for which the COMPRESSION clause was specified in the SSL.
- There are records with variable items in the realm.
- SEARCH key tables have been created with DUPLICATES ALLOWED and TYPE IS DATABASE-KEY-LIST.

Reference values needed for correction purposes are output in a correction table. These values are needed to make corrections if the percentage of space saved for the records is not as high as the default assumption for the ESTIMATE-REPORT.

The following table lists all options together with an explanation of the variables:

Entries in ESTIMATE-REPORT	Explanation of variables
`**** FOR USER-REALM` *`realm-ref`* `NAME IS:` *`realm-name`*	Realm number; Realm name
`A SIZE OF` *`size`* `BLOCKS WAS ESTIMATED`	Realm size in data pages; serves as a reference value for the amount of space needed for the specified user realm

`** THE RECORD rec-ref NAME IS record-name`	Record type number; name of the record type for which the COMPRESSION clause applies
`WITH * COMPRESSION * WAS CALCULATED WITH A PROFIT OF 50%`	The calculation of the realm size was based on the assumption that the saving due to COMPRESSION for the specified record type would be 50%.

`CORRECTION-TABLE:`
`0% 25% 75%`
`n+ n+ n-`
	Correction table (only for COMPRESSION);
	n indicates the number of data pages that must be added (+) to or subtracted from (-) the realm size *size* for savings of 0% / 25% / 75% .

`**** IN SET set-ref`
`NAME IS: set-name`
`FOR RECORD rec-ref`
	Set number;
	Set name;
	Record reference number;

`A SEARCH-KEY-TABLE TYPE *`
`DATABASE-KEY-LIST *`
`WAS CALCULATED WITH 50% DUPLICATES`
	Calculation of the size of the DATABASE-KEY-LIST was based on the assumption that 50% of the key values would be duplicates.

`CORRECTION-TABLE:`
`0% 75% 90%`
`n+ n- n-`
	Correction table;
	n indicates the number of data pages that must be added to or subtracted from the realm size *size* on the assumption that 0% / 75% / 90% of the key values are duplicates.

Table 14: Variables in the ESTIMATE-REPORT

The suggested realm sizes are intended as an aid to determining orders of magnitude. They may be imprecise for the following reasons:

- SSL population specifications (DBTT, RECORD POPULATION, SET POPULATION) are inaccurate.
- The saving cannot be predicted with 100% accuracy (e.g. for record types subject to the COMPRESSION clause or containing a variable data item).
- The number of key duplicates is not known for SEARCH key tables with DUPLICATES ALLOWED, TYPE IS DATABASE-KEY-LIST.
- The number of overflow pages cannot be predicted for LOCATION MODE IS CALC or for CALC SEARCH keys.
- The size of the unused storage space cannot be calculated due to mixed storage.
- In tables, the size may vary according to the order in which the data is stored.
- INCREASE is not taken into account.

The ESTIMATE-REPORT assumes maximum values for records. SINCE these values are unlikely to be achieved at the beginning, smaller user realms can be set up initially. These realms can be extended later by using the BREORG utility routine (see the "Recovery, Information and Reorganization" manual).

You can also configure realms in such a way that they can be extended online when required (see the “Database Operation” manual, The online realm extension process).

Example

```
ESTIMATE-REPORT
********** FOR USER-REALM     3 NAME IS : CUSTOMER-ORDER-RLM
     A SIZE OF         56 BLOCKS WAS ESTIMATED

********** FOR USER-REALM     4 NAME IS : PURCHASE-ORDER-RLM
     A SIZE OF         78 BLOCKS WAS ESTIMATED

********** FOR USER-REALM     5 NAME IS : CLOTHING
     A SIZE OF         67 BLOCKS WAS ESTIMATED
     *** THE RECORD     8 NAME IS : ART-DESCR
          WITH *COMPRESSION* WAS CALCULATED WITH A PROFIT OF 50%
          CORRECTION-TABLE :     0%       25%     75%
                                 17+      8+      8-

     **** IN SET    28 NAME IS : SYS_INSTALLMENT
          FOR RECORD     5 NAME IS : INSTALLMENT
          A SEARCH-KEY-TABLE TYPE *DATABASE-KEY-LIST*
          WAS CALCULATED WITH 50% DUPLICATES
          CORRECTION-TABLE:      0%       75%     90%
                                 0+       0-      0-

********** FOR USER-REALM     6 NAME IS : HOUSEHOLD-GOODS
     A SIZE OF         33 BLOCKS WAS ESTIMATED
     *** THE RECORD     8 NAME IS : ART-DESCR
          WITH *COMPRESSION* WAS CALCULATED WITH A PROFIT OF 50%
          CORRECTION-TABLE :     0%       25%     75%
                                 8+       4+      4-
.
.
.
     **** IN SET    12 NAME IS : ARTICLES-AVAILABLE
          FOR RECORD     9 NAME IS : ARTICLE
          A SEARCH-KEY-TABLE TYPE *DATABASE-KEY-LIST*
          WAS CALCULATED WITH 50% DUPLICATES
          CORRECTION-TABLE:      0%       75%     90%
                                 3+       0-      3-
.
.
.
********** FOR USER-REALM    11 NAME IS : ARTICLE-RLM
     A SIZE OF         85 BLOCKS WAS ESTIMATED
END OF ESTIMATE-REPORT
```

3.3 Formatting user realms with BFORMAT

Formatting of the user realms is carried out by the BFORMAT utility routine. BFORMAT

- adds information on the user realms to the SIA,
- stores in every realm an act-key-0 page, an act-key-N page and at least one FPA page and formats the DBTT and CALC pages,
- stores the note 'BFORMAT EXECUTED' in the DBDIR.

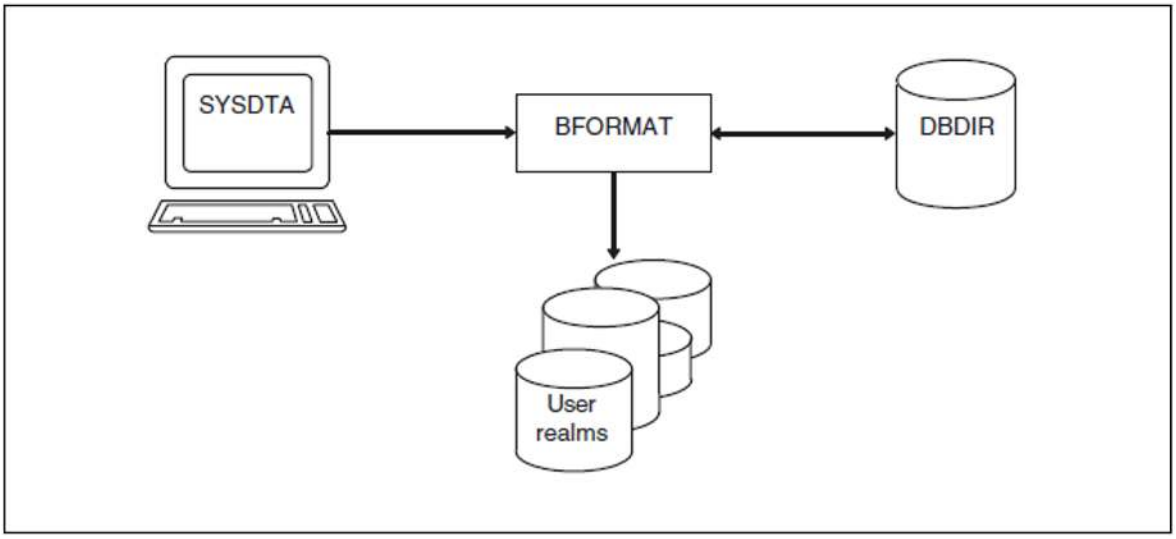


Figure 12: System environment for BFORMAT

BFORMAT must be called in the user ID under which the database is cataloged.

If the user realms have not yet been set up, they must be set up before the BFORMAT run (see section "Setting up the user realms").

When required, BFORMAT automatically extends the realms of the processed database (provided the realms are extendable). For details, please see the "Database Operation" manual, Automatic realm extension by means of utility routines.

At startup BFORMAT takes into account any assigned UDS/SQL pubset declaration (see the "Database Operation" manual, Pubset declaration job variable). Faulty assignment leads to the program aborting.

Statements for BFORMAT

The BFORMAT statement REALM identifies the realms which are to be formatted. Realms can be formatted in several BFORMAT runs, but each realm can only be formatted once.

Database creation cannot be continued until all realms have been formatted.

Statement	Default value	Meaning
REALM NAME IS {ALL [EXCEPT *realm-name1,..*] | *realm-name1,..* }	ALL	Optional; The specified realms are/are not to be formatted ALL all realms defined in the Schema DDL are to be formatted ALL EXCEPT *realm-name* exclusion list, i.e. all realms other than those specified are to be formatted *realm-name* specifies a user realm
END	-	Mandatory; Terminates statement input

Table 15: Statements for BFORMAT

> It is advisable to format realms one at a time. If a BFORMAT run formatting more than one realm aborts without the normal termination procedures owing to an operating system failure, realms that have already been successfully formatted will also be affected. The BFORMAT run will then have to be repeated for them.
> The BFORMAT run executes very quickly, since it only formats hash areas and FPA and DBTT pages.

Command sequence to start BFORMAT

It is assumed for the command sequence described here that UDS/SQL was installed with IMON (see section “START commands for the UDS/SQL programs”).

```
01 /CREATE-FILE FILE-NAME=dbname.realm-name ...
02 /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR
03 /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,
SCOPE=*TASK
04 /START-UDS-BFORMAT
05 [bformat-statement]
06 END
```

01 See section “Setting up the user realms”.

03 The specified version of BFORMAT is selected.
It is generally recommended that you specify the version since it is possible for several UDS/SQL versions to be installed in parallel.

04 The UDS/SQL utility routine can also be started with the alias BFORMAT.

05 If the REALM statement is omitted, all realms are formatted

Example

```
/CREATE-FILE FILE-NAME=TRAVEL.TRAVEL-RLM,SUPPORT=PUBLIC-DISK(SPACE=RELATIVE -
/    (PRIMARY-ALLOCATION=220,SECONDARY-ALLOCATION=60))
/ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=TRAVEL.DBDIR
/SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,    VERSION=02.9B00
/START-UDS-BFORMAT
***** START        BFORMAT      (UDS/SQL  V2.9 1801 )    2019-01-29   09:26:56
* VERSION RECORDS EXPANDED
REALM NAME IS ALL
END
* TRAVEL-RLM SUCCESSFULLY FORMATTED
* QUERY-RLM INITIALISED IN DBDIR
* BFORMAT-CONTROL-RECORD WRITTEN TO DBDIR
***** ALL REALMS FORMATTED

***** DIAGNOSTIC SUMMARY OF BFORMAT

              NO WARNINGS
              NO ERRORS
              NO SYSTEM-ERRORS

***** END OF DIAGNOSTIC SUMMARY
***** NR OF DATABASE ACCESSES  :          120
***** NORMAL END   BFORMAT      (UDS/SQL  V2.9 1801 )     2019-01-29   09:26:59
```

3.4 Generating the subschema

The following programs are used to generate a subschema:

DDL compiler	compiles the subschema description.
BGSSIA	generates the Subschema Information Area (SSIA).

3.4.1 Compiling the Subschema DDL

The Subschema DDL (SDDL) is compiled by the same DDL compiler as the Schema DDL. The SDDL compiler statements are described in table 12 of "Compiling the Schema DDL".

The Subschema DDL must be assigned as an input file to the DDL compiler. When compilation is finished, the compiled subschema description is stored in the DBCOM (Database Compiler Realm). Later BGSSIA uses this information to set up the SSIA in the DBDIR (Database Directory). In addition, the DDL compiler stores the transformed subschema derived from the compiled subschema description in the COSSD and creates a check table (CHECK-TABLE) for this subschema. This information is required by the COBOL compiler for the syntax and semantic checks of the DML statements.
When a subschema which is also to be used in KDBS applications is compiled, you must specify the DDL compiler statement “SUBSCHEMA FORM IS OLD” (see Table12 of "Compiling the Schema DDL"). The DDL compiler then creates the transformed subschema and the CHECK-TABLE in the format of UDS/SQL V1.2 (“old” format with 1-byte long reference numbers for record types and sets; see Table12 of "Compiling the Schema DDL").

The COSSD is also used as input for the BCALLSI utility routine (see "Additional measures for CALL DML programs with BCALLSI"). BCALLSI generates the SSITAB module, which makes the subschema information available to CALL DML programs.

After the subschemas are compiled, the DB administrator should save the database (see section “Saving the database”). This ensures that a consistent backup of the database exists.

> The subschema determines the RECORD AREA.
> The length of this record area is equal to the total lengths of all record types contained in the underlying subschema (aligned on a double-word boundary) and all implicitly defined data items, i.e. ALIAS items and AREA-IDs for distributed record types.
> The DDL compiler aborts the compilation of the subschema with an error as soon as the associated record area exceeds 65 535 bytes (or 61 328 bytes if SUBSCHEMA FORM IS OLD was specified).

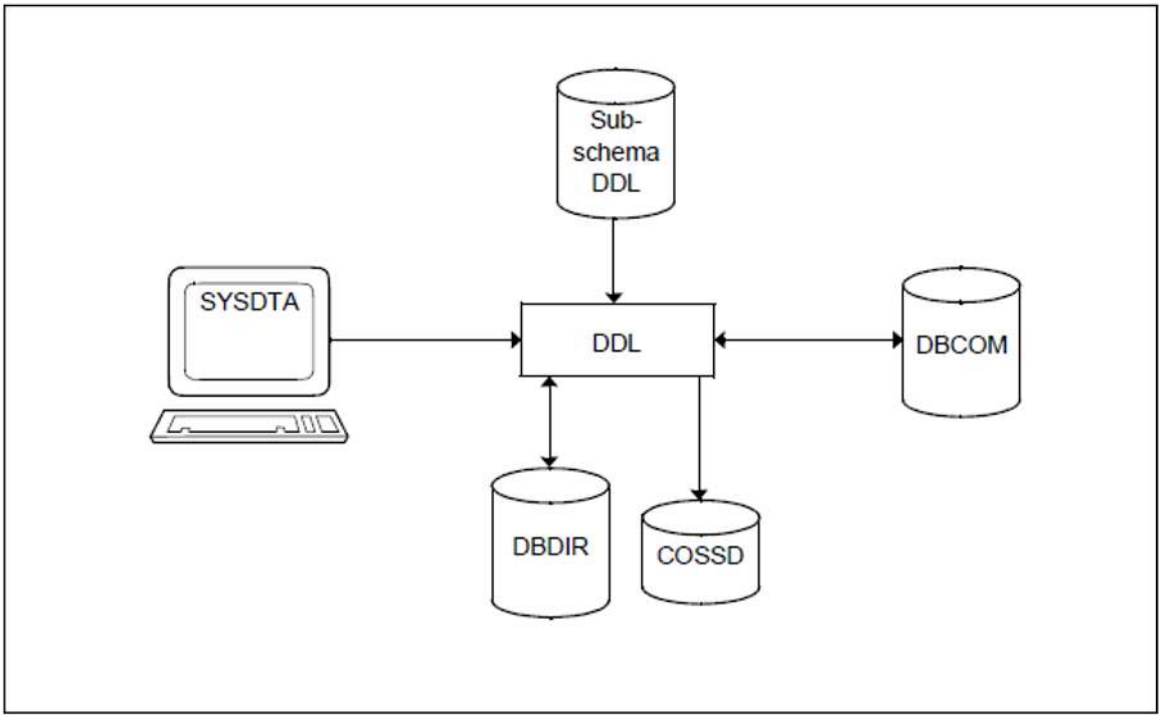


Figure 13: System environment for subschema compilation

Command sequence for compiling the subschema

```
01 /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR
02 /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,SCOPE=*TASK
03 /START-UDS-DDL
04 sddl-compiler-statements
05 END
```

02 The version-independent module of the linked-in DBH of the relevant version is loaded dynamically (see the section entitled "Compiling, linking and loading UDS/SQL-TIAM appliction programs" in the "Application Programming" manual).

03 The UDS/SQL utility routine can also be started with the alias DDL.

04 The individual statements can be entered in one line if they are separated by commas or blanks.

Example

```
/ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=TRAVEL.DBDIR
/SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,    VERSION=02.9C00
/START-UDS-DDL
***** START        DDLCOMP      (UDS/SQL  V2.9  1801 )    2019-01-29   09:26:59
*  DDLCOMP: INPUT SYSTEMPARAMETERS
SOURCE IS 'S.TRAVEL.SUBDDL'
END
*  DDLCOMP: READ SCHEMA/SUBSCHEMA
%  UDS0215 UDS STARTING UDS/SQL V2.9 (LINKED-IN), DATE=2019-01-29 (ILL2038,09:26:59/4TE7)
%  UDS0746 UDS PUBSET DECLARATION (CURRENT) FOLLOWS (ILL1746,09:26:59/4TE7)
4TE7: UDS-PUBSET-JV:  :IUDS:$XXXXXXXX.PUBSDECL.PUBS
4TE7: PUBSETS:        IUDS
4TE7: DEFAULT PUBSET: IUDS
4TE7: ----------------------------------------
*  DDLCOMP: START SUBSCHEMA-PHASE
*  DDLCOMP: CHECK SUBSCHEMA RULES
*  DDLCOMP: CHECK DATA ALLOCATION
*  DDLCOMP: SUBCOPY
*  DDLCOMP: ERROR DIAGNOSTIC
*  DDLCOMP: NO ERRORS IN SUBSCHEMA-PHASE
*  DDLCOMP: WRITE SUBSCHEMA ON COSSD
*  DDLCOMP: NO ERRORS DETECTED
%  UDS0758 NUMBER OF DML-STATEMENTS AND I/O COUNTERS PER DATABASE (ILL1758,09:26:59/4TE7)
 4TE7: DATABASE NAME      DMLS   LOG READ  PHYS READ  LOG WRITE PHYS WRITE
 4TE7: ------------------------------------------------------------------
 4TE7: TRAVEL             3011       5388         77       1249         57
%  UDS0213 UDS NORMAL SYSTEM TERMINATION WITH ************3011 DML-STATEMENTS 2019-01-29
(ILLY033,09:27:00/4TE7)

***** DIAGNOSTIC SUMMARY FOR DDL-SUBSCHEMA

              NO ERRORS
              NO WARNINGS

***** END OF DIAGNOSTIC SUMMARY
***** NORMAL END   DDLCOMP      (UDS/SQL  V2.9  1801 )     2019-01-29   09:27:00
```

3.4.2 Generating the Subschema Information Area (SSIA) with BGSSIA

The compiled subschema description is available in DBCOM as the end product of compilation by the DDL compiler. The BGSSIA utility routine requires this description in order to generate the Subschema Information Area (SSIA).

The SSIA is stored in the DBDIR as a record of the internal record type SSIA-RECORD.The SSIA contains subschema information needed by the DBH so that it can access the database. The information can be printed out by means of the DISPLAY statement in BGSSIA or by means of the BPSIA utility routine (see "SSIA PRINT REPORT" in the "Recovery, Information and Reorganization" manual).

When required, BGSSIA automatically extends the DBDIR and DBCOM of the database being processed or the DBTTs of the record types in the DBCOM. For details, please refer to the "Database Operation" manual, Automatic realm extension by means of utility routines.

At startup BGSSIA takes into account any assigned UDS/SQL pubset declaration (see the "Database Operation" manual, Pubset declaration job variable). Faulty assignment leads to the program aborting.

During execution BGSSIA works with the linked-in DBH.

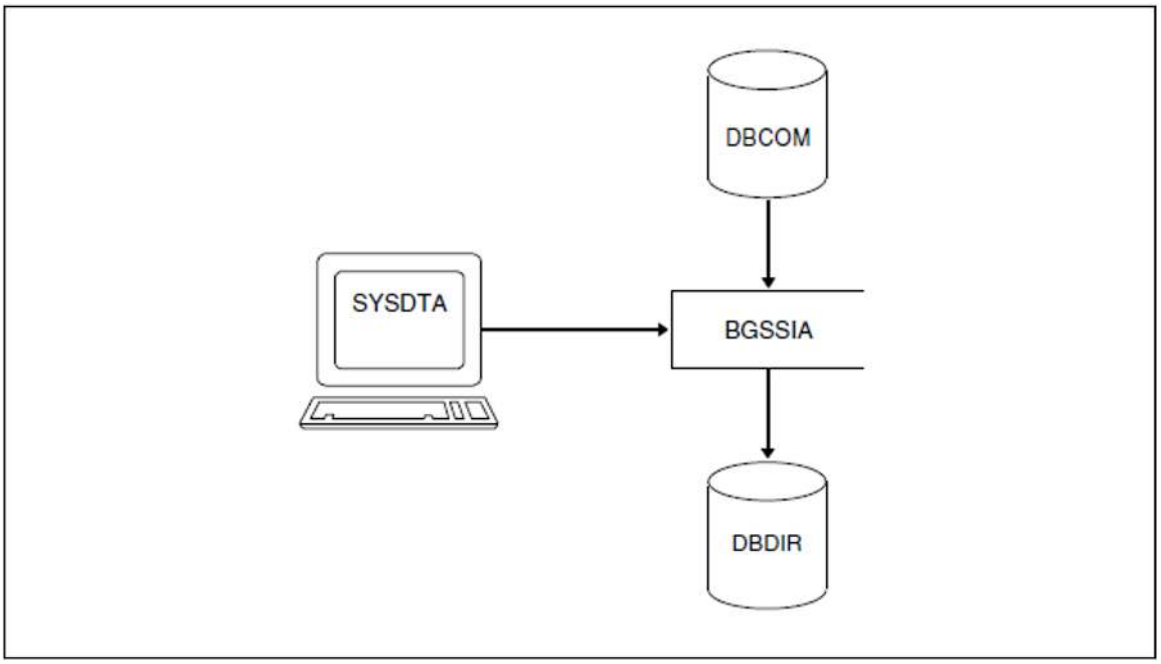


Figure 14: System environment for BGSSIA

Statements for BGSSIA

Statement	Default value	Meaning
GENERATE SUBSCHEMA *subschema-name* OF SCHEMA *schema-name*	-	Optional; subschema-name: name of the subschema schema-name: name of the schema Checks whether an SSIA is available for a specific subschema and generates an SSIA with information on realms, record types and sets lists of individual items lists of all names contained in the subschema.
DELETE SUBSCHEMA *subschema-name* OF SCHEMA *schema-name*	-	Optional; Deletes a previously generated SSIA from the DBDIR.
REGENERATE SUBSCHEMA *subschema-name* OF SCHEMA *schema-name*	-	Optional; Deletes the old SSIA and generates a new SSIA (combines DELETE and GENERATE functions). Suitable for correction of a subschema.
DISPLAY[SUBSCHEMA *subschema-name* OF SCHEMA *schema-name*]	-	Optional; Can only be used in conjunction with the GENERATE or REGENERATE statement. To print out SSIA, DISPLAY by itself is sufficient.
END	-	Mandatory; Terminates entry of statements.

Table 16: Statements for BGSSIA

Command sequence to start BGSSIA

It is assumed for the command sequence described here that UDS/SQL was installed with IMON (see section “START commands for the UDS/SQL programs”).

```
01 /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR
02 /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,SCOPE=*TASK
03 /START-UDS-BGSSIA
04 bgssia-statements
05 END
```

02 The version-independent module of the linked-in DBH of the relevant version is loaded dynamically (see the section entitled "Compiling, linking and loading UDS/SQL-TIAM appliction programs" in the "Application Programming" manual).

03 The UDS/SQL utility routine can also be started with the alias BGSSIA.

Example

```
/ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=TRAVEL.DBDIR
/SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,    VERSION=02.9C00
/START-UDS-BGSSIA
***** START        BGSSIA       (UDS/SQL  V2.9  1801 )    2019-01-29   09:27:00
GENERATE SUBSCHEMA RESERVATION OF SCHEMA TRAVEL-AGENCY
DISPLAY
END
%  UDS0215 UDS STARTING UDS/SQL V2.9 (LINKED-IN), DATE=2019-01-29 (ILL2038,09:27:00/4TE7)
%  UDS0746 UDS PUBSET DECLARATION (CURRENT) FOLLOWS (ILL1746,09:27:00/4TE7)
4TE7: UDS-PUBSET-JV:  :IUDS:$XXXXXXXX.PUBSDECL.PUBS
4TE7: PUBSETS:        IUDS
4TE7: DEFAULT PUBSET: IUDS
4TE7: ----------------------------------------
*** SSIA GENERATION NORMALLY ENDED.
*GENERATION OF ITEM-TABLE AND NAME-TABLE STARTED.
*GENERATION OF ITEM-TABLE AND NAME-TABLE FINISHED.
%  UDS0758 NUMBER OF DML-STATEMENTS AND I/O COUNTERS PER DATABASE  (ILL1758,09:27:00/4TE7)
 4TE7: DATABASE NAME      DMLS   LOG READ  PHYS READ  LOG WRITE PHYS WRITE
 4TE7: ------------------------------------------------------------------
 4TE7: TRAVEL             1800       2529         76        296         27
%  UDS0213 UDS NORMAL SYSTEM TERMINATION WITH ************1800 DML-STATEMENTS 2019-01-29
(ILLY033,09:27:00/4TE7)

***** DIAGNOSTIC SUMMARY OF BGSSIA

              NO WARNINGS
              NO ERRORS
              NO SYSTEM-ERRORS

***** END OF DIAGNOSTIC SUMMARY
***** NORMAL END   BGSSIA       (UDS/SQL  V2.9  1801 )     2019-01-29   09:27:00
```

3.5 Additional measures for CALL DML programs with BCALLSI

The BCALLSI utility routine must be executed if you have CALL-DML programs or work with DMLTEST.

BCALLSI generates the SSITAB module (SUBSCHEMA INFORMATION TABLE) with the subschema information needed by a CALL-DML program at program runtime.

At startup BCALLSI takes into account any assigned UDS/SQL pubset declaration (see the "Database Operation" manual, Pubset declaration job variable). Faulty assignment leads to the program aborting.

Providing subschema information

In order to execute DML statements, the DBH requires information on the subschema being used. The following information is available in the COSSD as stored by the DDL compiler:

- the transformed subschema
- the so-called CHECK TABLE

This information is gathered for the DBH in a number of different ways.

- In the case of COBOL DML programs, the COBOL compiler needs the subschema information when compiling the application program.
- For CALL DML programs, the subschema information is required at program runtime. Since access to the COSSD would be too time-consuming at runtime, you must generate the SSITAB module with BCALLSI beforehand. BCALLSI uses the information of the COSSD for this purpose. At program runtime, the SSITAB module is loaded from the module library by the CALL DML connection module. Thus, in the case of CALL DML programs, a BCALLSI run must be added between compilation of the Subschema DDL and the program run.

Editing special subschemas in the “old” format

In addition to the standard format in which the transformed subschema and the associated check table exist, the "old" format up to and including UDS/SQL V1.2 with 1-byte long reference numbers for record types and sets (see Table12 of "Compiling the Schema DDL") is still accepted by BCALLSI. The "old" format is required for subschemas which are processed in KDBS applications. A COSSD can contain transformed subschemas in both the standard format and in the "old" format. From a transformed subschema in the "old" format BCALLSI generates an SSITAB module in UDS/SQL V1.2 format, which is still supported by the current CALL-DML converter.

BCALLSI functions

BCALLSI can access a COSSD of UDS/SQL > V1.2 as well as a COSSD of UDS/SQL V1.2 or UDS/SQL V1.1.

BCALLSI performs the following functions:

- Compilation of the transformed subschema in realm, set, record and item tables
- Printing out the transformed subschema
- Checking the realm, set, record and item names for unique identification by means of the first eight or thirty characters. If the names are not unique, a warning is appended to the printout of the transformed subschema.
- Copying the check table from the COSSD in order to complete the SSITAB.
- Outputting the SSITAB module to the EAM file under the name *subschema*##, where *subschema* comprises the first six characters of the full subschema name.

The SSITAB module generated must then be entered in a module library using the BS2000 utility routine LMS. The name of the library is freely selectable. The DBH gives first priority to loading SSITAB modules from a library assigned with the link name $UDSSSI. If the SSITAB modules are stored in more than one library, e.g. in a separate library for each database, other libraries can be assigned with the link names BLSLIB00 to BLSLIB99 (see the section “DBH start commands” in the "Database Operation" manual and the section "Compiling, linking and loading UDS/SQL-TIAM appliction programs" in the "Application Programming" manual).

> The first six characters of the subschema names must ensure unique identification, since the name of the SSITAB module is formed from the first six characters plus ’##’.

System environment for BCALLSI

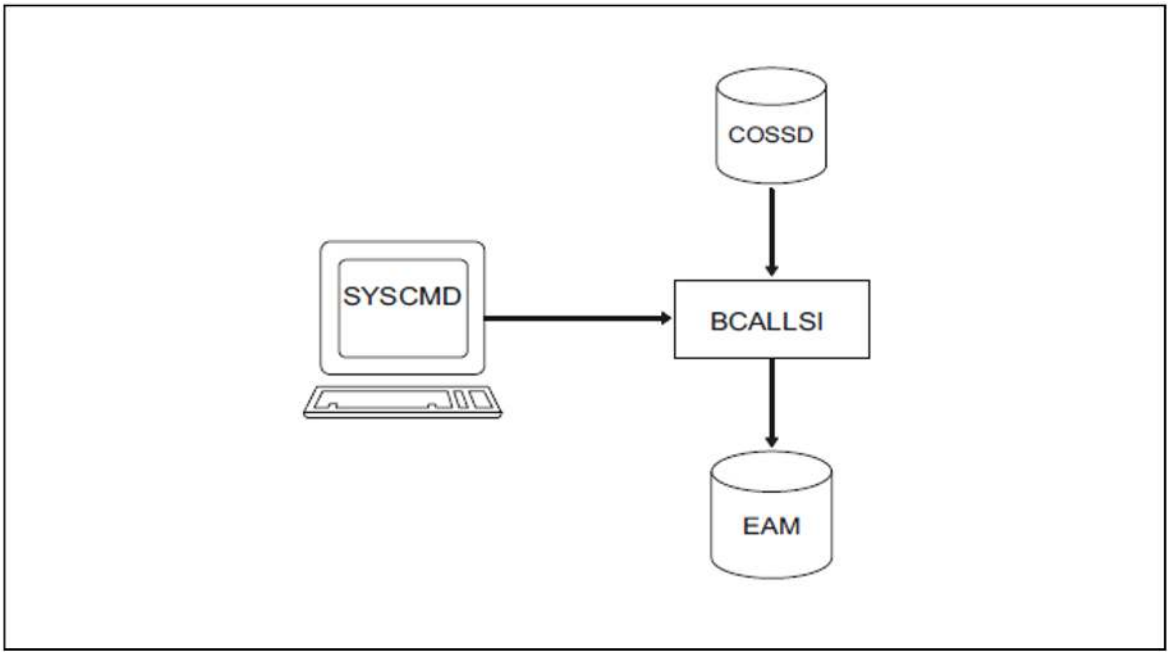


Figure 15: System environment of BCALLSI

Statements for BCALLSI

Statement	Default value	Meaning
{SCHEMA | S}=*schema-name*, {SUBSCHEMA | SS}=*subschema-name*	-	Mandatory; Assigns the name of the schema and subschema to BCALLSI: *schema-name* Name of the schema assigned in the Schema DDL. *subschema-name* Name of the subschema assigned in the Subschema DDL.
[{,MESSAGE | ,M}= {*ALL|N[O-AMBIGUITY-8]}]	*ALL	*ALL All cases of ambiguity, including those in the first 8 characters, are output individually to SYSLST. NO-AMBIGUITY-8 Cases of ambiguity in the first 8 characters of a name, are not output individually to SYSLST.

Table 17: Statements for BCALLSI

Command sequence for starting BCALLSI

It is assumed for the command sequence described here that UDS/SQL was installed with IMON (see section “START commands for the UDS/SQL programs”).

```
01 /DELETE-SYSTEM-FILE FILE-NAME=*OMF
02 /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR
03 /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,SCOPE=*TASK
04 /START-UDS-BCALLSI
05 bcallsi-statement
```

03 The specified version of BCALLSI is selected.
It is generally recommended that you specify the version since it is possible for several UDS/SQL versions to be installed in parallel.

04 The UDS/SQL utility routine can also be started with the alias BCALLSI.

05 There is no END statement for BCALLSI!

Entering the SSITAB module in the module library

```
01 /START-LMS
02 //OPEN-LIB LIB=modlib,MODE=*UPDATE
03 //ADD-ELEMENT FROM-FILE=*OMF,TO-ELEMENT=*LIBRARY-ELEMENT(TYPE=R)
04 //END
```

Example

```
/DELETE-SYSTEM-FILE FILE-NAME=*OMF
/ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=SHIPPING.DBDIR
/SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,    VERSION=02.9C00
/START-UDS-BCALLSI
***** START        BCALLSI      (UDS/SQL  V2.9  1801 )    2019-01-29   09:27:25
SCHEMA=MAIL-ORDERS,SUBSCHEMA=ADMIN,MESSAGE=*ALL
SSITAB ADMIN## WRITTEN TO EAM-OMF

***** DIAGNOSTIC SUMMARY OF BCALLSI

              NO WARNINGS
              NO ERRORS
              NO SYSTEM-ERRORS

***** END OF DIAGNOSTIC SUMMARY
***** NR OF DATABASE ACCESSES  :            0
***** NORMAL END   BCALLSI      (UDS/SQL  V2.9  1801 )    2019-01-29   09:27:25
/MODIFY-JOB-SWITCHES ON=(1,4)
/START-LMS
//MOD-LOG-PAR LOG=*MAX
//OPEN-LIB LIB=LMS.SSITAB,MODE=*UPD(STATE=*ANY)
//ADD-ELEMENT FROM-FILE=*OMF,TO-ELEM=*LIB(TYPE=R),WRITE-MODE=*ANY
INPUT  OMF
OUTPUT LIBRARY= :IUDS:$XXXXXXXX.LMS.SSITAB
        ADD ADMIN## AS (R)ADMIN##/@(0001)/2019-01-29

//SHOW-ELEM-ATTR ELEM=*LIB-ELEM()

INPUT  LIBRARY= :IUDS:$XXXXXXXX.LMS.SSITAB
TYP NAME     VER (VAR#) DATE         NAME     VER (VAR#) DATE
(R) ADMIN##  @   (0002) 2019-01-29 MANAGE## @   (0002) 2019-01-29
(R) UDSHASH  @   (0002) 2019-01-29 VERWAL## @   (0002) 2019-01-28
      4 (R)-ELEMENT(S) IN THIS TABLE OF CONTENTS

//END
PRINTOUT:
.
.
.
             SCHEMANAME       :  MAIL-ORDERS
             SUBSCHEMANAME    :  ADMIN
             MODUL-ENTRY      :  ADMIN##
             LENGTH OF MODUL  :         6408 BYTES
             SSITAB-VERSION   :            2
```

4 Specifying access authorizations (ONLINE-PRIVACY, BPRIVACY)

In UDS/SQL the ONLINE-PRIVACY and BPRIVACY utility routines are available for specifying who (which user groups) are allowed to access a database in what manner (access rights).

The scope of functions and the syntax and semantics for assigning rights are identical for ONLINE-PRIVACY and BPRIVACY.

- You use ONLINE-PRIVACY online, i.e. while the database is activated for an independent DBH session. You can also query or change the access authorizations for a database while the database is running.
- You use BPRIVACY offline, e.g. when creating a database, to specify the access authorizations for the database.

ONLINE-PRIVACY and BPRIVACY offer the following functions:

- define user groups with or without access rights
- delete user groups
- grant or revoke user groups access rights
- output information on user groups

If UDS/SQL is used with openUTM, the access protection of UDS/SQL can work in conjunction with that of openUTM (see the openUTM manual "Generating Applications").

4.1 User groups

User groups are groups of users with access to the database.

The name of a user group generally has the following structure (for more information see table 18 of a section "Checking access rights"):

host + *appl* + *grp*

host	Name of the host computer
appl	Name of the UDS/SQL/openUTM application or “_“
grp	BS2000 identification or, in the case of a UDS/SQL-openUTM application, the KSET name associated with the openUTM user ID.

UDS/SQL uses underscores “_“ internally to pad each of the three components to eight characters.

openUTM or UDS/SQL ascertains the information required. Further entries on access rights are therefore generally not required in the application programs (e.g. PRIVACY-RECORD in the SUB-SCHEMA SECTION, PERMIT in SQL programs). UDS/SQL ignores specifications in existing programs.

Before a user group can execute database calls, you must define a name for it and assign it access rights with ONLINE-PRIVACY or with BPRIVACY.

You will find further information in the "Database Operation" manual, which also contains an example of how to define UDS/SQL user groups and assign access rights.

4.2 Access rights

Access rights allow user groups read access (RETRIEVAL) or read and write access (UPDATE) to database objects, for example. Access rights can be granted and revoked.

A user group can be assigned read (RETRIEVAL) or read and write (UPDATE) access rights to the following database objects.

Access rights in CODASYL applications to:

- realms
- record types (RECORDs)
- sets

Access rights in SQL applications to:

- base tables
- foreign keys

RETRIEVAL, UPDATE and ALL access can be:

- granted using
  the ADD-USER-GROUP and GRANT-ACCESS statements
- revoked using
  the REVOKE-ACCESS statement
- qualified using
  the GRANT-ACCESS and REVOKE-ACCESS statements.

When access rights are assigned using the GRANT-ACCESS statement, any existing access rights for a database object are retained and the newly defined ones are added. In the same way, when the REVOKE-ACCESS statement is used, only the access rights specified with this statement are withdrawn; the user group retains any other access rights it has.

4.3 Checking access rights

UDS/SQL checks access rights only by means of the user group names.

The user group name must be defined with ONLINE-PRIVACY or BPRIVACY and the access rights must have been assigned before the users in the group can execute database calls.

If the DBH cannot identify the user group, the application program is supplied with a status code or the IQS session is terminated.

The table below indicates how user group names are structured, which configuration is checked with which group name and how to define the user groups in the ADD-USER-GROUP statement. The terms "local" and "remote" are meant in relation to the location of the database.

Configuration	**Value**			**Definition in the ADD-USER-GROUP statement**
	host	**appl**	**grp**	
openUTM Appl. w/o KSET	*host*	*appl*	-	*KSET-FORMAT(HOST=*host*,APPLICATION= *appl*,KSET=*NONE)
openUTM Appl. w/o KSET	*host*	*appl*	*kset*	*KSET-FORMAT(HOST=*host*,APPLICATION=*appl*,KSET=*kset*)
TIAM	*host*	' _'	*id*	*FREE-FORMAT(HOST=*host*,USER-ID=*id*)
linked-in	*host*	' _'	*id*	*FREE-FORMAT(HOST=*host*,USER-ID=*id*)

Table 18: Structure of user group names

Key

host Name of the host computer on which the UDS/SQL-openUTM application or the UDS/SQL application program runs.
Here you must specify the name of your processor from the standpoint of DCAM. If no DCAM is available in the TIAM case, you specify HOST=LOCAL.

appl Name of the openUTM application

kset KSET name associated with the corresponding openUTM user ID

id BS2000 user ID

In application programs (COBOL DML, CALL DML, SQL) and for IQS, "old" PRIVACY user specifications (< UDS/SQL V1.2) are still made in some instances or may be required (IQS):

- "Old" specifications in the so-called PRIVACY RECORD (< UDS/SQ Version 1.2) in application programs (COBOL DML, CALL DML, SQL) or for IQS are ignored by UDS/SQL.
- The PRIVACY specifications for IQS of any version <= 3.1 must not be empty, but are otherwise arbitrary.

Access rights are checked by means of the user group name, which comprises the name of the host computer and the runtime identification of the TIAM application or the name of the openUTM application (with or without the KSET specification).

The KSET name may be omitted if no KSET name was defined in the corresponding openUTM application. If no openUTM users are defined, a defined KSET name of a logical terminal (LTERM) is used for checking access rights.openUTM uses predefined KSET names, which you can display with KDCINF KSET. Access rights for the database must also be defined for these predefined KSET names. In distributed transaction processing with openUTM-D, the KSET name from the associated LPAP entry must be used (see the openUTM manual "Generating Applications").

4.4 System environment for ONLINE-PRIVACY

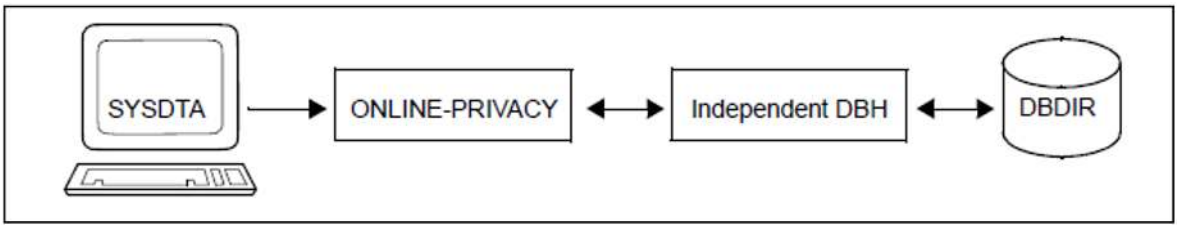


Figure 16: System environment for ONLINE-PRIVACY

The ONLINE-PRIVACY utility routine runs as a UDS/SQL TIAM application program in an independent DBH session.

You can start ONLINE-PRIVACY at any time while the database is operating when the following requirements are fulfilled:

- ONLINE-PRIVACY is called under the user ID under which the database is cataloged.
- The database is activated as an independent DBH session.
- The database is not attached to the session in the SHARED-RETRIEVAL mode.

With ONLINE-PRIVACY you can grant new user groups access to an existing database application and change the access rights of existing user groups as many times as needed.You need to update the access rights after making changes to a schema (for example, after adding new record types or new base tables).

Activating online extensibility of the DBDIR using ACT INCR enables you to ensure that the DBDIR can, when required, be extended by the DBH. However, no online extension of the DBTTs of the DBDIR's record types takes place.

> ¡ Access to a database in a remote configuration via UDS-D is not possible with ONLINE-PRIVACY.

Access locks

An access lock (ACCESS LOCK) or access restriction (ACCESS RETRIEVAL) set with the DAL command ACCESS on the database or realm level has no effect on the ONLINE-PRIVACY utility routine.

ONLINE-PRIVACY therefore provides you with the ability to change the access rights for a database for which only read transactions are permitted. The database must be in the EXCLUSIVE-UPDATE attach mode and be locked for other transactions with the DAL command ACCESS RETRIEVAL,DB=dbname.

Effect of changing rights

Changes to the access rights of a database that you have made using ONLINE-PRIVACY affect all processing chains started after terminating the ONLINE-PRIVACY run (FINISH statement of the ONLINE-PRIVACY transaction).

Processing chains started before the end of the ONLINE-PRIVACY run continue to work with the old privacy information.

Effect on the communication pool (CUP)

The ONLINE-PRIVACY utility routine connects to the communication pool (CUP) just like other UDS/SQL TIAM application programs. The space required by ONLINE-PRIVACY within the communication pool is about the same as the size of the SIA of the database to be processed.

Proceed as follows to ensure that the size of the communication pool is large enough to use ONLINE-PRIVACY:

- Determine the size of the SIA with the BPSIA utility routine.
- Take the size of the SIA into account as an additional space requirement for ONLINE-PRIVACY when calculating the minimum size of the communication pool.
- Set the value of the DBH load parameter PP CUP-SIZE accordingly.

4.5 System environment for BPRIVACY

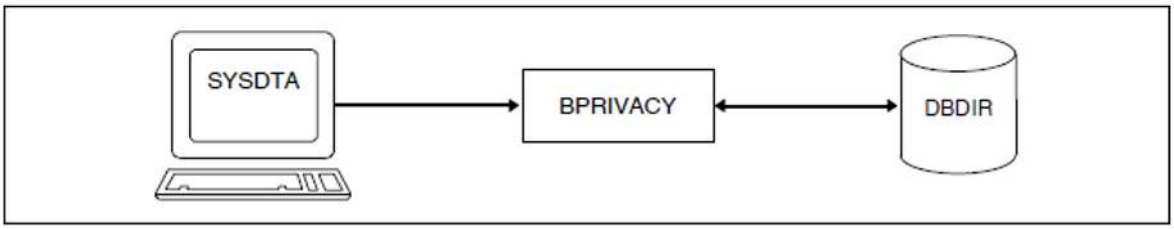


Figure 17: System environment for BPRIVACY

The BPRIVACY utility routine must be called in the identification under which the database is cataloged.
You can start BPRIVACY at any point after the BFORMAT routine has executed. It is thus possible to grant new groups access to an existing database application and to change the access rights of existing user groups any number of times.
After changes have been made to a schema (e.g. after new record types or new base tables have been introduced), you must update the access rights with BPRIVACY.

In the case of CODASYL access, BPRIVACY can run in the creation phase of a UDS/SQL database before the subschemas are compiled.

In the case of SQL access, you can start BPRIVACY after compilation and entry of the relational schema (UDS /SQL subschema) in the database.

When required, BPRIVACY automatically extends the DBDIR of the database being processed or the DBTTs of the record types in the DBDIR. For details, please refer to the “Database Operation” manual, Automatic realm extension by means of utility routines.

At startup BPRIVACY takes into account any assigned UDS/SQL pubset declaration (see the “Database Operation” manual, Pubset declaration job variable). Faulty assignment leads to the program aborting.

The BPRIVACY utility routine uses the linked-in DBH during execution.

4.6 Rules for the statements

The statement formats of the ONLINE-PRIVACY and BPRIVACY utility routines comply with the rules of SDF (System Dialog Facility, see the "SDF Dialog Interface" and "Commands" manuals).

Incorrectly entered statements can be corrected. You can undo any correctly entered statement with the UNDO statement or the inverse function (if available).

If entries are contradictory, the last one always applies.

Valid statements are not executed until after the END statement.

The OPEN-DATABASE statement is an exception to this.

The ONLINE-PRIVACY and BPRIVACY utility routines always apply to all objects (realms, records and sets) of the PRIVACY schema (PRIVACY-AND-IQF-SCHEMA).

4.7 Overview of statements

Statement	Meaning
ADD-USER-GROUP **USER-GROUP-NAME** = list-poss(6): ***KSET-FORMAT**(...) / ***FREE-FORMAT**(...) ,**OBJECT** = **NONE** / list-poss(6): ***REALM**(...) / ***RECORD**(...) / ***SET**(...)	Defines a user group, possibly with access rights
END	Terminates command input
GRANT-ACCESS **USER-GROUP-NAME** = list-poss(6): ***KSET-FORMAT**(...) / ***FREE-FORMAT**(...) ,**OBJECT** = list-poss(6): ***REALM**(...) / ***RECORD**(...) / ***SET**(...)	Assigns access rights to a user group
OPEN-DATABASE **DATABASE-NAME** = <dbname>	Opens a database
REMOVE-USER-GROUP **USER-GROUP-NAME** = **ALL** / ***ALL-EXCEPT**(...) / list-poss(6): ***KSET-FORMAT**(...) / ***FREE-FORMAT**(...)	Deletes one or more user group(s)
REVOKE-ACCESS **USER-GROUP-NAME** = list-poss(6): ***KSET-FORMAT**(...) / ***FREE-FORMAT**(...) ,**OBJECT** = list-poss(6): ***REALM**(...) / *RECORD(...) / ***SET**(...)	Withdraws access rights from a user group
SHOW-USER-GROUP **USER-GROUP-NAME** = **ALL** / ***ALL-EXCEPT**(...) / list-poss(6): ***KSET-FORMAT**(...) / ***FREE-FORMAT**(...) ,**OUTPUT** = list-poss: **SYSLST** / **SYSOUT**	Outputs information on one or more user group(s)
UNDO	Undoes a statement

Table 19: Overview of statements

ADD-USER-GROUP
(Defining a user group with or without assigning access rights)

The ADD-USER-GROUP statement allows you to define new user groups. You can define the associated access rights at the same time.

The structure of the user group name depends on the environment in which you are working (see table 18 on "Checking access rights").

You can specify the name of the user group in two different formats:

*KSET-FORMAT and *FREE-FORMAT

*KSET-FORMAT is available for openUTM operation, for example. In this case, the user group name comprises the three parts openUTM host name, openUTM application name and KSET name.

If you do not define access rights for the specified user group, the user group is created, but users in the group cannot access database objects. Access rights must be assigned by means of subsequent GRANT-ACCESS statements (see page GRANT-ACCESS).

ADD-USER-GROUP

```
USER-GROUP-NAME = list-poss(6): *KSET-FORMAT(...) / *FREE-FORMAT(...)

    *KSET-FORMAT(...)

        |    HOST = <host>

        |    ,APPLICATION = <appl>

        |    ,KSET = *NONE / list-poss(30): <kset>

    *FREE-FORMAT(...)

        |    HOST = <host>

        |    ,USER-ID = list-poss(30): <userid> / *NONE

,OBJECT = NONE / list-poss(6): *REALM(...) / *RECORD(...) / *SET(...)

    *REALM(...)

        |    NAME = *ALL / *ALL-EXCEPT(...) / list-poss(30): <realm-name>

        |         |    *ALL-EXCEPT(...)

        |         |         NAME = list-poss(30): <realm-name>

        |    ,RIGHT = ALL / RETRIEVAL

    *RECORD(...)

        |    NAME = *ALL / *ALL-EXCEPT(...) / list-poss(30): <record-name>

        |         |    *ALL-EXCEPT(...)

        |         |         NAME = list-poss(30): <record-name>

        |    ,RIGHT = ALL / RETRIEVAL

    *SET(...)

        |    NAME = *ALL / *ALL-EXCEPT(...) / list-poss(30): <set-name>

        |         |    *ALL-EXCEPT(...)

        |         |         NAME = list-poss(30): <set-name>

        |    ,RIGHT = ALL / RETRIEVAL
```

USER-GROUP-NAME = list-poss(6): *KSET-FORMAT(...) / *FREE-FORMAT(...)
Name of the user group.

***KSET-FORMAT(...)**
Specification of the user group.

HOST = <host>
Host computer of the openUTM application (see *host*, "Checking access rights").

APPLICATION = <appl>
Name of the openUTM application.

KSET = *NONE
No KSET name is specified.

KSET = list-poss(30): <kset>
KSET name of the openUTM application.

***FREE-FORMAT(...)**
Specification of the user group.

HOST = <host>
Host computer of the application.

USER-ID = list-poss(30): <userid>
Identification of the application.

USER-ID = *NONE
No identification is specified. The specification *NONE is now only permitted for reasons of compatibility. The corresponding user group name can no longer be used as of UDS/SQL V2.0.

OBJECT = NONE / list-poss(6): *REALM(...) / *RECORD(...) / *SET(...)
The access rights are specified.

NONE
No access rights are assigned.

***REALM(...)**
The realm rights are assigned.

NAME = *ALL
The specified access applies to all realms of the database.
This operand must be defined for SQL applications.

NAME = *ALL-EXCEPT(...)
The specified access applies to all realms except those entered here.

NAME = list-poss(30): <realm-name>
The specified access does not apply to these realms.

NAME = list-poss(30): <realm-name>
The specified access applies only to these realms.

RIGHT = ALL
Both read and write access is granted for the realms.

RIGHT = RETRIEVAL
Only read access is granted for the realms.

***RECORD (...)**
The rights for record types (in SQL applications: base tables) are assigned.

NAME = *ALL
The specified access applies to all record types (in SQL applications: base tables) in the database.

NAME = *ALL-EXCEPT(...)
The specified access applies to all record types except those entered here.

NAME = list-poss(30): <record name>
The specified access does not apply to these record types.

NAME = list-poss(30): <record-name>
The specified access does not apply to these record types.

RIGHT = ALL
Both read and write access is granted for the record types.

RIGHT = RETRIEVAL
Only read access is granted for the record types.

***SET(...)**
The rights for sets (in SQL applications: foreign keys) are assigned.

NAME = *ALL
The specified access applies to all sets (in SQL applications: foreign keys) in the database.

NAME = *ALL-EXCEPT(...)
The specified access applies to all sets except those entered here.

NAME = list-poss(30): <set-name>
The specified access does not apply to these sets.

NAME = list-poss(30): <set-name>
The specified access applies only to these sets.

RIGHT = ALL
Both read and write access is granted for the sets.

RIGHT = RETRIEVAL

Only read access is granted for the sets.

> In the case of applications that work with BPRIVACY group names of a version < UDS/SQL V1.2 or < UDS-D V1.4, the access rights can be defined by specifying the old group name for <host> in the *FREE-FORMAT operand (see table 18 in chapter "Checking access rights").
>
> BPRIVACY group names of a version < UDS/SQL V1.2 containing one or more blanks can no longer be defined via the new interface. In these cases, you must define a new group name in compliance with SDF rules of syntax.

Example

The following user groups are defined for the SHIPPING database:

- "D017ZE07_____XXXXX___" : all rights
- "D017ZE07_____YYYYYY__" : retrieval right
- "D017ZE07_____ZZZZZZ__" : no rights

```
/ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=SHIPPING.DBDIR
/SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,   VERSION=02.9C00
/START-UDS-BPRIVACY
***** START        BPRIVACY     (UDS/SQL  V2.9  1801 )     2019-01-29   09:27:25
%  UDS0215 UDS STARTING UDS/SQL V2.9 (LINKED-IN), DATE=2019-01-29 (ILL2038,09:27:25/4TE7)
%  UDS0746 UDS PUBSET DECLARATION (CURRENT) FOLLOWS (ILL1746,09:27:25/4TE7)
4TE7: UDS-PUBSET-JV:  :IUDS:$XXXXXXXX.PUBSDECL.ALL
4TE7: PUBSETS:        *
4TE7: DEFAULT PUBSET: IUDS
4TE7: ----------------------------------------
%  UDS0722 UDS ORDER ADD RLOG 190129082724 IN EXECUTION (ILL1283,09:27:25/4TE7)
%  UDS0356 UDS EXECUTION OF ORDERS FOR SHIPPING TERMINATED (ILL1309,09:27:25/4TE7)
//ADD-USER-GROUP USER-GROUP-NAME=*KSET-FORMAT(HOST=UTMHOST1,APPLICATION=UTMAWVER,
KSET=KSETUPD), -
//  OBJECT=(*REALM(NAME=*ALL,RIGHT=ALL),*RECORD(NAME=*ALL,RIGHT=ALL), -
//          *SET(NAME=*ALL,RIGHT=ALL))
//ADD-USER-GROUP USER-GROUP-NAME=*KSET-FORMAT(HOST=UTMHOST1,APPLICATION=UTMAWVER,
KSET=KSETRTR), -
//  OBJECT=(*REALM(NAME=*ALL,RIGHT=RETRIEVAL),*RECORD(NAME=*ALL,RIGHT=ALL), -
//          *SET(NAME=*ALL,RIGHT=ALL))
//ADD-USER-GROUP USER-GROUP-NAME=*KSET-FORMAT(HOST=UTMHOST1,APPLICATION=UTMAWVER,
KSET=KSETRTR), -
//  OBJECT=(*REALM(NAME=*ALL,RIGHT=RETRIEVAL),*RECORD(NAME=*ALL,RIGHT=RETRIEVAL), -
//          ,*SET(NAME=*ALL,RIGHT=RETRIEVAL))
//END
%  UDS0758 NUMBER OF DML-STATEMENTS AND I/O COUNTERS PER DATABASE  (ILL1758,09:27:25/4TE7)
 4TE7: DATABASE NAME      DMLS   LOG READ  PHYS READ  LOG WRITE PHYS WRITE
 4TE7: ------------------------------------------------------------------
 4TE7: SHIPPING             13        112         59         42         20
%  UDS0213 UDS NORMAL SYSTEM TERMINATION WITH **************13 DML-STATEMENTS 2019-01-29
(ILLY033,09:27:25/4TE7)

***** DIAGNOSTIC SUMMARY OF BPRIVACY

              NO WARNINGS
              NO ERRORS
              NO SYSTEM-ERRORS

***** END OF DIAGNOSTIC SUMMARY
***** NORMAL END   BPRIVACY    (UDS/SQL  V2.9  1801 )     2019-01-29   09:27:25
```

END (Terminating command input)

Command input is terminated. Execution begins.

END

This statement has no operands.

GRANT-ACCESS (Assigning access rights to a user group)

The GRANT-ACCESS statement allows you to assign a user group access rights for realms, record types (RECORDs) and sets.

If a user group already has access rights for an object, the rights specified with this statement are added to them.

GRANT-ACCESS

```
USER-GROUP-NAME = list-poss(6): *KSET-FORMAT(...) / *FREE-FORMAT(...)

    *KSET-FORMAT(...)

        |    HOST = <host>

        |    ,APPLICATION = <appl>

        |    ,KSET = *NONE / list-poss(30): <kset>

    *FREE-FORMAT(...)

        |    HOST = <host>

        |    ,USER-ID = *NONE / list-poss(30): <userid>

,OBJECT = list-poss(6): *REALM(...) / *RECORD(...) / *SET(...)

    *REALM(...)

        |    NAME = *ALL / *ALL-EXCEPT(...) / list-poss(30): <realm-name>

        |         |    *ALL-EXCEPT(...)

        |         |         NAME = list-poss(30): <realm-name>

        |    ,RIGHT = ALL / RETRIEVAL

    *RECORD(...)

        |    NAME = *ALL / *ALL-EXCEPT (...) / list-poss(30): <record-name>

        |         |    *ALL-EXCEPT(...)

        |         |         NAME = list-poss(30): <record-name>

        |    ,RIGHT = ALL /RETRIEVAL

    *SET(...)

        |    NAME = *ALL / *ALL-EXCEPT(...) / list-poss(30): <set-name>

        |         |    *ALL-EXCEPT(...)

        |         |         NAME = list-poss(30): <set-name>

        |    ,RIGHT = ALL / RETRIEVAL
```

USER-GROUP-NAME = list-poss(6): *KSET-FORMAT(...) / *FREE-FORMAT(...)
Name of the user group.

***KSET-FORMAT(...)**
Name of the user group.

HOST = <host>
Host computer of the openUTM application (see *host*, "Checking access rights").

APPLICATION = <appl>
Name of the openUTM application.

KSET = *NONE
No KSET name is specified.

KSET = list-poss(6): <kset>
KSET name of the openUTM application.

***FREE-FORMAT(...)**
Name of the user group.

HOST = <host>
Host computer of the application.

USER-ID = *NONE
No identification is specified.

USER-ID = list-poss(30): <userid>
Identification of the application.

OBJECT = list-poss(6): *REALM(...) / *RECORD(...) / *SET(...)
The access rights are specified.

***REALM (...)**
The existing realm rights are changed.

NAME = *ALL
The specified access applies to all realms in the database. This operand must be defined for all SQL applications.

NAME = ALL-EXCEPT (...)
The specified access applies to all realms except those entered here.

NAME = list-poss(30): <realm-name>
The specified access does not apply to these realms.

NAME = list-poss(30): <realm-name>
The specified access applies only to these realms.

RIGHT = ALL
Both read and write access is granted for the realms.

RIGHT = RETRIEVAL
Only read access is granted for the realms.

***RECORD(...)**
The existing rights for record types (in SQL applications: base tables) are changed.

NAME = *ALL
The specified access applies to all record types (in SQL applications: base tables in the database.

NAME = ALL-EXCEPT (...)
The specified access applies to all record types except those entered here.

NAME = list-poss(30): <record-name>
The specified access does not apply to these record types.

NAME = list-poss(30): <record-name>
The specified access applies only to these record types.

RIGHT = ALL
Both read and write access is granted for the record types.

RIGHT = RETRIEVAL
Only read access is granted for the record types.

***SET(...)**
The existing rights to sets (in SQL applications: foreign keys) are changed.

NAME = *ALL
The specified access applies to all sets (in SQL applications: foreign keys) in the database.

NAME = ALL-EXCEPT
The specified access applies to all sets except those entered here.

NAME = list-poss(30): <set-name>
The specified access does not apply to these sets.

NAME = list-poss(30): <set-name>
The specified access applies only to these sets.

RIGHT = ALL
Both read and write access is granted for these sets.

RIGHT = RETRIEVAL
Only read access is granted for these sets.

Example

The user group "D017ZE07_____ZZZZZZ__" (no rights) is assigned all rights for the realm CUSTOMER-ORDER-RLM.

```
/ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=SHIPPING.DBDIR
/SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,    VERSION=02.9C00
/START-UDS-BPRIVACY
***** START        BPRIVACY     (UDS/SQL  V2.9  1801 )     2019-01-29   09:27:59
%  UDS0215 UDS STARTING UDS/SQL V2.9 (LINKED-IN), DATE=2019-01-29 (ILL2038,09:27:59/4TE7)
%  UDS0746 UDS PUBSET DECLARATION (CURRENT) FOLLOWS (ILL1746,09:27:59/4TE7)
4TE7: UDS-PUBSET-JV:  :IUDS:$XXXXXXXX.PUBSDECL.ALL
4TE7: PUBSETS:        *
4TE7: DEFAULT PUBSET: IUDS
4TE7: ----------------------------------------
%  UDS0722 UDS ORDER ADD RLOG 190129082758 IN EXECUTION (ILL1283,09:27:59/4TE7)
%  UDS0356 UDS EXECUTION OF ORDERS FOR SHIPPING TERMINATED (ILL1309,09:27:59/4TE7)
//GRANT-ACCESS USER-GROUP-NAME=*FREE-FORMAT(HOST=IBAPROD1,USER-ID=ZZZZZZ),-
//  OBJECT=(*REALM(NAME=CUSTOMER-ORDER-RLM,RIGHT=ALL),*RECORD( NAME=*ALL,RIGHT=ALL), -
//          *SET(NAME=*ALL,RIGHT=ALL))
//SHOW-USER-GROUP USER-GROUP-NAME=*FREE-FORMAT(HOST=IBAPROD1,USER-ID=ZZZZZZ),OUTPUT=SYSOUT
//END

DATABASE NAME : $XXXXXXXX.SHIPPING
SCHEMA NAME   : MAIL-ORDERS
```

```
*****************************************************************

ACCESS RIGHTS FOR USERGROUP : IBAPROD1________ZZZZZZ__

RIGHTS ON REALMS

+-------------------------------+----------------------+
!                               !       R I G H T      !
!                               +-----------+----------+
! REALM NAME                    ! RETRIEVAL !  UPDATE   !
+-------------------------------+-----------+----------+
! CUSTOMER-ORDER-RLM            !     Y     !     Y    !
! PURCHASE-ORDER-RLM            !     N     !     N    !
! CLOTHING                      !     N     !     N    !
.
.
.
 RIGHTS ON RECORDS
 +-------------------------------+----------------------+
 !                               !       R I G H T      !
 !                               +-----------+----------+
 ! RECORD NAME                   ! RETRIEVAL !  UPDATE   !
 +-------------------------------+-----------+----------+
 ! CUSTOMER                      !     Y     !     Y    !
 ! CST-ORDERS                    !     Y     !     Y    !
 ! ORD-ITEM                      !     Y     !     Y    !
.
.
.
 RIGHTS ON SETS
 +-------------------------------+----------------------+
 !                               !       R I G H T      !
 !                               +-----------+----------+
 ! SET NAME                      ! RETRIEVAL !  UPDATE   !
 +-------------------------------+-----------+----------+
 ! CST-ORD-PLACED                !     Y     !     Y    !
 ! CST-O-CONTENTS                !     Y     !     Y    !
 ! OUTSTANDING                   !     Y     !     Y    !
.
.
.
%  UDS0758 NUMBER OF DML-STATEMENTS AND I/O COUNTERS PER DATABASE (ILL1758,09:27:59/4TE7)
 4TE7: DATABASE NAME       DMLS   LOG READ  PHYS READ  LOG WRITE PHYS WRITE
 4TE7: ------------------------------------------------------------------------
 4TE7: SHIPPING              16         84         57         18         16
%  UDS0213 UDS NORMAL SYSTEM TERMINATION WITH **************16 DML-STATEMENTS 2019-01-29
(ILLY033,09:27:59/4TE7)

 ***** DIAGNOSTIC SUMMARY OF BPRIVACY

             NO WARNINGS
             NO ERRORS
             NO SYSTEM-ERRORS

 ***** END OF DIAGNOSTIC SUMMARY
 ***** NORMAL END   BPRIVACY     (UDS/SQL  V2.9  1801 )     2019-01-29   09:27:59
```

OPEN-DATABASE (Opening the database)

The OPEN-DATABASE statement must be the first one entered.

The OPEN-DATABASE statement allows you to specify the database to be processed by the subsequent ONLINE-PRIVACY or BPRIVACY statements.

OPEN-DATABASE
DATABASE-NAME = <dbname>

DATABASE-NAME = <dbname>
Specifies the database for which access authorizations are to be changed. A user of ONLINE-PRIVACY or BPRIVACY can process a database only if it is in his or her identification. A database in a different identification can only be processed using the TSOS identification of the system administrator.

> The OPEN-DATABASE statement cannot be used if the database is assigned by means of LINK-NAME=DATABASE for BPRIVACY.
> The OPEN-DATABASE statement is always required for ONLINE-PRIVACY.In this case the configuration name of the independent DBH session is specified via LINK-NAME=DATABASE.

REMOVE-USER-GROUP (Deleting one or more user group(s))

The REMOVE-USER-GROUP statement allows you to delete one or more user groups together with their access rights.

REMOVE-USER-GROUP

USER-GROUP-NAME = **ALL** / ***ALL**/ ***ALL-EXCEPT**(...) / list-poss(6): ***KSET-FORMAT**(...) / ***FREE-FOR-MAT**(...)

***ALL-EXCEPT**(...)

| **NAME** = list-poss(6): ***KSET-FORMAT**(...) / ***FREE-FORMAT**(...)

| ***KSET-FORMAT**(...)

| | **HOST** = <host>

| | ,**APPLICATION** = <appl>

| | ,**KSET** = ***NONE** / list-poss(30): <kset>

| ***FREE-FORMAT**(...)

| | **HOST** = <host>

| | ,**USER-ID** = ***NONE** / list-poss(30): <user-id>

***KSET-FORMAT**(...)

| **HOST** = <host>

| ,**APPLICATION** = <appl>

| ,**KSET** = ***NONE** / list-poss(30): <kset>

***FREE-FORMAT**(...)

| **HOST** = <host>

| ,**USER-ID** = ***NONE** / list-poss(30): <userid>

USER-GROUP-NAME = ALL / *ALL / *ALL-EXCEPT(...) / list-poss(6): *KSET-FORMAT(...) /*FREE-FORMAT(...)
It is specified which user group(s) is (are) to be deleted.

ALLALL / *ALL
All existing user groups are deleted.

***ALL-EXCEPT (...)**
All user groups except those entered here are deleted.

NAME = list-poss(6): *KSET-FORMAT(...) / *FREE-FORMAT(...)
The user groups specified here are not deleted.

***KSET-FORMAT(...)**

HOST = <host>
Host computer of the openUTM application (see *host*, "Checking access rights").

APPLICATION = <appl>
Name of the openUTM application.

KSET = *NONE
No KSET name is specified.

KSET = list-poss(30): <kset>
KSET name of the openUTM application.

***FREE-FORMAT(...)**

HOST = <host>
Host computer of the application.

USER-ID = *NONE
No identification is specified.

USER-ID = list-poss(30): <userid>
Identification of the application

***KSET-FORMAT(...)**
All user groups specified here are deleted.

HOST = <host>
Host computer of the openUTM application (see *host*, "Checking access rights").

APPLICATION = <appl>
Name of the openUTM application.

KSET = *NONE
No KSET name is specified.

KSET = list-poss(30): <kset>
KSET name of the openUTM application.

***FREE-FORMAT(...)**
All user groups specified here are deleted.

HOST = <host>
Host computer of the application.

USER-ID = *NONE
No identification is specified.

USER-ID = list-poss(30): <userid>
Identification of the application.

Example

The user group "`D017ZE07_____YYYYYY__`" is deleted.

```
/ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=SHIPPING.DBDIR
/SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,    VERSION=02.9C00
/START-UDS-BPRIVACY
***** START        BPRIVACY     (UDS/SQL  V2.9  1801 )    2019-01-29   09:27:59
%  UDS0215 UDS STARTING UDS/SQL V2.9 (LINKED-IN), DATE=2019-01-29 (ILL2038,09:27:59/4TE7)
%  UDS0746 UDS PUBSET DECLARATION (CURRENT) FOLLOWS (ILL1746,09:28:00/4TE7)
4TE7: UDS-PUBSET-JV:  :IUDS:$XXXXXXXX.PUBSDECL.ALL
4TE7: PUBSETS:        *
4TE7: DEFAULT PUBSET: IUDS
4TE7: ----------------------------------------
%  UDS0722 UDS ORDER ADD RLOG 190129082800 IN EXECUTION (ILL1283,09:28:00/4TE7)
%  UDS0356 UDS EXECUTION OF ORDERS FOR SHIPPING TERMINATED (ILL1309,09:28:00/4TE7)
//REMOVE-USER-GROUP USER-GROUP-NAME=*FREE-FORMAT(HOST=IBAPROD1,USER-ID=YYYYYY)
//END
%  UDS0758 NUMBER OF DML-STATEMENTS AND I/O COUNTERS PER DATABASE (ILL1758,09:28:00/4TE7)
 4TE7: DATABASE NAME      DMLS   LOG READ  PHYS READ  LOG WRITE PHYS WRITE
 4TE7: ------------------------------------------------------------------
 4TE7: SHIPPING             7         93         60         25         22
%  UDS0213 UDS NORMAL SYSTEM TERMINATION WITH ***************7 DML-STATEMENTS 2015-06-28
(ILLY033,09:28:00/4TE7)

***** DIAGNOSTIC SUMMARY OF BPRIVACY
              NO WARNINGS
              NO ERRORS
              NO SYSTEM-ERRORS

***** END OF DIAGNOSTIC SUMMARY
***** NORMAL END   BPRIVACY    (UDS/SQL  V2.9  1801 )     2019-01-29   09:28:00
```

REVOKE-ACCESS (Withdrawing access rights from a user group)

The REVOKE-ACCESS statement allows you to withdraw from a user group access rights for realms, record types (RECORDs) and sets.

Only the access rights specified here are withdrawn. All other access rights are retained.

REVOKE-ACCESS

USER-GROUP-NAME = list-poss(6): ***KSET-FORMAT**(...) / ***FREE-FORMAT**(...)

***KSET-FORMAT**(...)

| **HOST** = <host>

| ,**APPLICATION** = <appl>

| ,**KSET** = ***NONE** / list-poss(30): <kset>

***FREE-FORMAT**(...)

| **HOST** = <host>

| ,**USER-ID** = ***NONE** / list-poss(30): <userid>

,**OBJECT** = list-poss(6): ***REALM**(...) / ***RECORD**(...) / ***SET**(...)

***REALM**(...)

| **NAME** = ***ALL** / ***ALL-EXCEPT**(...) / list-poss(30): <realm-name>

| | ***ALL-EXCEPT**(...)

| | **NAME** = list-poss(30): <realm-name>

| ,**RIGHT** = **ALL** / **UPDATE** / **RETRIEVAL**

***RECORD**(...)

| **NAME** = ***ALL** / ***ALL-EXCEPT**(...) / list-poss(30): <record-name>

| | ***ALL-EXCEPT**(...)

| | **NAME** = list-poss(30): <record-name>

| ,**RIGHT** = **ALL** / **UPDATE** / **RETRIEVAL**

***SET**(...)

| **NAME** = ***ALL** / ***ALL-EXCEPT**(...) / list-poss(30): <set-name>

| | ***ALL-EXCEPT**(...)

| | **NAME** = list-poss(30): <set-name>

| ,**RIGHT** = **ALL** / **UPDATE** / **RETRIEVAL**

USER-GROUP-NAME = list-poss(6): *KSET-FORMAT(...) / *FREE-FORMAT(...)
Name of the user group.

***KSET-FORMAT(...)**
Name of the user group.

HOST = <host>
Host computer of the openUTM application (see *host*, "Checking access rights").

APPLICATION = <appl>
Name of the openUTM application.

KSET = *NONE
No KSET name is specified.

KSET = list-poss(30): <kset>
KSET name of the openUTM application.

***FREE-FORMAT(...)**
Name of the user group.

HOST = <host>
Host computer of the application.

USER-ID = *NONE
No identification is specified.

USER-ID = list-poss(30): <userid>
Identification of the application.

OBJECT = list-poss(6): *REALM(...) / *RECORD(...) / *SET(...)
The access rights are specified.

***REALM(...)**
The existing realm rights are changed.

NAME = *ALL
The specified access is withdrawn for all realms in the database.

NAME = *ALL-EXCEPT (...)
The specified access is withdrawn for all realms except those entered here.

NAME = list-poss(30): <realm-name>
The specified access is not withdrawn for these realms.

NAME = list-poss(30): <realm-name>
The specified access is withdrawn for these realms only.

RIGHT = ALL
The realms can no longer be accessed.

RIGHT = UPDATE
The realms can no longer be write-accessed.

RIGHT = RETRIEVAL
This is only possible if the specified user group does not have write (UPDATE) access.
The realms can no longer be accessed.

***RECORD(...)**
The existing rights for record types (in SQL applications: base tables) are changed.

NAME = *ALL
The specified access is withdrawn for all record types (in SQL applications: base tables) in the database.

NAME = *ALL-EXCEPT (...)
The specified access is withdrawn for all record types except those entered here.

NAME = list-poss(30): <record-name>
The specified access is not withdrawn for these record types.

NAME = list-poss(30): <record-name>
The specified access is withdrawn only for these record types.

RIGHT = ALL
The record types can no longer be accessed.

RIGHT = UPDATE
The record types can no longer be write-accessed.

RIGHT = RETRIEVAL
This is only possible if the specified user group does not have write (UPDATE) access.
The record types can no longer be accessed.

***SET(...)**
The existing rights for sets (in SQL applications: foreign keys) are changed.

NAME = *ALL
The specified access is withdrawn for all sets (in SQL applications: foreign keys) in the database.

NAME = *ALL-EXCEPT (...)
The specified access is withdrawn for all sets except those entered here.

NAME = list-poss(30): <set-name>
The specified access is not withdrawn for these sets.

NAME = list-poss(30): <set-name>
The specified access is withdrawn for these sets only.

RIGHT = ALL
The sets can no longer be accessed.

RIGHT = UPDATE
The sets can no longer be write-accessed.

RIGHT = RETRIEVAL
This is only possible if the specified user group does not have write (UPDATE) access.
The sets can no longer be accessed.

Example

The update right for the realm CUSTOMER-ORDER-RLM is withdrawn from user group "D017ZE07_____ZZZZZZ__".

```
/ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=SHIPPING.DBDIR
/SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,    VERSION=02.9C00
/START-UDS-BPRIVACY
***** START        BPRIVACY     (UDS/SQL  V2.9  1801 )     2019-01-29   09:28:02
%  UDS0215 UDS STARTING UDS/SQL V2.9 (LINKED-IN), DATE=2019-01-29 (ILL2038,09:28:02/4TE7)
%  UDS0746 UDS PUBSET DECLARATION (CURRENT) FOLLOWS (ILL1746,09:28:02/4TE7)
4TE7: UDS-PUBSET-JV:  :IUDS:$XXXXXXXX.PUBSDECL.ALL
4TE7: PUBSETS:        *
4TE7: DEFAULT PUBSET: IUDS
4TE7: ----------------------------------------
%  UDS0722 UDS ORDER ADD RLOG 190129082802 IN EXECUTION (ILL1283,09:28:02/4TE7)
%  UDS0356 UDS EXECUTION OF ORDERS FOR SHIPPING TERMINATED (ILL1309,09:28:02/4TE7)
//REVOKE-ACCESS USER-GROUP-NAME=*FREE-FORMAT(HOST=IBAPROD1,USER-ID=ZZZZZZ), -
//  OBJECT=(*REALM(NAME=CUSTOMER-ORDER-RLM,RIGHT=UPDATE))
//SHOW-USER-GROUP USER-GROUP-NAME=ALL,OUTPUT=SYSOUT
//END

DATABASE NAME : $XXXXXXXX.SHIPPING
SCHEMA NAME   : MAIL-ORDERS

*********************************************************************

ACCESS RIGHTS FOR USERGROUP : IBAPROD1________ZZZZZZ__

RIGHTS ON REALMS

+--------------------------------+----------------------+
!                                !       R I G H T      !
!                                +-----------+----------+
! REALM NAME                     ! RETRIEVAL !  UPDATE   !
+--------------------------------+-----------+----------+
! CUSTOMER-ORDER-RLM             !     Y     !     N    !
! PURCHASE-ORDER-RLM             !     Y     !     Y    !
! CLOTHING                       !     Y     !     Y    !
.
.
%  UDS0758 NUMBER OF DML-STATEMENTS AND I/O COUNTERS PER DATABASE (ILL1758,09:28:02/4TE7)
 4TE7: DATABASE NAME      DMLS   LOG READ  PHYS READ  LOG WRITE PHYS WRITE
 4TE7: ------------------------------------------------------------------
 4TE7: SHIPPING             17         86         58         18         18
%  UDS0213 UDS NORMAL SYSTEM TERMINATION WITH **************17 DML-STATEMENTS 2019-01-29
(ILLY033,09:28:02/4TE7)

***** DIAGNOSTIC SUMMARY OF BPRIVACY

              NO WARNINGS
              NO ERRORS
              NO SYSTEM-ERRORS

***** END OF DIAGNOSTIC SUMMARY
***** NORMAL END   BPRIVACY     (UDS/SQL  V2.9  1801 )     2019-01-29   09:28:02
```

SHOW-USER-GROUP (Outputting information on user groups)

Information is output on user groups, i.e. on the access rights the groups have for database objects. The realms, record types and sets are output in ascending order by reference number.

SHOW-USER-GROUP

USER-GROUP-NAME = **ALL** / ***ALL** / ***ALL-EXCEPT**(...) / list-poss(6): ***KSET-FORMAT**(...) / ***FREE-FOR-MAT**(...)

***ALL-EXCEPT**(...)

| **NAME** = list-poss(6): ***KSET-FORMAT**(...) / ***FREE-FORMAT**(...)

| ***KSET-FORMAT**(...)

| | **HOST** = <host>

| | ,**APPLICATION** = <appl>

| | ,**KSET** = ***NONE** / list-poss(30): <kset>

| ***FREE-FORMAT**(...)

| | **HOST** = <host>

| | ,**USER-ID** = ***NONE** / list-poss(30): <userid>

***KSET-FORMAT**(...)

| **HOST** = <host>

| ,**APPLICATION** = <appl>

| ,**KSET** = ***NONE** / list-poss(30): <kset>

***FREE-FORMAT**(...)

| **HOST** = <host>

| ,**USER-ID** = ***NONE** / list-poss(30): <userid>

,**OUTPUT** = list-poss: **SYSLST** / **SYSOUT**

USER-GROUP-NAME = ALL / *ALL / *ALL-EXCEPT(...) / list-poss(6): *KSET-FORMAT(...)/*FREE-FORMAT(...)
The rights are output.

ALL / *ALL
The rights of all user groups are output.

***ALL-EXCEPT (...)**
The rights of all user groups except those entered here are output.

NAME = list-poss(6): *KSET-FORMAT(...) / *FREE-FORMAT(...)
The rights of the specified user groups are not output.

***KSET-FORMAT(...)**

HOST = <host>
Host computer of the openUTM application (see *host*, "Checking access rights").

APPLICATION = <appl>
Name of the openUTM application.

KSET = *NONE
No KSET name is specified.

KSET = list-poss (30): <kset>
KSET name of the openUTM application.

***FREE-FORMAT(...)**

HOST = <host>
Host computer of the application.

USER-ID = *NONE

USER-ID = list-poss(30): <userid>
Identification of the application.

***KSET-FORMAT(...)**
The rights of the specified user groups are output.

HOST = <host>
Host computer of the openUTM application (see *host*, "Checking access rights").

APPLICATION = <appl>
Name of the openUTM application.

KSET = *NONE
No KSET name is specified.

KSET = list-poss(30): <kset>
KSET name of the openUTM application.

***FREE-FORMAT(...)**

HOST = <host>
Host computer of the application.

USER-ID = *NONE
No identification is output.

USER-ID = list-poss(30): <userid>
Identification of the application.

OUTPUT = list-poss: SYSLST / SYSOUT
The information is output.

SYSLST
Output is to SYSLST.

SYSOUT
Output is to SYSOUT.

Beispiel

See example of GRANT-ACCESS.

UNDO (Undoing a statement)

The last correctly entered statement (except UNDO itself) is not executed. A subsequent UNDO statement undoes the penultimate statement (apart from UNDO), and so on.

UNDO

This statement has no operands.

4.8 Command sequence for starting ONLINE-PRIVACY

It is assumed for the command sequence described here that UDS/SQL was installed with IMON (see section “START commands for the UDS/SQL programs”).

```
01 /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version
02 /SET-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=configurationname
03 /START-UDS-ONLINE-PRIVACY
04 OPEN-DATABASE DATABASE-NAME=dbname
05 other online-privacy statements
06 END
```

Explanation

01 Use the SELECT-PRODUCT-VERSION command to specify which UDS/SQL version is to be used since more than one UDS/SQL version may be installed in parallel with IMON in the Software Configuration Inventory (SCI) and more than one version of the UDS/SQL subsystem may be loaded.

02 You assign the configuration name FILE-NAME=*configurationname* via the link name DATABASE with the SET-FILE-LINK command.
The UDS/SQL configuration that is to work with ONLINE-PRIVACY and that the database to be processed is attached to must be made known to the system using this command.

03 ONLINE-PRIVACY must run under the user ID under which the database to be processed was created. If this is not the case, then access to the database is rejected by the DBH with status code 901.

04 You specify the database to be processed with the OPEN-DATABASE DATABASE-NAME=... statement. In contrast to the BPRIVACY utility routine, this statement must be specified when ONLINE-PRIVACY is used.
You can only process a single database in each ONLINE-PRIVACY run.
You cannot access a database in a remote configuration via UDS-D with ONLINE-PRIVACY.

4.9 Command sequence for starting BPRIVACY

It is assumed for the command sequence described here that UDS/SQL was installed with IMON (see section “START commands for the UDS/SQL programs”).

```
01 /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,SCOPE=*TASK
02 [/ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR]
03 /START-UDS-BPRIVACY
04  bprivacy-statements
05 END
```

Explanation

01 The version-dependent module of the linked-in DBH of the relevant version is loaded dynamically (see the section entitled "Compiling, linking and loading UDS/SQL-TIAM application programs" in the "Application Programming" manual).

02 If you assign the database using LINK-NAME=DATABASE, you must not specify the BPRIVACY statement OPEN-DATABASE.
If you do not assign the database using LINK-NAME=DATABASE, the BPRIVACY statement OPEN-DATABASE is mandatory, i.e. must be specified.

03 BPRIVACY must run in the ID in which the database to be processed is located. The UDS/SQL utility routine can also be started with the aliases BPRIVACY and START-UDS-AUTHORIZATION.

5 Storing and unloading data (BINILOAD, BOUTLOAD)

This chapter describes the utility routines BINILOAD and BOUTLOAD, which allow you to store and unload data.

The BINILOAD utility routine can be run at any time in order to store data in an empty or partially loaded database.

The BOUTLOAD utility routine is used to copy, delete or unload entire record types from the database, e.g. for the purpose of restructuring.

At startup both utility routines take into account any assigned UDS/SQL pubset declaration (see the “Database Operation” manual, Pubset declaration job variable). Faulty assignment leads to the program aborting.

5.1 Storing records in the database with BINILOAD

The following table is intended to help the administrator decide whether to use BINILOAD or an application program for this purpose.

Keyword	Application program	BINILOAD
Generally used	to update the database.	to load a database.
Stores	every record of a record type individually in the database.	large numbers of records of a record type in the database in one or more runs.
Suitable for loading	a database containing many set occurrences with few member records.	a database with set occurrences which have large numbers of member records.
When inserting into sets	the correct set occurrence and the position within this set occurrence must be determined separately for each member record.	all member records which are to belong to a specific set occurrence are made available as a sequence of records of an input file and processed jointly.
Efficiency in loading the database	bears no relation to the number of records stored.	increases in proportion to the number of records to be stored.

Table 20: Comparison of application program and BINILOAD

BINILOAD is designed for loading a database and can efficiently load a large number of records of one record type. Whether BINILOAD represents a saving in time compared to an application program, and how much time it saves, depends on the number of records to be loaded, and the structure of the database.

> ¡ Since BINILOAD does not use before-images, DBDIR and at least the realms required on loading should be saved before a run, especially if a check run is not going to be carried out.

BINILOAD enables records of the same record type to be stored in an empty or partially empty database. It is important to make all the records to be loaded available in an input file for BINILOAD. All records loaded during a run are stored in the same realm by BINILOAD. It does not matter if records of the same record type are already contained in the database. In each run, BINILOAD only processes records of a single record type.

BINILOAD can

- load data from CSV files;
- presort the records of the input file;
- store records with items of fixed length and variable length;
- generate empty set occurrences for owner records which it has stored;
- when storing member records, insert all records of the input file into a set occurrence nominated by the DB administrator, or select, using the key value of the owner record concerned, the individual set occurrences into which the records are to be inserted.

BINILOAD determines the sequence of member records in compliance with the sort criteria specified in each case and sets up the set occurrence tables so that relatively few accesses to the pages of the database are necessary.

It is also possible however to stipulate the member record sequence desired by specifying a character string in the records of the input file (only for sets where ORDER is not SORTED or SORTED INDEXED, e.g. ORDER IS LAST).

BINILOAD accesses the database directly and not via the DBH. DB records (including set connection data) and tables (pointer arrays, lists, SEARCH key tables etc.) are generated by BINILOAD in work files or in main memory. BINILOAD then assembles the complete pages before transferring them to the database.

5.1.1 Description of functions

Inserting in sets

BINILOAD can insert records it stores in the database in sets; in this case it is very important to distinguish between owner and member record types:

- Owner record type:
  In a CHAIN set, for example, BINILOAD automatically creates an empty set occurrence for each owner record stored.
- Member record type of one or more sets:
  You must specify with the INSERT statement (see section “Statements for BINILOAD”) into which sets BINILOAD is to insert the records.

  This also applies to sets in which the record type has been defined as AUTOMATIC member.

Selecting the set occurrence

Before BINILOAD inserts member records into a set, it needs to select the set occurrence into which the member records are to be inserted. It does this by selecting the owner record using the following key values:

- the value of the CALC key (see "LOCATION MODE clause" in the "Design and Definition" manual)
- the value of the SEARCH key. Defined at record type level either as INDEX-SEARCH key or as CALC-SEARCH key (see "SEARCH KEY clause in the "Design and Definition" manual).
- the value of the database key. This method is always possible, since the database key value is a unique identifier within the database.

If the owner record is a member in a SYSTEM set, BINILOAD can select it using the following key values:

- the value of the ASC/DESC key (for a sorted SYSTEM set)
- the value of the SEARCH key. Defined at set level as INDEX-SEARCH key or as CALC-SEARCH key (see "Direct access" in the "Design and Definition" manual).

BINILOAD can be informed in two ways of the key of the owner record by means of which the set occurrence is to be selected:

- Firstly by specifying the position of the key in the records of the input file. The value of the key may then vary from record to record. Consequently BINILOAD inserts records with different key values in different set occurrences.
- Secondly by specifying the key as a literal in the OWNER statement of BINILOAD. In this case the key value is the same for all records to be stored. BINILOAD therefore inserts all records of the input file in a single set occurrence.

The following points must be taken into consideration when selecting a set occurrence:

- Duplicates
  If the set occurrence is selected by specifying the key value (CALC key, ASC/DESC key, SEARCH key) of the owner record type, and if there are several records with the same key value (DUPLICATES ARE ALLOWED), BINILOAD selects one owner record from the owner records having the same key value, without the user knowing which one will be chosen.

- MANUAL member
  If the records of the input file are to be inserted into a set in which the record type is defined as a MANUAL member, the DB administrator must specify whether or not the record is to be inserted by using the INSERT statement and identifying the input records for optional insertion in a BINILOAD run.
- Member record sequence
  If the set into which BINILOAD is to insert the records to be stored as member records is defined with ORDER IS FIRST/LAST/NEXT/PRIOR or IMMATERIAL, the DB administrator can specify with the SET-ORDER statement of BINILOAD whether BINILOAD should insert the records in the set occurrences
  - in ascending order, by the contents of one item of the input records, or
  - in the order in which they occur in the input file.

  In all other cases the sequence defined in the ORDER clause of the set description is used.
- Set occurrence not empty
  If the records to be stored are to be inserted as member records in a set occurrence in which member records have already been inserted, the structure of the existing set occurrence must be taken into consideration.

  This structure is determined by the following clauses:
  - the MODE clause (SSL) with POINTER-ARRAY, LIST, CHAIN
  - the ORDER clause (DDL) with FIRST, LAST, NEXT, PRIOR, IMMATERIAL, SORTED (can be used with CHAIN only), SORTED INDEXED.

  For DDL and SSL, see the "Design and Definition" manual.

  The following points must be observed:
  - If MODE IS CHAIN is specified with SORTED, SORTED INDEXED, an unfavorable chaining structure may arise;
  - If MODE IS LIST is specified, no further records can be inserted into an existing set occurrence by BINILOAD. This also applies for distributable lists.
  - If MODE IS LIST is specified, BINILOAD requires at least one page for level 0 of each set cccurrence.

Storing in the database

BINILOAD is designed to store large numbers of records in the database. Since BINILOAD does not use partially filled pages, memory space can be more efficiently used than when loading by means of an application program.

It can occur that input files, BINILOAD statements and Schema DDL and SSL specifications are incompatible. BINILOAD therefore offers a facility for checking the input file data before it is finally stored.

Specifying EXECUTION WITH CHECK causes the first phase of BINILOAD storage, the table creation phase, to be executed without actual alterations to the database. Any errors are detected and corresponding messages are output.

If no errors have occurred, BINILOAD repeats its run from the beginning and stores the input data in the database.

Specifying EXECUTION WITHOUT CHECK causes alterations to be written to the database immediately. If an **error** occurs, the BINILOAD run is aborted, error messages are output and the **database is inconsistent**.

- Interdependencies between Schema DDL and BINILOAD statements
  - DUPLICATES ARE NOT ALLOWED and presence of duplicates:

    If, despite specification of DUPLICATES ARE NOT ALLOWED, there are records in the input file that have the same key value, the following message is issued:

    ```
    DUPLICATE KEYS OR DBKEYS FOUND / REC REF‘S OR RSQ‘S OUT OF RANGE
    SEE PRINTER OUTPUT
    ```

    In addition, up to the first 60 bytes of the key are output in the BINILOAD listing in character representation and up to the first 30 bytes of the key are output in hexadecimal form. The message is repeated for each key value found to be present more than once. After checking all the input records, BINILOAD terminates with the message:

    ```
    ABNORMAL END BINILOAD
    ```

    BINILOAD is only able to detect duplicates within the input file. It cannot identify duplication between input records and database records.
  - WITHIN clause and RECORD AREA statement of BINILOAD

    If more than one realm has been defined in the WITHIN clause of the DDL, the realm in which the records are to be stored must be specified in the RECORD AREA statement of BINILOAD.
- Interdependence of SSL and BINILOAD statements:
  - DATABASE-KEY-TRANSLATION-TABLE clause

    This clause specifies the size and location of the DBTT for a record type, and thereby determines the maximum number of records which may be stored in the database. Consequently the input file must not contain more records than there are free entries in the DBTT of the corresponding record type.
  - PLACEMENT OPTIMIZATION clause

    This clause packs the member records for the set contiguously in consecutive pages in accordance with the sort criterion. The member is not stored in the owner page.
  - SSL specifications with ATTACHED

    MODE IS POINTER-ARRAY ATTACHED

    MODE IS LIST ATTACHED

    INDEX NAME IS *indexname* PLACING IS ATTACHED

    Specifying ATTACHED in the SSL has no effect.
    BINILOAD stores pointer arrays, lists and indexes of all levels in empty database pages. These information elements are always stored DETACHED within the realm of the owner.
  - SSL specifications with DETACHED

    Specifying DETACHED WITHIN *realm-name* in the SSL causes the pointer arrays and indexes to be stored in the specified realm. The only exception here are the distributable lists. Here the table part (levels >0) and a possible indirect hash area are stored in *realm-name*.
  - POPULATION clause in the set entry of the SSL and the FILLING statement in BINILOAD

    The FILLING statement can be used to specify the occupancy level for table pages. In this way additional member records can be loaded by means of the Database Handler without direct extension and reorganization for these information elements. The storage space to be reserved for these table pages is determined exclusively by the occupancy level and not by the SET POPULATION clause in the SSL.

For the Schema DDL and SSL, see the "Design and Definition" manual.

5.1.2 Readying the input file and preparing the BINILOAD run

The records to be stored using the BINILOAD utility routine must be readied in an input file.

The records of the input file must all have the same structure. The same input file can, however, be used to store database records of varying structure if different items of the input record are selected.

An input file record may contain, in addition to items for the database record, items with the following information:

- User information. Ignored by BINILOAD.
- Key values which are interpreted by BINILOAD in order to determine the correct set occurrence.
- Information interpreted by BINILOAD in order to determine the sequence of member records.
- Information interpreted by BINILOAD in order to indicate the insertion or non-insertion of member records.

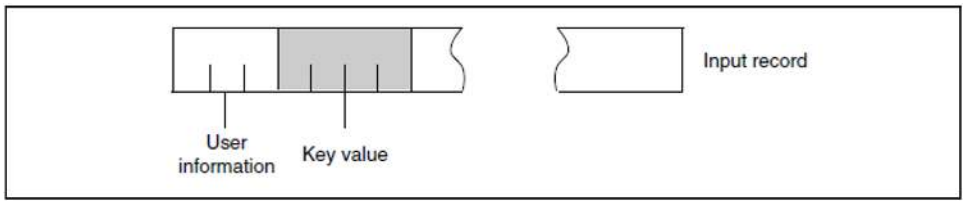


Figure 18: Example of the structure of an input file record

BINILOAD statements are used to specify the position and length of the items to be transferred to the database record (see table 23 RECORD DISPL statement, "Statements for BINILOAD").

The item contents are stored in the database in the format in which they appear in the records of the input file. This means that the item contents specified are not converted to the type defined for these items in the Schema DDL (see "Defining an alphanumeric item of fixed length" in the "Design and Definition" manual).

The input file may be stored on disk or tape. In both cases the input file can be a SAM or ISAM file (EDT format) containing records of the same length in either fixed or variable record format (RECFORM=V or F). (See table 21 USER RECORD LENGTH statement, "Statements for BINILOAD".)

If the input comes from an uncataloged tape file, STATE=FOREIGN must also be specified in the /CREATE-FILE command for the input file.

The name of the input file is specified in the INPUT-FILE statement, its record length in the USER RECORD LENGTH statement.

At the end of this chapter there is an example for the BINILOAD command sequence, followed by an example of a BINILOAD input file.

5.1.3 Reading the input file in CSV format and preparing the BINILOAD run

BINILOAD can load data from CSV files, generated by BOUTLOAD or created by any tools, according to the CSV file format rules.

An input file in CSV format generated by BOUTLOAD may contain lines with the following information:

```
(1)  BOUTLOAD;CSV V1.20;2018-01-16;07:45:27;

(2)  DBNAME;DATABASE NAME;BINICSV;

(3)  INFO01;RECORD NAME;REC-001;

(4)  INFO02;RECORD REF;2;

(5)  INFO03;REALM NAME;AREA1

(6)  INFO04;REALM REF;3

(7)  FIELDS;DB Key Ref;DB Key RSQ;Member SYS-1;Owner DB Key Ref S1;Owner DB Key RSQ
     S1;R1;R2;

(8)  RECORD;2;1;Y;3;2;”AAA”;1;
```

The line (1) may contain the name of the BOUTLOAD utility routine, the corresponding utility routine’s output format version, and the date and time of creation of the CSV output.

The next lines (2) – (6) contain database name, record name and record reference. If realm name was specified, the header also contains realm name and realm reference.

Lines (1)-(6) are optional in CSV file while loading by BINILOAD and ignored.

The header line (7) and the content line (8) are mandatory, but control fields with name FIELDS and RECORD are optional. If control field FIELDS is present, then control field RECORDS must be present as well, and vice versa.

The header line (7) and lines (1) - (6) can be in any order in the beginning of CSV file, but not be mixed with content lines.

The header line should contain name of fields according to the following naming rules:

- “DB Key Ref” and “DB Key RSQ” – field names for the database key of the record

  These fields are optional.

  If "DB Key Ref" is not present, BINILOAD will then use the record reference number (REC-REF) of the record type specified for STORE RECORD.

  If both "DB Key Ref" and "DB Key RSQ" are not present, database key values are to be assigned in the same sequence as the order of records in the CSV file (see section "Assigning the database key value to a record" in chapter "Statements for BINILOAD").
- “Member *set-name*” – field name for a one-byte long item with the content

  Y = Member inserted into the SYSTEM set set-name

  N = Member not inserted into the SYSTEM set set-name

  This field is mandatory for all singular sets specified in a INSERT statement.

- “Owner DB Key Ref *set-name*” and “Owner DB Key RSQ *set-name*” - field names for the database key of the owner in *set-name*.

  For each regular set set-name specified in a INSERT statement the field "Owner DB Kef Ref set-name" is optional, the field "Owner DB Kef RSQ set-name" is mandatory.

  If "Owner DB Kef Ref set-name" is not present, BINILOAD will then use the record reference number (REC-REF) of the owner record type of the set specified in the INSERT statement.

  (see section "Format 3: Using the database key to define the owner" in chapter "Statements for BINILOAD").

- The item names according to the user schema;
  If an item name of the schema is omitted, a default value is used for it. This default value is blank X'40' for alphanumeric fields, unicode blanks X'0020' for national fields, low value for DBKEY fields and 0 in the respective numeric format (unpacked, packed or binary).
- Area ref - in the case of record type which is distributed to realms if its records are copied from multiple realms. This field is ignored in BINILOAD.

A semicolon (";") is used to separate the individual values.

Values in content line (8) must correspond to the order of fields in the header line (7).

Fields in CSV file can be in arbitrary order.

The correct order of fields is determined based on fields’ names in the header line.

Alphanumeric values in CSV can be enclosed in double quotes (") to represent special characters like semicolon (;). If a double quote character is contained in a value, which is enclosed in double quotes, it must be duplicated.

The items’ contents are converted to the type defined for these items in the Schema DDL and are stored in the database in the correct format.

In case if item content can’t be converted to the item type due to incorrect value in CSV file, error 908 will be output, the current record will not be stored and BINILOAD will continue to process with the next line in the CSV file.

If a field of DB record is missed in CSV file, then BINILOAD will fill the field by default value (0, X’40’ or X'0020' depending on data type).

If there is a field in CSV, which doesn’t exist in DB record type, and doesn’t correspond any field of system information part (like ‘DB Key Ref’ , ‘DB Key RSQ, ‘DB Key Ref set-name’, ‘DB Key RSQ set-name’, ‘Member set-name’, then BINILOAD will reject loading.

The input file may be created by BOUTLOAD or by any tool. The input CSV file can be a SAM or ISAM file (EDT format) containing records of different length in variable record format (RECFORM=V).

The name of the input file is specified in the INPUT-FILE statement.

For loading data from an input file in CSV format, the DBCOM must be available.

At the end of this chapter there is an example for the BINILOAD command sequence for loading data from an input file in CSV format.

5.1.4 BINILOAD system environment

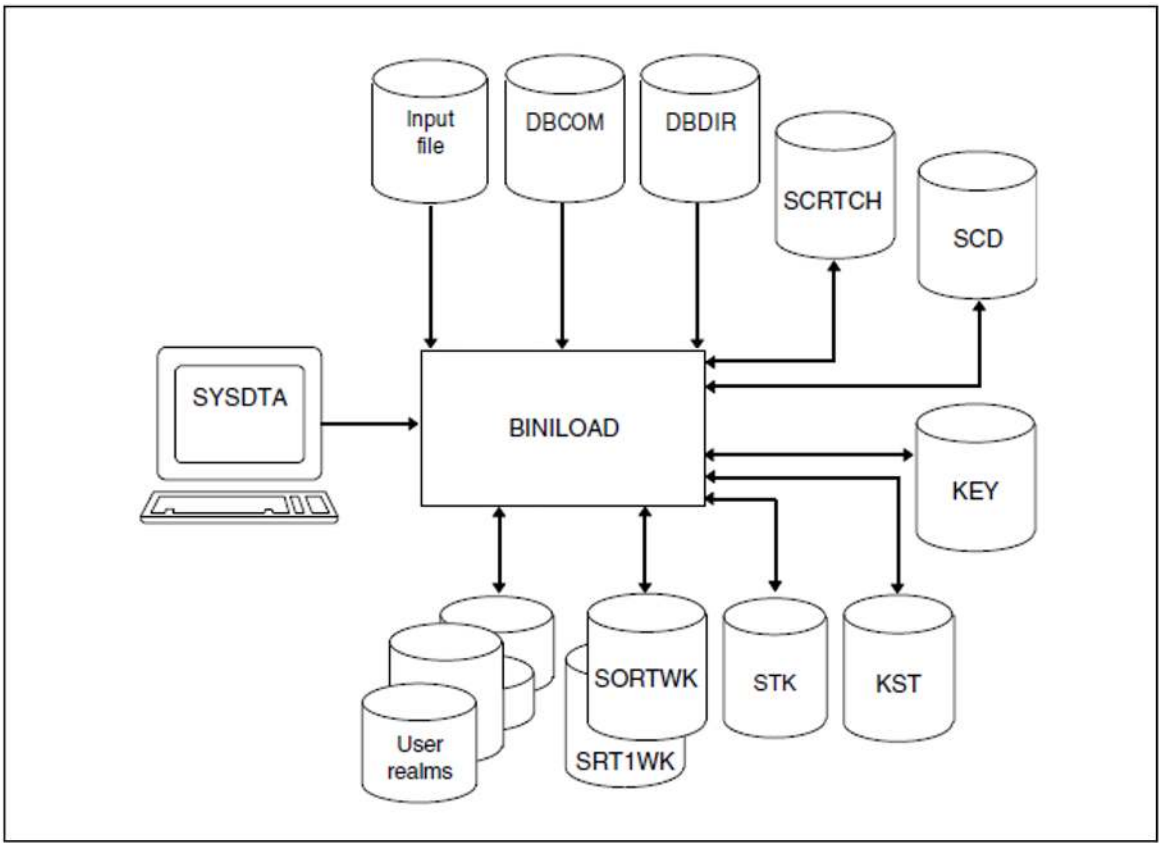


Figure 19: System environment for BINILOAD

BINILOAD requires several work files, which it creates automatically in the correct size on public volumes under the name UTI.SAMWORK.*tsn.timestamp* and deletes again following normal termination of the loading operation.

The files have the following default link names:
SCRTCH1, SCRTCH2, SCRTCH3, SORTWK, SRT1WK, SCD*nnnnn*, STK*nnnnn*, KEY*mmmmm* and KST*nnnnn*.

SCRTCH1	contains a follow-up version of the input file during execution.
SCRTCH2 SCRTCH3	are used to allocate space for records to be stored.
SORTWK SRT1WK	requires the SORT used by BINILOAD for sorting internal evaluation records (see the manual “SORT (BS2000)”).
SCD*nnnnn*	contain, during execution of BINILOAD, the SCD information of the records for the set with the five-digit set number *setref*, of which the record type to be loaded is a member.
STK*nnnnn*	contain, during execution of BINILOAD, the SEARCH key information of the records for the set with the five-digit set number *setref*, of which the record type to be loaded is a member.
KEY*mmmmm*	contain, during execution of BINILOAD, the keys of the records for the key with the five-digit key number *keyref*, from which the access tables are to be set up.

KST*nnnnn*	contain, during execution of BINILOAD, storage information for the records for the set with the five-digit set number *setref*, for which there is no user-defined sort key. No key numbers *keyref* are identified for the set in the BPSIA log.

Database recovery

BINILOAD writes after-images if AFIM logging has been previously specified for the current database using the utility routine BMEND (see "BMEND" in the "Recovery, Information and Reorganization" manual).

ALOG files

If AFIM logging has been turned on, the current ALO file must be present. If an error on the ALOG file occurs during the execution of BINILOAD or the ALOG file overflows, AFIM logging is turned off, and BINILOAD continues to completion without ALOG files. A logging gap results.

On completion of the BINILOAD run, the ALOG file is switched, i.e. a new ALOG file isset up.

5.1.5 Statements for BINILOAD

In order to execute BINILOAD, a series of statements must be specified. BINILOAD recognizes four kinds of statements:

- Control statements
- Program statements
- STORE statements
- INSERT statements

Statements which are optional are indicated as such. It is advisable to observe the sequence of statements, even if certain statements are not used. The sequence is mandatory for STORE and INSERT statements.

Control statements

These control execution of the UDS/SQL utility routine BINILOAD.

Statement	Default value	Meaning
[EXECUTION {WITH | WITHOUT} CHECK.]	WITH	Checks/does not check input data
[SORTCORE IS *nnn*.]	150	Specifies size of main memory for sort/merge routine

Table 21: Control statements for BINILOAD

Program statements

These determine the schema, subschema, input file, and the occupancy level of tables.

Statement	Default value	Meaning
SCHEMA NAME IS *schema-name*. SUBSCHEMA NAME IS *subschema-name*.	-	Name of schema and subschema
FILLING IS *nnn* PERCENT.	-	Specifies occupancy level for table pages
USER FILE RECORD LENGTH IS *n*.	-	Length of input records in bytes
USER FILE BUFFER LENGTH IS *n*.	-	Block length of input file; must be a multiple of 2048
INPUT FILE NAME '*file-name*' [FORMAT IS CSV].	-	File name of input file
INPUT RECORDNUMBER IS n.	-	None, only tolerated for reasons of compatibility

Table 22: Program statements for BINILOAD

In case if FORMAT IS CSV is specified, the statement **USER FILE RECORD LENGTH** is not allowed, as the CSV file is a variable-length file with maximal length of 32752 bytes, and each time actual record length will be taken from the file record. The **USER BUFFER LENGTH** statement is also not allowed; buffer length from the file properties will be used.

STORE statements

These provide BINILOAD with information on the record type and its relation to the input records.

Statement	Default value	Meaning
STORE RECORD NAME IS *record-name*.	-	Record type to be stored
RECORD-DBKEY IS DISPL IS n, LENGTH IS {4 | 8}	-	Assigns database key value; • displacement and length of the database key value
RECORD-RSQ IS DISPL IS n, LENGTH IS {3 | 6}	-	Assigns database key value; • displacement and length of the record sequence number (RSQ) The associated record reference number (REC-REF) is determined by BINILOAD.
RECORD-DISPL IS *n*, {DISPL IS *n*,LENGTH IS *n* | VALUE IS '*literal*' }.	-	Structures database record. For specified record type; • displacement and length of the items of this record • character string to be inserted in the database records
RECORD-AREA NAME IS *realm-name*.	-	Realm into which the records are to be loaded.

Table 23: STORE statements for BINILOAD

In case if FORMAT IS CSV is specified, statements for the description of the record RECORD-DBKEY, RECORD-RSQ and RECORD-DISPL are not allowed.
In case of FORMAT IS CSV no fix displacements are used, but the position of the DB-KEY (Ref and RSQ) are determined by the header line.

INSERT statements

These indicate to BINILOAD the sets into which the records are to be inserted.

Statement	Default value	Meaning
INSERT INTO SET NAME IS *set-name*.	-	Specifies the set into which the records are to be inserted as member records.
SET ORDER {USING {DISPL IS *n*, LENGTH IS *n* | FIELD NAME IS *field-name*|*'field-name'*} | VIA USER FILE SEQUENCE}.	VIA USER FILE SEQUENCE	Specifies sort sequence of records within the sets with ORDER IS FIRST, LAST, NEXT, PRIOR, IMMATERIAL; specifies length of sort item. FIELD NAME IS is only allowed when FORMAT IS CSV is specified. *field-name* is a field name used in CSV header line. It must be specified in single quotes if field name consists of several parts separated with spaces. For the database key fields name can be specified as 'DB Key Ref' resp. 'DB Key RSQ'. For items of datatype DBKEY, the field name can be specified as 'item-name:DB Key Ref' resp. 'item-name:DB Key RSQ'.
OWNER CALCKEY IS {DISPL IS *n*, LENGTH IS *n* | VALUE IS *'literal'*}, AREA NAME IS *realm-name*.	-	Selects set occurrence by selecting the owner • displacement and length of the CALC key values in the input file records by means of which the owner is to be selected • character string with CALC key • name of the realm in which the owner record is stored.

OWNER SEARCHKEY IS {DISPL IS *n*, LENGTH IS *n* | VALUE IS '*literal*'}, [VIA SET NAME IS *set-name*,] SEARCHKEY TABLE {COLUMN-NR IS *n* | ORDER-NR IS *keyref*}.	-	Selects set occurrence by selecting the owner via SEARCH key • displacement and length of the SEARCH key values in the input file records by means of which the owner is to be selected • character string with SEARCH key table • name of the SYSTEM set in which the owner is a member • DBTT column number of SEARCH key table • key reference number.
OWNER DBKEY IS {DISPL IS *n*, LENGTH IS {4 | 8} | VALUE IS *dbkey* }.	-	Selects set occurrence by selecting the owner via its database key value: • displacement and length of the database key value in the input file records by means of which the owner is to be selected • character string with database key value.

OWNER RSQ IS {DISPL IS *n*, LENGTH IS {3 | 6} | VALUE IS *rsq* }.	-	Selects set occurrence by selecting the owner via its database key value: • displacement and length of the record sequence number (RSQ) in the input file records by means of which the owner is to be selected • character string with record sequence number (RSQ). The associated record reference number (REC-REF) is determined by BINILOAD.
OWNER KEY IS DISPL IS *n*,LENGTH IS 1.	-	Position of the item in the input records, which specifies whether or not the record is to be inserted in the SYSTEM sets.

Table 24: INSERT statements for BINILOAD

In case if FORMAT IS CSV is specified, in the SET ORDER statement order can be specified using item specified by item-name, via user file sequence, but specification of displacement is not allowed.
OWNER statements (OWNER CALCKEY, OWNER SEARCHKEY, OWNER DBKEY, OWNER RSQ and OWNER KEY) are only allowed, if FORMAT IS CSV is not specified.

EXECUTION (Checking/not checking input data)

The EXECUTION statement is optional.

```
EXECUTION { WITH | WITHOUT } CHECK.
```

WITH — Before changes are made to the database, BINILOAD checks whether the input data and the database structure are compatible and whether there is enough space available in the database. If it finds discrepancies between input data and the structure of the database, it issues appropriate messages and terminates the run. BINILOAD stores the tables and records only if the input data is compatible with the structure of the database.

If there is not enough space in a realm, this is indicated at the end of the test run by the following runtime message:

`MODIFY-REALM-SIZE <realm-name>, DIFFERENCE = n .`

However, the program run is not aborted if SECONDARY_ALLOCATION > 0 is set in the realm as the free space required is then obtained by means of automatic realm extension (see also the "Database Operation" manual).

No automatic DBTT extension by the BINILOAD utility routine takes place.

WITHOUT — BINILOAD suppresses the check run and stores the data from the input file immediately.
If errors occur, the BINILOAD run is abnormally terminated and appropriate messages are issued.
The database is then inconsistent

Default value:

WITH

BINILOAD does not check the actual contents of the input file.

SORTCORE (Specifying the size of the sort buffer)

The SORTCORE statement is optional.

```
SORTCORE IS nnn.
```

nnn Specifies, in units of 4 Kbytes, the size of the memory space for the sort buffer assigned to the BS2000 SORT utility routine (see the "ALLOC statement" in the "SORT (BS2000)" manual). The population of the data that is to be sorted is the same as that on which the size of the work files with the link names SORTWK and SRT1WK is based (see "Creating work files").
Default value:150

SCHEMA (Specifying the name of the schema)

The SCHEMA statement is optional.

```
SCHEMA NAME IS schema-name.
```

schema-name

is the schema name specified in the Schema DDL

SUBSCHEMA (Specifying the name of the subschema)

The SUBSCHEMA statement is mandatory.

```
SUBSCHEMA NAME IS subschema-name.
```

subschema-name

is the subschema name specified in the Subschema DDL

With the aid of the names specified, BINILOAD obtains information on the database in which the new records are to be stored.

FILLING (Specifying the occupancy level of table pages)

The FILLING statement is optional.

```
FILLING IS nnn PERCENT.
```

nnn Specifies the percentage of filling for table pages on level 0.
This allows for future insertions in these data elements. Pages not used for tables are filled to the maximum.

nnn = 1 ... 100

For table pages on level 1, the default level of occupancy is 95%. On every higher level, one table entry is left free.

If FILLING is omitted, one entry is left free on level 0 as well.

If *nnn* is made too small, BINILOAD makes sure that there is room for at least one entry.

USER RECORD LENGTH (Specifying the length of the input records)

The USER RECORD LENGTH statement is mandatory in case if input file isn’t in CSV format.

In case if input file is in CSV format the statement USER FILE RECORD LENGTH is not allowed, as the CSV file is a variable-length file with maximal length of 32752 bytes, and each time actual record length will be taken from the file record.

```
USER FILE RECORD LENGTH IS n.
```

n Total length of an input file record in bytes.
The input file records may contain, in addition to the item contents of the records to be stored in the database, additional user information and control information.

If "variable" record format is specified for the input file (RECFORM=V), the length specification will be the record length minus the record length item (RECSIZE - 4).

n > 0

USER BUFFER LENGTH (Specifying the block length of the input file)

You can omit the USER BUFFER LENGTH statement if the input has been generated with fixed record length (RECFORM=F).

In case if input file is in CSV format the USER BUFFER LENGTH statement is not allowed; buffer length from the file properties will be used.

```
USER FILE BUFFER LENGTH IS n.
```

n Block length of the input file in bytes.
BINILOAD sets up a buffer of the specified length.

n must be a multiple of 2048.

INPUT FILE (Specifying the name of the input file)

The INPUT FILE statement is mandatory.

```
INPUT FILE NAME IS 'file-name' [FORMAT IS CSV].
```

' *file-name* '

Is the name of the input file containing the records to be stored. The file can be a SAM or ISAM file (EDT format). It may contain not only records of variable length (RECFORM=V), but also records of fixed length (RECFORM=F).

If FORMAT IS CSV is specified, it must be a SAM file with records of variable length (RECFORM=V).

file-name must be specified in literal format, since the name can be qualified at multiple levels.

STORE RECORD (Specifying the record type)

The STORE RECORD statement is mandatory.

```
STORE RECORD NAME IS record-name.
```

record-name

Is the name of the record type whose records are to be stored in the database. The name must be defined in the appropriate schema and subschema.

RECORD-DBKEY (Assigning the database key value to a record)

If you want to explicitly assign the database key value for each record of the record type specified for STORE RECORD (see section "STORE RECORD (Specifying the record type)"), you must specify either the RECORD-DBKEY statement or the RECORD-RSQ statement as follows:

- If you are using RECORD-DBKEY, specify the complete database key value in the input file.
- If you are using RECORD-RSQ, specify only the record sequence number (RSQ) in the input file. BINILOAD will then use this RSQ to determine the database key value of the input record and the record reference number (REC-REF) of the record type which you specified for STORE RECORD.

If the database key values are to be assigned in the same sequence as the order of records in the input file, the RECORD-DBKEY or RECORD-RSQ statement is optional.

In case if input file is in CSV format the RECORD-DBKEY or RECORD-RSQ statement are not allowed.

RECORD-DBKEY statement

```
RECORD-DBKEY IS DISPL IS n, LENGTH IS {4 | 8}
```

DISPL IS *n*

specifies the displacement within the input record (relative to the beginning of the record) of the database key value to be assigned.

LENGTH IS { 4 | 8 }

The length of a database key value is always 4 or 8 bytes.
Database key values of 8-byte length with a record reference number (REC-REF) > 254 and/or a record sequence number (RSQ) > 2^{24} -1 can only be used for input in databases with a page length of 4000 or 8096 bytes.

RECORD-RSQ statement

```
RECORD-RSQ IS DISPL IS n, LENGTH IS {3 | 6}
```

DISPL IS *n*

specifies the displacement within the input record (relative to the beginning of the record) of the record sequence number (RSQ) to be assigned.

LENGTH IS { 3 | 6 }

The length of a record sequence number (RSQ) is always 3 or 6 bytes.
Record sequence numbers with a length of 6 bytes and a value > 2^{24} -1 can only be used for input in databases with a page length of 4000 or 8096 bytes.

RECORD-DISPL (Creating the database record)

The RECORD-DISPL statement is mandatory if the input file records contain, in addition to the database records, user information and control information or items for other record types, i.e. if items are to be relocated.

It is to be specified an appropriate number of times if several items are to be transferred from the input record to appropriate locations in the database record.

It is optional if the input file records are identical to the records to be stored in the database; with "variable" record format each database record corresponds to the data part without the record length item.

In case if input file is in CSV format the RECORD-DISPL statement is not allowed.

```
RECORD-DISPL IS n, {DISPL IS n,LENGTH IS n | VALUE IS 'literal'}.
```

RECORD-DISPL IS *n*

specifies the displacement within the database record (relative to the beginning of the record) of the item to be transferred.

DISPL IS *n*

specifies the displacement within the input record (relative to the beginning of the record) of the item to be transferred.
With variable-length records the record length item must be disregarded.

LENGTH IS *n*

specifies the length of the item to be transferred.

VALUE IS ’*literal* ’

specifies a value which is inserted in each stored record at the location specified by *n* in the RECORD-DISPL statement.

The literal may be:

- a character string, e.g. ’date’ (max. 64 bytes)
- a hexadecimal character string, e.g. ’014F’X, ’FFFF’X, etc.
  (max. 32 bytes)

If an apostrophe is to be included in a character string, two apostrophes should be entered.

You can enter up to five RECORD-DISPL statements with a VALUE clause.

Example of the RECORD-DISPL statement:

RECORD-DISPL IS 0

0 is the number of bytes between the beginning of the database record and the first byte to which the item is to be transferred.

DISPL IS 3

3 is the number of bytes (displacement) between the beginning of the input record and the first byte of the item to be transferred.

LENGTH IS 6

6 is the length of the item to be transferred.

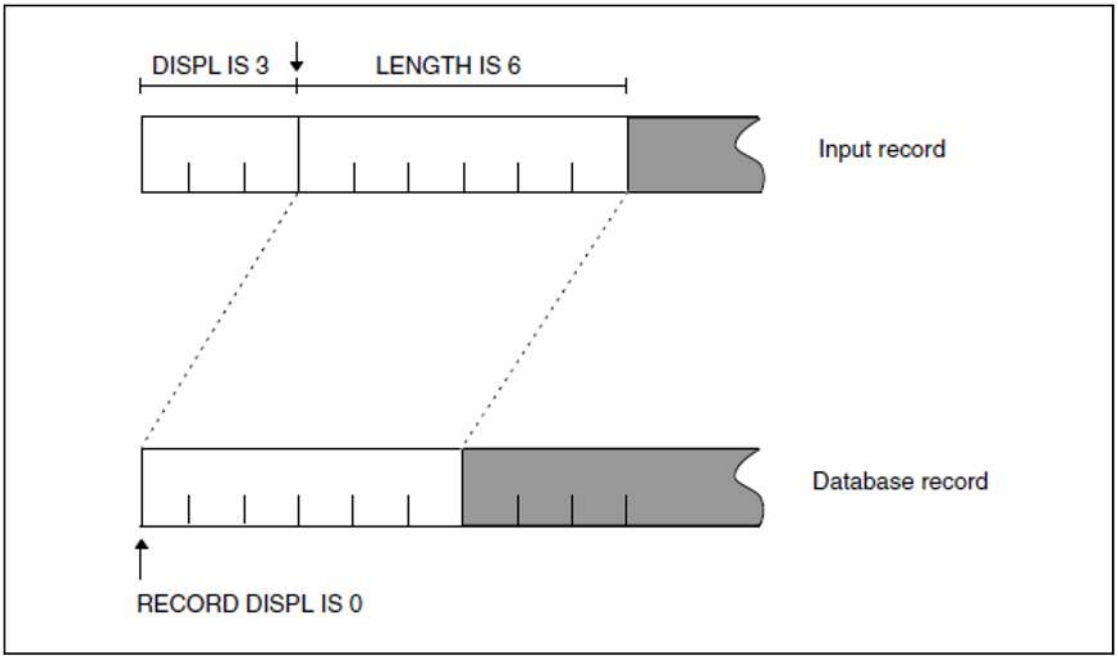


Figure 20: Input record and database record with the item to be transferred

Every RECORD-DISPL statement generates or extends a MOVE statement. Parts of the input record are inserted (character strings are inserted if VALUE is specified). If more than one RECORD-DISPL statement is specified, they are executed in the sequence given. Parts of the record inserted by previous RECORD-DISPL statements can be overwritten.
The RECORD-DISPL statement must not refer to a displacement position outside the database record. There is no check to ascertain whether the value of the literal or the type of input record matches the element in the database record as defined in the schema. BINILOAD likewise performs no conversions.

RECORD-AREA (Specifying the realm)

The RECORD-AREA statement is mandatory if the WITHIN clause of the Schema DDL contains more than one realm name and the record type to be stored is not the member record type of a distributable list.

The RECORD-AREA statement can optionally be specified in the following cases:

- When the WITHIN clause of the Schema DDL contains only one realm name
- When the record type to be stored is a member of a distributable list:

  If no realm is specified for distributable lists, BINILOAD stores the records approximately evenly in all realms which are specified in the WITHIN clause of the Schema DDL.
  If a realm is specified for distributable lists with the RECORD-AREA statement, this must be the table realm. The records are then stored in this realm. The associated list remains distributable.

When the RECORD-AREA statement is used to specify a realm, this realm must be defined in the subschema. If the record type to be stored is the member record type of a distributable list, all realms of the Schema DDL's WITHIN clause must be defined in the subschema.

```
RECORD-AREA NAME IS realm-name.
```

realm-name
: Name of the realm into which the records are to be loaded.

> The SSL specifications must be observed!
> If the record type to be stored is a member of a set specified with MODE IS LIST but without DETACHED WITHIN *realm-name*, or if it is a member of a set specified with PLACEMENT OPTIMIZATION, the owners of the set occurrences to be stored must also be contained in the realm specified for the member record type.

INSERT (Specifying the set)

Wether or not you specify the INSERT statement depends on the membership of the member records in the set (see the "Design and Definition" manual).

- Standard set MANDATORY AUTOMATIC
  You must specify the INSERT statement followed by the OWNER statement.
- Standard set OPTIONAL or MANUAL
  - You must specify the INSERT statement if all or some of the records are to be inserted; the OWNER statement must then follow.
  - You must omit the INSERT statement if none of the records is to be inserted into the set occurrence; the OWNER statement is also omitted.
- SYSTEM set MANDATORY AUTOMATIC
  You must specify INSERT, but omit the subsequent OWNER statement.
- SYSTEM set OPTIONAL or MANUAL
  - You must specify INSERT if only some of the records are to be inserted; the OWNER statement must then follow.
  - You must specify INSERT without a subsequent OWNER statement if all the member records are to be inserted.
  - You must omit INSERT if none of the records is to be inserted into the set occurrence of the SYSTEM set; the OWNER statement is also omitted.
  - The set must be defined in the specified subschema.
- SYSTEM set IMPLICIT
  You must omit INSERT.

```
INSERT INTO SET NAME IS set-name.
```

set-name

Specifies in which set the input file records are to be inserted as members.

> The INSERT and OWNER statements must be specified if BOUTLOAD generates the set connection data (SCD) when unloading and subsequently BINILOAD is to restore the old set memberships.

SET ORDER (Specifying the sort sequence)

You have the option of specifying the statement if sorting within the set was defined with FIRST, LAST, NEXT, PRIOR or IMMATERIAL (see the "Design and Definition" manual) in the ORDER clause of the Schema DDL and the sequence of the records in the set occurrence does not match the sequence in the input file.

In this case the sequence within the set occurrence can be specified during loading with BINILOAD by defining a sort item in each input record. The content of this item is used to sort the member records in ascending order.

You need not specify the SET ORDER statement if the database records to be stored occur in the same sequence in the input file as they are to be inserted in the set occurrence.

You should not specify the statement if sorting within the set was defined with SORTED, SORTED INDEXED in the ORDER clause of the Schema DDL.

```
SET ORDER {USING {DISPL IS n, LENGTH IS n | FIELD NAME IS item-name} | VIA USER FILE
SEQUENCE}.
```

USING DISPL IS *n*

specifies the displacement (relative to the beginning of the record) of the sort item in the input record.
In case if input file is in CSV format the option **USING DISPL IS n** is not allowed.

USING FIELD NAME IS *item-name*

specifies the name of the sort item in the input record. Allowed only if input file is in CSV format.

VIA USER FILE SEQUENCE

causes the sequence of records in the input file to be retained in the set occurrences.

Default value

VIA USER FILE SEQUENCE

The SET ORDER statement must precede the associated INSERT statement.

OWNER (Defining the owner)

If input file is not in CSV format, you must specify the OWNER statement for all sets other than SYSTEM sets, irrespective of whether the sets have MANUAL or AUTOMATIC members, if an INSERT statement has previously been entered.
With CSV format the owner is determined by its DB-KEY according to the header line.

BINILOAD can be used to define the owner if you specify the following values in one of the formats 1 through 3 (with SYSTEM sets format 4 applies):

Value	Condition	Format
CALC key	-	Format 1
ASC-/DESC key	if the owner is, at the same time, member in a SYSTEM set	Format 2
SEARCH key	if the SEARCH key was defined at record type level	
DB key	-	Format 3

All key values can be specified as the content of an item in the input records, or as a literal in the OWNER statement for this set. If the key value is specified as a literal, all the records of the input file are assigned to the same owner.

If the owner set is the member record type of a distributable list, the realms of the owner record type's DDL-WITHIN clause must be defined in the subschema.

> If DUPLICATES ARE ALLOWED has been specified in the Schema DDL and CALC, ASC/DESC or SEARCH keys have been used, duplicate key values may occur, in which case it is impossible to predict which owner records BINILOAD will select.
> If in a MANUAL or OPTIONAL set certain member records are not to be inserted, HIGH-VALUE should be entered in the item for owner selection in the input file.

Format 1: Using the CALC key to define the owner

In case if input file is in CSV format the OWNER CALCKEY statement is not allowed.

```
OWNER CALCKEY IS {DISPL IS n, LENGTH IS n | VALUE IS 'literal'},
    AREA NAME IS realm-name.
```

DISPL IS *n*

specifies the displacement in the input record of the item containing the CALC key.

LENGTH IS *n*

specifies the length of the item containing the CALC key (length of the CALC key).

VALUE IS ' *literal* '

specifies the CALC key which selects the owner for all records of the input file.

realm-name

designates a realm specified in the DDL WITHIN clause of the owner record type. If the owner set is the member record type of a distributable list, its table realm must be specified here as the indirect CALC area is located there.

The AREA entry is mandatory in format 1.

Format 2: Using the SEARCH key to define the owner

In case if input file is in CSV format the OWNER SEARCHKEY statement is not allowed.

```
OWNER SEARCHKEY IS {DISPL IS n, LENGTH IS n | VALUE IS 'literal'},
    [VIA SET NAME IS set-name,]
    SEARCHKEY TABLE {COLUMN-NR IS n | ORDER-NR IS keyref}.
```

DISPL IS *n*

is the displacement in the input record of the item containing the SEARCH key.

LENGTH IS *n*

s the length of the item containing the SEARCH key.

VALUE IS ' *literal* '

is the SEARCH key value which defines the owner for all records of the input file.

VIA SET NAME IS *set-name*

is the name of the SYSTEM set in which the owner record type is a member; may not be specified for SYSTEM sets created by the DDL compiler on the basis of a record SEARCH key.

SEARCHKEY TABLE

The owner is selected using the ASC/DESC or SEARCH key. The key value must be contained in a table. Consequently the ASC/DESC key can only be used for SYSTEM sets with MODE IS CHAIN (see the "Design and Definition" manual) if ORDER IS SORTED INDEXED was specified in the Schema DDL.

COLUMN-NR IS *n*

is the DBTT column number of the corresponding SEARCH key or sort key table (see "SIA PRINT REPORT" in the "Recovery, Information and Reorganization" manual).

ORDER-NR IS *keyref*

must be specified if the SEARCH key of the owner is a CALC-SEARCH key. *keyref* specifies the key number, which is obtained from the sequence in which the keys were defined within the record type description or set description in the DDL.

It can also be used instead of the COLUMN-NR option.

Format 3: Using the database key to define the owner

In case if input file is in CSV format the OWNER DBKEY statement is not allowed.

The OWNER DBKEY and OWNER RSQ statements can be used to determine the owner record by means of its database key value:

- If you are using OWNER DBKEY, specify the complete database key value of the owner record.
- If you are using OWNER RSQ, specify the record sequence number (RSQ) of the owner record. BINILOAD will then use this RSQ to determine the database key value of the owner record and the record reference number (REC-REF) that is assigned to the owner record of set that you listed in the last specified INSERT statement (see section "INSERT (Specifying the set)").

OWNER DBKEY statement

```
OWNER DBKEY IS {DISPL IS n, LENGTH IS {4 | 8} | VALUE IS dbkey}.
```

DISPL IS *n*

specifies the displacement in the input record of the item containing the database key value.

LENGTH IS {4 | 8}

is the length of the database key value.
You must supply the database key value in the specified length in the records of the input file: the item defining the owner record must contain the database key value in binary. For more information on the binary representation of database key values, see the "Design and Definition" manual.
If the database key value of the owner record contains a record reference number (REC-REF) > 254 and/or a record sequence numbers (RSQ) >2^{24} -1, you must specify ”LENGTH IS 8“.

VALUE IS *dbkey*

is the database key that selects the owner record for all records of the input file. This database key value must be specified as follows:
record reference number (REC-REF) : record sequence numbers (RSQ)

The following applies to the value range of REC-REF and RSQ:

- for “LENGTH IS 4“: 1 < REC-REF <= 254 and 0 < RSQ <= 2^{24}-1
- for “LENGTH IS 8“: 1 < REC-REF <= 2^{15}-1 and 0 < RSQ <= 2^{31}-1

Example for the input of a database key value

The database key value with a REC-REF = 22 and an RSQ = 10 596 can be specified as follows:

1. In the VALUE IS clause:

```
VALUE IS  22 : 10596
          |       |
       REC-REF   RSQ
```

2. In the input file:
   - for ”LENGTH IS 4“: X 16002964

- for ”LENGTH IS 8“: X 0016000000002964

OWNER RSQ statement

In case if input file is in CSV format the OWNER RSQ statement is not allowed.

```
OWNER RSQ IS {DISPL IS n, LENGTH IS {3 | 6} | VALUE IS rsq}.
```

DISPL IS *n*

specifies the displacement in the input record of the item containing the record sequence number (RSQ).

LENGTH IS {3 | 6}

is the length of the record sequence number (RSQ).
You must supply the RSQ in the specified length in the records of the input file: the item defining the owner record must contain the RSQ in binary.

If the RSQ of the owner record is greater than 2^{24} -1, you must specify ”LENGTH IS 6“.

VALUE IS *rsq*

is record sequence number (RSQ) that selects the owner record for all records of the input file. *rsq* must be specified with the following value range:

- for ”LENGTH IS 3“: 0 < RSQ <= 2^{24} -1
- for ”LENGTH IS 6“: 0 < RSQ <= 2^{31} -1

Example for the input of a record sequence numbers (RSQ)

An RSQ = 10 596 can be specified as follows:

1. In the VALUE IS clause:

```
VALUE IS 10596
           |
         RSQ
```

2. In the input file:
   - for ”LENGTH IS 3“: X 002964
   - for ”LENGTH IS 6“: X 000000002964

Format 4: Defining set membership in the SYSTEM set

In case if input file is in CSV format the OWNER KEY statement is not allowed.

```
OWNER KEY IS DISPL IS n, LENGTH IS 1
```

DISPL IS *n*

is the displacement of the item in each input record that specifies whether the record is to be inserted into the SYSTEM set.

LENGTH IS 1

The length of the item is always 1.

Insert: X’00’ (LOW-VALUE)
Do not insert: X’FF’ (HIGH-VALUE)

If you do not specify the OWNER statement, all records are inserted into the SYSTEM set.

5.1.6 Command sequence for starting BINILOAD

It is assumed for the command sequence described here that UDS/SQL was installed with IMON (see section “START commands for the UDS/SQL programs”).

```
01 /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR
02 [/CREATE-FILE FILE-NAME=input-tape-file,...]
03 /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,
SCOPE=*TASK
04 /START-UDS-BINILOAD
05 biniload-statements
06 END
```

03 The specified version of BINILOAD is selected.
It is generally recommended that you specify the version since it is possible for several UDS/SQL versions to be installed in parallel.

04 The UDS/SQL utility routine can also be started with the alias BINILOAD.

> The BINILOAD statements are read in via SYSDTA! A file generated by BOUTLOAD can also be used for this purpose.

5.1.7 Creating work files

If you wish to create the work files explicitly, you must issue the appropriate CREATE-FILE commands. If you specify too little storage space (with SPACE), the value you specify will be corrected internally by BINILOAD.

```
/CREATE-FILE FILE-NAME=workfile-n [,SUPPORT ...]

/ADD-FILE-LINK LINK-NAME={SCRTCH1 | SCRTCH2 |SCRTCH3 |SCDnnnnn | STKnnnnn |
                          KEYnnnnn | KSTnnnnn | SORTWK | SRT1WK},
                         FILE-NAME=workfile-n[,ACCESS-METHOD={PAM | SAM}]
```

workfile-n Freely selected name for the work file

SUPPORT The storage space size can be specified with the SPACE entry in the SUPPORT operand:

ACCESS-METHOD

The work files SORTWK and SRT1WK are created as PAM files. The other work files are created as SAM files.

Link names for the work file

SCRTCH1
SCRTCH2
SCRTCH3
SCD*nnnnn*
STK*nnnnn*
KEY*nnnnn*
KST*nnnnn*
SORTWK
SRT1WK

Calculating primary requirements for work files

The primary allocation for work files should be based on the data population that is to be buffered. There should always be an appropriate secondary allocation in case it should be necessary to extend the storage space.

The approximate space requirements for individual work files can be calculated by using the formulas below.

SCRTCH1

(*total key lengths* + 12) x *number of input records* Bytes

total keylengths:
is the total length of the following keys:

- keys by which the owners of the set are selected
  (CALC keys, ASC/DESC keys, SEARCH keys or database keys)
- keys that do not belong to a set (CALC keys)
- keys for all sets in which these records are to be inserted
  (ASC/DESC keys, SEARCH keys).

SCRTCH2

```
12 x number of input records Bytes
```

SCRTCH3

```
3 x number of input records Bytes
```

SCD*nnnnn*

with 2048-byte page length:

```
40 x number of input records Bytes
```

with 4000/8096-byte page length:

```
50 x number of input records Bytes
```

STK *nnnnn*

with 2048-byte page length:

```
(8 + reclength_1) x number of input records Bytes
```

with 4000/8096-byte page length:

```
(12 + reclength_1) x number of input records Bytes
```

reclength_1:
reclength_1 is the sum of the key lengths of all SEARCH keys in the set withSET-REF *nnnnn*.

KEY*nnnnn* and for SEARCH key

with 2048-byte page length:

```
(16 + keylength_1) x number of input records Bytes
```

with 4000/8096-byte page length:

```
(24 + keylength_1) x number of input records Bytes
```

keylength_1:
Key length of the key with KEY-REF *nnnnn*

KEY *mmmmm* and KST *nnnnn* for SORT-Key

with 2048-byte page length:

```
(keylength_1 + 12 + keylength_2) x number of input records Bytes
```

with 4000/8096-byte page length:

```
(keylength_1 + 16 + keylength_2) x number of input records Bytes
```

keylength_1:
Length of the key used to specify the owner of the set.
This may be:
CALC keys, ASC-/DESC keys, SEARCH keys or database keys.

keylength_2:
Key length of the key with KEY-REF*mmmmm*
For KST*nnnnn* with SET-REF *nnnnn*: *keylength_1* = 0.

SORTWK and SRT1WK

SORT needs the two work files with the link names SORTWK and SRT1WK if there is not enough virtual memory for pre-sorting. The primary allocation should be based on the data population that is to be sorted while taking account of the safety factor recommended by SORT (see the discussion of work files in the manual "SORT (BS2000)"). There should always be an appropriate secondary allocation in case it is necessary to extend the storage space.

The volume of the data that is to be sorted is calculated using the formula:

`(`*`reclength_2`*`+SCD+12) x `*`number of input records`*` Bytes`

reclength_2:
is the length of the database record.

SCD:
Length of the Set Connection Data. You can take over this value from the BPSIA log where it is located in the SYSINFO column under the heading
RECORD-INFORMATION (see the manual "Recovery, Information and Reorganization").

When the load operation is complete, it is up to you to delete any work files that were explicitly created.

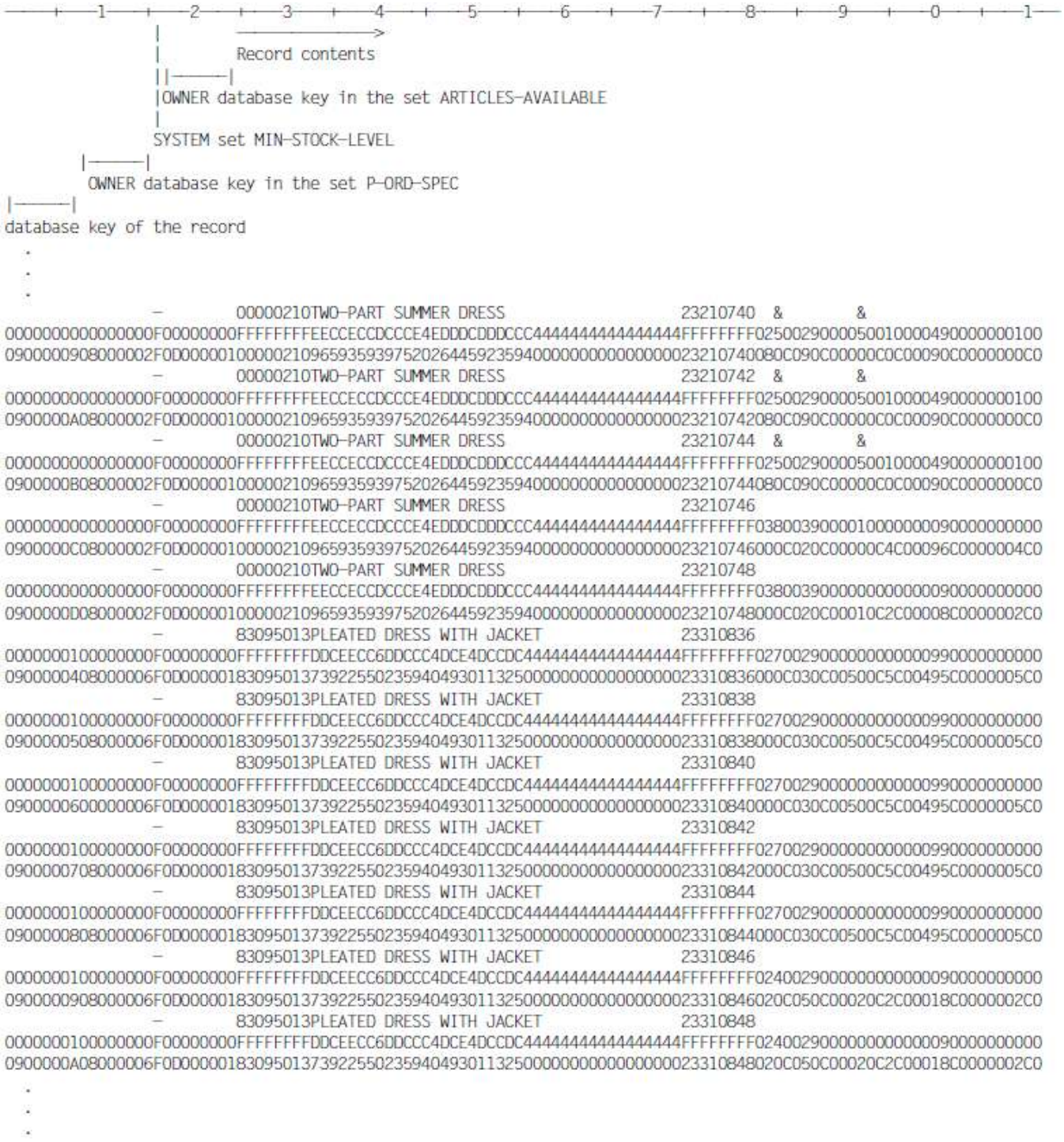
5.1.8 BINILOAD example

The record type ARTICLE is stored in the database SHIPPING. The file SHIPPING.REC00009.00005 generated by BOUTLOAD is used as the input file.

```
/ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=SHIPPING.DBDIR
/SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,    VERSION=02.9C00
/START-UDS-BINILOAD
***** START        BINILOAD     (UDS/SQL  V2.9  1801 )     2019-01-29   09:27:54
 EXECUTION WITH CHECK.
 SCHEMA NAME IS MAIL-ORDERS
 SUBSCHEMA NAME IS ADMIN
 USER FILE RECORD LENGTH IS  112
 USER FILE BUFFER LENGTH IS 8192
 INPUT FILE 'SHIPPING.REC00009.00005
 INPUT RECORDNUMBER IS       55
 STORE RECORD NAME IS ARTICLE
 RECORD-DBKEY IS DISPL IS 0 , LENGTH IS 8
 RECORD-DISPL IS 0 , DISPL IS   25 , LENGTH IS   87
 RECORD-AREA NAME IS CLOTHING
 INSERT INTO SET NAME IS P-ORD-SPEC
 OWNER DBKEY IS DISPL IS    8 , LENGTH IS 8
 INSERT INTO SET NAME IS MIN-STOCK-LEVEL
 OWNER KEY IS DISPL IS   16 , LENGTH IS 1
 INSERT INTO SET NAME IS ARTICLES-AVAILABLE
 OWNER DBKEY IS DISPL IS   17 , LENGTH IS 8
 END
 BEGIN CHECK-RUN
 *** DATE AND TIME 2019-01-29  09:27:54
 BEGIN ALLOCATION
 *** DATE AND TIME 2019-01-29  09:27:54

 SET_NAME: P-ORD-SPEC
 SET_REF:     7
 SORTKEY TABLE, DBTT_COLUMN_NR:     1
 *** ICRELES: MOVE_ROUTINE SORTKF CREATED
 CALCKEY TABLE - INDIRECT
 *** DATE AND TIME 2019-01-29  09:27:54
 BEGIN TABLE-PROCESSOR
 *** DATE AND TIME 2019-01-29  09:27:54

 SET_NAME: MIN-STOCK-LEVEL
 SET_REF:     8
 SORTKEY TABLE, DBTT_COLUMN_NR:     1
 BEGIN TABLE-PROCESSOR
 *** DATE AND TIME 2019-01-29  09:27:54

 SET_NAME: ARTICLES-AVAILABLE
 SET_REF:    12
 SORTKEY TABLE, DBTT_COLUMN_NR:     1
 BEGIN TABLE-PROCESSOR
 *** DATE AND TIME 2019-01-29  09:27:55
 SEARCHKEY TABLE, DBTT_COLUMN_NR:     2
 INDIRECT CALC-SEARCH-KEY BUCKETS
 INDIRECT CALC-SEARCH-KEY BUCKETS
 *** NO ERRORS DETECTED DURING CHECK-RUN
 END CHECK-RUN
 *** DATE AND TIME 2019-01-29  09:27:55
```

```
 BEGIN ALLOCATION
 *** DATE AND TIME 2019-01-29  09:27:55
 *** DATABASE IS IN USE

 SET_NAME: P-ORD-SPEC
 SET_REF:     7
 SORTKEY TABLE, DBTT_COLUMN_NR:     1
 *** ICRELES: MOVE_ROUTINE SORTKF CREATED
 CALCKEY TABLE - INDIRECT
*** DATE AND TIME 2019-01-29  09:27:55
 BEGIN TABLE-PROCESSOR
*** DATE AND TIME 2019-01-29  09:27:55

 SET_NAME: MIN-STOCK-LEVEL
 SET_REF:     8
 SORTKEY TABLE, DBTT_COLUMN_NR:     1
 BEGIN TABLE-PROCESSOR
 *** DATE AND TIME 2019-01-29  09:27:55

 SET_NAME: ARTICLES-AVAILABLE
 SET_REF:    12
 SORTKEY TABLE, DBTT_COLUMN_NR:     1
 BEGIN TABLE-PROCESSOR
 *** DATE AND TIME 2019-01-29  09:27:55
 SEARCHKEY TABLE, DBTT_COLUMN_NR:     2
 INDIRECT CALC-SEARCH-KEY BUCKETS
 INDIRECT CALC-SEARCH-KEY BUCKETS
 BEGIN STORE DB-RECORD
 *** DATE AND TIME 2019-01-29  09:27:55
 STORING DATABASE RECORDS
 END STORE DB-RECORD

 *** DATE AND TIME 2019-01-29  09:27:55
 END CLOSE
 *** DATE AND TIME 2019-01-29  09:27:55

 ***** DIAGNOSTIC SUMMARY OF BINILOAD

              NO WARNINGS
              NO ERRORS
              NO SYSTEM-ERRORS

              55 RECORDS  STORED

 ***** END OF DIAGNOSTIC SUMMARY
 ***** NR OF DATABASE ACCESSES  :          196
 ***** NORMAL END   BINILOAD     (UDS/SQL  V2.9  1801 )    2019-01-29   09:27:55
```

BINILOAD input file:




```
                -       00000210TWO-PART SUMMER DRESS                   23210740  &       &
0000000000000000F00000000FFFFFFFFEECCECCDCCCE4EDDDCDDDCCC4444444444444444FFFFFFFF0250029000050010000490000000100
0900000908000002F0D000001000002109659359397520264459235940000000000000000232107400800090C00000C0C00090C0000000C0
                -       00000210TWO-PART SUMMER DRESS                   23210742  &       &
0000000000000000F00000000FFFFFFFFEECCECCDCCCE4EDDDCDDDCCC4444444444444444FFFFFFFF0250029000050010000490000000100
0900000A08000002F0D000001000002109659359397520264459235940000000000000000232107420800090C00000C0C00090C0000000C0
                -       00000210TWO-PART SUMMER DRESS                   23210744  &       &
0000000000000000F00000000FFFFFFFFEECCECCDCCCE4EDDDCDDDCCC4444444444444444FFFFFFFF0250029000050010000490000000100
0900000B08000002F0D000001000002109659359397520264459235940000000000000000232107440800090C00000C0C00090C0000000C0
                -       00000210TWO-PART SUMMER DRESS                   23210746
0000000000000000F00000000FFFFFFFFEECCECCDCCCE4EDDDCDDDCCC4444444444444444FFFFFFFF0380039000010000000090000000000
0900000C08000002F0D000001000002109659359397520264459235940000000000000000232107460000200C00000C4C00096C0000004C0
                -       00000210TWO-PART SUMMER DRESS                   23210748
0000000000000000F00000000FFFFFFFFEECCECCDCCCE4EDDDCDDDCCC4444444444444444FFFFFFFF0380039000000000000090000000000
0900000D08000002F0D000001000002109659359397520264459235940000000000000000232107480000200C00010C2C00008C0000002C0
                -       83095013PLEATED DRESS WITH JACKET               23310836
0000000100000000F00000000FFFFFFFFDDCEECC6DDCCC4DCE4DCCDC4444444444444444FFFFFFFF0270029000000000009900000000000
0900000408000006F0D000001830950137392255023594049301132500000000000000002331083600000300C00500C5C00495C0000005C0
                -       83095013PLEATED DRESS WITH JACKET               23310838
0000000100000000F00000000FFFFFFFFDDCEECC6DDCCC4DCE4DCCDC4444444444444444FFFFFFFF0270029000000000009900000000000
0900000508000006F0D000001830950137392255023594049301132500000000000000002331083800000300C00500C5C00495C0000005C0
                -       83095013PLEATED DRESS WITH JACKET               23310840
0000000100000000F00000000FFFFFFFFDDCEECC6DDCCC4DCE4DCCDC4444444444444444FFFFFFFF0270029000000000009900000000000
0900000600000006F0D000001830950137392255023594049301132500000000000000002331084000003030C00500C5C00495C0000005C0
                -       83095013PLEATED DRESS WITH JACKET               23310842
0000000100000000F00000000FFFFFFFFDDCEECC6DDCCC4DCE4DCCDC4444444444444444FFFFFFFF0270029000000000009900000000000
0900000708000006F0D000001830950137392255023594049301132500000000000000002331084200003030C00500C5C00495C0000005C0
                -       83095013PLEATED DRESS WITH JACKET               23310844
0000000100000000F00000000FFFFFFFFDDCEECC6DDCCC4DCE4DCCDC4444444444444444FFFFFFFF0270029000000000009900000000000
0900000808000006F0D000001830950137392255023594049301132500000000000000002331084400003030C00500C5C00495C0000005C0
                -       83095013PLEATED DRESS WITH JACKET               23310846
0000000100000000F00000000FFFFFFFFDDCEECC6DDCCC4DCE4DCCDC4444444444444444FFFFFFFF0240029000000000009000000000000
0900000908000006F0D000001830950137392255023594049301132500000000000000002331084602003050C00020C2C00018C0000002C0
                -       83095013PLEATED DRESS WITH JACKET               23310848
0000000100000000F00000000FFFFFFFFDDCEECC6DDCCC4DCE4DCCDC4444444444444444FFFFFFFF0240029000000000009000000000000
0900000A08000006F0D000001830950137392255023594049301132500000000000000002331084802003050C00020C2C00018C0000002C0
```

For an example of how BINILOAD and BOUTLOAD work together, refer to "Examples".

5.2 Copying, deleting and unloading records with BOUTLOAD

BOUTLOAD allows you to copy, delete and unload record types from a database quickly.

You can use BOUTLOAD when you want to unload a database partly or fully and then reload it, or when you want to evaluate the data.
For restructuring purposes with BALTER, it is only necessary to unload record types in a few exceptional cases, e. g. if you want to include the DUPLICATES NOT ALLOWED clause in order to detect duplicates.

The files generated by BOUTLOAD can be reread by BINILOAD. Furthermore, BOUTLOAD outputs the control statements for a following BINILOAD run to SYSLST if the parameter SET-INFORMATION is set to YES. If SYSLST is assigned to a file, this file can be subsequently modified and used for BINILOAD.

BOUTLOAD can also unload records with items of variable length.

5.2.1 BOUTLOAD functions

You can use BOUTLOAD to perform the following functions:

- copy record types from a database to output files
- delete record types from a database
- unload record types from a database to output files.

Furthermore, all records of a record type stored in one realm can be copied to an output file.

It is possible to delete the contents of an entire database and then use BOUTLOAD to reformat the database.

When copying and unloading, BOUTLOAD stores the records of each specified record type in one output file each. These output files can at the same time be used as input files for BINILOAD.

BOUTLOAD copies or deletes the records of the specified record types in a single sweep, i.e. it not only handles several record types concurrently, but also fills the output files concurrently. In both cases, the consistency of the database is maintained.

Copying of record types with set information output is also possible parallel to a DBH run with retrieval access or with a shadow database.

Copying without set information output is also possible parallel to the DBH, provided the database is attached and has not been updated since mounting and up to termination of the BOUTLOAD run. You must employ appropriate organizational measures to ensure that this condition is respected. It is not checked by BOUTLOAD.

Copying record types (COPY-RECORD)

One or more, or all record types can be copied from a database in a BOUTLOAD run.

If no output files have been created, they are created by BOUTLOAD: one output file per record type.

The database remains unchanged.

BOUTLOAD reads the records in the physical sequence in which they are stored in thedatabase.

BOUTLOAD copies the user part of the database record. Compressed records are decompressed; records with variable items are filled with blanks to their maximum length; the length item for the variable item is maintained. BINILOAD cannot process a file generated by BOUTLOAD with variable items.

The structure of the output record is described on "Preparing the output files and the BOUTLOAD run".

Copying with SET-INFORMATION=NO is also possible for an inconsistent database.

Copying records of one record type from one realm (COPY-RECORD, REALM-NAME)

It is also possible in a BOUTLOAD run to copy the records of one or more record types from only one realm. When this is done, the area reference is not stored in the output records.

If you have not created any output files, they are created by BOUTLOAD, one output file per record type.

The realm remains unchanged.

> For sets with ORDER IS FIRST/LAST/NEXT/PRIOR/IMMATERIAL, the sorting sequence of the records in the set occurrences may change when they are reinserted.

Deleting record types (REMOVE-RECORD)

One or more record types can be deleted from a database in a BOUTLOAD run. The record types are deleted together with all pointers to the records in associated tables, owner records and DBTT entries.

When multiple record types are to be deleted, BOUTLOAD deletes them all simultaneously in a single pass through the database. The hierarchy in the database (member-owner relations) must be observed: Member record types must be deleted either before or together with the owner record types.

¡ AFIM logging is permitted for the REMOVE-RECORD function if individual record types are specified, but not for REMOVE-RECORD *ALL. If you specify REMOVE-RECORD *ALL, you will need to first turn off AFIM logging with BMEND if required.

Unloading record types (EXPORT-RECORD)

One or more record types can be unloaded from a database in a BOUTLOAD run.

This function is a combination of the copy and delete functions. As when deleting, the hierarchy in the database must be observed.

In the case of record types whose records are distributed across several realms, it is not possible to unload records from only one realm.

When unloading all record types the database is reformatted, just as when deleting all record types.

¡ AFIM logging is permitted for the EXPORT-RECORD function if individual record types are specified, but not for EXPORT-RECORD *ALL. If you specify EXPORT-RECORD *ALL, you will need to first turn off AFIM logging with BMEND if required.

Deleting and unloading all record types from a database (REMOVE/EXPORT-RECORD,RECORD-NAME=*ALL)

It is possible in a BOUTLOAD run to delete or unload all record types from a database. When this is done the database is reformatted. The schema and subschema structures of the database are retained. The FPA pages, DBTTs, CALC pages and anchor records are relocated
(see "BFORMAT", "Formatting user realms with BFORMAT").
A formatting run can be performed only if all realms of the database are available.

Formatting with REMOVE-RECORD,RECORD-NAME=*ALL is also permitted for an inconsistent database (e.g. following abnormal termination of a BOUTLOAD run).

¡ AFIM logging is permitted in REMOVE-RECORD and EXPORT-RECORD functionsif individual record types are specified, but not for REMOVE-RECORD *ALL or EXPORT-RECORD *ALL. If you specify REMOVE-RECORD *ALL or EXPORT-RECORD *ALL, you will need to first turn off AFIM logging with BMEND if required.

The current setting for the online DBTT extensibility of the record types is retained.

Rights of access with the individual functions

	only administrator id	RETRIEVAL	EXCLUSIVE
copy	-	x	-
delete	x	-	x
unload	x	-	x

Table 25: Rights of access with individual functions

If a database is to be completely unloaded, for reasons of efficiency it is often advisable to first execute multiple BOUTLOAD runs in parallel with the copy function and thereafter to execute a BOUTLOAD run with the delete function.

5.2.2 Preparing the output files and the BOUTLOAD run

The individual output files for BOUTLOAD can be created using the following commands:

```
/CREATE-FILE FILE-NAME=dbname.RECnnnnn[.mmmmm] [,SUPPORT= ...]
/ADD-FILE-LINK LINK-NAME=linkname,FILE-NAME=dbname.RECnnnnn[.mmmmm]
    [,BUFFER-LENGTH=xxx][,FILE-SEQUENCE=*NEW]
```

dbname

Name of the database to be processed.

nnnnn

5-digit record reference number with leading zeros.

mmmmm

5-digit area reference number with leading zeros; this specification is required if records are to be copied from only one realm.

SUPPORT

You can specify the size of the storage space by means of SPACE in the SUPPORT operand. This is only permitted for a disk.

linkname

A freely selectable link name must be specified if FILE-SEQUENCE=*NEW is set or BUFFER-LENGTH is changed. Only one TFT entry may then exist for the output file.

xxx

Default

- for disk:*STD(SIZE=4)
- for tape:
  BUFFER-LENGTH depends on the length of the records: At least 4 PAM pages.
  The value is rounded up to a whole multiple of the record length and a whole multiple of a a doubleword.

If output is to be on tape, an explicit number should be entered for BUFFER-LENGTH, rather than using standard blocks, since specifying standard blocks (STD) increases the size of the output file.

FILE-SEQUENCE=*NEW

permitted only for tape, if the same set of tapes is to be accessed in a series of BOUTLOAD runs.

The volume of data for output is calculated as follows:

number of records x *reclength* Bytes

The record length is calculated as follows

- for records containing set information in a 2-Kbyte database:

```
reclength = record length as per SIA report - length of system information
          + 4 * (number of non-singular sets in which the record is a
                  member + 1)
          + 1 * (number of singular sets in which the record is a
                  member, except for MANDATORY AUTOMATIC members)
```

- for records containing set information in a 4-Kbyte or 8-Kbyte database:

```
reclength = record length as per SIA report - length of system information
          + 8 * (number of non-singular sets in which the record is a
                  member + 1)
          + 1 * (number of singular sets in which the record is a
                  member, except for MANDATORY AUTOMATIC members)
```

- for records not containing set information:

```
reclength = record length as per SIA report - length of system information
```

In the case of record types which are distributed to realms, five bytes for the area reference are added to the record length when the records are copied or extracted from multiple realms.

The records are always copied into one output file per record type. An origin from more than one realm is therefore required for the area reference to be specified.

The number of records can be obtained using the BSTATUS utility routine.

If the output files have not been previously created, they are created by BOUTLOAD on public disk. The size of each file is calculated by BOUTLOAD from the maximum number of DBTT entries for the corresponding record type.

> For output files on tape it must be ensured that as many tape units are available as there are record types to be simultaneously unloaded, since BOUTLOAD copies the record types at the same time.

CSV output file

It is not obligatory to create the CSV output file. It is always created by the BOUTLOAD utility on a public disc.

The CSV output file name consists of the file name of the output file and the suffix ‘CSV’:

dbname.REC*nnnnn*[.*mmmmm*].CSV

dbname

Name of the database to be processed.

nnnnn

5-digit record reference number with leading zeros.

mmmmm

5-digit area reference number with leading zeros. This specification is required if records are to be copied from only one realm.

CSV

Suffix for CSV output file.

The records are always copied into one CSV output file per record type.

If CSV-OUTPUT = *YES is specified, the DBCOM must be available.

Creating the output record

If BOUTLOAD has also output the set information on account of the
SET-INFORMATION=YES statement, the output record is created with the following structure:

```
| Database Key | Item | Database keys of all owners | User part | Area ref. |
---------------+------+-----------------------------+-----------+------------
```

- The record’s database key
- A one-byte long item with the content
  X’00’ = Member inserted
  X’FF’ = Member not inserted
  (for all singular sets in which the record is a member, except for MANDATORY AUTOMATIC members)
- The owners’ database keys are not singular sets in which the set is a member
- If the record is not inserted in the set, the owner’s database key is set to High Value (X'FFFFFFFF' in the case of a 2-KB database, or X'FFFFFFFFFFFFFFFF' in the case of a 4/8-KB database)
- User part
- The five-byte area reference (realm reference) in the case of record types which are distributed to realms if their records are copied from multiple realms. The records are always copied into one output file per record type. An origin from more than one realm is therefore required for the area reference to be specified.

When BOUTLOAD outputs set information on the individual sets, the length of the database key values is specified in the BOUTLOAD log which contains the statements for a subsequent BINILOAD run (length “4” in the case of a 2-Kbyte database, length “8” in the case of a 4-Kbyte/8-Kbyte database).

Without any set information, the output record consists of the user part only.

CSV output data

The following example shows part of a data output to a CSV output file in a presentation mode similar to Microsoft EXCEL. In contrast, in a CSV file, the single values are separated by a semicolon (;).

BOUTLOAD	CSV V1.20	29.01.2019	14:38:53									
DBNAME	DATABASE NAME	DB1										
INFO01	RECORD NAME	RECORD2										
INFO02	RECORD REF	2										
INFO03	REALM NAME	AREA1										
INFO04	REALM REF	3										
FIELDS	DB Key Ref	DB Key RSQ	Member SYS-1	Owner DB Key Ref S1	Owner DB Key RSQ S1	R1	R2-1 (1,1)	R2-1 (1,2)	R2-2 (1,1,1)	R2-2 (1,1,2)	...	Area-ref.
RECORD	2	1	Y	1	6	15,7	Y	A	YZ	AC	...	3
RECORD	2	3	N	1	3	-47,1	A	B	BC	AB	...	3

The header of the data output contains up to 6 rows, if a realm name was specified in COPY-RECORD:

- The first row contains the name of the utility routine, the corresponding utility routine's output format version, and the date and time of the CSV output creation.
- The next rows contain database name, record name, and record reference. If a realm name was specified, the header also contains realm name and realm reference.

The header line of the record output can contain the following fields:

DB Key Ref, DB Key RSQ

Field names for the database key of the record

Member *set-name*

Field name for a one-byte long item with the following content:

Y Member inserted into the SYSTEM set *set-name*

N Member not inserted into the SYSTEM set *set-name*

(for all singular sets in which the record is a member, except for MANDATORY AUTOMATIC members)

Owner DB Key Ref *set-name*, Owner DB Key RSQ *set-name*

If the record is a member record, database keys of all owners are output additionally.

The item names according to the user schema

Area ref

In the case of a record type that is distributed to realms if its records are copied from multiple realms

If an item is part of a repetition group or a vector the index value is attached to the item name of such an element.

Items of the type DBKEY are output in the following format:
DB KEY REF and DB KEY RSQ.

BOUTLOAD converts binary and numeric data into a printable format as follows:

- The decimal point is represented by the character "comma" (",").
- Alphanumeric items of variable length are output according to the current length of the variable item.
- Items of national type are converted to a user-defined character set, if it is possible.

To assign or get a user default character set, proceed as follows:

> To assign a user default character set use the ADD-USER or MODIFY-USER commands.

> To get the user default character set, use the SHOW-USER-ATTRIBUTES command.

For conversion from a national data type, the XHCS subsystem must be available in the system. If national characters cannot be converted due to an XHCS subsystem error, the warning 3935 is output, the output to this CSV file is terminated, and the CSV file is deleted.

`3935 NATIONAL CHARACTERS CANNOT BE CONVERTED: XHCS RETURN CODE: returncode`

If a record type contains a national data type, the corresponding CSV file is created with the CODED-CHARACTER-SET from the USER-ID.
If the CODED-CHARACTER-SET of the user cannot be determined, the warning 3936 is output, the CSV output for this record type is terminated, and the CSV file is deleted.

`3936 USER CODED CHARACTER SET CANNOT BE DETERMINED: SRMUINFI RETURN CODE: returncode`

If a national character cannot be converted to a user-defined character set (because there is no equivalent), this national character is output as character "period" (".") A warning message is output additionally:

`3932 STRING CONVERSION WITH SUBSTITUTION BY DEFAULT CHARACTERS PERFORMED FOR RECORD recordname`.

A semicolon (";") is used to separate the individual values.

Alphanumeric values can contain some characters such as separators/delimiters (";"), new lines, or double quotes, which have special meanings in different system environments. All alphanumeric values are enclosed by double quotes so that these values can be processed correctly in other system environments. If a value contains embedded (double) quote characters, this double quote is represented by two (double) quote characters.

5.2.3 BOUTLOAD log for the output record format

BOUTLOAD generates a log on SYSLST if the SET-INFORMATION parameter is set to YES. If SYSLST is assigned to a file, this file can be used as input file for BINILOAD, and the BINILOAD statements can be applied. See also the examples starting on "Examples".

Example

```
BINILOAD PARAMETERS FOR RECORD : ARTICLE                    , REC-REF :    9
SCHEMA NAME IS MAIL-ORDERS
SUBSCHEMA NAME IS
USER FILE RECORD LENGTH IS  112
USER FILE BUFFER LENGTH IS 8192
INPUT FILE 'SHIPPING.REC00009.00005                           '
INPUT RECORDNUMBER IS       55
STORE RECORD NAME IS ARTICLE
RECORD-DBKEY IS DISPL IS 0 , LENGTH IS 8
RECORD-DISPL IS 0 , DISPL IS   25 , LENGTH IS   87
RECORD-AREA NAME IS CLOTHING
INSERT INTO SET NAME IS P-ORD-SPEC
OWNER DBKEY IS DISPL IS    8 , LENGTH IS 8
INSERT INTO SET NAME IS MIN-STOCK-LEVEL
OWNER KEY IS DISPL IS   16 , LENGTH IS 1
INSERT INTO SET NAME IS ARTICLES-AVAILABLE
OWNER DBKEY IS DISPL IS   17 , LENGTH IS 8
END
```

If BOUTLOAD generates output to CSV, then necessary statements for further uploading by BINILOAD from CSV file are generated and output also to SYSLST, after usual statements for BINILOAD.

For instance, “FORMAT: CSV” in the "BINILOAD PARAMETERS FOR RECORD" line means that the following statements are suitable for loading from CSV file by BINILOAD.

Example

```
BINILOAD PARAMETERS FOR RECORD : ARTICLE , REC-REF : 9, FORMAT: CSV
SCHEMA NAME IS MAIL-ORDERS
SUBSCHEMA NAME IS
INPUT FILE 'SHIPPING.REC00009.00005.CSV ' FORMAT IS CSV
INPUT RECORDNUMBER IS 55
STORE RECORD NAME IS ARTICLE
INSERT INTO SET NAME IS P-ORD-SPEC
INSERT INTO SET NAME IS MIN-STOCK-LEVEL
INSERT INTO SET NAME IS ARTICLES-AVAILABLE
END
```

5.2.4 BOUTLOAD system environment

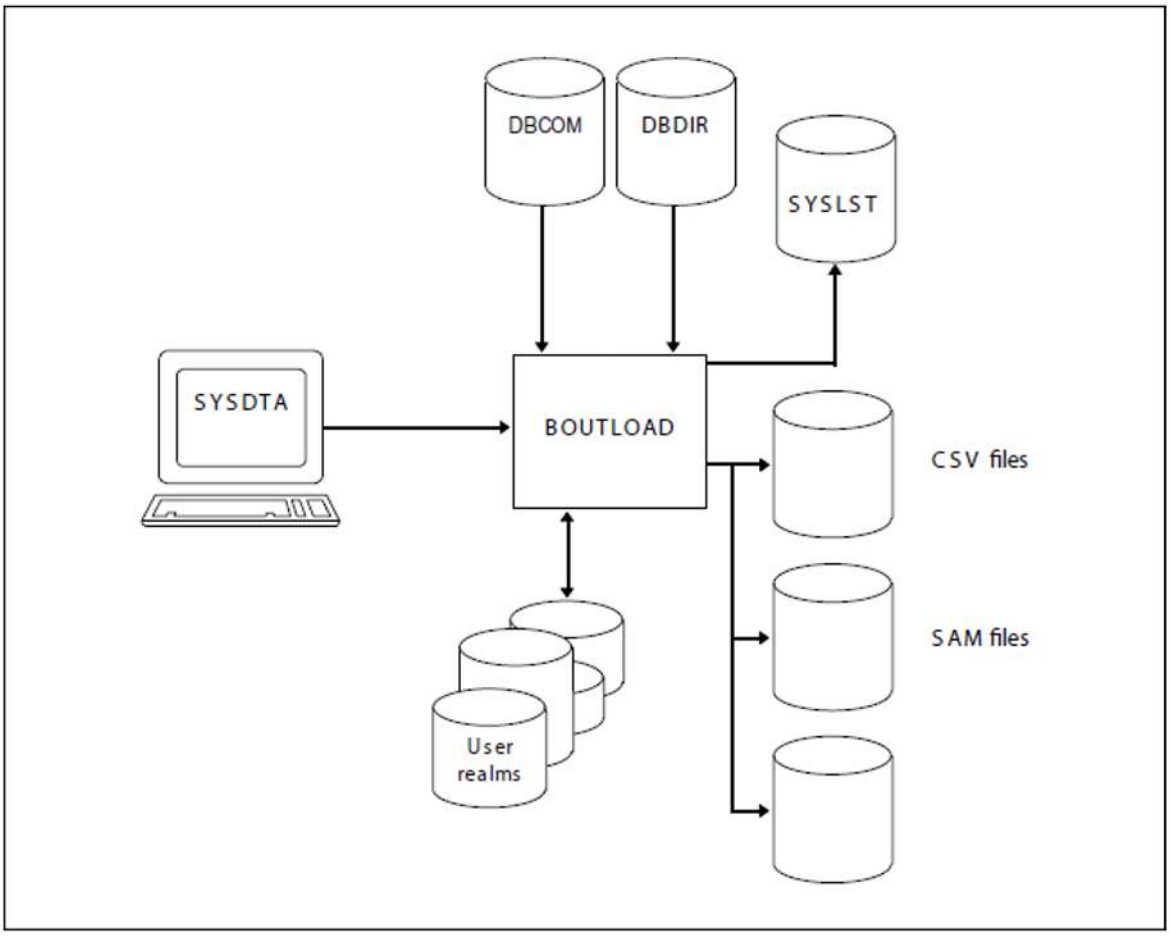


Figure 21: System environment for BOUTLOAD

In this description it is assumed that UDS/SQL was installed with IMON (see section “START commands for the UDS/SQL programs”).

5.2.5 BOUTLOAD statements

The statement formats of the BOUTLOAD utility routine conform to the SDF rules (System Dialog Facility, see the manuals "SDF Dialog Interface" and "Commands").

The data types used in SDF formats are described in table 5 on "SDF syntax representation".

Overview of statements for BOUTLOAD

Statement	Meaning
COPY-RECORD **RECORD-NAME** = ***ALL** / list-poss(20): <record-name> / ***ALL-EXCEPT(...)** **,REALM-NAME** = ***ALL** / <realm-name> **,SET-INFORMATION** = **YES** / **NO** **,CSV-OUTPUT = *NO / *YES (...)**	Copy all records of the specified record types to output files
END	Terminate BOUTLOAD
EXPORT-RECORD **RECORD-NAME** = ***ALL** / list-poss(20): <record-name> / ***ALL-EXCEPT(...)** **,SET-INFORMATION** = **YES** / **NO**	Unload all records of the specified record types to output files
OPEN-DATABASE **DATABASE-NAME** = <dbname> **,COPY-NAME** = ***NONE** / <copyname> **,USER-IDENTIFICATION** = ***OWN** / <userid>	Assign the database
REMOVE-RECORD **RECORD-NAME** = ***ALL** / list-poss(20): <record-name> / ***ALL-EXCEPT(...)**	Delete all records of the specified record types

Table 26: Statements for BOUTLOAD

COPY-RECORD (Copying records to output files)

This statement is used to copy all records of the specified record types to output files. It is also possible to get the output in CSV format. The database key values are output in the same form in which they exist in the database, i.e. BOUTLOAD does not convert them from the short form to the long form, and vice versa. The database itself remains unchanged.

COPY-RECORD

RECORD-NAME = ***ALL** / list-poss(20): <record-name> / ***ALL-EXCEPT(...)**

***ALL-EXCEPT(...)**

| **EXCEPT-NAME=** list-poss(20):<recordname>

,**REALM-NAME** = ***ALL** / <realm-name>

,**SET-INFORMATION** = **YES** / **NO**

,CSV-OUTPUT = *NO/ *YES (...)

***YES(...)**

| **OUTPUT = *STD / *FULL / *CSV-COMPATIBLE**

RECORD-NAME = *ALL / list-poss(20): <record-name>/ *ALL-EXCEPT(...)

***ALL**
With this specification only this one function is permitted in the BOUTLOAD run, i.e. the END statement must follow.
All records of all record types are copied.

<record-name>
All records of the specified record type(s) are copied.

***ALL-EXCEPT(...)**
With this specification only this one function is permitted in the BOUTLOAD run, i.e. the END statement must follow.
All records with the exception of the specified record types are copied.

EXCEPT-NAME= list-poss(20): <recordname>
Names of the record types which are not to be copied.

REALM-NAME = *ALL / <realm-name>

***ALL**
All records of the specified record type are copied from all realms in which they can occur.

<realm-name>
If a single realm has been specified, only the records of that realm are copied to output files.

With this specification only this one function is permitted in the BOUTLOAD run, i.e. the END statement must follow.

SET-INFORMATION = YES / NO
Determines whether set information is to be stored in the corresponding output file for every record and whether statements for BINILOAD are to be written to SYSLST.

Before copying the database is not checked as to whether it contains records which can be copied. If no such records exist, the associated output file is either invalidated, if it has been created by the user, or deleted, if it has been created by BOUTLOAD.

CSV-OUTPUT = *NO / *YES

***NO**
BOUTLOAD outputs the data to output files, but not in CSV format.

***YES (...)**
BOUTLOAD additionally outputs the data in CSV format.

OUTPUT = *STD / *FULL / *CSV-COMPATIBLE

*STD and *FULL produce a full CSV output. *CSV-COMPATIBLE produces an output that only consists of the header line (containing the field names, without the entry FIELD) and the data lines (without the entry RECORD).

END (Terminating the BOUTLOAD run)

This statement is used to terminate the BOUTLOAD run. It must be the last statement you enter.

END

This statement has no operands.

EXPORT-RECORD (Unloading records to output files)

This statement is used to unload all records of the specified record types from the database to output files. The database key values are output in the same form in which they exist in the database, i.e. BOUTLOAD does not convert them from the short form to the long form, and vice versa.

This statement may only be used if BOUTLOAD has been loaded under the identification under which the database is cataloged.

EXPORT-RECORD

RECORD-NAME = ***ALL** / list-poss(20): <record-name>/ ***ALL-EXCEPT(...)**

***ALL-EXCEPT(...)**

| **EXCEPT-NAME=** list-poss(20):<recordname>

,**SET-INFORMATION** = **YES** / **NO**

RECORD-NAME = *ALL / list-poss(20): <record-name>/ *ALL-EXCEPT(...)

***ALL**
Implies that this is the only function permitted in the BOUTLOAD run; i.e. the END statement must follow.The database is reformatted; all realms have to be available.

<record-name>
All records of the specified record type are copied and deleted from the database.

***ALL-EXCEPT(...)**
With this specification only this one function is permitted in the BOUTLOAD run, i.e. the END statement must follow.
All records with the exception of the specified record types are copied and deleted in the database.

EXCEPT-NAME= list-poss(20): <recordname>
Names of the record types which are not to be copied and are to be deleted in the database.

SET-INFORMATION = YES / NO

Determines whether set information is to be stored in the corresponding output file for every record and whether statements for BINILOAD are to be written to SYSLST.
When BOUTLOAD outputs set information for the individual records, the BOUTLOAD log, which contains the statements for a subsequent BINILOAD run, indicates the length of the database key values (length “4” for a 2-Kbyte database and length “8” for a 4-Kbyte or 8-Kbyte database).

Record types may be copied with SET-INFORMATION=NO even if the database is inconsistent.

Before copying the database is not checked as to whether it contains records which can be copied. If no such records exist, the associated output file is either invalidated, if it has been created by the user, or deleted, if it has been created by BOUTLOAD.

When entering the record types, the hierarchical structure of the database schema must be observed; i.e. the member record types must be unloaded either before or together with the owner record types.
If the entries are incorrect, the BOUTLOAD run is aborted.

If you store records in the database subsequently, the assignment of the DB keys begins again at RSQ=1. The DB keys can now also be used (once) if they are locked for reuse by the DBH by means of the BMODTT statement KEEP. An additional BMODTT run with the REMOVE statement is not required.

OPEN-DATABASE (Assigning the database)

This statement is used to assign the database.

You must enter it first if you have not assigned the database with:

`/ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=`*`dbname`*`.DBDIR`

If the SET-FILE-LINK command has been entered, an OPEN-DATABASE statement is rejected as invalid.

OPEN-DATABASE
DATABASE-NAME = <dbname> ,**COPYNAME** = ***NONE** / <copyname> ,**USER-IDENTIFICATION** = ***OWN** / <userid>

DATABASE-NAME = <dbname>
Name of the database. A user can edit only databases that are in his or her userid. A database from another userid can only be processed using the TSOS identification of the system administrator.

COPY-NAME = *NONE / <copyname>

***NONE**
The original database is opened.

<copyname>
The shadow database is opened.

USER-IDENTIFICATION = *OWN / <userid>

***OWN**
The database is in the user's userid.

<userid>
The database is in another userid. This specification is permitted only from the TSOS identification.

The link name DATABASE remains in effect until it is canceled with the REMOVE-FILE-LINK command.

The OPEN-DATABASE statement remains in effect up to completion of the BOUTLOAD run.

REMOVE-RECORD (Deleting records)

This statement is used to delete all records of the specified record types from the database.

This statement may only be used if BOUTLOAD has been loaded under the identification under which the database is cataloged.

REMOVE-RECORD
RECORD-NAME = ***ALL** / list-poss(20): <record-name> ***ALL-EXCEPT(...)** ***ALL-EXCEPT(...)** | **EXCEPT-NAME=** list-poss(20):<recordname>

RECORD-NAME = *ALL / list-poss(20): <record-name>/ *ALL-EXCEPT(...)

***ALL**
Implies that this is the only function permitted in the BOUTLOAD run; i.e. the END statement must follow.
The database is reformatted; all realms have to be available.
This statement is also allowed for an inconsistent database (with the system break bit set).

<record-name>
All records of the specified record type are deleted from the database.

***ALL-EXCEPT(...)**
With this specification only this one function is permited in the BOUTLOAD run, i.e. the END must follow.
All records with the exception of the specified record types are copied and deleted in the database.

EXCEPT-NAME= list-poss(20): <recordname>
Names of the record types which are not to be deleted.

When entering the record types, the hierarchical structure of the database schema must be observed; i.e. the member record types must be deleted either before or together with the owner record types.
If the entries are incorrect, the BOUTLOAD run is aborted.
If you store records in the database subsequently, the assignment of the DB keys begins again at RSQ=1. The DB keys can now also be used (once) if they are locked for reuse by the DBH by means of the BMODTT statement KEEP. An additional BMODTT run with the REMOVE statement is not required.

5.2.6 Command sequence to start BOUTLOAD

It is assumed for the command sequence described here that UDS/SQL was installed with IMON (see section “START commands for the UDS/SQL programs”).

```
01 [/ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR]

02 /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,
SCOPE=*TASK

03 /START-UDS-BOUTLOAD

04 [OPEN-DATABASE DATABASE-NAME=dbname]

05 boutload-statements

06 END
```

01,04 One of the two statements must be used to assign the database.

02 The specified version of BOUTLOAD is selected.
It is generally recommended that you specify the version since it is possible for several UDS/SQL versions to be installed in parallel.

03 BOUTLOAD can be called from any user ID. The UDS/SQL utility routine can also be started with the aliases BOUTLOAD and START-UDS-OUTLOAD.

5.2.7 Examples

Example of BOUTLOAD

Record type COLORS from database SHIPPING is copied.

```
/SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,    VERSION=02.9C00
/START-UDS-BOUTLOAD
***** START        BOUTLOAD     (UDS/SQL  V2.9  1801 )     2019-01-29   09:27:53
//OPEN-DATABASE DATABASE-NAME=SHIPPING
//COPY-RECORD RECORD-NAME=COLORS,REALM-NAME=ARTICLE-RLM
3903  AFTER " REALM-NAME = <NAME> " ONLY THE " END " STATEMENT IS ALLOWED
//END
***** INPUT CHECK SUCCESSFULLY TERMINATED
***** BEGIN SCAN OF USER-REALMS
***** REALM: ARTICLE-RLM
***** SCAN OF USER-REALMS SUCCESSFULLY TERMINATED

***** DIAGNOSTIC SUMMARY OF BOUTLOAD

              NO WARNINGS
              NO ERRORS
              NO SYSTEM-ERRORS

***** END OF DIAGNOSTIC SUMMARY
***** NR OF DATABASE ACCESSES  :           32
***** NORMAL END   BOUTLOAD     (UDS/SQL  V2.9  1801 )     2019-01-29   09:27:53
```

Example of BOUTLOAD and BINILOAD

The record type MATERIALS is copied and deleted with BOUTLOAD and stored with BINILOAD. BINILOAD uses the output generated by BOUTLOAD for the input of data and BINILOAD statements.

```
/SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,    VERSION=02.9C00
/START-UDS-BOUTLOAD
***** START        BOUTLOAD     (UDS/SQL  V2.9  1801 )     2019-01-29   09:27:53
//OPEN-DATABASE DATABASE-NAME=SHIPPING
//COPY-RECORD RECORD-NAME=MATERIALS,REALM-NAME=ARTICLE-RLM
3903  AFTER " REALM-NAME = <NAME> " ONLY THE " END " STATEMENT IS ALLOWED
//END
***** INPUT CHECK SUCCESSFULLY TERMINATED
***** BEGIN SCAN OF USER-REALMS
***** REALM: ARTICLE-RLM
***** SCAN OF USER-REALMS SUCCESSFULLY TERMINATED

BINILOAD PARAMETERS FOR RECORD : MATERIALS             , REC-REF :   12  - (1)

SCHEMA NAME IS MAIL-ORDERS                                              - (1)
SUBSCHEMA NAME IS                                                       - (1)
USER FILE RECORD LENGTH IS 29                                           - (1)
USER FILE BUFFER LENGTH IS 8192                                         - (1)
INPUT FILE 'SHIPPING.REC00012.00011'                                    - (1)
INPUT RECORDNUMBER IS 10                                                - (1)
STORE RECORD NAME IS MATERIALIEN                                        - (1)
RECORD-DBKEY IS DISPL IS 0 , LENGTH IS 8                                - (1)
RECORD-DISPL IS 0 , DISPL IS 8 , LENGTH IS 21 v
RECORD-AREA NAME IS ARTICLE-RLM                                         - (1)
END                                                                     - (1)

***** DIAGNOSTIC SUMMARY OF BOUTLOAD
              NO WARNINGS
              NO ERRORS
              NO SYSTEM-ERRORS

***** END OF DIAGNOSTIC SUMMARY
***** NR OF DATABASE ACCESSES  :           32
***** NORMAL END   BOUTLOAD     (UDS/SQL  V2.9  1801 )     2019-01-29   09:27:53
     .
     .
     .
```

(1) - BINILOAD statements that are generated by BOUTLOAD and used as input for BINILOAD.

```
/SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,    VERSION=02.9C00
/START-UDS-BOUTLOAD
***** START        BOUTLOAD     (UDS/SQL  V2.9  1801 )     2019-01-29   09:27:54
//OPEN-DATABASE DATABASE-NAME=SHIPPING
//REMOVE-RECORD RECORD-NAME=MATERIALS
//END
***** INPUT CHECK SUCCESSFULLY TERMINATED
***** NO OCCURRENCES OF MEMBER RECORDS DETECTED
***** BEGIN SCAN OF USER-REALMS
***** REALM: ARTICLE-RLM
***** SCAN OF USER-REALMS SUCCESSFULLY TERMINATED

***** DIAGNOSTIC SUMMARY OF BOUTLOAD

              NO WARNINGS
              NO ERRORS
              NO SYSTEM-ERRORS

***** END OF DIAGNOSTIC SUMMARY
***** NR OF DATABASE ACCESSES  :           100
***** NORMAL END   BOUTLOAD     (UDS/SQL  V2.9  1801 )     2019-01-29   09:27:54
.
.
.
```

```
/SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,    VERSION=02.9C00
/START-UDS-BINILOAD
***** START        BINILOAD     (UDS/SQL  V2.9  1801 )     2019-01-29   09:27:55
EXECUTION WITH CHECK.                                                  - (1)
SCHEMA NAME IS MAIL-ORDERS                                             - (1)
SUBSCHEMA NAME IS ADMIN                                                - (1)
USER FILE RECORD LENGTH IS 29                                          - (1)
USER FILE BUFFER LENGTH IS 8192                                        - (1)
INPUT FILE 'SHIPPING.REC00012.00011'                                   - (1)
INPUT RECORDNUMBER IS      10                                          - (1)
STORE RECORD NAME IS MATERIALS                                         - (1)
RECORD-DBKEY IS DISPL IS 0 , LENGTH IS 8                               - (1)
RECORD-DISPL IS 0 , DISPL IS 8 , LENGTH IS 21                          - (1)
RECORD-AREA NAME IS ARTICLE-RLM                                        - (1)
END                                                                    - (1)
BEGIN CHECK-RUN
*** DATE AND TIME 2019-01-29  09:27:55
BEGIN ALLOCATION
*** DATE AND TIME 2019-01-29  09:27:55
SEARCHKEY ON RECORD-LEVEL:
--------------------------
REC REF:    12
RECORD NAME: MATERIALS
SEARCHKEY TABLE, DBTT_COLUMN_NR:     1
*** ICRELES: MOVE_ROUTINE SORTKF CREATED
BEGIN TABLE-PROCESSOR
*** DATE AND TIME 2019-01-29  09:27:56
SEARCHKEY ON RECORD-LEVEL:
--------------------------
REC REF:    12
RECORD NAME: MATERIALS
```

```
SEARCHKEY TABLE, DBTT_COLUMN_NR:     2
*** NO ERRORS DETECTED DURING CHECK-RUN
END CHECK-RUN
*** DATE AND TIME 2019-01-29  09:27:56
BEGIN ALLOCATION
*** DATE AND TIME 2019-01-29  09:27:56
*** DATABASE IS IN USE
SEARCHKEY ON RECORD-LEVEL:
--------------------------
REC REF:    12
RECORD NAME: MATERIALS
SEARCHKEY TABLE, DBTT_COLUMN_NR:     1
*** ICRELES: MOVE_ROUTINE SORTKF CREATED
BEGIN TABLE-PROCESSOR
*** DATE AND TIME 2019-01-29  09:27:56
SEARCHKEY ON RECORD-LEVEL:
--------------------------
REC REF:    12
RECORD NAME: MATERIALS
SEARCHKEY TABLE, DBTT_COLUMN_NR:     2
BEGIN STORE DB-RECORD
*** DATE AND TIME 2019-01-29  09:27:56
STORING DATABASE RECORDS
END STORE DB-RECORD
*** DATE AND TIME 2019-01-29  09:27:56
END CLOSE
*** DATE AND TIME 2019-01-29  09:27:56

***** DIAGNOSTIC SUMMARY OF BINILOAD

              NO WARNINGS
              NO ERRORS
              NO SYSTEM-ERRORS

              10 RECORDS  STORED

***** END OF DIAGNOSTIC SUMMARY
***** NR OF DATABASE ACCESSES  :           108
***** NORMAL END   BINILOAD     (UDS/SQL  V2.9  1801 )     2019-01-29   09:27:56
```

(1) - BINILOAD statements that are generated by BOUTLOAD and used as input for BINILOAD.

6 Restructuring the database (BCHANGE, BALTER)

Restructuring a database which already contains data means changing the schema and/or the storage structure.

You can perform actions purely for the purpose of renaming which only affect the schema in a renaming cycle (see chapter “Renaming database objects (BRENAME, BALTER)”).

The activities necessary for restructuring can be divided into three categories as follows:

- preparatory measures
- restructuring process
- follow-up activities.

Preparatory measures

- Analyzing and modifying the database schema and storage structure
- Checking database consistency
- Analyzing memory space statistics
- If After Image Logging is activated deactivate it using BMEND (see also section “Saving the database”)

- Either
  - save the complete database including DBCOM, COSSD and HASHLIB before the restructuring process

  or
  - save HASHLIB, COSSD, DBDIR and DBCOM before the restructuring process
  - determine which user realms are required in an analysis run with the statements REPORT IS YES and EXECUTION IS NO
  - save these user realms before the BALTER execution phase

  Detailed information on saving is provided in section “Saving the database”.

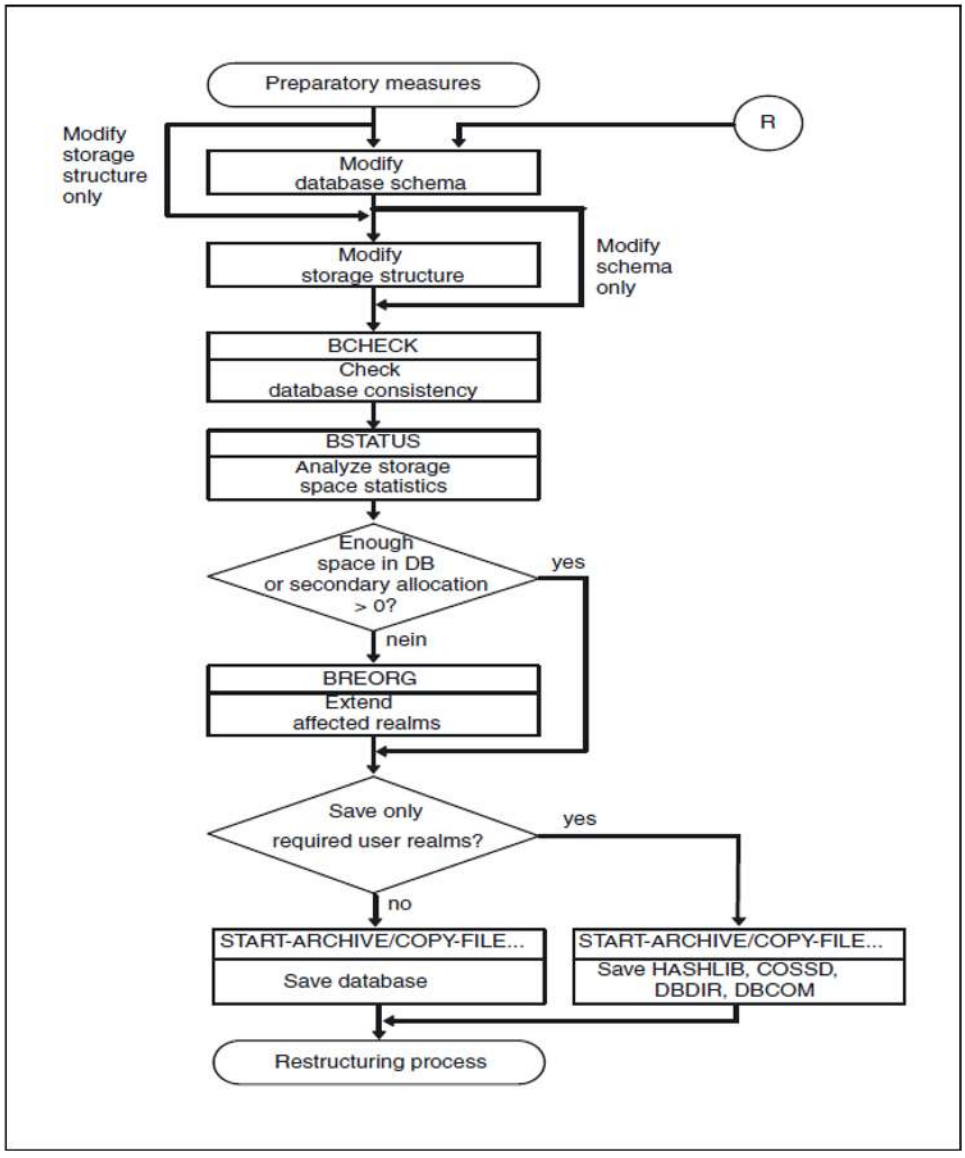


Figure 22: Preparatory measures for restructuring the database

Restructuring process

This is a process that resembles the creation of a database:

- BCHANGE prepares the DBDIR to accept a new SIA
- New DDL and SSL definitions are then compiled and the new SIA is entered in the DBDIR

- Finally, BALTER adapts the data to the modified schema.

The restructuring cycle of BCHANGE/BALTER cannot be combined with the renaming cycle of BRENAME/BALTER. Renaming in a restructuring cycle is interpreted as deleting the old item and inserting the new item. This can result in the loss of data.

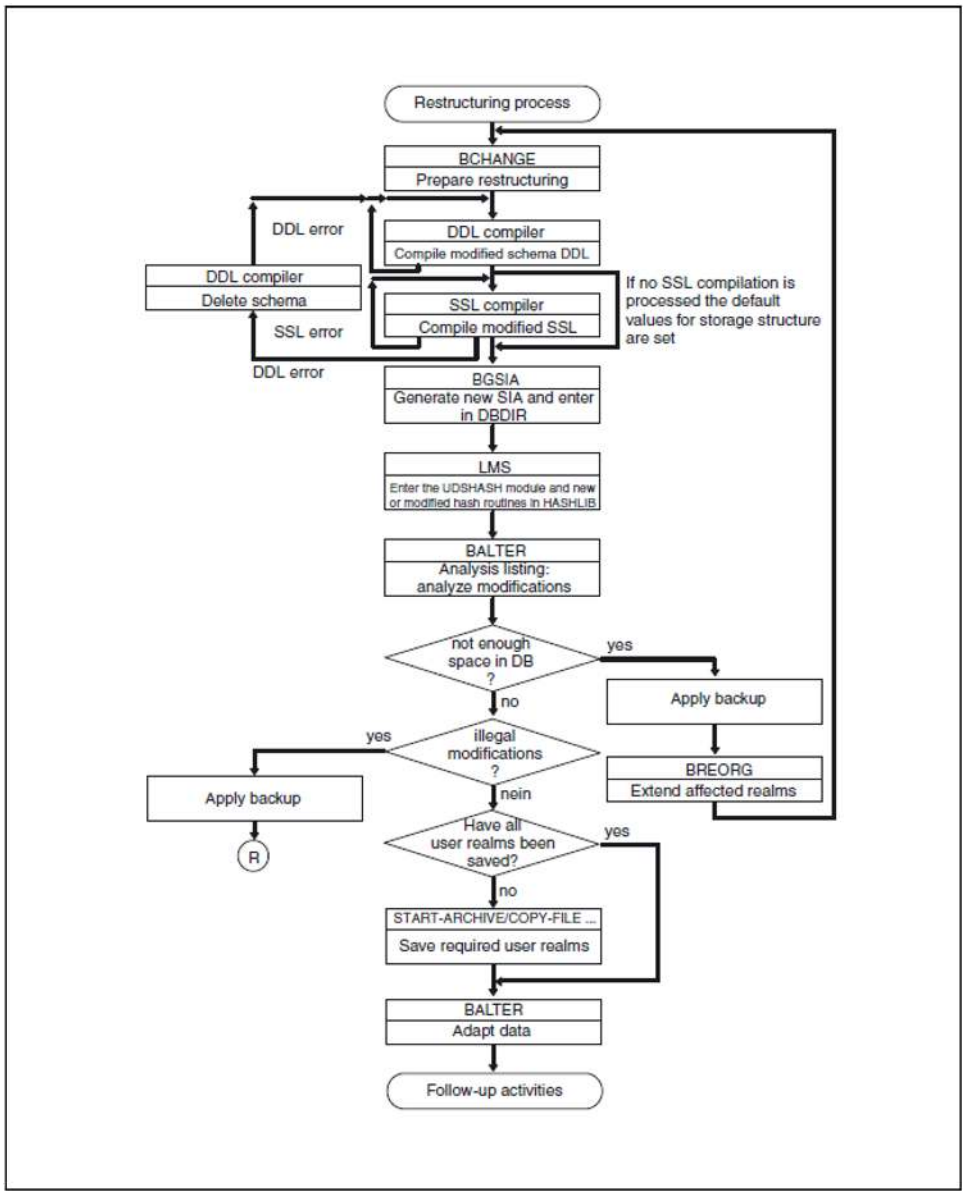


Figure 23: Restructuring process

Follow-up activities

After restructuring, the following activities have to be carried out:

- Access rights have to be updated if user group names are defined for access rights in the output database.
- Subschemas have to modified to the schema.
- DB application programs have to be adapted to the new schema.
- Sets and hash areas have to be reorganized using BREORG.

- It may also be necessary to use the utility routine BMODTT to reassign DB keys that have been released (see "BMODTT" in the "Recovery, Information and Reorganization" manual).

> **i** A logging gap occurs because of the restructuring cycle (see the "Database Operation" manual, Media recovery). After the restructuring cycle you must therefore establish a new basis for media recovery by copying the modified files together with the unmodified files. You must then use BMEND to activate After Image Logging again.

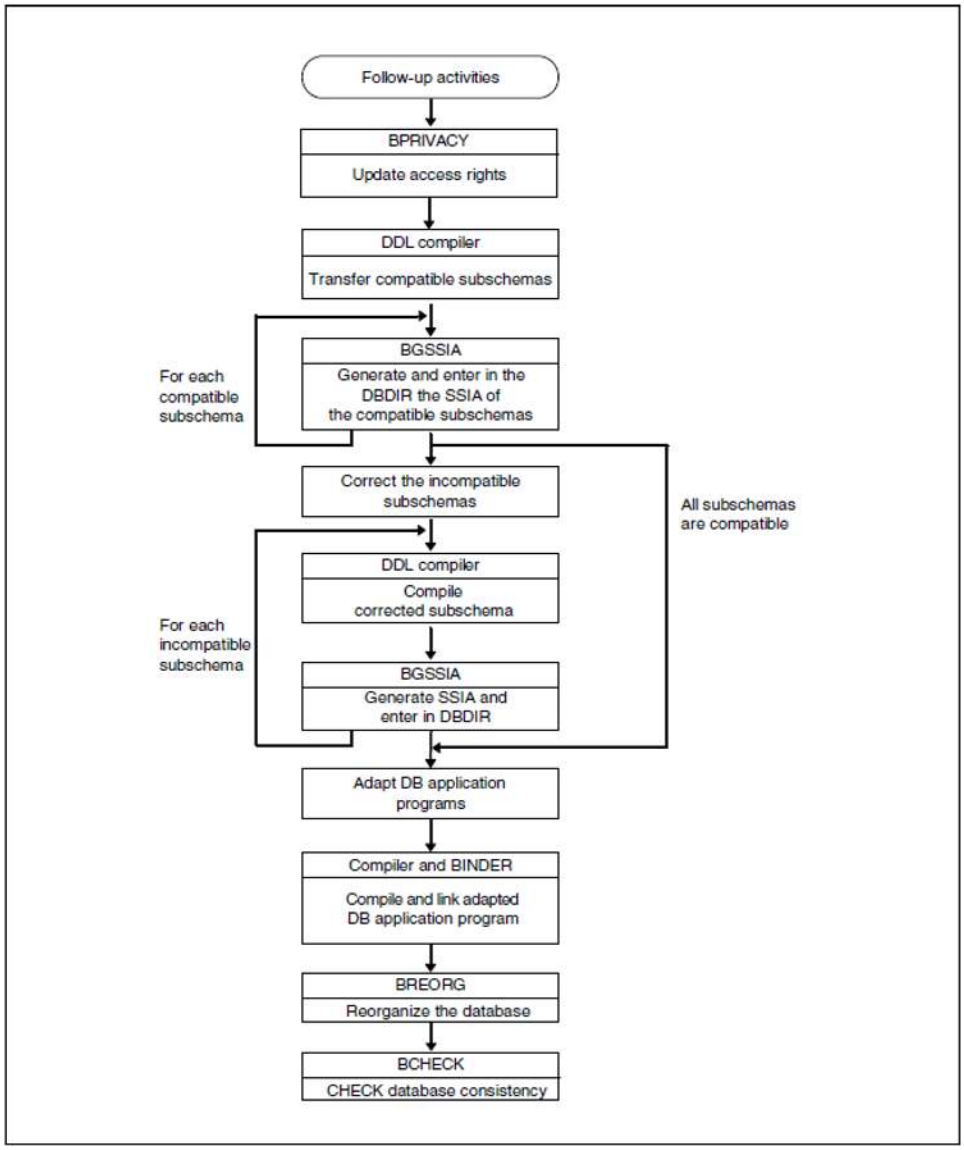


Figure 24: Activities after database restructuring

6.1 Modifying the Schema DDL

If the DB administrator wishes to modify the Schema DDL, he or she must generate a complete new Schema definition and have it compiled. BALTER enables the following modifications to be made to the Schema DDL:

- at realm level
  - adding or deleting realms
- at record type level
  - adding or deleting record types
  - changing LOCATION MODE
  - redefining, deleting or modifying SEARCH keys
  - adding, omitting, redefining, lengthening or shortening items
- at set level
  - adding or deleting sets (with restrictions)
  - modifying the ORDER clause (with restrictions)
  - changing the sort criteria
  - redefining, deleting or modifying SEARCH keys.

During restructuring operations BALTER identifies the database elements (realms, record types, sets, keys, etc.) solely by means of their names:

- BALTER recognizes elements as identical if they are of the same type (e.g. record type) and their names occur in the old and new Schema DDL.
- BALTER deletes elements if the name for this element type does not occur in the new Schema DDL.
- BALTER adds elements if the name for this element type does not occur in the old Schema DDL.

Consequently it is impossible to rename elements or in the same restructuring run delete an element and replace it by an element of the same type and name. The BRENAME utility routine is provided for renaming items (see chapter “Renaming database objects (BRENAME, BALTER)”).

When the BGSIA run is carried out during restructuring, the numbers of the database elements remain the same. BGSIA allocates the numbers in question according to the element name. Consequently the sequence in which the elements are defined in the new Schema DDL is immaterial.

When restructuring is performed, the clauses of the Schema DDL and the SSL are subject to the same rules which apply when a database is defined (see the "Design and Definition" manual). This also means that when a clause of the Schema DDL is modified, all other associated clauses in the Schema DDL and SSL must be adapted so as to comply with these rules.

The following is a detailed description of the modifications possible in the Schema DDL and their effects on the stored data.

Schema entry

```
SCHEMA NAME IS schema-name
```

schema-name

may be changed if required.

```
[PRIVACY LOCK FOR COPY IS literal-1[ OR literal-2]].
```

The PRIVACY LOCK specifications can be changed as required.

Effects on stored data:

The DDL compiler enters the new PRIVACY LOCK specifications in the new DBCOM.

A subschema whose PRIVACY KEY does not match the new PRIVACY LOCK specifications can still be used (see section “Copying compatible subschemas”). The new PRIVACY LOCK specifications need only be taken into account when the new or modified subschemas are compiled.

Realm entry

```
AREA NAME IS realm-name
```

Realms can be added or deleted.

If a realm containing records of a record type which also occurs in the new schema is deleted, these records must be unloaded before restructuring takes place, since BALTER does not transfer records to other realms. BALTER simply removes a deleted realm from the database. When restructuring is complete, the file of the realm must be deleted with the ERASE command.

Realms which have been added need not be formatted with BFORMAT. BALTER formats them automatically and adds them to the database. The realms must, however, be set up using the CREATE-FILE command before the BALTER run takes place.
Make sure that the secondary assignment is set to a value greater than 0 if the realm is to be extendable online (see the “Database Operation” manual, ACT INCR)

```
[AREA IS TEMPORARY].
```

Adding or deleting the temporary realm is possible.

Changing a temporary realm into a non-temporary realm or vice versa is however not permissible.

Record entry

```
RECORD NAME IS record-name
```

record-name

It is possible to add or delete record types or to modify their definitions.

A record type which is to be deleted need not first be unloaded. BALTER deletes all related information such as records, hash areas, DBTT and tables, with one exception: compressed records or records with variable items must be unloaded before restructuring, since otherwise BALTER cannot process them and therefore cannot delete them.

```
[LOCATION MODE IS {
  {DIRECT | DIRECT-LONG} {itemname-1 {IN | OF} record-name | identifier-1} |
  CALC[ hash-routine] USING itemname-2,... DUPLICATES ARE[ NOT] ALLOWED}]
```

The LOCATION MODE clause can be modified, added or omitted as required.

CALC/DIRECT or CALC/DIRECT-LONG

Allows the conversion of DIRECT or DIRECT-LONG to CALC, and vice versa, or allows the LOCATION MODE clause with one of these specifications to be omitted or added.

Effects on stored data:

- DIRECT or DIRECT-LONG -> CALC:
  BALTER creates a new hash area and transfers the data records to it.
  If the record type is the member record type of a list, an indirect CALC is created. In the case of a distributable list the hash area for all records is located in one realm.
- CALC -> DIRECT or DIRECT-LONG:
  BALTER deletes all system information needed for hashing but does not transfer those records of this record type which have already been stored to the database.
- Addition/omission of LOCATION MODE IS CALC:
  This change has exactly the same effect on the data as changing the specification DIRECT or DIRECT-LONG to CALC, and vice versa.
- Addition/omission of LOCATION MODE IS DIRECT/DIRECT-LONG:
  This change has no effect on the stored data, but needs to be taken into account in application programs.

DIRECT / DIRECT-LONG

item-name-1:
the key item may be altered as required

identifier-1:
may be changed

Effects on data:

These changes have no effect on stored data, but needs to be taken into account in application programs.

CALC *hash-routine*: Change permitted.

It is permissible to change from the standard UDS/SQL hash routine to a user hash routine or vice versa; it is also permissible to replace the user hash routine by a new user hash routine.

item-name-2,...:
Any change in the key items permitted.

DUPLICATES...:
Duplicates may be permitted or prohibited as required.

Effects on stored data:

- If the key items or the hash routine are altered, BALTER creates a new hash area and transfers the records.
- If the DUPLICATES specification is changed to NOT ALLOWED, **it is important to remember** that BALTER only checks data for duplicates in those cases in which further alterations make processing of keys necessary.
- If BALTER finds records with duplicate key values, **it does not eliminate these values**. If duplicates are prohibited, a check must be carried out to ascertain whether records with duplicate key values occur. If so, duplicates must be removed before restructuring is performed.
- BALTER logs duplicate key values in the EXECUTION phase only, not in the analysis log. Restructuring is then continued.
- This treatment of duplicates should be applied to all clauses containing the DUPLICATES entry.

```
WITHIN realm-name-1[,realm-name-2,... AREA-ID IS identifier-2]
```

realm-name

It is permissible to change the allocation of record types to realms.

Effects on stored data:

- When a realm which is defined with LOCATION MODE IS CALC is added to a record type, BALTER creates a hash area for the record type in this realm (except in the case of distributable lists).
- When the record type is the member record type of a distributable list and this is defined with LOCATION MODE IS CALC, an indirect CALC area which is determined explicitly by the DETACHED WITHIN clause of the MODE IS LIST statement in the SSL or alternatively by the first realm name of the aforementioned WITHIN clause is used in one realm. A corresponding change causes the CALC area to be created anew. If only one or more realms are added for a member record type of a distributable list, the existing list remains unchanged.

- When *realm-name-1* is changed, BALTER relocates the record type's DBTT to the newly specified realm if no SSL specification prevents this.

  When a realm is omitted in the WITHIN clause, no records of the record type concerned may be stored in this realm, otherwise (except in the case of distributable lists) BALTER will abort the restructuring even if another realm is made available for these records.
  When a realm is omitted in the case of distributable lists, this results in the list being recreated even if this realm contains no records.
  When a distributable list is removed, for instance by changing the MODE clause to POINTER-ARRAY, the records of the member record type remain in the realm in which they were stored in the distributable list. If these records are accessed using LOCATION MODE IS CALC, it must be ensured that the AREA-ID is supplied with correct information after the list has been removed. To avoid any access problems occurring here, all pages in the list can be relocated with the UDS online utility to one realm before restructuring takes place, and just one realm can be declared for the record type when the list is removed.
  To improve the runtime of BALTER with large databases, you can proceed as follows:
  If you increase the realm allocation of an owner record type from one to two or more realms, you should restructure the database in a first cycle by changing the DETACHED-specifications (including the default values) to DETACHED WITHIN *realm-name* for the tables that depend on the owner record type. *realm-name* is the realm in which the owner records are currently stored. In a second restructuring cycle you should delete the DETACHED specification.

identifier-2

Can be changed as required.

Effects on stored data:

There are no effects on data; the change need only be taken into account in the application programs.

```
[SEARCH KEY IS item-name,...

    USING { CALC[ hash-routine] | INDEX} [ NAME IS name
]

    DUPLICATES ARE[ NOT] ALLOWED] ... .
```

In the SEARCH-KEY clause any change is permitted. It is possible to:

- change existing SEARCH keys
- define new SEARCH keys
- omit SEARCH keys which are no longer required

item-name-3,...

There are no restrictions on which data items can be used as a SEARCH key or on which data items are combined to form the SEARCH key.

USING ...

Any change is allowed.

Effects on stored data:

- CALC -> INDEX:
  BALTER creates a multi-level SEARCH key table and releases the memory space of the indirect hash area.
- INDEX -> CALC:
  BALTER creates the SEARCH key table as an indirect hash area and releases the previously allocated memory space.
- Any other hash routine:
  BALTER creates a new hash area and releases the memory space of the original hash area.

name This specification can be omitted, added or changed.

DUPLICATES...

For duplicates the same applies here as for the LOCATION MODE clause (see section “Record entry”).

> BALTER checks for illegal key value duplicates and logs them only if it has to create a multi-level SEARCH key table or an indirect hash area.

```
{[level-number] record-element-name
   {PICTURE IS {mask-string |
               LX (integer-1) DEPENDING ON item-name-4}
    TYPE IS {FIXED REAL {BINARY {15 | 31 | 63} |
                         DECIMAL[ integer-2[,integer-3]]} |
             CHARACTER[ integer-4[ DEPENDING ON item-name-5]] |
             DATABASE-KEY |
             DATABASE-KEY-LONG }
   [OCCURS integer-5 TIMES].}...
```

The structure of record types can be modified as required, but it is important to bear in mind the following:

- Length of record type
  Record types which are stored in a single-level list must not be lengthened!
  A single-level list means ORDER IS LAST, FIRST, PRIOR or IMMATERIAL and MODE IS LIST.
- Number and sequence of item
  You may change the order of items. You may also delete items.
  Newly defined items are initialized by BALTER dependent on item type:
  - alphanumeric items with blanks
  - national items with national blanks (Unicode)
  - numeric items with the value zero

  With the FILL statement user-defined values for initialization can be specified.
- Length of items
  Items can be lengthened or shortened. BALTER proceeds as follows, depending on the item type:
  - Alphanumeric items:
    When items are lengthened, BALTER pads to the right with blanks. When items are shortened, BALTER truncates to the right by the appropriate number of characters.
  - National items:
    When items are lengthened, BALTER pads to the right with blanks (Unicode). When items are shortened, BALTER truncates to the right by the appropriate number of characters.
  - Numeric items:
    When items are lengthened, BALTER pads to the left of the decimal point with zeros. When items are shortened, BALTER truncates to the left of the decimal point by the appropriate number of characters. Significant digits may be lost as a result. If the item has a sign, the sign is retained, provided the new item definition allows a sign.

- Database key items
  In the case of database key items, their type can be changed from DATABASE-KEY to DATABASE-KEY-LONG, and vice versa.

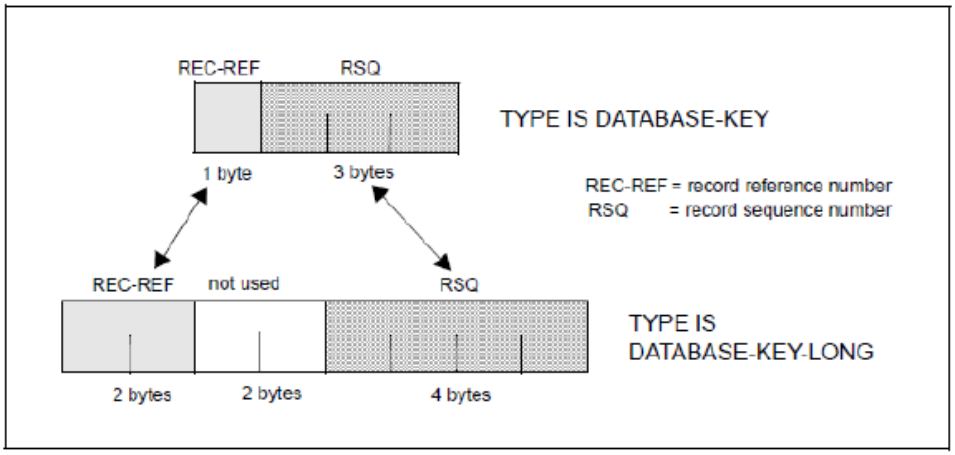


Figure 25: Changing the type of database key items

TYPE IS DATABASE-KEY -> TYPE IS DATABASE-KEY-LONG:
BALTER copies the 1-byte record reference number (REC-REF) of the DATABASE-KEY item right-justified into the corresponding 2-byte area of the DATABASE-KEY-LONG item. The 3-byte record sequence number (RSQ) of the DATABASE-KEY item is copied right-justified into the corresponding 4-byte area of the DATABASE-KEY-LONG item.

TYPE IS DATABASE-KEY-LONG -> TYPE IS DATABASE-KEY:
BALTER copies the right byte of the 2-byte long record reference number (REC-REF) of the DATABASE-KEY-LONG item right-justified into the corresponding 1-byte area of the DATABASE-KEY item. The right 3 bytes of the 4-byte long record sequence number (RSQ) of the DATABASE-KEY-LONG item are copied right-justified into the corresponding 3-byte area of the DATABASE-KEY item.

Due to the truncation of positions on the left for the REC-REF and RSQ when converting from TYPE IS DATABASE-KEY-LONG to TYPE IS DATABASE-KEY, data is lost if the REC-REF > 254 and/or the RSQ > 2^{24} -1. If this occurs, BALTER issues a warning, which contains the original DATABASE-KEY-LONG value; however, the BALTER run is not aborted. The database remains in a consistent state (see also “BCHECK” in the "Recovery, Information and Reorganization" manual). The logical consistency, i.e. the consistency of application data, will need to be verified and ensured.

- Type of items
  When you change the type of a **numeric** item to another numeric type (e.g. TYPE IS DECIMAL ->TYPE IS BINARY), BALTER converts the data.
  If you change the type of an **unpacked numeric** item to an alphanumeric type (of fixed length), BALTER will proceed as described below.
  For all other type changes, BALTER fills the item in accordance with the new type with blanks or with the value zero. This also applies in particular for all type changes from or to national.

unpacked numeric ->alphanumeric (fixed length):
If required, BALTER converts the data by proceeding as follows:

1. BALTER copies the numeric digit sequence left-justified into the alphanumeric target item. Leading zeros are retained.
   Any symbol “V” (decimal point) that may be in the definition of the source item is ignored by BALTER, i.e. the whole part **and** the decimal positions are copied. Existing “P” symbols (implicit multiplication with 10) in the definition of the source item are taken into account by BALTER to the extent possible.
   Depending on the size of the alphanumeric target item, BALTER proceeds as follows:
   - If the target item contains fewer positions than the sequence of digits to be copied (including the considered “P” symbols), the excess positions in the digit sequence are truncated.
   - If the target item contains more positions than the sequence of digits to be copied (including the considered “P” symbols), the digit sequence is padded on the right up to the target item length with X‘40‘.
2. In the hexadecimal representation of the digit sequence obtained in accordance with 1, BALTER converts the second-last half-byte (sign) to hexadecimal “F”. This applies, in particular, even if the definition of the source item contains the symbol “S“ (sign).

Examples

	Source item (unpacked numeric item)			**Target item (alphanumeric item)**	
	Item definition	**Item contents (hexadecimal)**		**Item definition**	**Item contents (hexadecimal)**
1)	PIC 9999	F8 F1 F2 C3	->	PIC XXXX	F8 F1 F2 F3
2)	PIC S999PP	F5 F2 D3	->	PIC XXXXX	F5 F2 F3 F0 F0
3)	PIC S99V99	F1 F4 F3 D5	->	PIC XXXX	F1 F4 F3 F5
4)	PIC 9999	F1 F2 F3 F4	->	PIC XXX	F1 F2 F3
5)	PIC 999P	F5 F2 E3	->	PIC XXXXXX	F5 F2 F3 F0 40 40
6)	PIC S999V99	F0 F2 F3 F4 D5	->	PIC XXXXXX	F0 F2 F3 F4 F5 40
7)	PIC SV999	F0 F2 B3	->	PIC XX	F0 F2
8)	PIC V999	F7 F2 A3	->	PIC XX	F7 F2
9)	PIC S999PP	F0 F2 B3	->	PIC X	F0
10)	PIC 999PP	F8 F2 F3	->	PIC X	F8

- Variable item
  In record types, you can only add or modify a variable item providing no records of this record type are stored in the database (see "Summary of restrictions").
  However, items of fixed length can be added to or omitted from records with a variable item when records are stored. You can also modify the length of such items. In addition, you can implement all changes in the schema which lead to a change in the system part (SCD).
- Position of the decimal point
  If the position of the decimal point or the scale factor is changed, BALTER shifts the digits within the item so that the former value is retained. Digits shifted beyond the left or right boundary of the item are lost; BALTER does not round up.
- Repetition factor
  When the repetition factor of vectors or repeating groups is reduced, BALTER truncates the items at the end of the vector or the repeating group. Increasing the repetition factor causes BALTER to initialize new items with blanks or zeros according to type. It is easy to check whether BALTER copies, initializes or omits the contents of the items when the repetition factor is changed:
  For indexed items the BALTER conversion routine uses a three-level index for both the old and new definitions. If the old item and the new item are represented with three-level indexing, they can be compared easily.

> **CAUTION!**
> Reducing the repetition factor of vectors or repeating groups can cause loss of data.

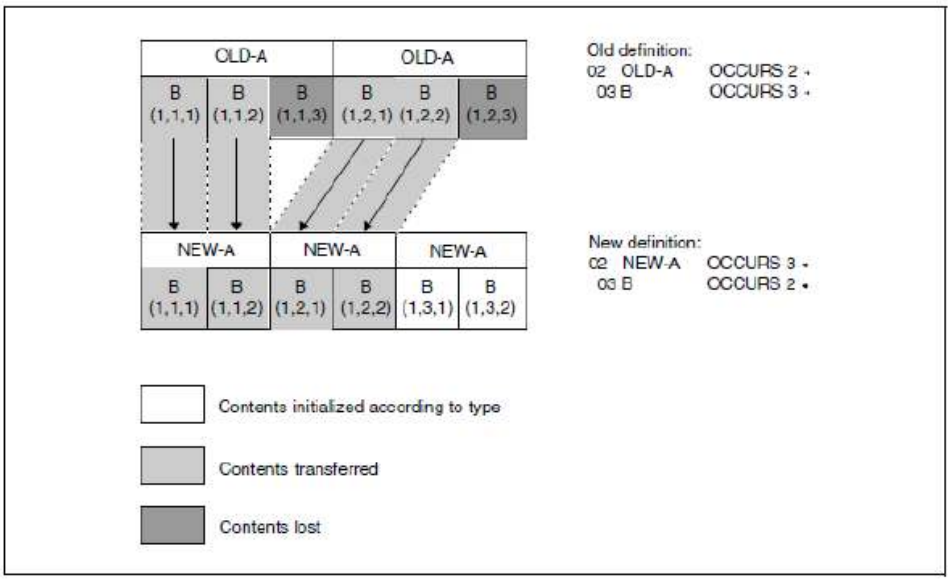


Figure 26: Changing the item contents of stored records by modifying the repetition factor

Set entry

```
SET NAME IS set-name
```

When changing the set entry, the following applies:

- Omission of sets is allowed without restriction.
- Addition of sets is permissible but subject to certain restrictions on the type of set membership (these do not apply, however, to the addition of SYSTEM sets).
- The changing of sets is subject to certain restrictions.

```
[SET IS DYNAMIC]
```

Dynamic sets may be added or omitted. The conversion of a set into a dynamic set and vice versa is, however, prohibited.

```
ORDER IS {LAST | FIRST | NEXT | PRIOR | IMMATERIAL |
          SORTED[ INDEXED[ NAME IS name]] BY
                 {DATABASE-KEY | DEFINED KEYS DUPLICATES ARE[ NOT] ALLOWED}}
```

Changes to the ORDER clause must be considered in relation to the MODE clause of the SSL (see "MODE clause" in the "Design and Definition" manual).

The following applies to changes to the ORDER clause:
If no records of the member record type of the set to be changed are stored, all modifications are allowed; if records of the member record type of the set to be changed are stored, the modification of the ORDER clause is subject to certain restrictions.

Which modifications of the ORDER clause are allowed when records of the member record type are stored can be seen in the following table:

old schema	**new schema**	ORDER IS {LAST | FIRST | NEXT | PRIOR | IMMATERIAL}	ORDER IS SORTED [INDEXED ...] .....
ORDER IS	MODE IS		
{LAST | FIRST | NEXT | PRIOR | IMMATERIAL}	POINTER-ARRAY	No restrictions on use	allowed without restrictions
	LIST	Only allowed if the member record type (incl. CD) is not lengthened as a result of the change.	
	CHAIN	Only allowed if the SCD of the owner/member record type need not be enlarged (see table 28)	
SORTED [INDEXED ...]	POINTER-ARRAY	Not allowed.	
	LIST	Not allowed.	
	CHAIN	Only allowed if the SCD of the owner/member record type need not be enlarged (see table 28)	

Table 27: Changes to the ORDER clause when member records are stored

When and how the set connection data (SCD) for a set with MODE IS CHAIN is altered is shown in the following table:

new schema **old schema**	ORDER IS	{FIRST | NEXT | PRIOR | IMMATERIAL| SORTED | [INDEXED...] ...}	{FIRST | NEXT | PRIOR | IMMATERIAL}	SORTED [INDEXED...] ...	LAST	
ORDER IS	MODE IS MODE IS	CHAIN	CHAIN LINKED TO PRIOR	CHAIN LINKED TO PRIOR	CHAIN	CHAIN LINKED TO PRIOR
{FIRST | NEXT | PRIOR | IMMATERIAL | SORTED | [INDEXED...]}	CHAIN	-	2) >Member-SCD >Owner-SCD	1) >Member-SCD >Owner-SCD	2) >Owner-SCD	2) >Owner-SCD >Member-SCD
	CHAIN LINKED TO PRIOR	1) <Owner-SCD <Member-SCD	-	-	1) <Member-SCD	-
LAST	CHAIN	1) <Owner-SCD	2) >Member-SCD	1) >Member-SCD	-	2) >Member-SCD
	CHAIN LINKED TO PRIOR	1) <Owner-SCD <Member-SCD	-	-	1) <Member-SCD	-

Table 28: Modifying the SCD for a set with MODE IS CHAIN

< is shortened

> is lengthened

- no change in SCD length

1) modification allowed since SCD unchanged or shorter or in the new schema ORDER IS SORTED [INDEXED...]...

2) modification not allowed since SCD longer and in the new schema not ORDER IS SORTED [INDEXED...]...

DUPLICATES ...

> The same applies as for the LOCATION MODE clause (see section “Record entry”)

NAME IS *name*

This specification can be modified, added or omitted as required.

```
OWNER IS {record-name | SYSTEM}
```

Changing the OWNER clause of a set is prohibited!

```
MEMBER IS record-name {MANDATORY | OPTIONAL} {AUTOMATIC | MANUAL}
```

record-name

Specification of a new member record type is prohibited.

set membership

If an existing set is changed and the database contains

- no records of the member record type, the set membership may be changed as required.
- records of the member record type, the set membership must not be converted from OPTIONAL to MANDATORY AUTOMATIC if certain member records are not allocated to an owner.

> If the set is not modified in any other way, this change causes the database to be inconsistent. If the set must be processed to carry out other changes, this change causes BALTER to terminate abnormally.

If a new set is defined, and the database contains

- no records of the member record type, the set membership can be selected as required.
- records of the member record type, then MANDATORY AUTOMATIC must not be defined as set membership for sets other than SYSTEM sets. BALTER cannot automatically allocate member records to owner records if the set is not singular, i.e. the DB administrator must decide which member records are to be allocated to which owner record and then carry out allocation by program with the aid of the CONNECT statement.

Effects on stored data:

Changing set membership has no effect on the stored data. Possible changes in set membership must, however, be taken into account in programs when records are stored or deleted.

```
[{ASCENDING | DESCENDING} KEY IS item-name-1,...]
```

All modifications are allowed.

Effects on stored data:

BALTER recreates the pointer arrays, lists or chains concerned in accordance with the modified sort criteria.

The recreation of a list involves transferring the member records to another storage area, since BALTER has to rearrange them in a new sort sequence.

```
[SEARCH KEY IS item-name-2,... USING {CALC | INDEX}[ NAME IS name]
        DUPLICATES ARE[ NOT] ALLOWED]...
```

The same modification possibilities exist for SEARCH keys at both set level and at record type level (see the section on changing the SEARCH KEY clause).

```
[SET OCCURRRENCE SELECTION IS THRU
  {CURRENT OF SET |
   LOCATION MODE OF OWNER
          [ALIAS FOR {item-name-3 | identifier-1} IS identifier-2]...} ].
```

Modification of the SET OCCURRENCE SELECTION clause is allowed.

Effects on stored data:

Modification has no effect on the stored data. The DB administrator must, however, take any modification into account in his DB applications.

6.2 Modifying the SSL

The runtime performance of the DB applications is very significantly affected by the storage structure of the database. In order to achieve optimum results, you must match the storage structure to time-critical applications by modifying the SSL clauses. This can bring about considerable improvements in DB application runtimes even if the schema and programs remain unchanged.

Even if the old SSL is used, it must be recompiled since otherwise BGSIA uses the default values of the SSL clauses.

The following modifications to the storage structure are allowed:

- at record type level
  - relocation of DBTTs to other realms
  - definition of PLACEMENT OPTIMIZATION
  - redefinition of location, type and number of pages for the purpose of reorganization in SEARCH key tables
  - compression or decompression of records (with restrictions)
- at set level
  - modification of MODE clause (with restrictions)
  - relocation of pointer arrays or lists of a set
  - linking the owners of a set via pointer arrays or lists, or cancelling an existing link
  - changing storage space in which the tables of the set occurrences of a set are created or enlarged
  - definition of the number of pages for the reorganization of tables of a set
  - definition of location, type and number of pages for the purpose of reorganization in SEARCH key and sort key tables
  - linking members with their appropriate owner or cancelling an existing link

All SSL clauses are optional, i.e. they can be specified for the first time, or omitted. BGSIA assumes the default value for every clause omitted.

When restructuring the clauses of the Schema DDL and of the SSL are subject to the same rules as during database definition (see the "Design and Definition" manual). If Schema DDL clauses have been modified, all related clauses in the SSL must be adapted to comply with these rules.

The following is a detailed description of possible modifications to the storage structure and their effects on the stored data.

Schema entry

```
STORAGE STRUCTURE OF SCHEMA schema-name.
```

schema-name

Must be the same as the name in the schema entry of the new Schema DDL.

Record entry

```
RECORD NAME IS record-name
```

record-name

Must designate a record type of the new Schema DDL.

```
[DATABASE-KEY-TRANSLATION-TABLE[ IS integer-1][ WITHIN realm-name-1]]
```

integer-1

The number of DBTT entries can be changed using the BREORG utility routine for example. BALTER ignores any change to the size specification.

realm-name-1

Can be modified.

Effects on stored data:

BALTER relocates the DBTT to the specified realm.

```
[PLACEMENT OPTIMIZATION FOR SET set-name]
```

Any modification is allowed.

Effects on stored data:

BALTER has no effect on the position of records already stored. New records are stored in the database by the DBH in accordance with the new specifications noted by BALTER.

If this clause is specified for the first time for a record type stored in a direct hash area, BALTER creates an indirect hash area but does not change the position of the records concerned; instead it simply converts the CALC pages of the former hash area into normal data pages.

```
[POPULATION IS {integer-2 WITHIN realm-name-2},...]
```

integer

The size of a hash area can be changed using the BREORG utility routine. BALTER ignores any change of the size specification.

Exception

BALTER must create the hash area anew in the event of a change:

- from direct to indirect hash area or vice versa
- of hash routine
- of composition of the CALC key
- of realm in which the hash area is to be located.

BALTER uses *integer-2* in order to calculate the new size of the hash area.

```
[INDEX NAME IS name[ PLACING IS WITHIN realm-name-3]
      [TYPE IS {DATABASE-KEY-LIST |
                REPEATED-KEY
                    [DYNAMIC REORGANIZATION SPANS integer-3 PAGES]}]]
```

PLACING IS ...

SEARCH key tables can be relocated to other realms.

TYPE IS ...

The structure of the SEARCH key tables can be modified by changing the TYPE clause to DATABASE-KEY-LIST or REPEATED-KEY.

DYNAMIC REORGANIZATION

The number of pages for reorganization purposes can be redefined as required.

Effects on stored data:

Changing the number of pages does not have an immediate effect during restructuring. It is not until later during database operation when storing records that the change takes effect.

```
[COMPRESSION FOR ALL ITEMS].
```

It is permissible to store in compressed format records which were previously stored without compression and vice versa, subject to certain restrictions.

- Changing from ’non-compressed’ to ’compressed’ format:

  BALTER does not compress records which have already been stored and therefore does not remove empty data items from data records. Instead it merely adds a 4-byte compression item to the set connection data (SCD) of the records of the record type concerned.
  In so doing BALTER lengthens each record of the record type by 4 bytes and must therefore transfer to other pages the records which have no more space in the pages previously occupied by this record type.
- Changing from ’compressed’ to ’non-compressed’ format:

  The change depends on how the stored records of the record type in question have been entered:
  - If the records have been entered in their full length (e.g. with BINILOAD), BALTER deletes the compression item in the SCD of the records. Since each record is thereby shortened by 4 bytes, BALTER pushes the records together in each page, thus releasing space.
  - If the records are compressed, i.e. not entered in their full length, BALTER abnormally terminates restructuring!

    BALTER cannot process compressed records even if database changes do not concern compression:

    Neither the user section nor the system information in the records may be altered. Tables or hash areas which contain items of this record type as keys cannot be created by BALTER.

Set entry

```
SET NAME IS set-name
```

set-name

Must be the same as the name of a set in the new Schema DDL.

```
[MODE IS
  {CHAIN[ LINKED TO PRIOR] |
   {POINTER-ARRAY | LIST} {ATTACHED TO OWNER |
                           DETACHED[ WITHIN realm-name-1]
                                   [ WITH PHYSICAL LINK]}}]
```

The MODE clause should be considered in relation to the ORDER clause of the Schema DDL (see the section on changing the ORDER clause on "Modifying the Schema DDL").

If, in the ORDER clause of the new Schema DDL

- SORTED[INDEXED] has been specified, the linkage method specified in the MODE clause may be changed as required;
- SORTED[INDEXED] has not been specified, changes of the linkage method are prohibited if records of the member record type have already been stored.

When a distributable list (MODE IS LIST) is changed to an address list (MODE IS POINTER-ARRAY), the member records remain in the realm in which they were currently stored in the list. It may be necessary to adjust the access logic of the application programs.

When a change is made in MODE IS LIST, a list is built. Records of the record type affected may possibly be relocated to another realm.

ATTACHED/DETACHED

Changing ATTACHED to DETACHED or vice versa is allowed with the restriction that records stored in a list must not be relocated to another realm.

Effects on stored data:

- Existing tables are reallocated by BALTER only if it is necessary to relocate them to another realm on the basis of a change in the WITHIN *realm-name-1* specification.
- The location of a list without a DETACHED WITHIN clause is determined by the position of the owner. Consequently records can be relocated in the event of a change from pointer array to list, and existing programs with direct access (FIND4) must be adjusted.

WITHIN *realm-name-1*

A pointer array may be relocated to another realm. Except in the case of distributable lists, a list may be relocated only when no member records are stored.

In the case of distributable lists *realm-name-1* implicitly determines the location of the table part (level 1 through level N pages) of the list and the location of an indirect CALC area in the case of LOCATION MODE IS CALC if the location is not specified explicitly in the SSL of the MODE IS LIST statement.

The table pages can be relocated to another realm even if member records are stored in the distributable list.

Effects on stored data:

The list and any CALC area will be recreated. In the process the member records will be distributed approximately evenly over the realms involved.

WITH PHYSICAL LINK

A pointer array or list which is stored separate from the owner can also be linked to the associated owner for the first time, or an existing link can be cancelled.

Effects on stored data:

- If an additional pointer is set up, BALTER adds the physical address of the highest level of the table to the set connection data (SCD) of the owner records. Since the owner records are thereby lengthened, BALTER relocates, within the database, the records for which no more space is available in the occupied pages.
- If the additional pointer is removed, BALTER deletes the physical address of the highest level of the table from the SCD of the owner records.

```
[POPULATION IS integer-1[ INCREASE IS integer-2]]
```

integer-1

Can be altered as required. BALTER takes this specification into account together with a corresponding FILLING WITH POPULATION statement. A modification also has an effect when saving owner records or deleting member records.
BALTER does not modify existing tables.

integer-2

Can be altered as required. Any change has no effect until table extensions become necessary when storing new member records.

```
[DYNAMIC REORGANIZATION SPANS integer-3 PAGES]
```

The number of pages for the reorganization of the set tables can be altered as required.

```
[INDEX NAME IS name
   [ PLACING IS {ATTACHED TO OWNER | DETACHED [WITHIN realm-name-2]}
    [TYPE IS {DATABASE-KEY-LIST |
              REPEATED-KEY [DYNAMIC REORGANIZATION SPANS integer PAGES]}]]
```

The usage of the INDEX clause for set entry is analogous to that of the INDEX clause for record entry (see the section on changing the INDEX clause above).

ATTACHED/DETACHED

Changing ATTACHED to DETACHED or vice versa is allowed without restriction.

Effects on stored data:

Changes have the same effects as in the MODE clause (see the section on changing the MODE clause above).

WITHIN *realm-name-2*

Can be added or omitted, or *realm-name-2* can be changed.

Effects on stored data:

In the event of a change, BALTER relocates the tables to the specified realm.

```
[MEMBER IS PHYSICALLY LINKED TO OWNER].
```

Additional linking of the member records to the associated owner record can be requested for the first time or cancelled.

Effects on stored data:

- If the clause is specified for the first time, BALTER adds to the SCD of the member records the physical pointer to the associated owner record. Since the member records are thereby lengthened, BALTER relocates, within the database, the records for which no more space is available in the occupied pages.
- If the clause is removed from the SSL, BALTER removes the physical pointer to the associated owner record from the SCD of the member records.

6.3 Summary of restrictions

This section provides an overview of which changes are not permitted in the Schema DDL and the SSL at all or only with restrictions when records of a record type concerned are stored. BALTER issues warnings relating to these modifications in the analysis listing.

- Compression
  Compressed records cannot be processed by BALTER. BALTER abnormally terminates restructuring if you do not unload the record type before restructuring and if you:
  - change the user or system part of the record type
  - change the structure of tables or hash areas which contain items of this record type as keys
- Variable items
  If records for a record type with a variable item are stored in the database, restrictions apply for restructuring. BALTER terminates restructuring abnormally in the following cases:
  - An item of variable length is added
  - An item of variable length is removed
  - The maximum length of an item of variable length is changed
- Lists
  Record types stored in a single-level list must not be lengthened. A single-level list means ORDER IS LAST, FIRST, NEXT, PRIOR or IMMATERIAL and MODE IS LIST.

  Lists which cannot be distributed may not be relocated to other realms.

  Distributable lists can be relocated as required. If only realms are added, the list remains unchanged and the restructuring process as a whole can be executed very quickly.
- Cyclic set structures
  If BALTER is to make a modification to all sets of a cyclic set structure (see "Cycle" in the "Design and Definition" manual), a modification in which the owner record type must be processed before the member record type, a deadlock situation will arise. Such modifications are either the creation of a new chain, or the additional linking of the member records to the appropriate owner record for sets in which the owner record type must be relocated in the course of further modifications.

  If modifications of this nature are planned for a cyclic set structure, restructuring must be split into two stages.

6.3.1 Schema DDL modifications

Schema entry

No restrictions

Realm entry

Conversion of a temporary realm into a non-temporary realm or vice versa is not allowed.

Record entry

WITHIN clause

If a realm is omitted in the WITHIN clause, no records of the record type concerned may be stored in this realm except in the case of distributable lists.

Record length

Record types which are members in a single-level list must not be lengthened.

Set entry

DYNAMIC clause

Conversion of a set into a dynamic set and vice versa is not allowed.

ORDER clause

If records of the member record type of the set to be modified have been stored, the following restrictions must be observed:

- for ORDER IS SORTED[INDEXED] and MODE IS POINTER-ARRAY/LIST the ORDER clause must not be changed to ORDER IS LAST, FIRST, NEXT, PRIOR or IMMATERIAL;
- for any ORDER clause and MODE IS CHAIN, the ORDER clause can only be changed to ORDER IS LAST, FIRST, NEXT, PRIOR or IMMATERIAL if, in so doing, the SCD of the owner record type or of the member record type is not lengthened;
- for ORDER IS LAST, FIRST, NEXT, PRIOR or IMMATERIAL and MODE IS LIST, the ORDER clause can only be changed to ORDER IS LAST, FIRST, NEXT, PRIOR or IMMATERIAL if, in so doing, the member record type including SCD is not lengthened.

OWNER clause

The OWNER clause of a set must not be modified.

MEMBER clause

record-name:
Specifying a new member record type is not allowed.

set membership

- If an existing set is modified and if records of the member record type are stored in the database, set membership must not be converted from OPTIONAL to MANDATORY AUTOMATIC if certain member records stored are not allocated to an owner.

- If a new set is defined and if records of the member record type are stored in the database, AUTOMATIC must not be defined as set membership, except for SYSTEM sets.

6.3.2 SSL modifications

Schema entry

No restrictions.

Record entry

DBTT clause

The number of DBTT entries can be changed by means of the BREORG utility routine for example. BALTER ignores any change to the size specification.

POPULATION clause

The size of a hash area can be changed by means of the BREORG utility routine. BALTER ignores any change to the size specification, unless it has to recreate the hash area in the course of further modifications.

COMPRESSION clause

BALTER cannot process compressed records, i.e. it cannot perform decompression, reallocation, sorting or deletion.

Set entry

MODE clause

If the ORDER clause in the new schema is LAST, FIRST, NEXT, PRIOR or IMMATERIAL, the linkage method in the MODE clause may only be modified if no records of the member record type are stored.

6.4 Checking the consistency of the database

A consistent database is the most important prerequisite for successful restructuring. The BCHECK utility routine should be used to ascertain the consistency of the database (see the "Recovery, Information and Reorganization" manual).

> If BCHECK detects any inconsistencies, the database should on no account be restructured before it has been recovered, otherwise errors may be compounded and this will make recovery of the inconsistent database even more difficult.

6.5 Checking free memory space

In order to adapt the stored data to the modified schema or modified storage structure, BALTER requires additional free memory space

- in order to create new tables, hash areas or DBTTs
- in order to re-store records, tables, hash areas or DBTTs.

If the free memory space in the realms is not large enough to do this, BALTER automatically extends the realms concerned, provided this is possible (requirement: secondary allocation for memory space > 0). For details, please see "Database Operation" manual, Automatic realm extension by means of utility routines.

If automatic realm extension is not possible (secondary allocation for memory space = 0 or no more disk storage is available), BALTER aborts restructuring! As a result your database is inconsistent and you must revert to the status before restructuring began and start again from the beginning.

Realms without automatic realm extension

When one or more realms with a secondary allocation = 0 are configured, e.g. because you do not want automatic realm extension, you must ensure that the available space is sufficient before you begin restructuring. For this purpose you must use BSTATUS to display how much memory space is still free in the various realms of your database and estimate whether the free space is sufficient for the planned restructuring.

> ¡ This should be done as early as possible, since BREORG can no longer enlarge the realms of the database once BCHANGE has processed the database for restructuring purposes.You can also make use of the analysis report for this purpose (see section “Description of the analysis report (REPORT phase)”).

BALTER selects the processing sequence in such a way that it first deletes those elements from the database which are no longer contained in the new schema. The space released can then later be used in the course of other activities, such as the creation of new tables, the creation of new hash areas, the re-storing of records etc. In the same way the space released after re-storage of a record type can be used again. Unfavorable configurations may however lead to a situation in which space which is free at the end of a realm is allocated first of all, leaving gaps at the beginning of the realm. If this is the case, it is possible, by means of the BREORG utility routine, to relocate hash areas and tables to the free pages at the beginning of the realm:

- hash areas with REORGANIZE-CALC
- tables with REORGANIZE-SET.

You can then reduce the realm by deleting the pages released at the end of the realm.

The overviews on the following pages are intended to help you estimate the size of the free memory space which must be available for restructuring purposes in the realms of the database. If insufficient free space is found in a realm, it must be extended using BREORG before BCHANGE is started for purposes of preparing restructuring.

DBTT

BALTER lists in an analysis listing the free memory space it needs to restore a DBTT or to create a new DBTT.

The new size of a DBTT can also be determined using the formulas specified in the table.

Modification in the database	**R e a s o n**	**Memory requirement**		
		Formula (see section "Calculation formulas")	**SSL clause specifying size**	**Standard size**
Creating a new DBTT	new record type defined	1	DBTT clause	1 page
Re-storing DBTT	Number of DBTT columns changed (only in owner record) by • modified linkage method in MODE clause • addition or omission of SEARCH key (set level) • changing the number of sets in which this record type is owner DBTT relocated by • new *realm-name-1* in the WIHTIN clause of the DDLL (record type level) • new *realm-name-1* in the DBTT clause of the SSL (record type level)	2	-	Orignal number of DBTT entries

Table 29: Memory requirement for DBTT modifications

Hash areas

The number of pages which BALTER needs to create a hash area is given in the analysis listing.

The size of the hash areas can also be determined with the aid of formulas.A hash area always occupies contiguous pages of a realm.

Modification in the database	R e a s o n	Memory requirement		
		Formula (see section "Calculation formulas")	SSL clause specifying size	Standard size
Creating a direct or indirect hash area (primary key)	• New record type defined with LOCATION CALC • LOCATION MODE changed to CALC • hash routine or CALC key changed in LOCATION CALC • record type with LOCATION CALC lengthened /shortened	3 or 4	POPULATION clause	1 page
Creating an indirect hash area (secondary key)	• new SEARCH key defined with using CALC • definition of a SEARCH key changed in USING CALC • CALC key or hash routine of a SE ARCH key changed • indirect hash area relocated to another realm	4	DBTT clause	1 page
Converting a direct into an indirect hash area (primary key)	A record type defined with LOCATION CALC • is owner or member in a set for which PLACEMENT OPTIMIZATION has been for first time • has become a member in a list • has been stored for the first time in compressed form in the new schema	4	POPULATION clause	1 page

Converting an indirect into a direct hash area (primary key)	In a record type defined with LOCATION CALC • PLACEMENT OPTIMIZATION is omitted in all sets in which this record type is a member • compressed storage is cancelled • the MODE clause of a set in which this record type is a member is changed from MODE IS LIST to POINTER-ARRAY/CHAIN	3		

Table 30: Memory requirement for modifications affecting hash areas

Tables

Relocating tables

BALTER can relocate only single-level tables in their entirety. If it relocates tables to another realm, the space required in that realm is equal to the space originally occupied by the table.

Creating new tables:

- SYSTEM set or TYPE IS DATABASE-KEY-LIST:
  each set occurrence table occupies at least one page.

  Non-singular set and not TYPE IS DATABASE-KEY-LIST:
  each set occurrence table takes up only as much space as it needs.

BALTER stores tables for the same set using a space-saving strategy such that there may be a number of tables in the same page, if the tables are stored DETACHED.

Table	**Change in Schema DDL or in SSL**	Change in DB		Memory requirement
		Create new table(s)	**Relocate table(s)**	
Indexed SEARCH key table (record type level)	new SEARCH key defined with USING INDEX	X	-	min. 1 page max. refer to formula 5
	Key item changed	X	-	
	Table type changed REPEATED-KEY <--> DATABASE-KEY-LIST	X	-	
	Definition of a SEARCH key changed to USING INDEX	X	-	
	SEARCH key table relocated to another realm	X	-	

Indexed SEARCH key table (set level)	Table type changed REPEATED-KEY --> DATABASE-KEY-LIST or key item modified in type DATABASE-KEY-LIST	X	-	min. 1 page per set occurrence table with SYSTEM set: min. 1 page max. refer to formula 5
	New SEARCH key defined with USING INDEX	X	-	with SYSTEM set: min. 1 page max. refer to formula 5 with non-singular set: minimum not specifiable. BALTER stores several set occurrence tables together, provided they are small enough and intended to go in the same realm. max. size of a single set occurrence table: refer to formula 5
	Key item changed	X	-	
	Table type changed DATABASE-KEY-LIST --> REPEATED-KEY	X	-	
	Definition of a SEARCH key changed to USING INDEX	X	-	
	SEARCH key table relocated to another realm	X	-	
Indexed pointer array	MODE clause changed to POINTER ARRAY	X	-	
	ASC/DSC key changed	X	-	
	Pointer array relocated to another realm	X	-	
	New set defined with MODE IS POINTER-ARRAY	X	-	
Indexed sort key table (MODE IS CHAIN)	New set defined with MODE IS CHAIN and ORDER SORTED INDEXED	X	-	
	Definition of a set changed to MODE IS and ORDER SORTED INDEXED	X	-	
	ASC/DESC key changed	X	-	
	Table relocated to another realm	X	-	

Non-indexed pointer array	Pointer array relocated to another realm	-	X	Original size
Indexed list	MODE clause changed to LIST	X	-	with SYSTEM set: min. 1 page max. refer to formula 5 with non-singular set: minimum not specifiable. BALTER stores several set occurrence tables together, provided they are small enough and intended to go in the same realm. max. size of a single set occurrence table: refer to formula 5
	ASC/DESC key changed	X	-	
	New set defined with MODE IS LIST	X	-	

Table 32: Memory requirements for table changes

If a pointer array, list or chain is added SORTED INDEXED to an empty SYSTEM set, BALTER creates an empty table for it one page in size.
The following changes are allowed only when there are no member records:

- creation of a single-level list or a single-level pointer array (caused e.g. by MODE definition, changed ORDER clause)
- relocation of a single-level list in the same realm (e.g. by record lengthening)
- relocation of a list to another realm

Record reallocation

When records are reallocated, a distinction is made by BALTER between a partial and a complete reallocation:

- A complete reallocation

  occurs when the new schema prescribes a special form of storage. This is the case with LOCATION MODE IS CALC for direct hash areas and record types stored in a list.

  A complete reallocation is carried out by BALTER when:

  - you use a hash routine other than the standard routine in the new schema
  - you change the CALC key for a direct hash area in the new schema
  - you do not specify the reason for indirect storage in the new schema
  - you introduce LOCATION MODE IS CALC in the new schema
  - you change the ASC/DESC key of a list in the new schema
  - you redefine MODE IS LIST in the new schema.

  In the case of a complete reallocation, BALTER relocates all records to new pages and releases the old pages. This means, however, that during the BALTER run double the storage space required for storing the records must be available.

  If a record type with a special form of storage is lengthened, a complete reallocation is automatically initiated, even if there is no change in the form of storage itself.

- A partial reallocation

  occurs when either the user or system part of the records is lengthened, thus making more pages necessary for storage purposes.

  In the case of a partial reallocation, as many records as possible are kept in their original pages, and the rest are relocated by BALTER to new pages.

- Shortening records

  If a record type with a special form of storage is shortened, a complete reallocation is automatically initiated as for the lengthening of records, even if there is no change in the form of storage.

  If a record type without a special form of storage is shortened, BALTER pushes all the records within a page together. This releases space within the individual pages. BALTER does not relocate the records to other pages, however, so no whole pages are released.

BALTER does not indicate the amount of space required to relocate records in the analysis listing. It is, however, possible to estimate the required storage space with the help of table 33 of and the calculation formulas which follow.

Modification	**R e a s o n**		**Memory requirement**		
			Formula (see section "Calculation formulas")	SSL clause specifying size	Standard size
Complete reallocation	Direct hash area	LOCATION MODE changed to CALC	3	POPULATION clause	1 page
		Hash routine/CALC key changed key			
		Hash area converted from indirect to direct (see table 30)			
	Indexed list	New set defined with MODE IS LIST	with SYSTEM set: min. 1 page max. refer to formula 5 with non-singular set: minimum not specifiable. BALTER stores several set occurrence tables together, provided they are small enough and intended to go in the same realm. max. size of a single set occurrence table: refer to formula 5		
		MODE clause changed to LIST			
		ASC/DESC key changed			
Partial reallocation (record type lengthened)	User part lengthened		BALTER enters as many records as possible in the previous pages and relocates the remaining records to other pages. Since most of the records are dispersed throughout the storage space, it is impossible to estimate the memory requirement (other than for record types with a special form of storage).		
	System part lengthened	Record type compressed			
		Owner linked to its table			
		Member linked to its owner			
		MODE IS CHAIN introduced (see table 28 in chapter "Modifying the Schema DDL")			
		Owner/member record type: new set with MODE IS CHAIN added			
		LINKED TO PRIOR or to MODE IS CHAIN (see table 28 in chapter "Modifying the Schema DDL")			

		Member record type: new set added	

(no reallocation) Record type shortened	User part shortened		BALTER pushes together the records in a page (except for record type with special form of storage). Additional free memory space is not required (except for record type with special form of storage).
	System part shortened	Record type decompressed	
		Linkage of owner to its table cancelled	
		Linkage of member to its owner cancelled	
		Sets defined with MODE IS CHAIN deleted	
		LINKED TO PRIOR and ORDER IS LAST omitted from MODE IS CHAIN (see table 28 in chapter "Modifying the Schema DDL")	
		MODE clause changed from MODE IS CHAIN to POINTER ARRAY / LIST (see table 28 in chapter "Modifying the Schema DDL")	
		Set in which record type was member deleted	

Table 33: Memory requirements for record reallocation

Calculation formulas

The calculation formulas indicated below can be used to calculate the following values:

- DBTT size for a new DBTT
- DBTT size for a re-stored DBTT
- size of a direct hash area
- size of an indirect hash area
- size of a multi-level SEARCH key table on record type level

1. Calculating the DBTT size for a new DBTT:

   `2044 / LENGTH = entries-per-page` [1] `(for 2048-byte page length)`

   `3980 / LENGTH = entries-per-page` [1] `(for 4000-byte page length)`

   `8076 / LENGTH = entries-per-page` [1] `(for 8096-byte page length)`

   `integer / entries-per-page = number-of-pages` [2]

   [1] *The result must be rounded down to an integer*

   [2] *The result must be rounded up to an integer*

   number-of-pages

   Number of DBTT pages

   entries-per-page

   Number of DBTT entries per page

   integer

   Number of planned records as in DBTT clause

   LENGTH

   Length of a DBTT entry; is contained in the BGSIA report under the heading: ’DBTT-INFORMATION’ (see the "Recovery, Information and Reorganization" manual)

2. Calculating the DBTT size for a re-stored DBTT:

   `2044 / LENGTH-new = entries-per-page-new` [1] (for 2048-byte page length)

   `3980 / LENGTH-new = entries-per-page-new` [1] (for 4000-byte page length)

   `8076 / LENGTH-new = entries-per-page-new` [1] (for 8096-byte page length)

   `prev.-total-no.-of-entries / entries-per-page-new = number-of-pages` [2]

   [1] *The result must be rounded down to an integer*
   [2] *The result must be rounded up to an integer*

   number-of-pages

   Number of DBTT pages

   prev.-total-no.-of-entries

   Number of entries in the original DBTT; this value can be determined using BSTATUS

   entries-per-page-new

   Number of DBTT entries per page in the new DBTT

   LENGTH-new

   Length of an entry in the new DBTT

   LENGTH-new = 4 x (*n* + 1)

   n
   Number of all the tables of the sets in which the record type is the owner record type

3. Calculating the size of a direct hash area:

   No allowance is made for the number of overflow pages which BALTER must also create.

   `(page-length - 30) / (record-length + calc-key-length + 15) = entries-per-page` [1]

   `(integer - 1) / entries-per-page + 1 = number-of-pages` [2]

   [1] *The result must be rounded up to an integer*
   [2] *The result must be rounded up to the next-higher prime number if no prime number is obtained*

   page-length

   Page length of the database (2048/4000/8096 bytes)

   number-of-pages

   Number of pages in the hash area

   calc-key-length

   Length of CALC key; is given in the BGSIA report under the heading ’CALC-INFORMATION’ in the LENGTH column (see the "Recovery, Information and Reorganization" manual).

   integer

   Number of records to be stored according to the POPULATION clause (record type level).

   record-length

   Length of record type including SCD; is given in the BGSIA report under the heading ’RECORD-INFORMATION’ in the LENGTH column (see the "Recovery, Information and Reorganization" manual).

   entries-per-page

   Number of entries (records or CALC index entries) per page.

4. Calculating the size of an indirect hash area:

   No allowance is made for the number of overflow pages which BALTER must also create.

   (*page-length* - 30) / (*calc-key-length* + 7) = *entries-per-page* [1)]

   (*integer* - 1) / *entries-per-page* = *number-of-pages* [2)]

   [1)] *The result must be rounded up to an integer*

   [2)] *The result must be rounded up to the next-higher prime number if no prime number is obtained*

   integer
   Number of records to be stored

   - according to POPULATION clause (primary key)
   - according to DBTT clause (secondary key).

   For other meanings, see the previous formula.

5. Calculating the size of a multi-level SEARCH key table (record type level)

   This formula is based on slightly simplified assumptions and therefore produces an estimate, not the exact numerical value.

   For a 2048-byte page length:

   - if an occupancy level was specified:

     $$\frac{\textit{no.-of-search-keys}}{\max(1,\ (2002 / a * \textit{occ. level} / 100))\ ^{1)}} * \frac{2002 - a}{2002 - 2a} = \textit{no.-of-pages}$$

     [1)] The result of the first division and the result of the first multiplication must each be rounded down to an integer

   - If no occupancy level was specified:

     $$\frac{\textit{no.-of-search-keys}}{(2002 / a\ ) -1)\ ^{2)}} * \frac{2002 - a}{2002 - 2a} = \textit{no.-of-pages}$$

     [2)] The result of the division must be rounded down to an integer

   where: a = *search-key-l* + 7

For a 4000-byte page length:

- if an occupancy level was specified:

```
        no.-of-search-keys                  3950 - b
------------------------------------ * ---------- = no.-of-pages
max(1, (3950 / b * occ. level / 100)) 1)  3950 - 2b
```

1) The result of the first division and the result of the first multiplication must each be rounded down to an integer

- If no occupancy level was specified:

```
no.-of-search-keys    3950 - b
------------------ * ---------- = no.-of-pages
(3950 / b ) -1) 2)    3950 - 2b
```

2) The result of the division must be rounded down to an integer

where: `b = search-key-l + 10`

For an 8096-byte page length:

- if an occupancy level was specified:

```
        no.-of-search-keys                  8046 - b
------------------------------------ * ---------- = no.-of-pages
max(1, (8046 / b * occ. level / 100)) 1)  8046 - 2b
```

1) The result of the first division and the result of the first multiplication must each be rounded down to an integer

- If no occupancy level was specified:

```
no.-of-search-keys    8046 - b
------------------ * ---------- = no.-of-pages
(8046 / b ) -1) 2)    8046 - 2b
```

2) The result of the division must be rounded down to an integer

where: `b = search-key-l + 10`

no.-of-pages

Number of pages required for the whole set occurrence table

no.-of-search-keys

Number of keys in the table

occ. level

Percentage occupancy level (see the FILLING clause, "BALTER statements")

search-key-l(ength)

Length of the SEARCH key; is given in the BGSIA report under the heading 'KEY-INFORMATION' (NO CALC SEARCH KEY'S)' in the LENGTH column (see the "Recovery, Information and Reorganization" manual).

6.6 Recovery measures and response to errors

Restructuring changes not only the user database but also the compiler database and the HASHLIB.

If errors occur during restructuring and you require the database in the state it was in before restructuring began, you also need the realms of the compiler database and the HASHLIB.

6.6.1 Saving the database

Before restructuring operations begin, i.e. before BCHANGE is started, you must switch off after-image logging (BALTER writes no after-images) and save the entire database or part of the database as required:

You must always save the following:

dbname.HASHLIB
dbname.COSSD
dbname.DBDIR
dbname.DBCOM

With regard to the user realms, you have two options:

- You save all the user realms before beginning the restructuring process, i.e.
  dbname.realmname-1
  .
  .
  dbname.realmname-n
- You use an analysis run with the statements
  REPORT IS YES
  EXECUTION IS NO
  to determine which user realms are needed
  and then you save only these realms in addition before the BALTER execution phase.

  Determining the user realms needed:

  After saving *dbname*.HASHLIB, *dbname*.COSSD, *dbname*.DBDIR and *dbname*.DBCOM, execute the restructuring process including the analysis run of BALTER with REPORT IS YES and EXECUTION IS NO. You can determine which user realms are needed from the analysis log for BALTER under REPORT OF ADDED, DELETED AND NEEDED AREAS.

  Realms may also be needed because anchor records of singular sets have to be recreated. Here BALTER attempts to prevent unnecessary relocations between the realms in order to limit the backup effort required in the user realms.

  Saving the needed user realms:

  Before restructuring using BALTER is performed and the EXECUTION IS YES statement is executed, save the user realms determined in the analysis phase:
  dbname.realmname-i
  .
  .
  dbname.realmname-j

For further information on saving a database, refer to the section "Saving and recovering a database in the event of errors" in the "Database Operation" manual.

6.6.2 Restoring the database

If, during restructuring, a program abnormally terminates processing with "ABNORMAL END", one of the following steps must be taken, depending on the gravity of the error:

- re-execute the terminated program, or
- make use of the backup and repeat restructuring

The sections of this chapter dealing with the individual restructuring programs describe when it is necessary to reset the database and to repeat the restructuring process, and when it is sufficient to repeat the programs terminated abnormally.

The following table shows which programs modify which files or realms in the database during restructuring.

	HASHLIB	**DBDIR**	**DBCOM**	**DBCOM.O**	**COSSD**	**COSSD.O**	**User realms which have to be accessed**
BCHANGE	-	RW	RW	W	R	W	-
DDL compiler	-	RW	RW	-	W	-	-
SSL compiler	-	RW	RW	-	W	-	-
BGSIA	-	RW	RW	-	-	-	-
LMS	W	-	-	-	-	-	-
BALTER (analysis phase)	-	R	R	R	-	-	-
BALTER (restructuring phase)	R	RW	R	R	-	-	RW
DDL compiler (subschemas)	-	RW	RW	-	W	R	-
BGSSIA	-	RW	R	-	-	-	-

Table 34: Access to files and realms of the database during restructuring

R read access

W write access

- no access

The following options are available for restoring the database:

- You can convert the shadow database to an original database by renaming it with the die MODIFY-FILE-ATTRIBUTES command.
- You can read in the ARCHIVE backup and then change the database name, if desired, with the MODIFY-FILE-ATTRIBUTES command. If the ARCHIVE backup was created on-line, you may have to mend it with the BMEND utility routine (see "BMEND" in the "Recovery, Information and Reorganization" manual).

For further information on restoring a database, refer to the section "Saving and recovering a database in the event of errors" in the "Database Operation" manual.

Steps required in case of memory shortage

If one of the programs terminates with database status (DBSTATUS) 14802 or 14804, you must

- expand the affected realm or record type with BREORG,
- restart the program and if necessary delete any incompletely generated information with the DELETE parameter.

6.7 Preparing the compiler database with BCHANGE

The tasks of BCHANGE when restructuring a database are comparable to the tasks of BCREATE when creating a database. BCHANGE prepares the compiler database to accept the new schema. It carries out the following preliminary functions prior to restructuring:

- It saves the old SIA in the DBDIR and prepares the DBDIR to accept a new SIA, so that a new and an old SIA are stored in the DBDIR after the BGSIA run for the new schema. BALTER needs both SIAs when adapting the stored data to the new schema so that it can recognize differences between the old and the new schemas. Make sure therefore that there are enough free pages available in the DBDIR before the BCHANGE run or that automatic realm extension is possible by means of secondary allocation > 0.
- It deletes all user SSIAs in the DBDIR.
- It saves the old DBCOM in the file dbname.DBCOM.O and reformats the DBCOM.

  BALTER needs the schema information of the old and the new DBCOMs in order to examine the planned modifications.
- It saves the old COSSD in the file dbname.COSSD.O.

  After restructuring the DDL compiler needs the old COSSD to accept the subschemas. You should therefore delete the *dbname*.COSSD.O file only after you have compiled or accepted all the other subschemas which are still required.

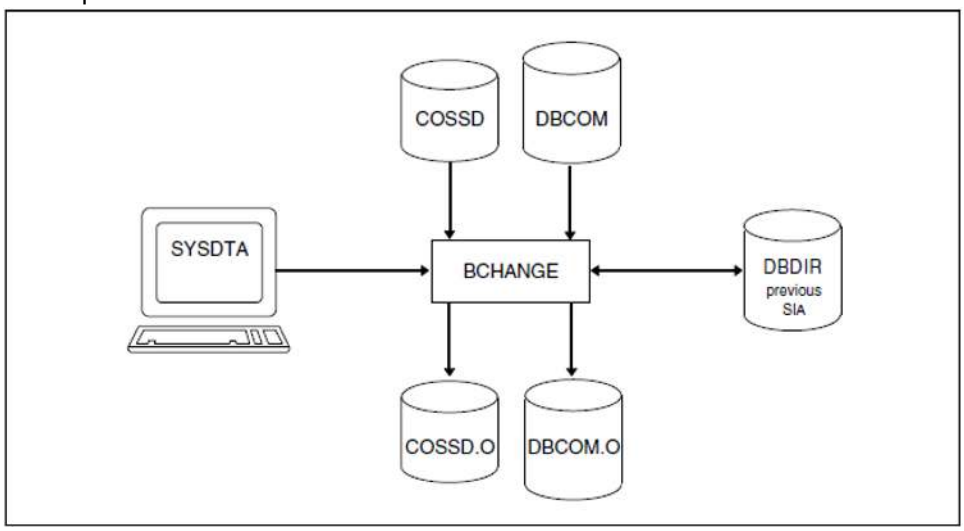


Figure 27: System environment when preparing the compiler database

BCHANGE automatically stores the copies of DBCOM and COSSD on public disks. It is not necessary to issue a CREATE-FILE command to set up the two files (before BCHANGE is started) unless the copies are to be stored on private disks.
Depending on the size of the files it is, however, advisable to set them up using a CREATE-FILE command with SPACE operand - even if they are to be stored on public disks (see section “Maximum size of UDS/SQL files” in chapter “Files and realms of a UDS/SQL database”).

When required, BCHANGE automatically extends the realms of the processed database. For details, please refer to the “Database Operation” manual, Automatic realm extension by means of utility routines.

At startup BCHANGE takes into account any assigned UDS/SQL pubset declaration (see the “Database Operation” manual, Pubset declaration job variable). Faulty assignment leads to the program aborting.

Command sequence for starting BCHANGE

It is assumed for the command sequence described here that UDS/SQL was installed with IMON (see section “START commands for the UDS/SQL programs”).

The BCHANGE utility routine is started by the following commands in the identification under which the database is cataloged:

```
01 [/CREATE-FILE FILE-NAME=dbname.DBCOM.0 ...]
02 [/CREATE-FILE FILE-NAME=dbname.COSSD.0 ...]
03 /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR
04 /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,
SCOPE=*TASK
05 /START-UDS-BCHANGE
```

01,02 See section “Setting up the compiler database”.

04 The specified version of BCHANGE is selected.
It is generally recommended that you specify the version since it is possible for several UDS/SQL versions to be installed in parallel.

05 The UDS/SQL utility routine can also be started with the alias BCHANGE.

¡ There are no BCHANGE statements.

6.8 Compiling the Schema DDL

If the compiler database has been prepared to accept a new schema with the aid of the BCHANGE utility routine, the current Schema DDL must then be compiled by the DDL compiler, even if the Schema DDL has not been modified.

The compilation procedure is the same as that used for database creation.

Once the Schema DDL has been compiled, the following are available:

- an old and a new DBCOM
- an old SIA in the DBDIR
- an old and a new COSSD.

Command sequence for compiling the current Schema DDL

It is assumed for the command sequence described here that UDS/SQL was installed with IMON (see section “START commands for the UDS/SQL programs”).

The commands listed here are described in detail in section “Compiling the Schema DDL”.

```
01 /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR
02 /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,
SCOPE=*TASK
03 /START-UDS-DDL
04 SOURCE IS 'schema-file'
05 END
06 /ASSIGN-SYSDTA TO=*SYSCMD
```

> It is essential that the DDL compiler should terminate compilation with the message ’NORMAL END’.
>
> If the message ’ABNORMAL END’ is received, compilation must be repeated with corrected DDL clauses.

6.9 Compiling the SSL

The option is available to compile a new SSL using the SSL compiler once the Schema DDL has been compiled.

If no SSL compilation is carried out, default values for the storage structure are used. If the previously defined storage structure is to be retained, it is necessary to recompile the original SSL clauses.

The compilation procedure is the same as that used for database creation (see section “Compiling the SSL” of chapter “Database creation (BCREATE, BFORMAT, DDL- and SSLCompiler, BGSIA, BGSSIA, BCALLSI)”).

Command sequence for compiling the SSL

It is assumed for the command sequence described here that UDS/SQL was installed with IMON (see section “START commands for the UDS/SQL programs”. The aliases for calling the utility routines are also listed in this section).
The commands listed here are described in detail in section “Compiling the SSL” of chapter “Database creation (BCREATE, BFORMAT, DDL- and SSLCompiler, BGSIA, BGSSIA, BCALLSI)”.

```
01 /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR
02 /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,SCOPE=*TASK
03 /START-UDS-SSL
04 SOURCE IS 'ssl-file'
05 END
06 /ASSIGN-SYSDTA TO=*SYSCMD
```

> It is essential that the SSL compiler should terminate compilation with ’NORMAL END’. If compilation ends with ’ABNORMAL END’, the following action should be taken:
>
> - for errors in the SSL clauses:
>
> the faulty SSL clauses should be corrected and the SSL compilation should be repeated;
> - for errors in the DDL clauses:
> - the faulty DDL clauses should be corrected
> - the faulty schema should be deleted in a DDL run by means of the statement DELETE SCHEMA *schemaname*
> - and the restructuring process should be repeated from ’Compiling the Schema DDL’ onwards.

6.10 Generating a new SIA and entering it in the DBDIR with BGSIA

Once the Schema DDL and the SSL (optional) have been successfully compiled, the SIA of the new schema must be generated and entered in the DBDIR using the BGSIA utility routine.

The saved SIA of the old schema remains in DBDIR so that, after the BGSIA run, DBDIR contains the SIAs of both the old and the new schemas. BALTER needs both in order to adapt the stored data to the modified schema.

The BGSIA run corresponds to the run carried out for the creation of the database (see section “Setting up the Schema Information Area (SIA) with BGSIA”). After the BGSIA run, the module UDSHASH generated by BGSIA must be entered in the HASHLIB.

If hash routines written by the DB administrator are used, these must also be entered in the HASHLIB with the attributes RMODE=ANY and AMODE=ANY before BALTER is started by means of EXECUTION IS YES.

Generating SIA and entering it in DBDIR

```
01 /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR
02 /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,SCOPE=*TASK
03 /DELETE-SYSTEM-FILE FILE-NAME=*OMF
04 /START-UDS-BGSIA
05 GENERATE SCHEMA schema-name
06 [DISPLAY[ SCHEMA schema-name]]
07 END
```

Entering the module UDSHASH in the HASHLIB

```
01 /START-LMS
02 //OPEN-LIB LIB=dbname.HASHLIB,MODE=*UPDATE
03 //ADD-ELEMENT FROM-FILE=*OMF,TO-ELEMENT=*LIBRARY-ELEMENT(TYPE=R)
04 //END
```

6.11 Analyzing schema modifications and adapting stored data with BALTER

Analyzing the modifications to the database schema and adapting stored data to the modified schema is the task of the BALTER utility routine. BALTER controls these processes in two phases:

- in the analysis phase BALTER analyzes the modifications to the database schema
- In the optional REPORT phase BALTER outputs the analysis report
- in the restructuring phase BALTER adapts the stored data and the definition of the database to the modified schema

In order to run BALTER successfully, you must first use the BGSIA utility routine to create the new SIA and enter it into the DBDIR (see section “Generating a new SIA and entering it in the DBDIR with BGSIA”). Otherwise, the BALTER run will abort with the message “BGSIA HAS NOT BEEN EXECUTED”.

When required, BALTER automatically extends the realms of the database being processed. For details, please refer to the “Database Operation” manual, Automatic realm extension by means of utility routines.

At startup BALTER takes into account any assigned UDS/SQL pubset declaration (see the “Database Operation” manual, Pubset declaration job variable). Faulty assignment leads to the program aborting.

6.11.1 Analysis phase

Using the old and new versions of both the DBCOM and the SIA, BALTER determines, in the analysis phase, the differences between the old and the new schema description and checks that all the modifications are permissible.

At the end of the analysis phase BALTER offers an optional logging function which logs the following to SYSLST (see "REPORT statement", table 42 of "BALTER statements"): the user realms that will be added, deleted and needed during restructuring, as well as the modifications to be made to stored data in the order in which they are subsequently carried out in the restructuring phase.
This analysis listing indicates how much free memory space BALTER requires for individual restructuring operations in the database, and whether any of the planned modifications are illegal.

It is therefore important that you output the analysis listing and to read through it carefully, and that you save any need user realms that have not yet been saved before you initiate restructuring.

> If you need to create a unique key (sort key, CALC key, SEARCH KEY USING INDEX) that consists of only newly-defined items for a record type for which there are already existing records in the database, the following approach is recommended:
>
> 1. Create the new keys with DUPLICATES ARE ALLOWED.
> 2. Assign unique values or value combinations to the new keys (e.g. by using the DML statement MOVE).
> 3. Then change the definition of the key to DUPLICATES ARE NOT ALLOWED.

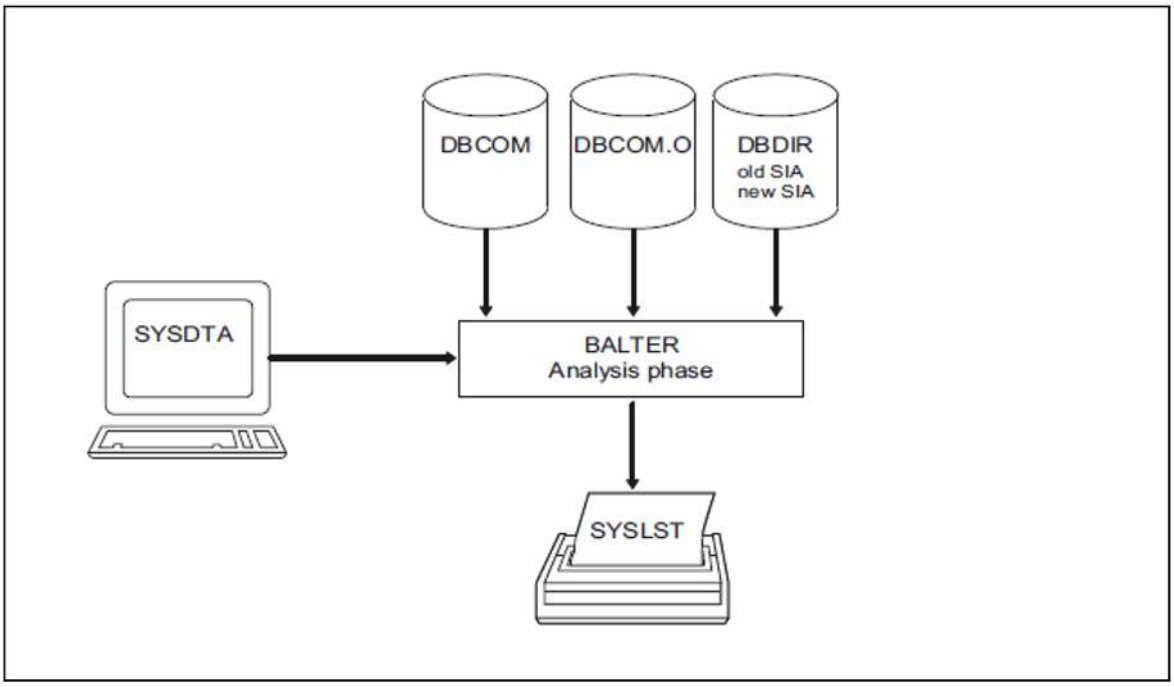


Figure 28: System environment in the analysis phase

BALTER issues error messages and warnings via SYSOUT if illegal modifications are planned (see section “Description of BALTER messages”). If REPORT IS YES is specified, BALTER outputs information on which realms are needed and which realms are not needed to SYSOUT.

6.11.2 Description of the analysis report (REPORT phase)

If REPORT IS YES is specified for a BALTER run, BALTER initiates the REPORT phase after the analysis phase. In the REPORT phase it outputs the analysis report via SYSLST. In the analysis report BALTER lists the changes to be carried out in the order in which it actually implements them during the restructuring phase.

If BALTER requires free memory space in the database for a modification, it also specifies:

- the number of data pages required and
- the realm in which it requires the space.

In addition, BALTER logs whether, and how much, free space will become available as a result of each restructuring operation, and it reports changes which are illegal or only permissible under certain circumstances.

> **Free memory space**
> When required, BALTER automatically extends the realms of the processed database. For details, please refer to the "Database Operation" manual, Automatic realm extension by means of utility routines. If the free memory space is not sufficient for a change, sufficient space is generally made available by this automatic realm extension. Only if the requirement for automatic realm extension is not satisfied can it happen that there is actually not enough free memory space.

REPORT OF ADDED, DELETED AND NEEDED AREAS

Message	Meaning
realm-name ADDED	Realm *realmname* added
realm-name DELETED	Realm *realm-name* deleted
realm-name NEEDED	Realm *realm-name* needed Realm *realm-name* must be saved before the restructuring phase.

Table 35: Report of added, deleted and needed realms

If REPORT IS YES is specified, BALTER also outputs to SYSOUT the names of the realms that are needed and those that are not needed

Message	Meaning
REALM NEEDED: *realm-name*	Realm *realm-name* needed Realm *realm-name* must be saved before the restructuring phase.
REALM NOT NEEDED: *realm-name*	Realm *realm-name* not needed

Table 36: SYSOUT report on realms that are needed and not needed

REPORT OF CHANGES IN DBTT FOR RECORD: record-name

Log of DBTT changes for record type *record-name*

Message	Meaning

`NUMBER OF ENTRIES RESERVED IN` `  OLD DBTT PAGE = integer`	Self-explanatory (page is synonymous with page)

NUMBER OF ENTRIES RESERVED IN NEW DBTT PAGE = *integer*	Self-explanatory
TOTAL NUMBER OF PAGES IN OLD DBTT = *integer*	Number of pages reserved for the previous DBTT (DBTT base and DBTT extents)
TOTAL NUMBER OF PAGES, NEEDED FOR NEW DBTT IN AREA *realm-name* = *integer*. IF THERE IS NOT ENOUGH SPACE AVAILABLE, THE RESTRUCTURING PROCESS WILL ABEND.	Total number of pages that the new DBTT requires in the realm *realm-name*. If there is not enough space available then the restructuring phase is aborted; contiguous empty pages must be available for the new DBTT extents that are to be created (cf. “Free memory space”).
TABLE ANCHORED IN DBTT COLUMN: *integer* WILL BE DELETED.	Self-explanatory
TABLE ANCHORED IN DBTT-COLUMN-NR: *integer* WILL BE MOVED FROM {AREA *realm-name-1* | OWNER AREA} TO {AREA *realm-name-2* | OWNER AREA}.	Self-explanatory
[UP TO]{*integer* | NO} OLD DBTT PAGE(S) WILL BE FREED IN AREA *realm-name*	Number of pages released by the previous DBTT in the realm *realm-name*. The precise number of pages actually released in the execution phase may in some cases be up to 64 pages smaller than indicated in *integer*.
UP TO 256 CONSECUTIVE EMPTY PAM PAGES ARE NEEDED FOR NEW DBTT.	If parts of an existing DBTT with extents are re-used then a new DBTT base, which is greater in size than a DBTT extent, is created.
THE RECORD TYPE IS NOW OWNER IN SOME SETS.	As a result of restructuring, a record type which was previously only a set member is now also a set owner.
THE DBTT HAS TO BE SHORTENED TO 16 711 679 ENTRIES.	Databases with 2-Kbyte page format permit only 16711679 DBTT entries for owner record types (cf section “Declaring the population” in the "Design and Definition" manual).
IF THERE ARE EXISTING ENTRIES WITH HIGHER RSQ THE RESTRUCTURING PROCESS WILL END ABNORMALLY.	If DBTT entries above those possible for the owner record types exist in the previous member record type then BALTER aborts the restructuring phase. The planned restructuring is not possible in 2-Kbyte page format.

Table 37: Report of DBTT changes

If the new DBTT occupies more pages than the previous one then the previously used pages continue to be used, if possible. If new DBTT extents are to be created then contiguous empty pages with the fixed size of these extents must be available. There must be a minimum of 128 contiguous free PAM- pages for each of the DBTT components.

If the new DBTT is exactly the same size as the previous one, or smaller, BALTER uses the pages of the previous DBTT to create the new DBTT. DBTT extents that are no longer required are released.

REPORT OF DATABASE CHANGES FOR SINGULAR SET: set-name

Message	Meaning
`LENGTH OF OLD SYSTEM RECORD = integer`	Self-explanatory (system record = anchor record)
`LENGTH OF NEW SYSTEM RECORD = integer`	Self-explanatory

`TABLE OCCURRENCE ANCHORED IN SYSTEM RECORD`
`COLUMN integer`
`WILL BE DELETED IF PRESENT.`
	Self-explanatory

`TABLE ANCHORED IN DBTT-COLUMN-NR integer`
`WILL BE MOVED FROM`
`AREA realm-name-1 TO AREA realm-name-2.`
	Self-explanatory

`FORWARD CHAIN POINTER WILL BE REMOVED.`	Self-explanatory
`BACKWARD CHAIN POINTER WILL BE REMOVED.`	Self-explanatory
`THE SYSTEM RECORD WILL BE CREATED` `IN AREA realm-name`	Self-explanatory
`THE SYSTEM RECORD WILL BE DELETED` `IN AREA realm-name`	Self-explanatory
`THE SYSTEM RECORD WILL BE MOVED FROM` `AREA realm-name-1 TO AREA realm-name-2`	Self-explanatory

`integer PAGES FOR CALC SEARCH KEY TABLES`
`WILL BE FORMATED IN AREA realm-name.`
`THEY ARE CONSECUTIVE.`
`IS THERE ENOUGH SPACE AVAILABLE?`
	Self-explanatory
	(page is synonymous with page)
	cf. “Free memory space”.

`integer PAGES FOR CALC SEARCH KEY TABLES` `WILL BE DELETED IN AREA realm-name`	Self-explanatory

Table 38: Report of database changes for singular sets

REPORT OF DATABASE CHANGES FOR DELETION OF RECORD: record-name

Message	Meaning
NUMBER OF ENTRIES RESERVED IN OLD DBTT PAGE = *integer*	Self-explanatory (page is synonymous with page)
integer OLD DBTT PAGES WILL BE FREED IN AREA *realm-name*	Self-explanatory
TABLE ANCHORED IN DBTT COLUMN *integer* WILL BE DELETED IF PRESENT.	Self-explanatory
integer PAGES WITH CALC KEY RECORDS AND TABLES WILL BE DELETED IN AREA *realm-name*	Self-explanatory
integer PAGES WITH CALC KEY TABLES WILL BE DELETED IN AREA *realm-name*	Self-explanatory
ALL RECORD INFORMATION WILL BE DELETED.	Self-explanatory

Table 39: Report of database changes for deletion of record types

REPORT OF DATABASE CHANGES FOR CREATION OF RECORD: record-name

Message	Meaning
NUMBER OF ENTRIES RESERVED IN NEW DBTT PAGE = *integer*	Self-explanatory (page is synonymous with page)
TOTAL NUMBER OF PAGES, NEEDED FOR NEW DBTT IN AREA *realm-name* = *integer*	Number of empty pages that the new DBTT requires in the realm *realmname*. Contiguous empty pages must be available for the DBTT base and the DBTT extents.
integer PAGES FOR CALC KEY RECORDS AND TABLES WILL BE FORMATED IN AREA *realm-name*. THEY ARE CONSECUTIVE.	Self-explanatory (direct hash area)
integer PAGES FOR CALC KEY TABLES WILL BE FORMATED IN AREA *realm-name*. THEY ARE CONSECUTIVE.	Self-explanatory (indirect hash area)
IF THERE IS NOT ENOUGH SPACE AVAILABLE, THE RESTRUCTURING PROCESS WILL END ABNORMALLY.	Self-explanatory cf. “Free memory space”.

Table 40: Report of database changes for creation of record types

REPORT OF DATABASE CHANGES FOR RECORD: record-name

Message	Meaning
integer PAGES WITH CALC KEY RECORDS AND TABLES WILL BE DELETED IN AREA *realm-name*	Self-explanatory
integer PAGES WITH CALC KEY TABLES WILL BE DELETED IN AREA *realm-name*	Self-explanatory
integer PAGES FOR CALC KEY RECORDS AND TABLES WILL BE FORMATED IN AREA *realm-name* . THEY ARE CONSECUTIVE. IS THERE ENOUGH SPACE AVAILABLE?	Self-explanatory cf. “Free memory space”.
integer PAGES FOR CALC KEY TABLES WILL BE FORMATED IN AREA *realm-name*. THEY ARE CONSECUTIVE. IS THERE ENOUGH SPACE AVAILABLE?	Self-explanatory cf. “Free memory space”.
AREA DELETED FROM RECORD-WITHIN-CLAUSE	Self-explanatory
FOLLOWING ACTIONS EXECUTED IF RECORD OCCURRENCES ARE PRESENT:	Self-explanatory
THE RESTRUCTURING PROCESS WILL END ABNORMALLY FOR NOT ALLOWED SCHEMA CHANGES.	Self-explanatory
THE RESTRUCTURING PROCESS WILL END ABNORMALLY IF RECORD OCCURRENCES ARE PRESENT IN AREAS WHICH ARE DELETED FROM RECORD-WITHIN-CLAUSE	Self-explanatory
A NON SINGULAR AUTOMATIC SET THAT WAS NOT PRESENT IN THE OLD SCHEMA HAS BEEN SPECIFIED IN THE NEW SCHEMA. THE RESTRUCTURING PROCESS WILL STOP BECAUSE THE SET OCCURRENCES TO WHICH EACH RECORD OCCURRENCE BELONGS ARE NOT KNOWN.	Self-explanatory
AS A CONSEQUENCE OF LOGICAL CHANGE THE RECORDTYPE WILL BE PLACED IN NEW PAGES. DURING THE PROCESS THE RECORDTYPE WILL RESIDE TWICE IN THE AREA(S). IS THERE ENOUGH SPACE AVAILABLE?	Self-explanatory cf. “Free memory space”.

SET *setname* DOES NOT HAVE MODE = LIST ANY- MORE. THE LIST TABLE HEADER WILL BE REMOVED FROM THE LIST-PAGES.	Self-explanatory

DUE TO A CHANGE IN LOCATION MODE THE CALC KEY INFORMATION WILL BE REMOVED.	Self-explanatory
CALC RECORDS AND KEYS WILL BE PLACED IN THE CALC KEY PAGES.	Self-explanatory
CALC KEYS WILL BE PLACED IN THE CALC KEY PAGES.	Self-explanatory
'DUPLICATES ARE NOT ALLOWED' HAS BEEN SPECIFIED FOR THE CALC KEY. IF DUPLICATES ARE DETECTED THE DUPLICATE VALUES WILL BE PRINTED AND THE RESTRUCTURING PROCESS WILL CONTINUE.	Self-explanatory. Note: If the table contains duplicates after the BALTER run, the table cannot be processed by means of these duplicates. This situation can be corrected as follows: First define DUPLICATES ARE ALLOWED in the Schema DDL; then fill the key items in accordance with DUPLICATES NOT, and finally change the DDL definition to DUPLICATES NOT.
'DUPLICATES ARE NOT ALLOWED' HAS BEEN SPECIFIED FOR THE CALC KEY. THIS KEY IS A NEW ONE ON {ONE NEW FIELD | ONLY NEW FIELDS}. THEREFORE THE TABLE WILL HAVE ONLY DUPLICATES. THIS IS INCONSISTENT WITH 'DUPLICATES ARE NOT ALLOWED'. SPECIFY 'DUPLICATES ARE ALLOWED' IF DUPLICATES ARE DETECTED THE DUPLICATE VALUES WILL BE PRINTED AND THE RESTRUCTURING PROCESS WILL CONTINUE.	DUPLICATES ARE NOT ALLOWED was defined for the CALC key. Each item of this key is new. If existing records of the record type for which the key is defined are used to create the table, the table will consist of all new items and can only contain duplicates. BALTER logs the found duplicates and continues restructuring. However, the table cannot be processed. These conflicting entries can be corrected as follows: First define DUPLICATES ARE ALLOWED in the Schema DDL; then fill the key items in accordance with DUPLICATES NOT, and finally change the DDL definition to DUPLICATES NOT..
FOR SET *setname* A SORTED CHAIN WILL BE BUILT.	Self-explanatory
FOR SET *setname* A LIST TABLE WILL BE BUILT.	Self-explanatory
FOR SET *setname* A POINTER ARRAY WILL BE BUILT.	Self-explanatory

'DUPLICATES ARE NOT ALLOWED' HAS BEEN SPECIFIED FOR THE SORT KEY. IF DUPLICATES ARE DETECTED THE DUPLICATE VALUES WILL BE PRINTED AND THE RESTRUCTURING PROCESS WILL CONTINUE.	Self-explanatory. Note: If the table contains duplicates after the BALTER run, the table cannot be processed by means of these duplicates. This situation can be corrected as follows: First define DUPLICATES ARE ALLOWED in the Schema DDL; then fill the key items in accordance with DUPLICATES NOT, and finally change the DDL definition to DUPLICATES NOT.
'DUPLICATES ARE NOT ALLOWED' HAS BEEN SPECIFIED FOR THE SORT KEY. THIS KEY IS A NEW ONE ON {ONE NEW FIELD | ONLY NEW FIELDS}. THEREFORE THE TABLE WILL HAVE ONLY DUPLICATES. THIS IS INCONSISTENT WITH 'DUPLICATES ARE NOT ALLOWED'. SPECIFY 'DUPLICATES ARE ALLOWED' IF DUPLICATES ARE DETECTED THE DUPLICATE VALUES WILL BE PRINTED AND THE RESTRUCTURING PROCESS WILL CONTINUE.	DUPLICATES ARE NOT ALLOWED was defined for the ASC/DESC key. Each item of this key is new. If existing records of the record type for which the key is defined are used to create the table, the table will consist of all new items and can only contain duplicates. BALTER logs the found duplicates and continues restructuring. However, the table cannot be processed. These conflicting entries can be corrected as follows: First define DUPLICATES ARE ALLOWED in the Schema DDL; then fill the key items in accordance with DUPLICATES NOT, and finally change the DDL definition to DUPLICATES NOT.
FOR SET *setname* CALC SEARCH KEYS WILL BE PLACED IN THE CALC KEY PAGES.	Self-explanatory
FOR SET *setname* AN INDEXED SEARCH KEY TABLE OF TYPE REPEATED KEY WILL BE BUILT	Self-explanatory
FOR SET *setname* AN INDEXED SEARCH KEY TABLE OF TYPE DATABASE KEY LIST WILL BE BUILT.	Self-explanatory
'DUPLICATES ARE NOT ALLOWED' HAS BEEN SPECIFIED FOR THE SEARCH KEY. IF DUPLICATES ARE DETECTED THE DUPLICATE VALUES WILL BE PRINTED AND THE RESTRUCTURING PROCESS WILL CONTINUE.	Self-explanatory. Note: If the table contains duplicates after the BALTER run, the table cannot be processed by means of these duplicates. This situation can be corrected as follows: First define DUPLICATES ARE ALLOWED in the Schema DDL; then fill the key items in accordance with DUPLICATES NOT, and finally change the DDL definition to DUPLICATES NOT.

'DUPLICATES ARE NOT ALLOWED' HAS BEEN SPECIFIED FOR THE SEARCH KEY. THIS KEY IS A NEW ONE ON {ONE NEW FIELD | ONLY NEW FIELDS}. THEREFORE THE TABLE WILL HAVE ONLY DUPLICATES. THIS IS INCONSISTENT WITH 'DUPLICATES ARE NOT ALLOWED'. SPECIFY 'DUPLICATES ARE ALLOWED' IF DUPLICATES ARE DETECTED THE DUPLICATE VALUES WILL BE PRINTED AND THE RESTRUCTURING PROCESS WILL CONTINUE.	DUPLICATES ARE NOT ALLOWED was defined for the SEARCH key. Each item of this key is new. If existing records of the record type for which the key is defined are used to create the table, the table will consist of all new items and can only contain duplicates. BALTER logs the found duplicates and continues restructuring. However, the table cannot be processed. These conflicting entries can be corrected as follows: First define DUPLICATES ARE ALLOWED in the Schema DDL; then fill the key items in accordance with DUPLICATES NOT, and finally change the DDL definition to DUPLICATES NOT.
ALL RECORD OCCURRENCES WILL BE READ AND WRITTEN.	Self-explanatory
ALL RECORD OCCURRENCES WILL BE READ, MODIFIED AND WRITTEN. THESE MODIFICATIONS ARE A CONSEQUENCE OF CHANGES IN: - THE SYSTEM-PART OF THE RECORD - THE USER-PART OF THE RECORD	Self-explanatory
THE SYSTEM WILL TRY TO USE SAME PAGE FOR THE NEW ALLOCATION OF THE RECORD OCCURRENCES.	Self-explanatory
LIST WILL BE REALLOCATED	A list will be reconstructed. Member records of a distributable SYSTEM-LIST set will be distributed approximately evenly over the realms involved.
RECORDS OF SYSTEM LIST SET CAN NOW BE STORED IN n REALMS	Following reconstruction the member records can be stored in *n* realms.

Table 41: Report of changes for record types

If the user part of a record type has been modified, BALTER outputs a table contrasting the layout of the old record with that of the new:

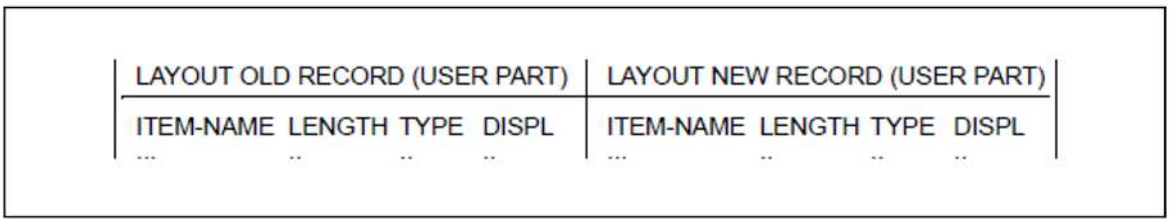


Figure 29: Comparison of the old and the new record (user-part)

LAYOUT OLD RECORD (USER PART)

Self-explanatory

LAYOUT NEW RECORD (USER PART)

Self-explanatory

ITEM-NAME

Item name

TYPE Type of item:

1 alphanumeric
2 unsigned decimal, unpacked
3 signed decimal, unpacked
5 packed decimal
6 half-word
7 Wort
8 database key item

DISPL Displacement of item from beginning of record type (incl. SCD).

What to do if there is a shortage of memory space in the realms

If the analysis report shows that the space required for restructuring in one or more database realms is greater than the space available, the following action must be taken:

- either fulfil the requirements to permit the realms concerned to be extended automatically (for details see the "Database Operation" manual)
- or create additional memory space in the realms concerned manually:
  - the realms of the database which have already been modified should be reset to their original state before the beginning of restructuring (see section “Restoring the database”),
  - the concerned realms should be extended using BREORG, and
  - the restructuring process should be repeated from the ’preparing the compiler database’ stage onwards.

6.11.3 Restructuring phase

The restructuring phase is initiated by BALTER when the DB administrator issues a control statement to that effect (see "EXECUTION statement", table 42, in chapter "BALTER statements") and if the analysis phase has detected no errors.

6.11.3.1 Effects of the restructuring on the content of the database

The various modifications which can be made to the schema and to the storage structure have varying effects on the content of the database:

- BALTER does not modify the content of the database during the restructuring phase when
  - *identifier* is renamed
  - LOCATION DIRECT or DIRECT-LONG is added or omitted
  - the specifications in the ORDER clause vary between: LAST, FIRST, NEXT, PRIOR or IMMATERIAL
  - set membership is redefined
  - duplicates are allowed or prohibited
  - the SET OCCURRENCE SELECTION clause is changed.

  Such modifications are noted by BALTER in the database definitions only; they have no effect on stored data and need only be taken into account when programming the DB applications.
- Again, BALTER does not modify the content of the database when any of the following are redefined:
  - the size of a DBTT
  - the number of pages for the dynamic reorganization of tables
  - the size of set tables.

  Such modifications must be carried out by the BREORG utility routine.
- BALTER modifies the content of the database to a certain degree when any of the following are redefined:
  - PLACEMENT OPTIMIZATION
  - the location of tables
  - the size of a hash area.

  Modifications of this kind do affect the content of the database, but DB consistency does not depend on whether or not BALTER adapts the data stored to the new definition. Since, in addition, BALTER generally has to re-store a large number of records and table lines when carrying out such modifications, it updates the overall database definition, but only adapts the data stored to the new definition if it has to relocate the records or tables in the course of other modifications.

- BALTER modifies the content of the database when
  - a realm is added to the database
  - a new record type or a new set is defined
  - LOCATION CALC is defined for the first time
  - the hash routine or the CALC keys are modified when LOCATION CALC is used
  - a DBTT is relocated to another realm
  - the definition of a SEARCH key is changed
  - the user or system information of a record type is changed
  - the structure or sort criteria of tables are redefined.

  When effecting such modifications, BALTER adapts the stored data to the modified schema or storage structure by carrying out the following steps in the order specified:
  - formatting the realms added
  - in the case of owner record types, modifying the DBTT and deleting the set tables or relocating them in the database
  - entering, deleting or modifying system records for SYSTEM sets
  - deleting or relocating tables
  - deleting all information concerning record types which no longer occur in the new schema
  - creating hash areas and DBTTs for new record types
  - modifying and re-storing record types (user and system information) and creating new tables
  - removing realms which no longer occur in the new schema from the database.

  > **i** If the reuse of database keys was disabled using the BMODTT utility routine, this run must be repeated after restructuring has been performed as BGSIA restores the default setting (database keys of deleted records are reusable or the search for free space begins in a contiguous free area at the end of the realm). Delete identifiers are retained by BALTER when DBTTs are recreated.

6.11.3.2 Logging the restructuring phase

All steps of the restructuring phase can be seen from the analysis report. During restructuring, BALTER only logs the following to SYSOUT:

- realms or record types which have been added or deleted
- the processing of record types, i.e. the creation of new tables, and the modification of record types.

6.11.3.3 System environment in the restructuring phase

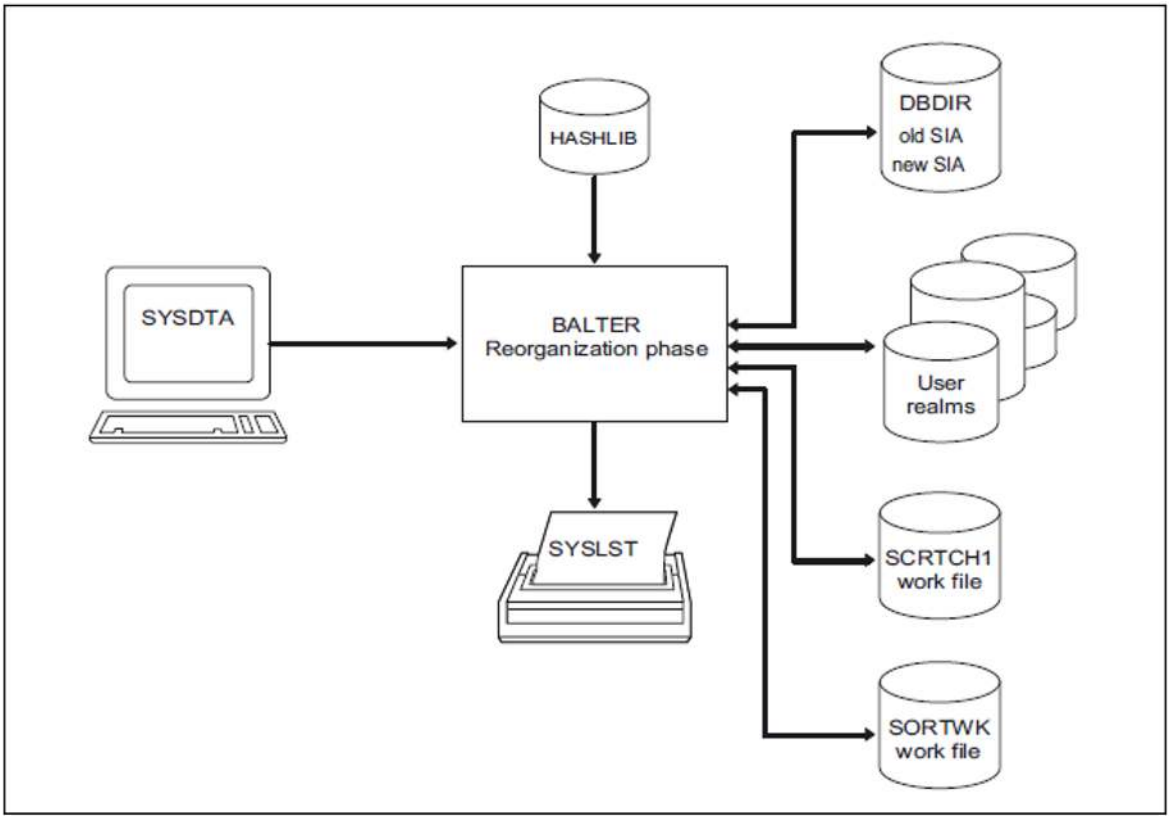


Figure 30: System environment in the restructuring phase

Realms

- User realms
  You can query which user realms BALTER needs during the restructuring phase in an analysis run for which REPORT IS YES EXECUTION IS NO has been specified.
  These user realms are not yet accessed. BALTER does not access the needed user realms until the restructuring phase itself.
- Added realms
  Unless temporary, these realms must be created with the CREATE-FILE command before the restructuring phase is started, under the file name:
  dbname.realm-name

Work files

During the restructuring phase, BALTER requires two work files on disk. It automatically creates these on public disks under the appropriate user identification and deletes them once the run has normally terminated.

The default names for these files are the link names SCRTCH1 and SORTWK:

SCRTCH1 is required by BALTER for buffering information concerning the re-storage and modification of records and the creation of tables

SORTWK requires the SORT used by BALTER for sorting internal evaluation records (see the manual "SORT (BS2000)").

If the two work files are to be created explicitly, they must have the following attributes:

work-file-1

The primary allocation for work file 1 should be based on the data population that is to be buffered. There should always be an appropriate secondary allocation in case it should be necessary to extend the storage space.

SCRTCH1 file link name

PAM access method

The data population for buffering can be calculated approximately using the following formula:

`max(`*`key-length`* `x` *`no.-of-records`*`) x 3 Bytes`

key-length

total length of all keys required. The value 8 should be chosen as the minimum.

max

maximum value obtained when processing various record types.

work-file-2

SORT needs work file 2 if there is not enough virtual memory for pre-sorting. The primary allocation should be based on the data population that is to be sorted while taking account of the safety factor recommended by SORT (see the discussion of work files in the manual "SORT (BS2000)"). There should always be an appropriate secondary allocation in case it is necessary to extend the storage space.

SORTWK file link name

PAM access method

The sort data population can be calculated approximately using the following formula:

`max(`*`rec-length`* `x` *`no.-of-records`*`) Bytes`

rec-length	length of a record incl. SCD
max	maximum value obtained when processing various record types

If you do not set up the two work files yourself, BALTER creates them with the following names and sizes:

UTI.*tsn*.SCRTCH1 (360,360)

UTI.*tsn*.SORTWK (120,120)

tsn is the task sequence number under which BALTER is started.

6.11.4 BALTER statements

BALTER recognizes the following statements:

Statement	Default value	Meaning
[SORTCORE IS *nnn*.]	150	Specifies size of sort area
EXECUTION IS {YES | NO}.	-	Starts/does not start restructuring phase
REPORT IS {YES | NO}.	-	Requests/suppresses logging
[FILL *itemname* OF *recordname* WITH *value*.]	-	Initializes the field *itemname* of *recordname* in all existing records with user-defined value, instead of zeros or blanks.
[FILLING IS *nnn* PERCENT [IN SET NAME IS {*setname*,... | *ALL[EXCEPT *setname* ,..]}].]	-	Specifies table occupancy level (Format 1)
[FILLING WITH POPULATION [IN SET NAME IS {*setname*,... | *ALL[EXCEPT *setname* ,..]}].]	-	Specifies table occupancy level (Format 2)
END.	-	Terminates entry of statement

Table 42: Statements for BALTER

The statements are described in detail in the following pages.

SORTCORE (Specifying the size of the sort area)

To sort elements (records/table rows), BALTER uses the BS2000 utility routine SORT. The SORTCORE statement allows you to specify the size of the main memory space required for the sort area of the SORT routine (see "ALLOC statement" in the "SORT (BS2000)" manual).

```
[SORTCORE IS nnn.]
```

nnn You specify the size of the sort buffer memory space to be made available to the BS2000 SORT utility routine in 4-Kbyte units (see "ALLOC statement" in the "SORT (BS2000)" manual).
Default value:150

The sort data population is the same as that on which the size of work file 2 is based (see "System environment in the restructuring phase").

EXECUTION (Starting/not starting the restructuring phase)

The EXECUTION statement specifies whether BALTER is to carry out the analysis phase and analyze the changes made to the schema and the storage structure, or, in addition, to implement the restructuring phase and adapt stored data to the changes.

The EXECUTION statement must be specified.

```
EXECUTION IS {YES | NO}.
```

NO	analysis phase only
YES	analysis and restructuring phases

REPORT (Requesting/suppressing logging)

The REPORT statement specifies whether or not BALTER is to print out an analysis report (see section “Description of the analysis report (REPORT phase)”).

The REPORT statement must be specified.

```
REPORT IS {YES | NO}.
```

YES	BALTER prints out an analysis report
NO	BALTER does not perform logging

FILL

The FILL statement can be used to initialize new items with a user-defined value, instead of zeros or blanks.

[FILL itemname OF recordname WITH value.]

Where *itemname* and *recordname* are identifiers of record and field in it. Itemname must denote a new field.

Value is:

- ***[+,-]n[,n]*** — for numeric, where n — sequence of decimal digits (0-9), for separation integer and fractional parts comma(‘,’) is used;
- ***‘Z’*** — for alphanumeric, where Z — sequence of character symbols;
- ***X’n’*** — for all types, value is entered in hexadecimal representation, where n — even amount of hexadecimal digits (0-F).

Notes for binary types and unpacked:

value must only contain ‘+’, ‘-’, comma and digits from 0 to 9. ‘+’ and ‘-’ must precede the first digit. The digits aligned to the right. If there is no sign in value, it has a positive value. Value can contain no more than one comma.

Notes for database keys:

value must be entered only in binary format using hexadecimal representation.

Example 1**:**

Database has following structure (after restructuring):

```
...
RECORD NAME IS      EMPLOYEE.
01 ID1              TYPE IS DATABASE-KEY-LONG.
01 FIRSTNAME        TYPE IS CHARACTER 6.
01 SALARY           TYPE IS FIXED REAL DECIMAL
                    7,3.
01 DEBT             TYPE IS BINARY 31.

RECORD NAME IS      IMPORTANT-DATA.
01 ID2              TYPE IS DATABASE-KEY-LONG.
01 DATA             PICTURE IS X(200).
...
```

Command to start BALTER:

```
01  /START-UDS-BALTER
02  REPORT IS YES.
03  EXECUTION IS YES.
04  FILL FIRSTNAME OF EMPLOYEE WITH ‘JOHN’.
05  FILL SALARY OF EMPLOYEE WITH 1535,89.
06  FILL DEBT OF EMPLOYEE WITH -120000.
07  FILL DATA OF IMPORTANT-DATA WITH X’A1B2C3’.
08  END
```

Value is converted to the data type of the field *itemname*. If this is not possible, BALTER execution will be stopped.

Maximal length of entered value is 255 characters.

In case if value is long and total length of the FILL statement is more than 72 characters, you can enter a value as shown in the example 2 below.

BALTER reads 72 characters per line until it finds a closing quote.

Example 2:

```
FILL DATA OF IMPORTANT-DATA WITH 'DataDataDataDataDataDataDataDataDataDa
taDataDataDataDataDataDataDataDataDataDataDataDataDataDataDataDataDataDa
taData'
```

Example 3:

If DB has the same structure and we enter FILL statements like in the example 1, then all records will be filled with:

FIRSTNAME	‘JOHN ’
SALARY	+1535,890
DEBT	-120 000
DATA	X’A1B2C3’

If invalid characters are entered in value or value can’t be converted to correspondent type, then BALTER execution will be stopped and error message occurs.

It is recommended to run BALTER with ”EXECUTION IS NO” statement at first and analyze error messages.

FILLING (Specifying the occupancy level of table pages)

The FILLING statement enables you to define the occupancy level of new tables.

Format 1 Specifies an occupancy level of the new tables in percent.

Format 2 Specifies a minimum size for new tables.
However, the minimum size actually used is limited by the number of table entries which fit onto a database page. Format 2 consequently only works for small tables, i.e. ones which fit onto a page.

Format 1

```
[FILLING IS nnn PERCENT
   [ IN SET NAME IS {setname,... | *ALL[ EXCEPT setname,..]}].]
```

Format 2

```
[FILLING WITH POPULATION
   [ IN SET NAME IS {setname,... | *ALL[ EXCEPT setname,..]}].]
```

nnn

Specifies to which percentage the new table pages are to be filled

nnn = 1 ... 100

POPULATION

Specifies that the POPULATION clause in the SSL is used to determine the size of new tables.

IN SET NAME IS ...

Specifies the sets in which the occupancy level specified for new tables applies.If IN SET NAME IS is omitted, FILLING applies to all new table pages.

setname,...

The specified occupancy level applies to new table pages in the specified sets

*ALL

The specified occupancy level applies to all new table pages

*ALL EXCEPT *setname*,...

The specified occupancy level applies to all new table pages other than those in the sets listed after EXCEPT

- Format 1 of the statement is effective for single-level tables and for level 0 of all multilevel tables which BALTER creates except lists.
  On level 1, tables are 95 % filled, and on every higher level one table entry is left free.

  If *nnn* is made too small, BALTER makes sure that there is room for at least one entry.
- Format 2 of the statement is effective for single-level tables which BALTER creates, including lists.
- If you do not specify FILLING, an entry on level 0 also remains free.
- You can specify both formats simultaneously for the same set name. If required, more free table entries can occur than are specified with Format 1.
- You can repeat the statements and consequently complement and correct preceding statements with the same format.
  The last entry therefore applies for each set name.

6.11.5 Command sequence to start BALTER

It is assumed for the command sequences described here that UDS/SQL was installed with IMON (see section “START commands for the UDS/SQL programs”).

The following commands are used under the identification in which the database is cataloged to initiate the BALTER analysis and restructuring phases (you can also start the program using the alias BALTER):

Analysis phase

```
01 /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR

02 /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,
SCOPE=*TASK

03 /START-UDS-BALTER

04 EXECUTION IS NO.

05 REPORT IS YES.

06 END.
```

Restructuring phase

```
01   /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR

02   [/CREATE-FILE FILE-NAME=work-file-1 ...
      /ADD-FILE-LINK LINK-NAME=SCRTCH1,FILE-NAME=work-file-1
          ,ACCESS-METHOD=*UPAM]

03   [/CREATE-FILE FILE-NAME=work-file-2 ...
      /ADD-FILE-LINK LINK-NAME=SORTWK,FILE-NAME=work-file-2
          ,ACCESS-METHOD=*UPAM]

04   /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,
SCOPE=*TASK

05   /START-UDS-BALTER

06   [SORTCORE IS nnn.]

07   [FILL itemname OF recordname WITH value.]

08   [FILLING IS nnn PERCENT
        [ IN SET NAME IS {setname,... | *ALL [ EXCEPT setname,..]}].]

09   [FILLING WITH POPULATION
        [ IN SET NAME IS {setname,... | *ALL [ EXCEPT setname,..]}].]

10   EXECUTION IS YES.

11   REPORT IS {YES | NO}.

12   END
```

6.11.6 Description of BALTER messages

The messages which BALTER issues to SYSOUT enable its activities to be monitored:

Message	Meaning
`*** ANALYSE-PHASE ***`	Start message for analysis phase
`*** REPORT-PHASE ***`	Start message for REPORT phase with the analysis report
`*** EXECUTION-PHASE ***`	Start message for restructuring phase
`NO ERRORS DETECTED IN SCHEMA CHANGES`	Self-explanatory
`ERRORS DETECTED IN SCHEMA CHANGES`	Self-explanatory
`DATABASE ALTERED`	Stored data adapted to schema changes
`DATABASE NOT ALTERED`	Stored data not adapted to schema changes
`NUMBER OF DATABASE ACCESSES` *integer*	Self-explanatory
`NUMBER OF FILE ACCESSES` *integer*	Self-explanatory
`NUMBER OF SORT ACCESSES` *integer*	Self-explanatory

Table 43: General BALTER messages

Action indicators

Message	Meaning

`PRINTOUT OF THE USED TAB2-INDICES`
`TAB2-INDEX` *number* `FOR RECORD` *rec-name*
`PRINTOUT OF TAB3- & TAB4-INDICES FOR MATCHING AND SINGULAR SETS`
`TAB3-INDEX` *number* `FOR SET` *set-name*
`TAB4-INDEX` *number* `FOR SET` *set-name* `KEY-REF` *keyref*
	The TAB-INDICES are used for diagnostic purposes only.
	They give information about BALTER activities and changes to record types, sets and keys.

Table 44: TAB-INDICES

Restructuring messages

During the restructuring phase BALTER logs to SYSOUT all changes it makes to record types, sets or keys:

Message	Meaning
REALM ADDED TO DATABASE: *realm-name*	Self-explanatory
RECORD DELETED FROM DATABASE: *record-name*	Self-explanatory
RECORD ADDED TO DATABASE: *record-name*	Self-explanatory
RECORD MODIFICATION STARTED FOR: REC NAME: *record-name* REC REF : *record-reference*	Self-explanatory
SET REF : *set-no* SET NAME: *set-name*	Modification/creation of set *set-name* begun
CALCKEY TABLE	Modification/creation of CALC-key table begun
SORTKEY TABLE, DBTT COLUMN NR: *integer*	Modification/creation of SORT key table begun; DBTT column no.: *integer*
SEARCHKEY TABLE, DBTT COLUMN NR: *integer*	Modification/creation of SEARCH key table begun; DBTT column no.: *integer*
TABLE FILLING IS *integer* PERCENT	Occupancy level for specified table is *integer* per cent; only shown if FILLING (Format 1) has been specified
MINIMUM TABLE SIZE FROM POPULATION: *integer* ENTRIES	The table size was determined on the basis of the POPULATION clause and the table contains *integer* entries; appears only if FILLING (Format 2) was specified
CALC SEARCHKEY TABLE	Creation of indirect hash area for CALC SEARCH key begun
ALLOCATION OF LIST RECORDS STARTED	Self-explanatory
STORING DATABASE RECORDS	Self-explanatory
DELETION OF REALM: *realm-name*	Self-explanatory

Table 45: Restructuring messages

6.12 Adapting access rights

The restructuring process has no effect on the access rights which have been entered in the old database with the aid of the BPRIVACY utility routine.

These access rights must be adapted to the new schema, i.e. they must be completely reentered.

If no user group names were assigned for access rights before restructuring took place, you can dispense with this processing step.

6.13 Adapting subschemas

When the compiler database is being prepared for restructuring, one of BCHANGE’s tasks is to delete all the SSIAs in the DBDIR and all subschema information in the DBCOM. The DDL compiler then readies the COSSD to accept new subschema information when compiling the new Schema DDL. Consequently all the old subschema information is deleted after the restructuring phase. No subschema information has as yet been entered in the new COSSD.

Therefore, once BALTER has restructured the database, all subschemas must be recompiled and a new SSIA must be generated for each and entered in the DBDIR.

6.13.1 Copying compatible subschemas

Often not all the subschemas will be affected by schema changes. BCHANGE therefore copies the COSSD into the file COSSD.O at the beginning of restructuring so that all the old subschema information is retained despite the restructuring activity. If copying of the old subschemas is required, it is necessary to carry out a DDL compiler run to copy the old subschemas after BALTER has successfully terminated the restructuring phase.

During this run for copying the subschemas, the DDL compiler reads all the old subschemas from the file COSSD.O, recompiles them and then checks them for compatibility with the new schema. It differentiates between three possible results:

- the old subschema description is incompatible with the new schema
- the old subschema is incompatible with the new schema because of logical and/or physical changes in the schema, i.e. the execution of DML statements is affected
- the old subschema is unaffected by changes in the new schema.

In the first two cases the DDL compiler does not store subschema information in either the DBCOM or the COSSD. Only in the third case, when a subschema is not affected by schema changes, does the computer copy the subschema from the COSSD.O, recompile it and enter the subschema information in the new DBCOM and in the new COSSD. For every subschema copied a new SSIA must be generated using the BGSSIA utility routine and entered in the DBDIR.

Please note that "compatibility" only means that the old subschema’s view of the new schema has remained the same as that of its view of the old schema. It does not mean, for example, that when the "COPY [ALL] RECORD[S]" clause is used the view of the (upwardcompatible) changes in the new schema is retained in the new schema. If you want to do this, you must recompile the subschema.

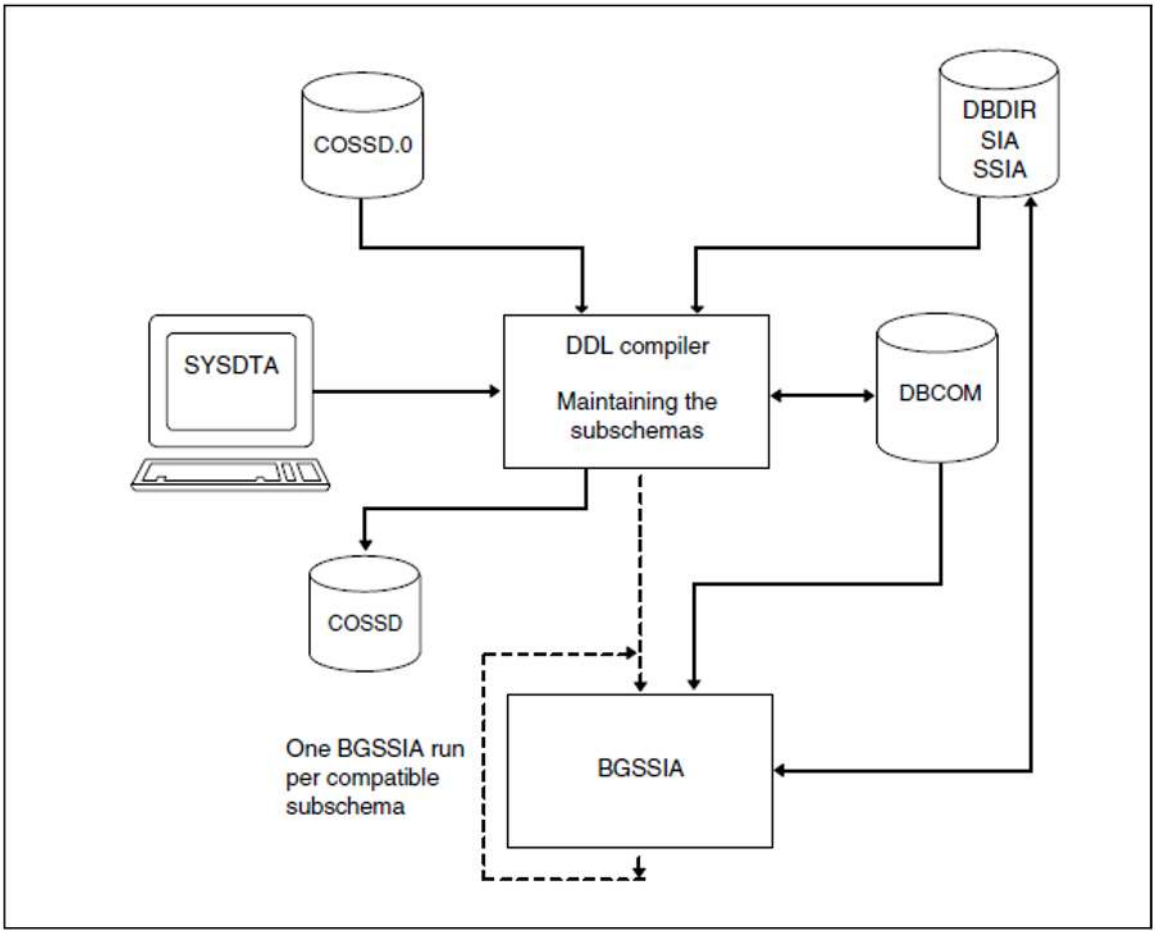


Figure 31: System environment for copying subschemas

The compiler run for copying compatible subschemas is optional; if omitted, all subschemas must be recompiled individually, and the corresponding SSIAs must be regenerated and entered in the DBDIR.

Subschema compatibility and incompatibility

schema-name and PRIVACY LOCK FOR COPY.....

A change of schema name and of PRIVACY LOCK specifications has no effect on copying the subschemas. Such changes need only be taken into account when subsequent subschema compilations are carried out.

PRIVACY LOCK FOR COMPILE

In the compiler run for copying the subschemas, the DDL compiler copies these PRIVACY specifications from the **old** subschema description so that access locks for the compilation of application programs are retained.

identifier

An old subschema is incompatible with the new schema if an *identifier* has been added, deleted or renamed in the LOCATION MODE clause, the WITHIN clause (record type level) or the SET OCCURRENCE SELECTION clause, and the corresponding record type or set is present in the subschema.

Statements for copying subschemas

The DDL compiler requires the following statements to copy the subschemas:

Statement	Default value	Meaning
COMPARE SUBSCHEMAS	-	Initiates copying of subschemas
[SORCLIST IS {YES | NO}]	YES	Prints out subschema listing
[DIAGNOSTIC {YES | NO}]	NO	Diagnoses incompatibilities of old subschemas with the new schema and lists them in the form of error messages
END	-	Terminates entry of the statements

Table 46: Statements for copying subschemas

Command sequence for copying subschemas

The following commands initiate a DDL compiler run for copying subschemas (see section “Compiling the Schema DDL”):

```
01 /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR
02 /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,SCOPE=*TASK
03 /START-UDS-DDL
04 COMPARE SUBSCHEMAS
05 [DIAGNOSTIC {YES | NO}]
06 [SORCLIST IS {YES | NO}]
07 END
```

An SSIA must then be generated for each subschema copied and entered in the DBDIR (see section “Generating the Subschema Information Area (SSIA) with BGSSIA”) using the following commands:

```
01 /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR
02 /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,SCOPE=*TASK
03 /START-UDS-BGSSIA
04 GENERATE SUBSCHEMA subschema-name OF SCHEMA schema-name
05 [DISPLAY[ SUBSCHEMA subschema-name OF SCHEMA schema-name]]
06 END
```

6.13.2 Adapting incompatible subschemas

For all subschemas which, in their original form, are not compatible with the new schema, it is necessary to do the following:

- correct the subschema description if required
- recompile the corrected subschema with the DDL compiler
- generate a new SSIA using BGSSIA and enter it in the DBDIR
- recompile and relink all relevant application programs.

It is assumed for the command sequence described here that UDS/SQL was installed with IMON (see section “START commands for the UDS/SQL programs”. The aliases for calling the utility routines are also listed in this section).

Command sequence for adapting the subschemas

Compiling the corrected subschema (see section “Compiling the Schema DDL”)

```
01 /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR
02 /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,SCOPE=*TASK
03 /START-UDS-DDL
04 [DELETE 'subschema-name':'new schema-name']
05 SOURCE IS 'subschema-file'
06 SORCLIST IS YES
07 END
08 /ASSIGN-SYSDTA TO=*SYSCMD
```

02 The version-independent module of the linked-in DBH of the relevant version is loaded dynamically (see the section entitled "Compiling, linking and loading UDS/SQL-TIAM application programs" in the "Application Programming" manual).

03 The UDS/SQL utility routine can also be started using the alias DDL.

04 The DELETE statement should be specified only if a subschema recognized as compatible and copied by the DDL compiler has been modified and requires recompilation.

Generating the SSIA and entering it in the DBDIR

See section “Generating the Subschema Information Area (SSIA) with BGSSIA”.

```
01 /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR
02 /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,SCOPE=*TASK
03 /START-UDS-BGSSIA
04 GENERATE SUBSCHEMA subschema-name OF SCHEMA schema-name
05 [DISPLAY[ SUBSCHEMA subschema-name OF SCHEMA schema-name]]
06 END
```

6.14 Adapting DB applications

Once the subschemas and the access rights have been adapted to the restructured database, all DB application programs that use an incompatible subschema must, if necessary, be corrected in accordance with the new definitions. In any case, however, they must be recompiled and relinked.

No DB application programs that use a compatible subschema need be recompiled or relinked; this also applies to SQL application programs.

Note for UDS-D:

If necessary, modified subschema modules must be transferred to the remote application program (see the "Database Operation" manual).

6.15 Updating the probable position pointers (PPP)

Pointers defined as probable position pointers (PPP) either in the old or in the new schema are not in every case updated when data is relocated during restructuring.

When records are relocated completely or partially, the following applies:

- Pointers in tables or indirect hash areas to records are updated by BALTER only if the tables or indirect hash areas have to be recreated as a result of schema modifications.
- Pointers within the records of a chain are not updated by BALTER.
- Pointers in member records to owner records are updated by BALTER when owner records are relocated.

  Pointers to records can be updated with the BREORG utility routine. You can use the REORGANIZE-POINTERS statement to update all the probable position pointers (PPP) in one realm in one go.

When recreating, deleting, relocating tables:

- Pointers in owner records to their tables are treated by BALTER as act-keys.

  The pointers are implemented in all relevant cases as act-keys:
  - When the table is recreated
  - When existing empty tables are deleted
  - When tables are relocated to another realm
  - When empty single-level lists are relocated to another realm
  - When the pointers are being newly added

If, after restructuring, probable position pointers (PPP) contain obsolete values, this may result in changes in the runtime behavior of DB applications.

6.16 Measures for restarting DB operation

If the After Image Logging was deactivated before the restructuring cycle was started, it wille result in a logging gap. After the restructuring cycle, the After Image Logging can be activated again with the BMEND utility (see the "Recovery, Information and Reorganization" manual, BMEND). Then a backup of the database has to be created again (see the "Database Operation" manual, Saving and recovering a database in the event of errors)

You then can delete the DBCOM.O and COSSD.O files, as well as user realms which are not present in the new schema.

6.17 Example

The INSURE database shown in the following diagram is to be restructured as follows:

- the realm TRANSPORT-RLM is to be added
- the record type TRANSPORT-INSURANCE is to be relocated to the realm TRANSPORT-RLM
- the set CONTR-PROP with the owner record type CUSTOMER and the member record type TRANSPORT-INSURANCE is to be added
- the set CLAIMS-TRANSPORT with the owner record type TRANSPORT-INSURANCE and the member record type DAMAGE-CLAIM is to be added
- the record type CUSTOMER is to be modified

The diagram below shows the schema of the INSURE database after restructuring (see Figure 3, "Sample databases", for a diagram of INSURE before restructuring).

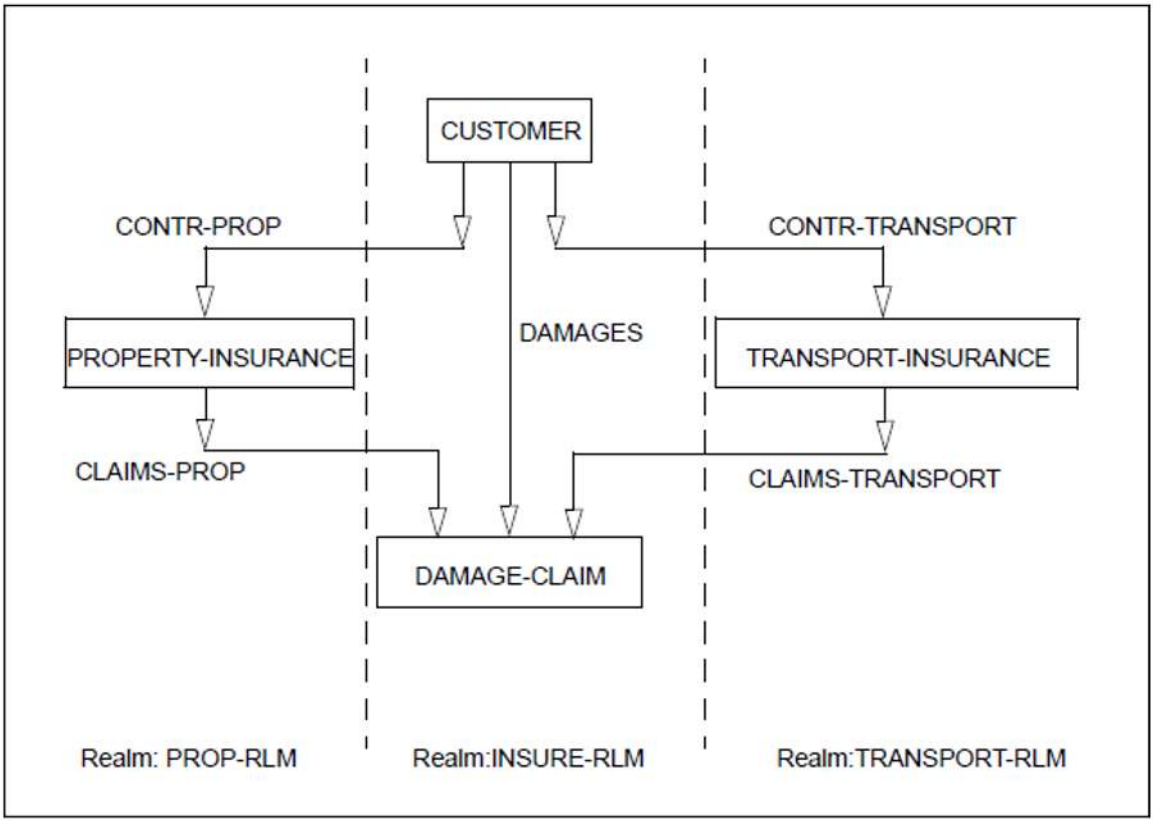


Figure 32: SHIPPINGDB database after restructuring

DBDIR, DBCOM, COSSD, HASHLIB and the user realms needed are saved before restructuring is performed (BEFRESTR). The entire database is saved after restructuring (AFTRESTR).

You should check the consistency of the database using the utility routine BCHECK before performing any save operation (see the "Recovery, Information and Reorganization" manual).

This example is only intended to illustrate the restructuring process; therefore a simple schema has been selected and the logs for Schema DDL, SSL etc. omitted.

Saving DBDIR, DBCOM, COSSD and HASHLIB

```
/COPY-FILE FROM-FILE=INSURE.DBDIR,TO-FILE=INSURE.DBDIR.BEFRESTR
/COPY-FILE FROM-FILE=INSURE.DBCOM,TO-FILE=INSURE.DBCOM.BEFRESTR
/COPY-FILE FROM-FILE=INSURE.COSSD,TO-FILE=INSURE.COSSD.BEFRESTR
/COPY-FILE FROM-FILE=INSURE.HASHLIB,TO-FILE=INSURE.HASHLIB.BEFRESTR
```

BCHANGE run and compiling new Schema DDL and SSL

The Schema DDL in this run still contains errors. The error involved is not detected until the SSL is compiled.

```
/START-UDS-BCHANGE
***** START       BCHANGE       (UDS/SQL  V2.9  1801 )     2019-01-29   09:35:40

***** THE FILE: :IUDS:$XXXXXXXX.INSURE.DBCOM IS COPIED TO:
     :IUDS:$XXXXXXXX.INSURE.DBCOM.O

***** THE FILE: :IUDS:$XXXXXXXX.INSURE.COSSD IS COPIED TO:
     :IUDS:$XXXXXXXX.INSURE.COSSD.O
1006 RESTRUCTURING SUCCESSFULLY INITIATED

***** DIAGNOSTIC SUMMARY OF BCHANGE

              NO WARNINGS
              NO ERRORS
              NO SYSTEM-ERRORS

***** END OF DIAGNOSTIC SUMMARY
***** NR OF DATABASE ACCESSES  :           94
***** NORMAL END   BCHANGE      (UDS/SQL  V2.9  1801 )     2019-01-29   09:35:41

/CREATE-FILE FILE-NAME=INSURE.DBSTAT
/CREATE-FILE FILE-NAME=INSURE.DBSTAT.SAVE

/START-UDS-DDL
***** START        DDLCOMP      (UDS/SQL  V2.9  1801 )    2019-01-29   09:35:41
*  DDLCOMP: INPUT SYSTEMPARAMETERS
SOURCE IS 'S.INSURE.DDL.NEW'
END
*  DDLCOMP: READ SCHEMA/SUBSCHEMA
%  UDS0215 UDS STARTING UDS/SQL V2.9 (LINKED-IN), DATE=2019-01-29 (ILL2038,09:35:41/4TE7)
%  UDS0746 UDS PUBSET DECLARATION (CURRENT) FOLLOWS (ILL1746,09:35:41/4TE7)
4TE7: UDS-PUBSET-JV:  :IUDS:$XXXXXXXX.PUBSDECL.DEFAULT
4TE7: DEFAULT PUBSET: IUDS
4TE7: ----------------------------------------
*  DDLCOMP: START SCHEMA-PHASE
*  DDLCOMP: CHECK SCHEMA RULES
*  DDLCOMP: CHECK DATA ALLOCATION
*  DDLCOMP: SEMANTIC TEST
*  DDLCOMP: CYCLUS TESTS
*  DDLCOMP: ERROR DIAGNOSTIC
*  DDLCOMP: NO ERRORS IN SCHEMA-PHASE
*  DDLCOMP: CREATE FILE COSSD
*  DDLCOMP: NO ERRORS DETECTED
%  UDS0758 NUMBER OF DML-STATEMENTS AND I/O COUNTERS PER DATABASE (ILL1758,09:35:42/4TE7)
 4TE7: DATABASE NAME      DMLS   LOG READ  PHYS READ  LOG WRITE PHYS WRITE
```

```
 4TE7: ------------------------------------------------------------------
 4TE7: INSURE              651       1999         67        914         39
%  UDS0213 UDS NORMAL SYSTEM TERMINATION WITH *************651 DML-STATEMENTS 2019-01-29
(ILLY033,09:35:42/4TE7)

***** DIAGNOSTIC SUMMARY FOR DDL-SCHEMA CUSTOMER-CARDS

              NO ERRORS
 +++++         9 WARNINGS

***** END OF DIAGNOSTIC SUMMARY
***** NORMAL END   DDLCOMP      (UDS/SQL  V2.9  1801 )     2019-01-29   09:35:42

/START-UDS-SSL
***** START        SSLCOMP      (UDS/SQL  V2.9  1801 )     2019-01-29   09:35:42
*  SSLCOMP: INPUT SYSTEMPARAMETERS
SOURCE IS 'S.INSURE.SSL.NEW'
END
*  SSLCOMP: READ SSL-SCHEMA
%  UDS0215 UDS STARTING UDS/SQL V2.9 (LINKED-IN), DATE=2019-01-29 (ILL2038,09:35:42/4TE7)
%  UDS0746 UDS PUBSET DECLARATION (CURRENT) FOLLOWS (ILL1746,09:35:42/4TE7)
4TE7: UDS-PUBSET-JV:  :IUDS:$XXXXXXXX.PUBSDECL.DEFAULT
4TE7: DEFAULT PUBSET: IUDS
4TE7: ----------------------------------------
*  SSLCOMP: START SSL-PHASE
*  SSLCOMP: CHECK SSL RULES
*  SSLCOMP: SEMANTIC TEST
*  SSLCOMP: ERROR DIAGNOSTIC
*  SSLCOMP: ERRORS DETECTED IN SSL-PHASE
*  SSLCOMP: ERRORS DETECTED
*  SSLCOMP: ALL SSL-OPTIONS ARE RESET
+++++ ERROR: 0012 UDS-DBH RETURNS WITH DATABASE-STATUS '04021'
%  UDS0758 NUMBER OF DML-STATEMENTS AND I/O COUNTERS PER DATABASE  (ILL1758,09:35:42/4TE7)
 4TE7: DATABASE NAME      DMLS   LOG READ  PHYS READ  LOG WRITE PHYS WRITE
 4TE7: ------------------------------------------------------------------
 4TE7: INSURE              303        387         61         71         25
%  UDS0213 UDS NORMAL SYSTEM TERMINATION WITH *************303 DML-STATEMENTS 2019-01-29
(ILLY033,09:35:42/4TE7)

***** DIAGNOSTIC SUMMARY FOR SSL - SCHEMA

+++++           2 ERRORS
               NO WARNINGS

***** END OF DIAGNOSTIC SUMMARY
+++++ ABNORMAL END SSLCOMP     (UDS/SQL V2.9  1801 )     2019-01-29   09:35:42
```

Compiling the corrected schema

Once you have corrected the Schema DDL according to the SSL-ERROR-DIAGNOSTIC, you must delete the errored schema that has already been entered. Only then can you compile the corrected Schema DDL and then the SSL.

```
/START-UDS-DDL
***** START        DDLCOMP       (UDS/SQL  V2.9  1801 )     2019-01-29   09:35:42
*  DDLCOMP: INPUT SYSTEMPARAMETERS
```

```
DELETE SCHEMA 'CUSTOMER-CARDS'
END
%  UDS0215 UDS STARTING UDS/SQL V2.9 (LINKED-IN), DATE=2019-01-29 (ILL2038,09:35:42/4TE7)
%  UDS0746 UDS PUBSET DECLARATION (CURRENT) FOLLOWS (ILL1746,09:35:43/4TE7)
4TE7: UDS-PUBSET-JV:  :IUDS:$XXXXXXXX.PUBSDECL.DEFAULT
4TE7: DEFAULT PUBSET: IUDS
4TE7: ----------------------------------------
*  DDLCOMP: SCHEMA HAS BEEN ERASED
*  DDLCOMP: NO ERRORS DETECTED
%  UDS0758 NUMBER OF DML-STATEMENTS AND I/O COUNTERS PER DATABASE (ILL1758,09:35:43/4TE7)
 4TE7: DATABASE NAME      DMLS   LOG READ  PHYS READ  LOG WRITE PHYS WRITE
 4TE7: ------------------------------------------------------------------
 4TE7: INSURE                6       1075         70        556         39
%  UDS0213 UDS NORMAL SYSTEM TERMINATION WITH ***************6 DML-STATEMENTS 2019-01-29
(ILLY033,09:35:43/4TE7)
***** NORMAL END   DDLCOMP      (UDS/SQL  V2.9  1801 )     2019-01-29   09:35:43

/START-UDS-DDL
***** START        DDLCOMP      (UDS/SQL  V2.9  1801 )     2019-01-29   09:35:43
*  DDLCOMP: INPUT SYSTEMPARAMETERS
SOURCE IS 'S.INSURE.DDL.KORR'
DISPLAY IS YES
END
*  DDLCOMP: READ SCHEMA/SUBSCHEMA
%  UDS0215 UDS STARTING UDS/SQL V2.9 (LINKED-IN), DATE=2019-01-29 (ILL2038,09:35:43/4TE7)
%  UDS0746 UDS PUBSET DECLARATION (CURRENT) FOLLOWS (ILL1746,09:35:43/4TE7)
4TE7: UDS-PUBSET-JV:  :IUDS:$XXXXXXXX.PUBSDECL.DEFAULT
4TE7: DEFAULT PUBSET: IUDS
4TE7: ----------------------------------------
*  DDLCOMP: START SCHEMA-PHASE
*  DDLCOMP: CHECK SCHEMA RULES
*  DDLCOMP: CHECK DATA ALLOCATION
*  DDLCOMP: SEMANTIC TEST
*  DDLCOMP: CYCLUS TESTS
*  DDLCOMP: ERROR DIAGNOSTIC
*  DDLCOMP: NO ERRORS IN SCHEMA-PHASE
*  DDLCOMP: DISPLAY SCHEMA
*  DDLCOMP: CREATE FILE COSSD
*  DDLCOMP: NO ERRORS DETECTED
%  UDS0758 NUMBER OF DML-STATEMENTS AND I/O COUNTERS PER DATABASE (ILL1758,09:35:43/4TE7)
 4TE7: DATABASE NAME      DMLS   LOG READ  PHYS READ  LOG WRITE PHYS WRITE
 4TE7: ------------------------------------------------------------------
 4TE7: INSURE              751       2120         66        914         40
%  UDS0213 UDS NORMAL SYSTEM TERMINATION WITH *************751 DML-STATEMENTS 2019-01-29
(ILLY033,09:35:44/4TE7)

***** DIAGNOSTIC SUMMARY FOR DDL-SCHEMA CUSTOMER-CARDS

             NO ERRORS
+++++         9 WARNINGS
***** END OF DIAGNOSTIC SUMMARY
***** NORMAL END   DDLCOMP     (UDS/SQL  V2.9  1801 )     2019-01-29   09:35:44

/START-UDS-SSL
***** START        SSLCOMP     (UDS/SQL  V2.9  1801 )     2019-01-29   09:35:44
*  SSLCOMP: INPUT SYSTEMPARAMETERS
SOURCE IS 'S.INSURE.SSL.NEW'
END
*  SSLCOMP: READ SSL-SCHEMA
```

```
%  UDS0215 UDS STARTING UDS/SQL V2.9 (LINKED-IN), DATE=2019-01-29 (ILL2038,09:35:44/4TE7)
%  UDS0746 UDS PUBSET DECLARATION (CURRENT) FOLLOWS (ILL1746,09:35:44/4TE7)
4TE7: UDS-PUBSET-JV:  :IUDS:$XXXXXXXX.PUBSDECL.DEFAULT
4TE7: DEFAULT PUBSET: IUDS
4TE7: ----------------------------------------
*  SSLCOMP: START SSL-PHASE
*  SSLCOMP: CHECK SSL RULES
*  SSLCOMP: SEMANTIC TEST
*  SSLCOMP: ERROR DIAGNOSTIC
*  SSLCOMP: NO ERRORS DETECTED
%  UDS0758 NUMBER OF DML-STATEMENTS AND I/O COUNTERS PER DATABASE (ILL1758,09:35:44/4TE7)
 4TE7: DATABASE NAME      DMLS   LOG READ  PHYS READ  LOG WRITE PHYS WRITE
 4TE7: ------------------------------------------------------------------
 4TE7: INSURE              127        253         63         34         23
%  UDS0213 UDS NORMAL SYSTEM TERMINATION WITH *************127 DML-STATEMENTS 2019-01-29
(ILLY033,09:35:44/4TE7)

***** DIAGNOSTIC SUMMARY FOR SSL - SCHEMA

              NO ERRORS
              NO WARNINGS

***** END OF DIAGNOSTIC SUMMARY
***** NORMAL END   SSLCOMP     (UDS/SQL  V2.9  1801 )    2019-01-29   09:35:44
/DELETE-SYSTEM-FILE FILE-NAME=*OMF

/START-UDS-BGSIA
***** START        BGSIA       (UDS/SQL  V2.9  1801 )    2019-01-29   09:35:44
GENERATE SCHEMA CUSTOMER-CARDS
DISPLAY
END
%  UDS0215 UDS STARTING UDS/SQL V2.9 (LINKED-IN), DATE=2019-01-29 (ILL2038,09:35:44/4TE7)
%  UDS0746 UDS PUBSET DECLARATION (CURRENT) FOLLOWS (ILL1746,09:35:44/4TE7)
4TE7: UDS-PUBSET-JV:  :IUDS:$XXXXXXXX.PUBSDECL.DEFAULT
4TE7: DEFAULT PUBSET: IUDS
4TE7: ----------------------------------------
ESTIMATE-REPORT

********** FOR USER-REALM     3 NAME IS : PROP-RLM
     A SIZE OF         24 BLOCKS WAS ESTIMATED

********** FOR USER-REALM     4 NAME IS : INSURE-RLM
     A SIZE OF        239 BLOCKS WAS ESTIMATED

********** FOR USER-REALM     6 NAME IS : TRANSPORT-RLM
     A SIZE OF         24 BLOCKS WAS ESTIMATED
END OF ESTIMATE-REPORT
%  UDS0758 NUMBER OF DML-STATEMENTS AND I/O COUNTERS PER DATABASE (ILL1758,09:35:45/4TE7)
 4TE7: DATABASE NAME      DMLS   LOG READ  PHYS READ  LOG WRITE PHYS WRITE
 4TE7: ------------------------------------------------------------------
 4TE7: INSURE              569        779         60        183         30
%  UDS0213 UDS NORMAL SYSTEM TERMINATION WITH *************569 DML-STATEMENTS 2019-01-29
(ILLY033,09:35:45/4TE7)

***** DIAGNOSTIC SUMMARY OF BGSIA

              NO WARNINGS
              NO ERRORS
              NO SYSTEM-ERRORS
```

```
***** END OF DIAGNOSTIC SUMMARY
***** NORMAL END   BGSIA       (UDS/SQL  V2.9  1801 )     2019-01-29   09:35:45

/MODIFY-JOB-SWITCHES ON=(4)
/START-LMS
//MODIFY-LOGGING-PARAMETERS LOG=*MAX
//OPEN-LIBRARY LIB=INSURE.HASHLIB,MODE=*UPDATE
//ADD-ELEMENT FROM-FILE=*OMF,TO-ELEM=*LIB-ELEM(TYPE=R),WRITE-MODE=*ANY
INPUT  OMF
OUTPUT LIBRARY= :IUDS:$XXXXXXXX.INSURE.HASHLIB
        ADD UDSHASH AS (R)UDSHASH/@(0002)/2019-01-29 , OUTPUT REPLACED
//SHOW-ELEM-ATTR
INPUT  LIBRARY= :IUDS:$XXXXXXXX.INSURE.HASHLIB
TYP NAME    VER (VAR#) DATE         NAME    VER (VAR#) DATE
(R) ADMIN## @   (0001) 2019-01-29  UDSHASH @   (0002) 2019-01-29
      2 (R)-ELEMENT(S) IN THIS TABLE OF CONTENTS
//END
/MODIFY-JOB-SWITCHES OFF=(4)
```

Analysis phase with REPORT IS YES and EXECUTION IS NO

```
/START-UDS-BALTER
***** START       BALTER       (UDS/SQL  V2.9  1801 )     2019-01-29   09:35:45
REPORT    IS YES.
EXECUTION IS NO.
END.

***  ANALYSE-PHASE    ***
*** DATE AND TIME 2019-01-29  09:35:45

+++++ WARNING: 1081 AREAS DELETED FROM RECORD-WITHIN-CLAUSE
RECORD: TRANSPORT-INSURANCE
      IF RECORD OCCURRENCES ARE PRESENT IN AREAS
      WHICH ARE DELETED FROM RECORD-WITHIN-CLAUSE
      THE RESTRUCTURING PROCESS WILL END  ABNORMALLY.

NO ERRORS DETECTED IN SCHEMA CHANGES

***  REPORT-PHASE     ***
*** DATE AND TIME 2019-01-29  09:35:45

REALM NOT NEEDED: PROP-RLM
REALM NEEDED: INSURE-RLM

DATABASE NOT ALTERED

NUMBER OF FILE ACCESSES:            0

***** DIAGNOSTIC SUMMARY OF BALTER

+++++           1 WARNINGS
               NO ERRORS
               NO SYSTEM-ERRORS

***** END OF DIAGNOSTIC SUMMARY
***** NR OF DATABASE ACCESSES  :          107
***** NORMAL END   BALTER      (UDS/SQL  V2.9  1801 )    2019-01-29   09:35:45
```

Since the database does not include a record of the record type TRANSPORT-INSURANCE, the warning can be ignored and the database restructured.

Of the two user realms INSURE.PROP-RLM and INSURE.INSURE-RLM, only INSURE.INSURE-RLM is needed for the restructuring process.

This realm is saved:

```
/COPY-FILE FROM-FILE=INSURE.INSURE-RLM, TO-FILE=INSURE.INSURE-RLM.BEFRESTR
```

Restructuring phase

```
/CREATE-FILE FILE-NAME=INSURE.TRANSPORT-RLM,SUPPORT=*PUBLIC-DISK( -
/   PRIMARY-ALLOCATION=50,SECONDARY-ALLOCATION=50)
/START-UDS-BALTER
***** START       BALTER        (UDS/SQL  V2.9  1801 )    2019-01-29   09:35:45
REPORT    IS NO .
```

```
EXECUTION IS YES.
END.

***  ANALYSE-PHASE    ***
*** DATE AND TIME 2019-01-29  09:35:45

+++++ WARNING: 1081 AREAS DELETED FROM RECORD-WITHIN-CLAUSE
RECORD: TRANSPORT-INSURANCE
      IF RECORD OCCURRENCES ARE PRESENT IN AREAS
      WHICH ARE DELETED FROM RECORD-WITHIN-CLAUSE
      THE RESTRUCTURING PROCESS WILL END  ABNORMALLY.

NO ERRORS DETECTED IN SCHEMA CHANGES

***  EXECUTION-PHASE  ***
*** DATE AND TIME 2019-01-29  09:35:45

REALM ADDED TO DATABASE: TRANSPORT-RLM
*** DATE AND TIME 2019-01-29  09:35:46

MODIFICATION  CONCERNING OWNER ATTRIBUTE  STARTED FOR
REC NAME: TRANSPORT-INSURANCE
REC REF:      3
*** DATE AND TIME 2019-01-29  09:35:46

MODIFICATION  CONCERNING OWNER ATTRIBUTE  STARTED FOR
REC NAME: CUSTOMER
REC REF:      4
*** DATE AND TIME 2019-01-29  09:35:46

RECORD MODIFICATION STARTED FOR:
REC NAME: TRANSPORT-INSURANCE
REC REF:      3
*** DATE AND TIME 2019-01-29  09:35:46

RECORD MODIFICATION STARTED FOR:
REC NAME: CUSTOMER
REC REF:      4
*** DATE AND TIME 2019-01-29  09:35:46

RECORD MODIFICATION STARTED FOR:
REC NAME: DAMAGE-CLAIM
REC REF:      5
*** DATE AND TIME 2019-01-29  09:35:46

DATABASE ALTERED
*** DATE AND TIME 2019-01-29  09:35:46

NUMBER OF FILE ACCESSES:            10

***** DIAGNOSTIC SUMMARY OF BALTER

+++++           1 WARNINGS
             NO ERRORS
             NO SYSTEM-ERRORS

***** END OF DIAGNOSTIC SUMMARY
***** NR OF DATABASE ACCESSES  :          248
***** NORMAL END   BALTER       (UDS/SQL  V2.9  1801 )     2019-01-29   09:35:46
```

Entering new access rights

```
/START-UDS-BPRIVACY
***** START        BPRIVACY     (UDS/SQL  V2.9  1801 )     2019-01-29   09:35:46
%  UDS0215 UDS STARTING UDS/SQL V2.9 (LINKED-IN), DATE=2019-01-29 (ILL2038,09:35:46/4TE7)
%  UDS0746 UDS PUBSET DECLARATION (CURRENT) FOLLOWS (ILL1746,09:35:46/4TE7)
4TE7: UDS-PUBSET-JV:  :IUDS:$XXXXXXXX.PUBSDECL.DEFAULT
4TE7: DEFAULT PUBSET: IUDS
4TE7: ----------------------------------------
%  UDS0722 UDS ORDER ADD RLOG 190129083546 IN EXECUTION (ILL1283,09:35:46/4TE7)
%  UDS0356 UDS EXECUTION OF ORDERS FOR INSURE   TERMINATED (ILL1309,09:35:46/4TE7)
//ADD-USER-GROUP USER-GROUP-NAME=*FREE-FORMAT(HOST=IBAPROD1,USER-ID=XXXXXXXX), -
//  OBJECT=(*REALM(NAME=*ALL,RIGHT=ALL),*RECORD(NAME=*ALL,RIGHT=ALL), -
//  *SET(NAME=*ALL,RIGHT=ALL))
//END
%  UDS0758 NUMBER OF DML-STATEMENTS AND I/O COUNTERS PER DATABASE (ILL1758,09:35:46/4TE7)
 4TE7: DATABASE NAME          DMLS    LOG READ  PHYS READ  LOG WRITE PHYS WRITE
 4TE7: ------------------------------------------------------------------------
 4TE7: INSURE                   11         115         57         36         23
%  UDS0213 UDS NORMAL SYSTEM TERMINATION WITH **************11 DML-STATEMENTS 2019-01-29
(ILLY033,09:35:46/4TE7)

***** DIAGNOSTIC SUMMARY OF BPRIVACY

              NO WARNINGS
              NO ERRORS
              NO SYSTEM-ERRORS

***** END OF DIAGNOSTIC SUMMARY
***** NORMAL END   BPRIVACY    (UDS/SQL  V2.9  1801 )     2019-01-29   09:35:46
```

Testing whether the subschema is compatible with the new schema

```
/START-UDS-DDL
***** START        DDLCOMP      (UDS/SQL  V2.9  1801 )     2019-01-29   09:35:46
*  DDLCOMP: INPUT SYSTEMPARAMETERS
COMPARE SUBSCHEMAS
DIAGNOSTIC IS YES
END
%  UDS0215 UDS STARTING UDS/SQL V2.9 (LINKED-IN), DATE=2019-01-29 (ILL2038,09:35:46/4TE7)
%  UDS0746 UDS PUBSET DECLARATION (CURRENT) FOLLOWS (ILL1746,09:35:46/4TE7)
4TE7: UDS-PUBSET-JV:  :IUDS:$XXXXXXXX.PUBSDECL.DEFAULT
4TE7: DEFAULT PUBSET: IUDS
4TE7: ----------------------------------------

*  DDLCOMP: READ SCHEMA/SUBSCHEMA      1
*  DDLCOMP: START SUBSCHEMA-PHASE
*  DDLCOMP: CHECK SUBSCHEMA RULES
*  DDLCOMP: CHECK DATA ALLOCATION
*  DDLCOMP: SUBCOPY
*  DDLCOMP: ERROR DIAGNOSTIC
*  DDLCOMP: ERRORS DETECTED IN SUBSCHEMA-PHASE
*  DDLCOMP: SUBSCHEMA HAS BEEN ERASED
%  UDS0758 NUMBER OF DML-STATEMENTS AND I/O COUNTERS PER DATABASE (ILL1758,09:35:47/4TE7)
 4TE7: DATABASE NAME      DMLS   LOG READ  PHYS READ  LOG WRITE PHYS WRITE
 4TE7: ------------------------------------------------------------------
 4TE7: INSURE              832       2616         74        936         44
%  UDS0213 UDS NORMAL SYSTEM TERMINATION WITH **************832 DML-STATEMENTS 2019-01-29
(ILLY033,09:35:47/4TE7)

***** DIAGNOSTIC SUMMARY FOR DDL-SUBSCHEMA

+++++           1 ERRORS
+++++           9 WARNINGS

***** END OF DIAGNOSTIC SUMMARY
***** NORMAL END   DDLCOMP     (UDS/SQL  V2.9  1801 )     2019-01-29   09:35:47
```

Modifying the subschema

Since the old Subschema DDL is not compatible with the new Schema DDL, the Subschema DDL is corrected and then recompiled.

```
/START-UDS-DDL
***** START        DDLCOMP       (UDS/SQL  V2.9  1801 )     2019-01-29   09:35:47
*  DDLCOMP: INPUT SYSTEMPARAMETERS
SOURCE IS 'S.INSURE.SUBDDL.NEW'
END
*  DDLCOMP: READ SCHEMA/SUBSCHEMA
%  UDS0215 UDS STARTING UDS/SQL V2.9 (LINKED-IN), DATE=2019-01-29 (ILL2038,09:35:47/4TE7)
%  UDS0746 UDS PUBSET DECLARATION (CURRENT) FOLLOWS (ILL1746,09:35:47/4TE7)
4TE7: UDS-PUBSET-JV:  :IUDS:$XXXXXXXX.PUBSDECL.DEFAULT
4TE7: DEFAULT PUBSET: IUDS
4TE7: ----------------------------------------
*  DDLCOMP: START SUBSCHEMA-PHASE
*  DDLCOMP: CHECK SUBSCHEMA RULES
*  DDLCOMP: CHECK DATA ALLOCATION
```

```
*  DDLCOMP: SUBCOPY
*  DDLCOMP: ERROR DIAGNOSTIC
*  DDLCOMP: NO ERRORS IN SUBSCHEMA-PHASE
*  DDLCOMP: WRITE SUBSCHEMA ON COSSD
*  DDLCOMP: NO ERRORS DETECTED
%  UDS0758 NUMBER OF DML-STATEMENTS AND I/O COUNTERS PER DATABASE (ILL1758,09:35:48/4TE7)
 4TE7: DATABASE NAME      DMLS   LOG READ  PHYS READ  LOG WRITE PHYS WRITE
 4TE7: ------------------------------------------------------------------
 4TE7: INSURE             1363       2581         76        631         49
%  UDS0213 UDS NORMAL SYSTEM TERMINATION WITH ************1363 DML-STATEMENTS 2019-01-29
(ILLY033,09:35:48/4TE7)

***** DIAGNOSTIC SUMMARY FOR DDL-SUBSCHEMA

              NO ERRORS
              NO WARNINGS

***** END OF DIAGNOSTIC SUMMARY
***** NORMAL END  DDLCOMP  (UDS/SQL  V2.9  1801 )    2019-01-29   09:35:48
/START-UDS-BGSSIA
***** START      BGSSIA   (UDS/SQL  V2.9  1801 )     2019-01-29   09:35:48
GENERATE SUBSCHEMA MANAGEMENT OF SCHEMA CUSTOMER-CARDS
DISPLAY
END
%  UDS0215 UDS STARTING UDS/SQL V2.9 (LINKED-IN), DATE=2019-01-29 (ILL2038,09:35:48/4TE7)
%  UDS0746 UDS PUBSET DECLARATION (CURRENT) FOLLOWS (ILL1746,09:35:48/4TE7)
4TE7: UDS-PUBSET-JV:  :IUDS:$XXXXXXXX.PUBSDECL.DEFAULT
4TE7: DEFAULT PUBSET: IUDS
4TE7: ----------------------------------------
*** SSIA GENERATION NORMALLY ENDED.
*GENERATION OF ITEM-TABLE AND NAME-TABLE STARTED.
*GENERATION OF ITEM-TABLE AND NAME-TABLE FINISHED.
%  UDS0758 NUMBER OF DML-STATEMENTS AND I/O COUNTERS PER DATABASE (ILL1758,09:35:48/4TE7)
 4TE7: DATABASE NAME      DMLS   LOG READ  PHYS READ  LOG WRITE PHYS WRITE
 4TE7: ------------------------------------------------------------------
 4TE7: INSURE              781       1359         76        286         29
%  UDS0213 UDS NORMAL SYSTEM TERMINATION WITH *************781 DML-STATEMENTS 2019-01-29
(ILLY033,09:35:48/4TE7)

***** DIAGNOSTIC SUMMARY OF BGSSIA

              NO WARNINGS
              NO ERRORS
              NO SYSTEM-ERRORS

***** END OF DIAGNOSTIC SUMMARY
***** NORMAL END   BGSSIA       (UDS/SQL  V2.9  1801 )     2019-01-29   09:35:48
```

The restructuring of the database has now been completed.

Now, of course, the DB application programs have to be modified if they reference the modified items and have to be recompiled and linked again due to the incompatibility between the old and new subschemas.

Reorganizing the restructured database

To save memory space, the hash area for the record type CUSTOMER is reorganized, i.e. moved back to the area that is now free at the front of the realm. This means that the size of INSURE-RLM can be minimized.

```
/START-UDS-BREORG
***** START        BREORG       (UDS/SQL  V2.9  1801 )    2019-01-29   09:35:48
//SPECIFY-SCHEMA SCHEMA-NAME=CUSTOMER-CARDS
//REORGANIZE-CALC RECORD-NAME=CUSTOMER,
  CALC-RECORD=*WITHIN-POPULATION(REALM=INSURE-RLM,POPULATION=500),CALC-SEARCHKEY=NONE
//END
***** BEGIN OF CALC-REORGANIZATION     AT 09:35:49
***** RESULTS OF CALC-REORGANIZATION OF RECORD CUSTOMER
     NEW CALC BEGIN            :           4-          5
     NEW NR OF PRIMARY BUCKETS :          59
     NEW NR OF OVERFLOW BUCKETS:           0
***** END   OF CALC-REORGANIZATION     AT 09:35:49

***** DIAGNOSTIC SUMMARY OF BREORG

              NO WARNINGS
              NO ERRORS
              NO SYSTEM-ERRORS

***** END OF DIAGNOSTIC SUMMARY
***** NR OF DATABASE ACCESSES  :           77
***** NORMAL END   BREORG       (UDS/SQL  V2.9  1801 )     2019-01-29   09:35:49

/START-UDS-BREORG
***** START        BREORG       (UDS/SQL  V2.9  1801 )     2019-01-29   09:35:49
//SPECIFY-SCHEMA SCHEMA-NAME=CUSTOMER-CARDS
//MODIFY-REALM-SIZE REALM-NAME=INSURE-RLM,REALM-SIZE=MINIMUM
//END
***** BEGIN OF REALM-SIZE-MODIFICATION     AT 09:35:50
***** RESULTS OF FPA-REORGANIZATION OF AREA INSURE-RLM
     NEW FPA FIRST PAGE        : NOT CHANGED
     NEW FPA LAST  PAGE        : NOT CHANGED
     NEW FPA SIZE              : NOT CHANGED
     NEW NR OF PAGES           :              80
***** END   OF REALM-SIZE-MODIFICATION     AT 09:35:50

***** DIAGNOSTIC SUMMARY OF BREORG

              NO WARNINGS
              NO ERRORS
              NO SYSTEM-ERRORS

***** END OF DIAGNOSTIC SUMMARY
***** NR OF DATABASE ACCESSES  :           70
***** NORMAL END   BREORG       (UDS/SQL  V2.9  1801 )     2019-01-29   09:35:50
```

Measures to be taken before resuming database operation

A shadow database with the suffix AFTRESTR is created. The data saved before restructuring took place is deleted along with the files INSURE.DBCOM.O and INSURE.COSSD.O.

```
/COPY-FILE FROM-FILE=INSURE.HASHLIB,TO-FILE=INSURE.HASHLIB.AFTRESTR
/COPY-FILE FROM-FILE=INSURE.COSSD  ,TO-FILE=INSURE.COSSD.AFTRESTR
/COPY-FILE FROM-FILE=INSURE.DBDIR  ,TO-FILE=INSURE.DBDIR.AFTRESTR
/COPY-FILE FROM-FILE=INSURE.DBCOM  ,TO-FILE=INSURE.DBCOM.AFTRESTR
/COPY-FILE FROM-FILE=INSURE.PROP-RLM,TO-FILE=INSURE.PROP-RLM.AFTRESTR
/COPY-FILE FROM-FILE=INSURE.INSURE-RLM,TO-FILE=INSURE.INSURE-RLM.AFTRESTR
/COPY-FILE FROM-FILE=INSURE.TRANSPORT-RLM,TO-FILE=INSURE.TRANSPORT-RLM.AFTRESTR

/DELETE-FILE FILE-NAME=INSURE*BEFRESTR*

/DELETE-FILE FILE-NAME=INSURE.DBCOM.O
/DELETE-FILE FILE-NAME=INSURE.COSSD.O
```

7 Renaming database objects (BRENAME, BALTER)

The renaming cycle of BRENAME/BALTER enables datenbase objects in existing databases to be renamed. To do this, only the structure information of the DBDIR, DBCOM and COSSD database files needs to be modified. When names of user realms are changed, some structure information in the relevant realms is also changed.
However, the actual user data (records and tables in the user realms) are neither checked nor changed in the renaming cycle. Consequently only changes which leave the physical database structure unchanged are permitted in the renaming cycle.

> ¡ As only structure information is modified, a renaming cycle can execute very quickly.

The activities which are required during renaming are divided into three sections:

- preparatory measures
- renaming process
- follow-up activities.

Preparatory measures

In contrast to the restructuring, in a renaming cycle the After Image Logging may remain activated. Only if the name of a realm is to be changed you have to deactivate the afterimage logging with the BMEND utility (see the "Recovery, Information and Reorganization" manual, BMEND)

Renaming process

This is a process that resembles the creation of a database:

- BRENAME prepares the DBDIR to accept a new SIA
- New DDL and SSL definitions are then compiled and the new SIA is entered in the DBDIR
- BALTER checks the renaming and updates the structure information

The following measures can be taken in a BRENAME/BALTER renaming cycle:

- Changing item names in record types
- Changing the types of items in record types
- Subdivision of an existing item into multiple adjacent individual items
- Conversion of an existing item into a vector
- Conversion of one or more consecutive items into a repeating group
- Grouping of multiple adjacent individual items into a new item
- Conversion of a vector into a new individual item
- Combination of a repeating group into one or more consecutive individual items
- Changing of record names
- Changing of set names
- Changing of realm names

- In a renaming cycle between BRENAME and BALTER BMEND cannot be executed.
- The renaming cycle of BRENAME/BALTER cannot be combined with the renaming cycle of BCHANGE /BALTER.

Names of SEARCH keys in the DDL/SSL source of the schema enable the declarations of the SEARCH keys of the DDL and SSL compilers to be allocated unambiguously. As with restructuring (BCHANGE/BALTER cycle), these names can be changed without the schema or subschema description in the SIA or SSIA changing. Consequently such changes in the renaming cycle are not taken account either in the analysis or in BALTER’s REPORT outputs.

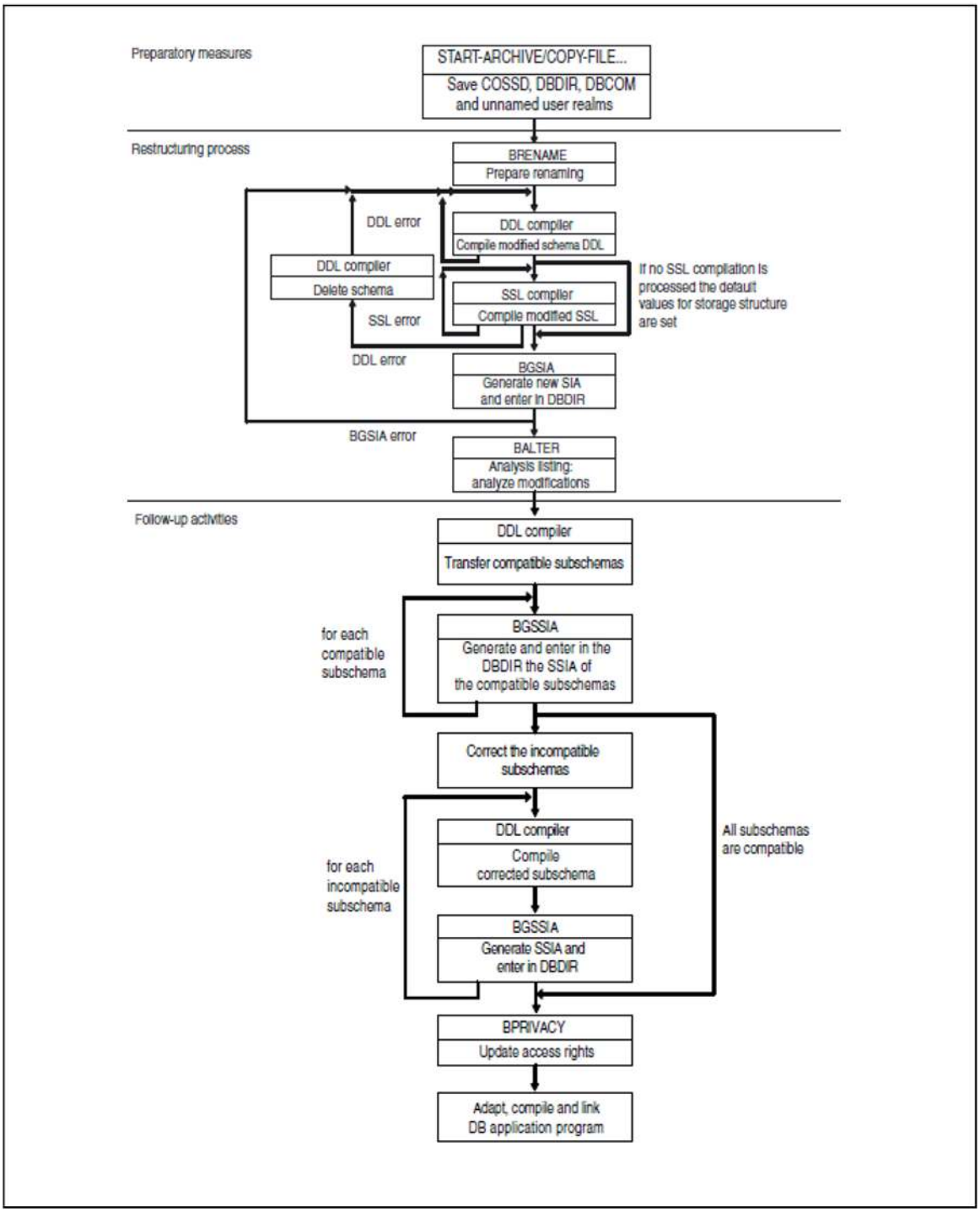


Figure 33: Restructuring process

Follow-up activities

The following activities must be performed after renaming:

- Adapting the subschemas to the changed schema
- Adapting DB application programs to the new schema
- If required, generating new SSITAB modules for CALL-DML programs using the BCALLSI utility routine
- If required, updating access rights
- If required, adapting, compiling and linking application programs
- If required, copying changed database files and activating the After Image Logging again with BMEND.

> ¡ A logging gap occurs because of the renaming cycle (see the “Database Operation” manual, Media recovery). After the renaming cycle you must therefore establish a new basis for media recovery by copying the modified files together with the unmodified files. You must then use BMEND to activate After Image Logging again.

Changing stored data

The user data are not changed in a renaming cycle. If renaming entails semantic changes to the user data, any necessary changes to the stored data must, for example, be executed using special application programs. This can take place in normal database operation either before or after the renaming cycle (see section “Adapting user data”).

7.1 Modifying the Schema DDL

If you want to modify your Schema DDL, you must create a new or modified complete schema definition and recompile this. The new names must be used at all places in the DDL and SSL sources.
BALTER checks the renaming and updates the structure information.

To ensure that BGSIA recognizes renamed realms, record types and sets and leaves the existing references unchanged, name changes which are to be made for realms, record types and sets must be specified in full in additional statements.

The following modifications are possible in the Schema DDL:

- Changing item names in record types
  The names must be changed at all places (e.g. also in key definitions).
- Changing types of items
  The length in bytes of the data items may not be changed. Type changes from CHAR and to CHAR are possible. In the case of conversion to NCHAR only half the characters are stored in the new item. Type changes of variable items and of items in compressed records are not permitted.
- The user data in the database also remains physically unchanged in the event of a type change. No check is made to see whether it is compatible with the new type. Type changes therefore require special measures to adapt the user data stored in the user realms (see “Adapting user data”).
  When an item is changed to a numeric type, you may need to take necessary alignments of the numeric type into account.
- Splitting an existing item of the type CHAR into multiple items
  The total length must be retained here. If the source item is used as a key item, it must be retained as such by using the new items. The special features regarding the modification of NCHAR and numeric types apply in accordance with the changes to the type of an individual item. Splitting an item into multiple items requires special measures to adapt the user data stored in the user realms (see “Adapting user data”).
- Grouping existing adjacent items to form an item of the type CHAR
  The total length must be retained here. Grouping is not possible if any of the individual items concerned is used as a key item. The grouping of multiple items requires special measures to adapt the user data stored in the user realms (see “Adapting user data”).
- Grouping a vector to form an item of the type CHAR
  The total length must be retained here. The grouping of a vector requires special measures to adapt the user data stored in the user realms (see “Adapting user data”).
- Converting an item of the type CHAR to a vector
  The total length must be retained here. The generation of a vector requires special measures to adapt the user data stored in the user realms (see “Adapting user data”).
- Grouping a repeating group to form one or more items of the type CHAR
  The total length must be retained here. The grouping of a repeating group requires special measures to adapt the user data stored in the user realms (see “Adapting user data”).
- Converting one item or multiple adjacent items of the type CHAR to a repeating group
  The total length must be retained here. The generation of a repeating group requires special measures to adapt the user data stored in the user realms (see “Adapting user data”).
- Using any existing free bytes in front of an implicitly aligned numeric item (FIXED BINARY) for new items on Level 0
  In repeating groups FIXED BINARY items in the stored records are not aligned. There are consequently no implicit free bytes in front of these items.

- Restructuring multiple existing items into multiple new items
  Two RENAME/BALTER cycles must be executed to permit restructuring. First the source items are grouped to form a CHAR item. In a second step this CHAR item can be split in the new structure. This procedure can be used above all for grouping or splitting NCHAR items.
- Renaming items of the type DBKEY or DBKEY-LONG
  Items of the type DBKEY or DBKEY-LONG which are used with LOCATION MODE DIRECT cannot be grouped with other items in a renaming cycle.
- Renaming record types
  Renaming must be performed at all places in DDL and SSL.
- Renaming sets
  Renaming must be performed at all places in DDL and SSL.
- Renaming realms
  Renaming must be performed at all places in DDL and SSL. An old realm name may not immediately be used in a renaming step as a new name for another realm. No file with the new file name of the user realm may exist.

7.2 Modifying the SSL

All name changes in the Schema DDL must be transferred to the Schema SSL.

Even if you can continue to use your existing SSL unchanged, you must recompile it as BGSIA otherwise uses the default values of the memory structure!

The following modifications to the memory structure are permitted:

- Changing item names in record types
  The names must be changed at all places (e.g. also in key definitions).
- Renaming items of the type DBKEY or DBKEY-LONG
  Items of the type DBKEY or DBKEY-LONG which are used with LOCATION MODE DIRECT cannot be grouped with other items in a renaming cycle.
- Renaming record types
  Renaming must be performed at all places in DDL and SSL.
- Renaming sets
  Renaming must be performed at all places in DDL and SSL.
- Renaming realms
  Renaming must be performed at all places in DDL and SSL. An old realm name may not immediately be used in a renaming step as a new name for another realm. No file with the new file name of the user realm may exist .

7.3 Recovery measures and response to errors

Renaming generally only changes the compiler database. So the After Image logging may remain activated.

Only when user realms are renamed do these also need to be saved as minor modifications are made in the user realms in the renaming cycle.

7.3.1 Saving the database

If an error occurs during the renaming the database has to be reset to the state before the renaming cycle was started. Therefore the following possibilities exist:

- Reading in a database backup and recovering the After Image Logging files up to the last consistency point before the renaming
- Use of a backup which was created directly before the renaming cycle. Therein the following files must be saved:
  - *dbname*.COSSD
  - *dbname*.DBDIR
  - *dbname*.DBCOM
  - user realms which are to be renamed.

For further information on saving a database, refer to the section "Saving and recovering a database in the event of errors" in the "Database Operation" manual.

7.3.2 Restoring the database

If a program aborts processing with “ABNORMAL END” during the renaming process, you must perform one of the following actions depending on the severity of the error and where it occurred in the renaming cycle:

- re-execute the terminated program, or
- fall back on the backup created and repeat the renaming process

When it is necessary to fall back on a backup of the database and to repeat the renaming process and when it is sufficient to repeat the aborted program is explained in the descriptions of the various programs.

The table below shows which programs modify files or realms of the database in the course of renaming:

	DBDIR	**DBCOM**	**DBCOM.O**	**COSSD**	**COSSD.O**	**User realms which have to be accessed**
BRENAME	RW	RW	W	R	W	-
DDL compiler	RW	RW	-	W	-	-
SSL compiler	RW	RW	-	W	-	-
BGSIA	RW	RW	-	-	-	-
BALTER (renaming phase)	RW	R	R	-	-	RW
DDL compiler (subschemas)	RW	RW	-	W	R	-
BGSSIA	RW	R	-	-	-	-

Table 47: Access to files and realms of the database during renaming

R read access

W write access

- no access

The following options are available for restoring the database:

- You can convert the shadow database to an original database by renaming it with the die MODIFY-FILE-ATTRIBUTES command.
- You can read in the ARCHIVE backup and then change the database name, if desired, with the MODIFY-FILE-ATTRIBUTES command. If the ARCHIVE backup was created on-line, you may have to mend it with the BMEND utility routine (see "BMEND" in the “Recovery, Information and Reorganization" manual).

For further information on restoring a database, refer to the section "Saving and recovering a database in the event of errors" in the "Database Operation" manual.

7.4 Initiating renaming using BRENAME

The task of BRENAME when renaming a database is comparable to that of BCREATE when creating a database: BRENAME prepares the compiler database to incorporate the new schema. Specifically, BRENAME performs the following preliminary work for renaming:

- It saves the old SIA in the DBDIR and prepares the DBDIR for incorporating a new SIA so that after the BGSIA run a new and an old SIA are present in the DBDIR for the schema. BALTER needs both SIAs when the structure data is adapted to the new schema to enable it to recognize the differences in the new schema compared to the old schema.
  Consequently you must ensure that enough free pages are available in the DBDIR or that automatic realm extension is possible by means of secondary allocation > 0.
- It deletes all user SSIAs in the DBDIR.
- It saves the old DBCOM in the file dbname.DBCOM.O and reformats the DBCOM.

  BALTER requires the schema information of the old and new DBCOM to check the planned renaming.
- It saves the old COSSD in the file dbname.COSSD.O.

  After renaming the DDL compiler requires the old COSSD to transfer the compatible subschemas. It is therefore advisable to keep the *dbname*.COSSD.O file available until all subschemas have been recompiled.

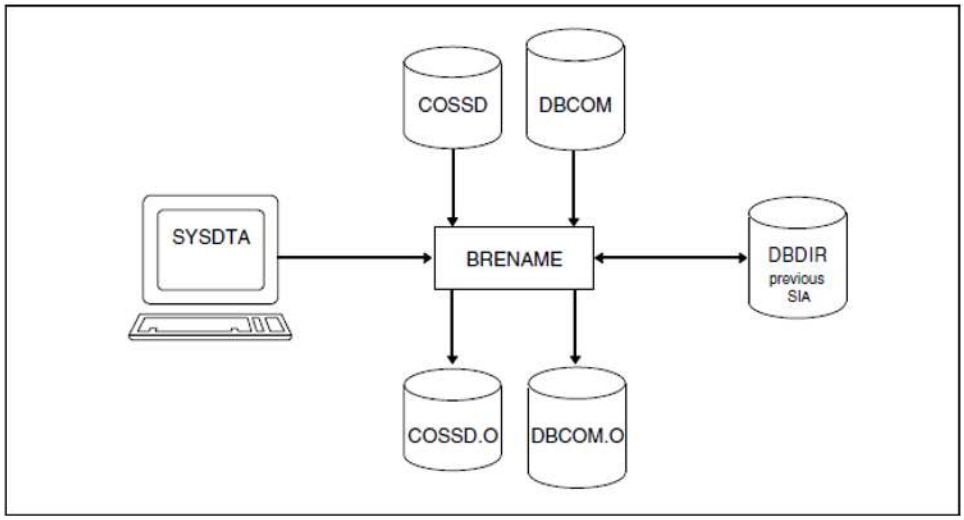


Figure 34: System environment when preparing the compiler database

BRENAME automatically stores the copies of DBCOM and COSSD on public disks. It is not necessary to issue a CREATE-FILE command to set up the two files (before BRENAME is started) unless the copies are to be stored on private disks.
Depending on the size of the files it is, however, advisable to set them up using a CREATE-FILE command with SPACE operand - even if they are to be stored on public disks (see "Maximum size of UDS/SQL files" in chapter "Files and realms of a UDS/SQL database").

When required, BRENAME automatically extends the realms of the database being processed. For details, please refer to the “Database Operation” manual, Automatic realm extension by means of utility routines.

At startup BRENAME takes into account any assigned UDS/SQL pubset declaration (see the “Database Operation” manual, Pubset declaration job variable). Faulty assignment leads to the program aborting.

Command sequence for starting BRENAME

It is assumed for the command sequence described here that UDS/SQL was installed with IMON (see section “START commands for the UDS/SQL programs”).

The BRENAME utility routine is started by the following commands in the identification under which the database is cataloged:

```
01  [/CREATE-FILE FILE-NAME=dbname.DBCOM.0 ...]
02  [/CREATE-FILE FILE-NAME=dbname.COSSD.0 ...]
03  /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR
04  /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,
SCOPE=*TASK
05  /START-UDS-BRENAME
06  END
```

01,02 See section “Setting up the compiler database”.

04 The specified version of BRENAME is selected.
It is generally recommended that you specify the version since it is possible for several UDS/SQL versions to be installed in parallel.

05 The UDS/SQL utility routine can also be started with the alias BRENAME.

06 BRENAME is terminated.

¡ The END statement is the only BRENAME statement.

7.5 Compiling the Schema DDL

If the compiler database has been prepared to accept a new schema with the aid of the BRENAME utility routine, the next thing you must do is to compile your Schema DDL with the new names using the DDL compiler.

The compilation procedure is the same as that used for database creation.

Once the Schema DDL has been compiled, the following are available:

- an old and a new DBCOM
- an old SIA in the DBDIR
- an old and a new COSSD.

Command sequence for compiling the current Schema DDL

It is assumed for the command sequence described here that UDS/SQL was installed with IMON (see section “START commands for the UDS/SQL programs”).

The commands listed here are described in detail in section “Compiling the Schema DDL”.

```
01  /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR
02  /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,
SCOPE=*TASK
03  /START-UDS-DDL
04  SOURCE IS 'schema-file'
05  END
06  /ASSIGN-SYSDTA TO=*SYSCMD
```

> It is essential that the DDL compiler should terminate compilation with the message ’NORMAL END’.
>
> If the message ’ABNORMAL END’ is received, compilation must be repeated with corrected DDL clauses.

7.6 Compiling the SSL

The option is available to compile a new SSL using the SSL compiler once the Schema DDL has been compiled.

If no SSL compilation is carried out, default values for the storage structure are used. If you want to retain the storage structure which has already been defined, you must recompile your original SSL clauses with new names which match the DDL source!

The compilation procedure is the same as that used for database creation.

Command sequence for compiling the SSL

It is assumed for the command sequence described here that UDS/SQL was installed with IMON (see section “START commands for the UDS/SQL programs”. The aliases for calling the utility routines are also listed in this section).
The commands listed here are described in detail in section “Compiling the SSL”.

```
01  /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR
02  /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,SCOPE=*TASK
03  /START-UDS-SSL
04  SOURCE IS 'ssl-file'
05  END
06  /ASSIGN-SYSDTA TO=*SYSCMD
```

It is essential that the SSL compiler should terminate compilation with ’NORMAL END’. If compilation ends with ’ABNORMAL END’, the following action should be taken:

- for errors in the SSL clauses:
  the faulty SSL clauses should be corrected and the SSL compilation should be repeated;
- for errors in the DDL clauses:
  - the faulty DDL clauses should be corrected
  - the faulty schema should be deleted in a DDL run by means of the statement
    DELETE SCHEMA *schemaname*
  - the renaming process should be repeated from “Compiling the Schema DDL”.

7.7 Generating a new SIA and entering it in the DBDIR with BGSIA

Once the Schema DDL and the SSL (optional) have been successfully compiled, the SIA of the new schema must be generated and entered in the DBDIR using the BGSIA utility routine.

The saved SIA of the old schema remains in the DBDIR so that, after the BGSIA run, DBDIR contains the SIAs of both the old and the new schemas. BALTER requires both to check and execute the planned renaming.

The BGSIA run corresponds to the run carried out for the creation of the database (see section “Setting up the Schema Information Area (SIA) with BGSIA”). After the BGSIA run, the module UDSHASH generated by BGSIA must be stored in the HASHLIB.

To ensure that BGSIA recognizes realms, record types and sets which remain the same in the renaming cycle despite the name change and leaves the existing references unchanged, name changes for realms, record types and sets must be specified in full in additional statements. The names in the DDL/SSL source are affected by this. The additional statements of BGSIA do not change any names in these sources.

Dynamic sets can be renamed in precisely the same way. The names of the dynamic sets required for using IQS are predefined and may not be changed. However, this is not checked. Implicit sets do not need to be specified explicitly. They are renamed automatically in accordance with the renamed record type (SYS_recordname) as soon as a search key is defined.

If you work with your own hash routines, you must also store these in the HASHLIB with the attributes RMODE=ANY and AMODE=ANY at the latest before BALTER is started.

If, when the SIA is generated, it is recognized that not just purely renaming is involved, a message is issued that the renaming cycle is being aborted because the references do not match. This message can be used for correcting the source. A precise analysis only takes place with BALTER. Illegal type changes and illegal splitting or grouping of items are examples of changes which conflict with pure renaming.

Generating SIA and entering it in DBDIR

```
01  /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR
02  /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,SCOPE=*TASK
03  /DELETE-SYSTEM-FILE FILE-NAME=*OMF
04  /START-UDS-BGSIA
05  GENERATE SCHEMA schema-name
06  [DISPLAY[ SCHEMA schema-name]]
07  END
```

Entering the module UDSHASH in the HASHLIB

```
01 /START-LMS
02 //OPEN-LIB LIB=dbname.HASHLIB,MODE=*UPDATE
03 //ADD-ELEMENT FROM-FILE=*OMF,TO-ELEMENT=*LIBRARY-ELEMENT(TYPE=R)
04 //END
```

7.8 Checking renaming and updating structure information using BALTER

In the renaming cycle the BALTER utility routine checks solely whether the planned modifications are really only restricted to renaming.

The only BALTER statement permitted (apart from the END statement) in the renaming cycle is the REPORT statement. REPORT IS YES is set by default. All other BALTER statements are rejected and result in the BALTER run being aborted.

If REPORT IS YES is set, after the analysis phase BALTER starts the REPORT phase in which it outputs the analysis report to SYSLST (see section “Description of the analysis report (REPORT phase)”). In the analysis report BALTER lists all the changes which are made to record types, sets or keys during the renaming phase. The analysis report also contains the analysis record concerning error messages and warnings relating to illegal changes (see section “Description of BALTER messages”).

You should only use REPORT IS NO when it is certain that renaming can be performed successfully. If this is not the case, it is more difficult to diagnose errors in the schema.

You can only run BALTER successfully if you have already generated the new SIA and stored it in the DBDIR using BGSIA (see section “Generating a new SIA and entering it in the DBDIR with BGSIA”). Otherwise the BALTER run terminates with the message “BGSIA HAS NOT BEEN EXECUTED”.

BALTER determines the differences between the old and the new schema descriptions using the old and the new DBCOM or the old and the new SIA and checks whether only renaming has taken place. If restructuring of the user data would be necessary because database records (e.g. Set Connection Data) and tables (address lists, lists, SEARCH key tables, etc.) are to be modified, the renaming cycle is aborted with an error message. If required, you must then fall back on the status of the database before the renaming cycle.

Changes which do not involve modifications to database records or tables but which mean that existing data cannot necessarily continue to be used without adaption are permitted but a corresponding warning is issued. Such changes could be type changes or the grouping and splitting of items (see also section “Adapting user data”).

7.8.1 Command sequence for starting BALTER

The command sequence described here requires that UDS/SQL was installed using IMON (see section “START commands for the UDS/SQL programs”).

You can start BALTER using the command below under the ID in which the database is cataloged (you can also start the program using the alias BALTER):

```
01  /START-UDS-BALTER
02  REPORT IS YES.
03  END.
```

7.8.2 Description of the BALTER check

BALTER checks the renaming in a predefined sequence.

BALTER begins by checking fundamental structures of the schemas ("SIA_CONTROL"). These include the internal numbers of realms, record types, sets and keys in the old and the new schemas. It is not permissible that these numbers have changed.

The area declarations are then checked. Only renaming of user realms is permitted.

The declarations of the record types are subsequently checked. The renaming of items, grouping of items and the splitting of items are checked here. A check is also made to see whether the declarations of the CALC key match.

Finally the set declarations and consequently also the logically associated key declarations are checked to see whether they match. Only sets may be renamed here.

The BALTER run finishes with any renaming of realms being implemented by recataloging the realms concerned. This is the only case in which user realms need to be opened and modified in a renaming cycle.

All renaming actions and actions for splitting and grouping items are documented in REPORT IS YES. The number of realms, record types (records) and sets which are unchanged is also output.

Modifications in the schema which conflict with pure renaming lead to error messages and to the recycling cycle being aborted. An error can result in multiple error messages as it can lead to related modifications at several places in the SIA. The analysis is not terminated after the first error which is detected but only after multiple errors have occurred.

7.9 Illegal schema modifications in the renaming cycle

To enable violations of the renaming rules to be corrected quickly, a current version of the old schema should be created using the BPSIA utility routine before the renaming cycle begins. In the renaming cycle you should also use the DISPLAY statement in the BGSIA utility routine. In some cases BGSIA already recognizes that contradictory declarations are contained in the new schema when renaming takes place.

In the renaming cycle the BALTER output concerning illegal differences between the old and new schemas largely relates to the corresponding information in the output of BPSIA and BGSIA. BALTER also outputs the differences in individual items in record types.

> ¡ When corrections are made, first of all the information relating to the record types should be observed because messages about illegal modifications in SIA_CONTROL, the sets and keys are very often due to illegal declarations in items of the record types.

Illegal schema modifications are checked for in five phases.

In phase one BALTER checks SIA-CONTROL. Modifications in SIA-CONTROL always indicate illegal modifications. The following illegal modifications can occur (message text: DIFFERENCE IN sia-content):

Message (sia-content)	**Meaning**
NR_AREAS	The highest realm number assigned (area reference) is different in the old and new schemas.
NR_RECORDS	The highest record type number assigned (REC-REF) is different in the old and new schemas.
NR_SETS	The highest set number assigned (SET-REF) is different in the old and new schemas.
NR_KEYS	The highest key number assigned (KEY-REF) is different in the old and new schemas.
SCHEMA_NAME	The old and new schema names are not the same.
IMPL_RESULT_SET	There are differences in the IMPLICIT_RESULT_SET of the schema.
SINGULAR_SET	There are differences in the first SYSTEM set of the schema.
DYNAMIC_SET	There are differences in the first dynamic set of the schema.
MAX_REC_LENGTH	The length of the largest record type is different in the old and new schemas.
MAX_ENTRY_LENGTH	The length of the longest key is different in the old and new schemas.
MAX_MEMBERSHIPS	The largest number of sets in which Member is a record type is different in the old and new schemas.
MAX_SPLIT_PARAMETER	The maximum number of pages in the SSL which is defined in the REORGANIZATION parameter is different.
LENGTH_KEY_BIT	The check information for the MODIFY information is different in the old and new schemas.

BLOCK_LENGTH	The length of the database pages is different in the old and new schemas.

In phase two the individual areas are checked for illegal modifications. The following illegal modifications can occur (message text: DIFFERENCE IN area-content):

Message (area-content)	**Meaning**
AREA_REF	The reference of the realm is different in the old and new schemas.
AREA_PROPERTIES	The central properties of the realm are different in the old and new schemas (TEMP, D/T).
NR_WITHIN_RECORDS	The number of record types which can be stored in the realm is different in the old and new schemas.
RECORD_LIST	In the old and new schemas there are differences between the references of the record types which can be stored in the realm.

The DIFFERENT USE OF AREA-REF message indicates that the assignment of this realm reference is so different in the new and old schemas that a more detailed differentiated analysis does not make sense.

The AREA RENAMING WITH ALOG message indicates that the renaming of areas is allowed only if After Image Logging is deactivated. If the database is operated with After Image Logging you must deactivate it before the renaming cycle is started.

In phase three the

- various record types are checked for illegal modifications.
- keys at record type level are checked for changes to the SIA data.
- key information which is only later stored in the SSIA is checked.
- renaming of items is checked.

The following illegal modifications can occur when the various record types are modified (message text: DIFFERENCE IN record-content):

Message (record-content)	**Meaning**
REC_REF	The reference of the record type is different in the old and new schemas.
LOCATION_MODE	There are differences with regard to the location of the DBKEY or DBKEY-LONG item which is used for LOCATION MODE DIRECT.
REC_PROPERTIES	The central properties of the record type are different in the old and new schemas.
IMPLICIT_SET	There are differences in the implicit set of the record type.
OWNER_CHAIN	There are differences in the first set in which the record type is Owner.
MEMBER_CHAIN	There are differences in the first set in which the record type is Member.

RECORD_LENGTH	The length of the record is different in the old and new schemas.

SYSTEM_INFO	The length of the set connection data is different in the old and new schemas.
PHYSICAL_CALC_INFO	The location of the CALC buckets is different in the old and new schemas.
DBTT_ENTRY_LENGTH	The length of the DBTT entries is different in the old and new schemas.
LOCATION_VIA	The type of location mode is different in the old and new schemas.

The DIFFERENT USE OF REC-REF message indicates that the assignment of this record reference is so different in the new and old schemas that a more detailed differentiated analysis does not make sense.

The following illegal modifications in the SIA data can occur when a key in a set is modified:

Message (record-content)	**Meaning**
KEY_LENGTH	The key length is different in the old and new schemas.
HASH	The reference to the hash routine is different in the old and new schemas.
DBTT_COLUMN	The DBTT entry corresponding to the key is different in the old and new schemas.
KEY_PROPERTIES	The properties of the key are different in the old and new schemas.

The following illegal modifications can occur in the key information which is only later entered in the SSIA (message text: DIFFERENCE IN key-content):

Message (key-content)	**Meaning**
KEY_LTH	The key length is different in the old and new schemas.
DBTT_COL_NR	The DBTT entry corresponding to the key is different in the old and new schemas.
INDEX_AREA_REF	The reference of the realm is different in the old and new schemas.
CALC_PROCEDURE_NAME	The name of the CALC routine is different in the old and new schemas.
CALC_PROCEDURE_NR	The number of the CALC routine is different in the old and new schemas.
KEY_BITS	The properties of the key are different in the old and new schemas.
KEY_REF_NR	The (internal) reference number is different in the old and new schemas.
NR_BUCKETS	The size of the CALC area is different in the old and new schemas.
USER_KEY_TYPE	The item type of the key is different in the old and new schemas.
KEY_INDICATOR	The properties of the key are different in the old and new schemas.
KEY_ITEM_DISPL	A key item has a different displacement in the old and new schemas.
KEY_ITEM_CONCAT	The grouping of key items in the new schema does not match the key items in the old schema.

KEY_ITEM_SPLIT	The splitting of key items in the new schema does not match the item fields in the old schema.

The DIFFERENT USE OF KEY-REF message indicates that the assignment of this key reference is so different in the new and old schemas that a more detailed differentiated analysis does not make sense.

The following illegal modifications can occur when items are renamed:

Message	Meaning
ITEM BOUNDARIES UNSUITABLE	Grouping or splitting items leads to overlapping.
CONCATENATION TO NON CHAR TYPE	Items were grouped but the grouped item in the new schema is not of the type CHAR.
SPLIT OF NON CHAR TYPE	An item is split but the item in the old schema is not of the type CHAR.
FORM GROUP FROM NON CHAR TYPE	Items were combined to form a group but not all items in the new schema are of the type CHAR.
SPLIT GROUP TO NON CHAR TYPE	A group was split into one or more items but not all items in the new schema are of the type CHAR.
LENGTH DIFFERENCE OF TYPE VARIABLE	The maximum length of a variable item at the end of the record type was modified illegally.
LENGTH DIFFERENCE OF LAST OLD ITEM	The length of the last item in the old schema indicates illegal renaming.
LENGTH DIFFERENCE OF LAST NEW ITEM	The length of the last item in the new schema indicates illegal renaming.
DIFFERENCE IN COMPRESSED RECORD	Illegal modifications have been made in a compressed record type.
DIFFERENCE IN ITEM TYPE	Indicates an illegal type change in the item
TYPE CHANGE OF NON CHAR TYPE	An illegal type change was detected.
VECTOR CHANGE OF NON CHAR TYPE	An illegal type change was detected when a vector was changed.

In phase four the

- various sets are checked for illegal modifications.
- keys at set level are checked for modifications in the SIA data. The same illegal modifications can occur as in phase three (see "Illegal schema modifications in the renaming cycle").

> As the keys have no explicit names, corresponding internal numbers are output for various keys or an unambiguous name is output for the key type in the case of record types and sets. Differences in the keys for record types or sets result from illegal modifications to the related data items from which the keys are formed. In order to eradicate these differences, the renaming errors must be corrected.

- key information which is only later stored in the SSIA is checked. The same keys with the corresponding number are output as in the schema's BPSIA report. The same illegal modifications can occur as in phase three (see "Illegal schema modifications in the renaming cycle").

The following illegal modifications can occur when the various sets are modified (message text: DIFFERENCE IN set-content):

Message (set-content)	**Meaning**
SET_REF	The reference of the set is different in the old and new schemas.
SET_PROPERTIES	The properties of the set are different in the old and new schemas.
SET_MODE	The set mode is different in the old and new schemas.
SET_ORDER	The sort sequence is different in the old and new schemas.
OWNER_OF_SET	The owner reference is different in the old and new schemas.
MEMBER_OF_SET	The member reference is different in the old and new schemas.
OWNER_CHAIN	Owner chaining is different in the old and new schemas.
MEMBER_CHAIN	Member chaining is different in the old and new schemas.
SINGULAR_SET_CHAIN	Chaining of the SYSTEM set is different in the old and new schemas.
DYNAMIC_SET_CHAIN	Chaining of the dynamic sets is different in the old and new schemas.
KEY_CHAIN	Chaining of the set is different in the old and new schemas.
SET_MEMBERSHIP	The POPULATION / INCREASE clause is different in the old and new schemas.
SET_CONNECTION	There are differences in the set connection data.
ANCHOR_DBKEY	The database key of the anchor of a singular set is different in the old and new schemas.

The DIFFERENT USE OF SET-REF message indicates that the assignment of this set reference is so different in the new and old schemas that a more detailed differentiated analysis does not make sense.

In the fifth phase the physical order of all keys of the SIA which are connected to the record types or sets is checked. The physical order of the keys must be unchanged as they are assigned an implicit numbering which the database uses. Illegal modifications can occur again here (see "Illegal schema modifications in the renaming cycle").

7.10 Adapting subschemas

When it prepares the compiler database for renaming, one of BRENAME's tasks is to delete all SSIAs in the DBDIR and all subschema information in the DBCOM; the DDL compiler then readies the COSSD to accept new subschema information when the new Schema DDL is compiled. Consequently all the old subschema information is deleted after the renaming phase; no subschema information has as yet been entered in the new COSSD.

Therefore, after BALTER has renamed your database, all subschemas must be recompiled and a new SSIA must be generated for each and entered in the DBDIR.

7.10.1 Copying compatible subschemas

Often not all subschemas are affected by modifications to the schema. Consequently BRENAME copies the COSSD into the COSSD.O file when renaming begins, as a result of which all old subschema information is retained despite the renaming activity. If you want to copy these old subschemas, you must execute a DDL compiler run to copy the old subschemas after BALTER has successfully completed the renaming phase.

During this run for copying the subschemas, the DDL compiler reads all the old subschemas from the file COSSD.O, recompiles them and then checks them for compatibility with the new schema. It differentiates between three possible results:

- the old subschema description is incompatible with the new schema
- the old subschema is incompatible with the new schema because of logical and/or physical changes in the schema, i.e. the execution of DML statements is affected
- the old subschema is unaffected by changes in the new schema.

In the first two cases the DDL compiler does not store subschema information in either the DBCOM or the COSSD. Only in the third case, when a subschema is not affected by schema changes, does the computer copy the subschema from the COSSD.O, recompile it and enter the subschema information in the new DBCOM and in the new COSSD. For every subschema copied a new SSIA must be generated using the BGSSIA utility routine and entered in the DBDIR.

Please note that "compatibility" only means that the old subschema’s view of the new schema has remained the same as its view of the old schema. It does not mean, for example, that when the "COPY [ALL] RECORD[S]" clause is used the view of the (upward-compatible) changes in the new schema is retained in the new schema. If you want to do this, you must recompile the subschema.

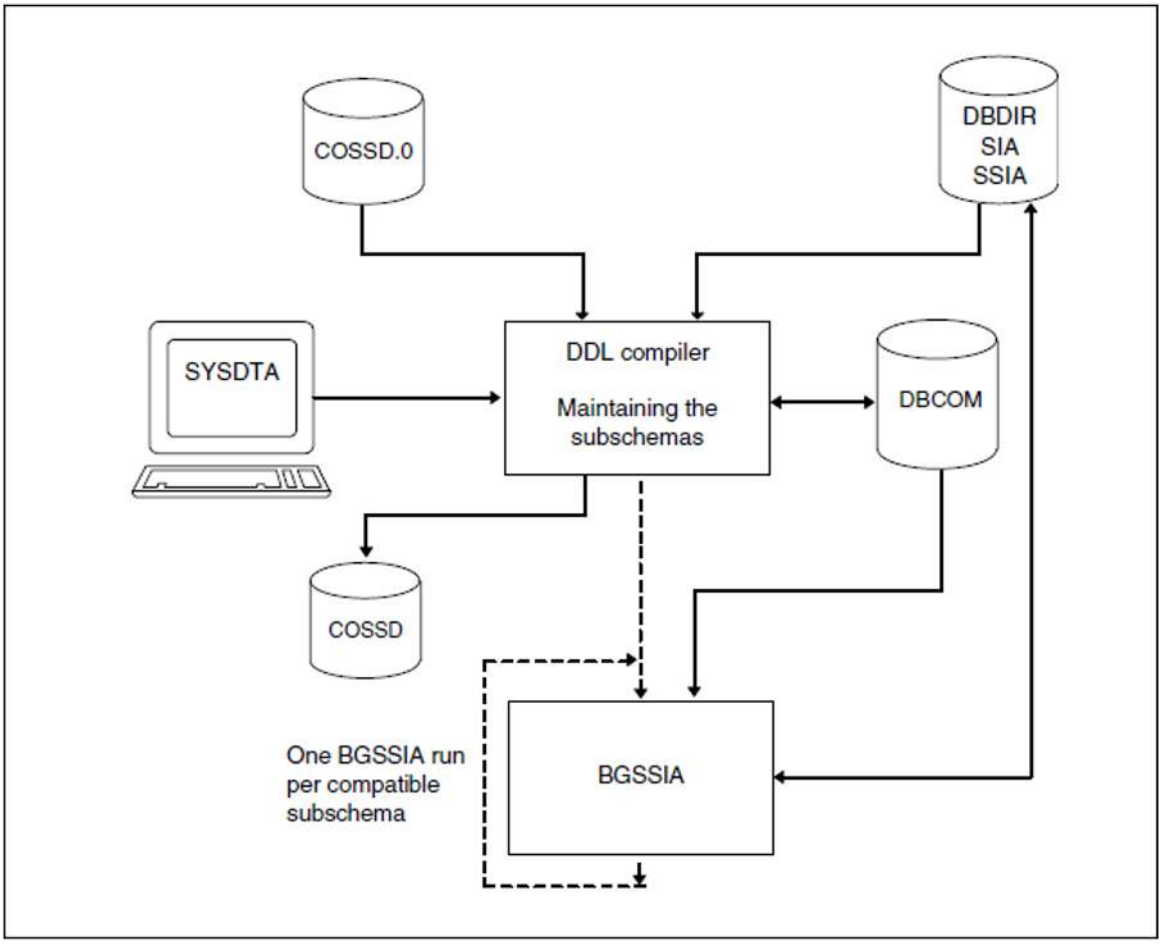


Figure 35: System environment for copying subschemas

The compiler run for copying compatible subschemas is optional; if omitted, all subschemas must be recompiled individually, and the corresponding SSIAs must be regenerated and entered in the DBDIR.

Subschema compatibility and incompatibility

schema-name and PRIVACY LOCK FOR COPY.....

A change of schema name and of PRIVACY LOCK specifications has no effect on copying the subschemas.
Such changes need only be taken into account when subsequent subschema compilations are carried out.

PRIVACY LOCK FOR COMPILE

In the compiler run for copying the subschemas, the DDL compiler copies these PRIVACY specifications from the **old** subschema description so that access locks for the compilation of application programs are retained.

identifier

An old subschema is incompatible with the new schema if an *identifier* has been added, deleted or renamed in the LOCATION MODE clause, the WITHIN clause (record type level) or the SET OCCURRENCE SELECTION clause, and the corresponding record type or set is present in the subschema.

Statements for copying subschemas

The DDL compiler requires the following statements to copy the subschemas:

Statement	Default value	Meaning
COMPARE SUBSCHEMAS	-	Initiates copying of subschemas
[SORCLIST IS {YES | NO}]	YES	Prints out subschema listing
[DIAGNOSTIC {YES | NO}]	NO	Diagnoses incompatibilities of old subschemas with the new schema and lists them in the form of error messages
END	-	Terminates entry of the statements

Table 48: Statements for copying subschemas

Command sequence for copying subschemas

The following commands initiate a DDL compiler run for copying subschemas (see section “Compiling the Schema DDL”):

```
01  /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR
02  /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,SCOPE=*TASK
03  /START-UDS-DDL
04  COMPARE SUBSCHEMAS
05  [DIAGNOSTIC {YES | NO}]
06  [SORCLIST IS {YES | NO}]
07  END
```

An SSIA must then be generated for each subschema copied and entered in the DBDIR (see section “Generating the Subschema Information Area (SSIA) with BGSSIA”) using the following commands:

```
01  /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR
02  /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,SCOPE=*TASK
03  /START-UDS-BGSSIA
04  GENERATE SUBSCHEMA subschema-name OF SCHEMA schema-name
05  [DISPLAY[ SUBSCHEMA subschema-name OF SCHEMA schema-name]]
06  END
```

7.10.2 Adapting incompatible subschemas

For all subschemas which, in their original form, are not compatible with the new schema, it is necessary to do the following:

- adapt the source with regard to renaming
- correct the subschema description if required
- recompile the corrected subschema with the DDL compiler
- generate a new SSIA using BGSSIA and enter it in the DBDIR
- recompile and link any application programs

It is assumed for the command sequence described here that UDS/SQL was installed with IMON (see section “START commands for the UDS/SQL programs”. The aliases for calling the utility routines are also listed in this section).

Command sequence for adapting the subschemas

Compiling the corrected subschema (see section “Compiling the Schema DDL”)

```
01  /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR
02  /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,SCOPE=*TASK
03  /START-UDS-DDL
04  [DELETE 'subschema-name':'new schema-name']
05  SOURCE IS 'subschema-file'
06  SORCLIST IS YES
07  END
08  /ASSIGN-SYSDTA TO=*SYSCMD
```

02 The version-independent module of the linked-in DBH of the relevant version is loaded dynamically (see the section entitled "Compiling, linking and loading UDS/SQL-TIAM application programs" in the "Application Programming" manual).

03 The UDS/SQL utility routine can also be started using the alias DDL.

04 The DELETE statement should be specified only if a subschema recognized as compatible and copied by the DDL compiler has been modified and requires recompilation.

Generating the SSIA and entering it in the DBDIR

See section “Generating the Subschema Information Area (SSIA) with BGSSIA”.

```
01  /ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=dbname.DBDIR
02  /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,SCOPE=*TASK
03  /START-UDS-BGSSIA
04  GENERATE SUBSCHEMA subschema-name OF SCHEMA schema-name
05  [DISPLAY[ SUBSCHEMA subschema-name OF SCHEMA schema-name]]
06  END
```

7.11 Adapting DB applications

It is not necessary to adapt the DB applications to new names immediately if purely renaming without type changes or the splitting or grouping of items is involved. In these cases adaptations are only required the next time the application program is compiled.

Applications which are not affected by the renaming do not need to be modified, for example if the subschema does not contain the renamed parts.

SSITAB modules which are not affected by renaming do not need to be created using BCALLSI.

When item types are changed or when items are split or grouped, all the applications affected must be recompiled on the basis of the new subschemas.

7.12 Updating access rights

The renaming process changes nothing in the access rights which you entered using the BPRIVACY utility routine. However, because record types, sets and user realms have been renamed, input files for assigning access rights may no longer be up-to-date. Following renaming you should therefore using BPRIVACY or ONLINE-PRIVACY and the SHOW-USER-GROUP statement generate a display of the current rights and adapt the input files as required.

7.13 Adapting user data

In the renaming cycle only the schema and subschema data is modified in the event of type changes or when items are split or grouped. The actual user data remains unchanged. If initialization or other input is required for the modified data items, this must be implemented independently of the renaming cycle. You must execute this before or after the renaming cycle parallel to normal operation, e.g. by means of suitable application programs.

7.14 Measures for restarting DB operation

If the After Image Logging was deactivated before the renaming cycle was started, it will result in a logging gap. After the renaming cycle, the After Image Logging can be activated again with the BMEND utility (see the "Recovery, Information and Reorganization" manual, BMEND). Then a backup of the database has to be created again (see the "Database Operation" manual, Saving and recovering a database in the event of errors)

You then can delete the DBCOM.O and COSSD.O files.

7.15 Example

BRENAME run and compiling new Schema DDL and SSL

A reserve item in the record type Customer is split into a new Unicode item for the copy of the address and the remaining reserve item.

/START-UDS-BRENAME

```
***** START        BRENAME      (UDS/SQL V2.9  1801 )     2019-01-29   09:35:50


***** THE FILE: :IUDS:$XXXXXXXX.INSURE.DBCOM IS COPIED TO:
      :IUDS:$XXXXXXXX.INSURE.DBCOM.O


***** THE FILE: :IUDS:$XXXXXXXX.INSURE.COSSD IS COPIED TO:
      :IUDS:$XXXXXXXX.INSURE.COSSD.O
1005 RENAMING SUCCESSFULLY INITIATED

***** DIAGNOSTIC SUMMARY OF BRENAME

               NO WARNINGS
               NO ERRORS
               NO SYSTEM-ERRORS

***** END OF DIAGNOSTIC SUMMARY
***** NR OF DATABASE ACCESSES  :          90
***** NORMAL END   BRENAME      (UDS/SQL V2.9  1801 )     2019-01-29   09:35:51
```

/START-UDS-DDL

```
***** START        DDLCOMP      (UDS/SQL V2.9  1801 )     2019-01-29   09:35:51
*  DDLCOMP: INPUT SYSTEMPARAMETERS
```

SOURCE IS 'S.INSURE.DDL.RENAME'
END

```
*  DDLCOMP: READ SCHEMA/SUBSCHEMA
%  UDS0215 UDS STARTING UDS/SQL V2.9 (LINKED-IN), DATE=2019-01-29 (ILL2038,09:35:51/4TE7)
%  UDS0746 UDS PUBSET DECLARATION (CURRENT) FOLLOWS (ILL1746,09:35:51/4TE7)
4TE7: UDS-PUBSET-JV:  :IUDS:$XXXXXXXX.PUBSDECL.DEFAULT
4TE7: DEFAULT PUBSET: IUDS
4TE7: ----------------------------------------
*  DDLCOMP: START SCHEMA-PHASE
*  DDLCOMP: CHECK SCHEMA RULES
*  DDLCOMP: CHECK DATA ALLOCATION
*  DDLCOMP: SEMANTIC TEST
*  DDLCOMP: CYCLUS TESTS
*  DDLCOMP: ERROR DIAGNOSTIC
*  DDLCOMP: NO ERRORS IN SCHEMA-PHASE
*  DDLCOMP: CREATE FILE COSSD
*  DDLCOMP: NO ERRORS DETECTED
%  UDS0758 NUMBER OF DML-STATEMENTS AND I/O COUNTERS PER DATABASE  (ILL1758,09:35:51/4TE7)
4TE7: DATABASE NAME            DMLS   LOG READ  PHYS READ  LOG WRITE PHYS WRITE
4TE7: ------------------------------------------------------------------------
4TE7: INSURE                    658       2026         67        926         39
%  UDS0213 UDS NORMAL SYSTEM TERMINATION WITH **********658 DML-STATEMENTS 2019-01-29
(ILLY033,09:35:51/4TE7)

***** DIAGNOSTIC SUMMARY FOR DDL-SCHEMA CUSTOMER-CARDS

               NO ERRORS
+++++           9 WARNINGS

***** END OF DIAGNOSTIC SUMMARY
***** NORMAL END   DDLCOMP      (UDS/SQL V2.9  1801 )     2019-01-29   09:35:51
```

/START-UDS-SSL

```
***** START        SSLCOMP      (UDS/SQL V2.9  1801 )     2019-01-29   09:35:52
*  SSLCOMP: INPUT SYSTEMPARAMETERS
```

SOURCE IS 'S.INSURE.SSL.NEW'
END

```
*  SSLCOMP: READ SSL-SCHEMA
%  UDS0215 UDS STARTING UDS/SQL V2.9 (LINKED-IN), DATE=2019-01-29 (ILL2038,09:35:52/4TE7)
%  UDS0746 UDS PUBSET DECLARATION (CURRENT) FOLLOWS (ILL1746,09:35:52/4TE7)
4TE7: UDS-PUBSET-JV:  :IUDS:$XXXXXXXX.PUBSDECL.DEFAULT
4TE7: DEFAULT PUBSET: IUDS
4TE7: ----------------------------------------
*  SSLCOMP: START SSL-PHASE
*  SSLCOMP: CHECK SSL RULES
*  SSLCOMP: SEMANTIC TEST
*  SSLCOMP: ERROR DIAGNOSTIC
*  SSLCOMP: NO ERRORS DETECTED
%  UDS0758 NUMBER OF DML-STATEMENTS AND I/O COUNTERS PER DATABASE  (ILL1758,09:35:52/4TE7)
4TE7: DATABASE NAME           DMLS   LOG READ  PHYS READ  LOG WRITE PHYS WRITE
4TE7: ------------------------------------------------------------------------
4TE7: INSURE                   127        251         60         34         23
%  UDS0213 UDS NORMAL SYSTEM TERMINATION WITH *************127 DML-STATEMENTS 2019-01-29
(ILLY033,09:35:52/4TE7)
***** DIAGNOSTIC SUMMARY FOR SSL - SCHEMA

               NO ERRORS
               NO WARNINGS

***** END OF DIAGNOSTIC SUMMARY
***** NORMAL END   SSLCOMP      (UDS/SQL V2.9  1801 )     2019-01-29   09:35:52
```

/START-UDS-BGSIA

```
***** START        BGSIA        (UDS/SQL V2.9  1801 )    2019-01-29   09:35:52
```

GENERATE SCHEMA CUSTOMER-CARDS
DISPLAY
END

```
%  UDS0215 UDS STARTING UDS/SQL V2.9 (LINKED-IN), DATE=2019-01-29 (ILL2038,09:35:52/4TE7)
%  UDS0746 UDS PUBSET DECLARATION (CURRENT) FOLLOWS (ILL1746,09:35:52/4TE7)
4TE7: UDS-PUBSET-JV:  :IUDS:$XXXXXXXX.PUBSDECL.DEFAULT
4TE7: DEFAULT PUBSET: IUDS
4TE7: ----------------------------------------
ESTIMATE-REPORT

********** FOR USER-REALM     3 NAME IS : PROP-RLM
      A SIZE OF         24 BLOCKS WAS ESTIMATED

********** FOR USER-REALM     4 NAME IS : INSURE-RLM
      A SIZE OF        239 BLOCKS WAS ESTIMATED

********** FOR USER-REALM     6 NAME IS : TRANSPORT-RLM
      A SIZE OF         24 BLOCKS WAS ESTIMATED
END OF ESTIMATE-REPORT
%  UDS0758 NUMBER OF DML-STATEMENTS AND I/O COUNTERS PER DATABASE (ILL1758,09:35:52/4TE7)
4TE7: DATABASE NAME            DMLS   LOG READ  PHYS READ  LOG WRITE PHYS WRITE
4TE7: ------------------------------------------------------------------------
4TE7: INSURE                    569        778         60        183         29
%  UDS0213 UDS NORMAL SYSTEM TERMINATION WITH *************569 DML-STATEMENTS 2019-01-29
(ILLY033,09:35:53/4TE7)

***** DIAGNOSTIC SUMMARY OF BGSIA

               NO WARNINGS
               NO ERRORS
               NO SYSTEM-ERRORS

***** END OF DIAGNOSTIC SUMMARY
***** NORMAL END   BGSIA        (UDS/SQL V2.9  1801 )     2019-01-29   09:35:53
```

/MODIFY-JOB-SWITCHES ON=(1,4)
/START-LMS
//MODIFY-LOGGING-PARAMETERS LOG=*MAX
//OPEN-LIBRARY LIB=INSURE.HASHLIB,MODE=*UPDATE
//ADD-ELEMENT FROM-FILE=*OMF,TO-ELEM=*LIB-ELEM(TYPE=R),WRITE-MODE=*ANY

```
INPUT OMF
OUTPUT LIBRARY= :IUDS:$XXXXXXXX.INSURE.HASHLIB
         ADD UDSHASH AS (R)UDSHASH/@(0003)/2019-01-29 , OUTPUT REPLACED
```

//SHOW-ELEM-ATTR

```
INPUT LIBRARY= :IUDS:$XXXXXXXX.INSURE.HASHLIB
TYP NAME    VER (VAR#) DATE          NAME    VER (VAR#) DATE
(R) ADMIN## @   (0001) 2019-01-29    UDSHASH @   (0003) 2019-01-29
       2 (R)-ELEMENT(S) IN THIS TABLE OF CONTENTS
```

//END
/MODIFY-JOB-SWITCHES OFF=(1)

Checking the renaming using BALTER

/START-UDS-BALTER

```
***** START        BALTER       (UDS/SQL V2.9  1801 )     2019-01-29   09:35:53
```

REPORT   IS YES .
END.

```
+++++ WARNING: 1091     1 CHANGE(S) OF ITEM TYPE
+++++ WARNING: 1096     1 CHANGE(S) WITH SPLIT OF ITEMS

NUMBER OF FILE ACCESSES:            0


***** DIAGNOSTIC SUMMARY OF BALTER

+++++           2 WARNINGS
               NO ERRORS
               NO SYSTEM-ERRORS

***** END OF DIAGNOSTIC SUMMARY
***** NR OF DATABASE ACCESSES  :         119
***** NORMAL END   BALTER       (UDS/SQL V2.9  1801 )     2019-01-29   09:35:53
```

List output of BALTER:

```
RENAME CHECK    FOR SCHEMA                    CUSTOMER-CARDS
RENAME CHECK    SIA_CONTROL
RENAME CHECK    AREA
NO CHANGES IN   ALL                        4 AREA(S)
RENAME CHECK    RECORD
|-------------------------------------------------------------------------------------------
|
| DIFFERENCE IN         RECORD              CUSTOMER
|
|-------------------------------------------------------------------------------------------
|
|
|
|   +++ WARNING +++     DIFFERENCE IN       ITEM_TYPE           NOT-USED
|
|
|
|   ITEM SPLIT
|
|       ITEM-NAME (OLD SIA)            LENGTH TYPE    |   ITEM-NAME (NEW SIA)   LENGTH TYPE
|
|       NOT-USED                           255 CHAR    |   ADRESSE-U               240 NCHAR
|
|                                                      |   NOT-USED                 15 CHAR
|
|-------------------------------------------------------------------------------------------
|
NO CHANGES IN   REMAINING                   3 RECORD(S)
RENAME CHECK    SET
NO CHANGES IN   ALL                        13 SET(S)
RENAME CHECK    KEYS IN SIA
NO CHANGES IN   ALL                         5 KEY(S)


+++++ WARNING: 1091     1 CHANGE(S) OF ITEM TYPE


+++++ WARNING: 1096     1 CHANGE(S) WITH SPLIT OF ITEMS


NUMBER OF FILE ACCESSES:            0


***** DIAGNOSTIC SUMMARY OF BALTER


+++++         2 WARNINGS
             NO ERRORS
             NO SYSTEM-ERRORS


***** END OF DIAGNOSTIC SUMMARY


***** NR OF DATABASE ACCESSES  :          119
```

Compiling the subschema

/START-UDS-DDL

```
***** START       DDLCOMP      (UDS/SQL (UDS/SQL  V2.9  1801 )     2019-01-29   09:35:53
*  DDLCOMP: INPUT SYSTEMPARAMETERS
```

SOURCE IS 'S.INSURE.SUBDDL.NEW'

END

```
*  DDLCOMP: READ SCHEMA/SUBSCHEMA
%  UDS0215 UDS STARTING UDS/SQL V2.9 (LINKED-IN), DATE=2019-01-29 (ILL2038,09:35:53/4TE7)
%  UDS0746 UDS PUBSET DECLARATION (CURRENT) FOLLOWS (ILL1746,09:35:53/4TE7)
4TE7: UDS-PUBSET-JV:  :IUDS:$XXXXXXXX.PUBSDECL.DEFAULT
4TE7: DEFAULT PUBSET: IUDS
4TE7: ----------------------------------------
*  DDLCOMP: START SUBSCHEMA-PHASE
*  DDLCOMP: CHECK SUBSCHEMA RULES
*  DDLCOMP: CHECK DATA ALLOCATION
*  DDLCOMP: SUBCOPY
*  DDLCOMP: ERROR DIAGNOSTIC
*  DDLCOMP: NO ERRORS IN SUBSCHEMA-PHASE
*  DDLCOMP: WRITE SUBSCHEMA ON COSSD
*  DDLCOMP: NO ERRORS DETECTED
%  UDS0758 NUMBER OF DML-STATEMENTS AND I/O COUNTERS PER DATABASE  (ILL2038,09:35:53/4TE7)
4TE7: DATABASE NAME            DMLS   LOG READ  PHYS READ  LOG WRITE PHYS WRITE
4TE7: ------------------------------------------------------------------------
4TE7: INSURE                   1379       2613         75        639         47
%  UDS0213 UDS NORMAL SYSTEM TERMINATION WITH ************1379 DML-STATEMENTS 2019-01-29
(ILLY033,09:35:54/4TE7)

***** DIAGNOSTIC SUMMARY FOR DDL-SUBSCHEMA

               NO ERRORS
               NO WARNINGS

***** END OF DIAGNOSTIC SUMMARY
***** NORMAL END   DDLCOMP      (UDS/SQL V2.9  1801 )     2019-01-29   09:35:54
```

/START-UDS-BGSSIA

```
***** START        BGSSIA       (UDS/SQL V2.9  1801 )     2019-01-29   09:35:54
```

GENERATE SUBSCHEMA MANAGEMENT OF SCHEMA CUSTOMER-CARDS

DISPLAY

END

```
%  UDS0215 UDS STARTING UDS/SQL V2.9 (LINKED-IN), DATE=2019-01-29 (ILL2038,09:35:54/4TE7)
%  UDS0746 UDS PUBSET DECLARATION (CURRENT) FOLLOWS (ILL1746,09:35:54/4TE7)
4TE7: UDS-PUBSET-JV:  :IUDS:$XXXXXXXX.PUBSDECL.DEFAULT
4TE7: DEFAULT PUBSET: IUDS
4TE7: ----------------------------------------
*** SSIA GENERATION NORMALLY ENDED.
*GENERATION OF ITEM-TABLE AND NAME-TABLE STARTED.
*GENERATION OF ITEM-TABLE AND NAME-TABLE FINISHED.
%  UDS0758 NUMBER OF DML-STATEMENTS AND I/O COUNTERS PER DATABASE (ILL1758,09:35:54/4TE7)
4TE7: DATABASE NAME            DMLS   LOG READ  PHYS READ  LOG WRITE PHYS WRITE
4TE7: ------------------------------------------------------------------------
4TE7: INSURE                    802       1392         76        297         29
%  UDS0213 UDS NORMAL SYSTEM TERMINATION WITH **********802 DML-STATEMENTS 2019-01-29
(ILLY033,09:35:54/4TE7)
***** DIAGNOSTIC SUMMARY OF BGSSIA

               NO WARNINGS
               NO ERRORS
               NO SYSTEM-ERRORS

***** END OF DIAGNOSTIC SUMMARY
***** NORMAL END   BGSSIA      (UDS/SQL V2.9  1801 )     2019-01-29   09:35:54
```

Renaming of the database objects has now been completed.

8 Converting databases to larger page formats (BPGSIZE)

UDS/SQL databases can be structured with the following page formats (see section “Formatting the compiler database with BCREATE”):

- 2-Kbyte format with a page length of 2048 bytes
- 4-Kbyte format with a page length of 4000 bytes
- 8-Kbyte format with a page length of 8096 bytes

You use the BPGSIZE utility routine to

- convert a database to a format with a larger page length (see section “Converting databases with BPGSIZE”)
- reduce the memory space requirements for realms of a database without changing the page length
- remove any FPA extents that may be present for each realm. BPGSIZE combines the entire FPA into a single FPA that consists solely of the FPA base.
- combine DBTTs. BPGSIZE combines the DBTT base and DBTT extents to form a new DBTT base.

In order to convert a database and its associated applications, you will need to perform the following individual steps:

- check the criteria required to convert the database to a larger format,
- convert the database to a larger format by using the BPGSIZE utility routine,
- adapt the database schema and subschemas (i.e. restructure the database), and
- determine and adapt applications affected by the database conversion if required. Note, however, that converting a database to a larger page format does not always involve the adaptation of application programs that work with the database.

The individual steps required for conversion, i.e. converting the database with BPGSIZE, restructuring it, and adapting application programs, need not be performed at the same time. In other words, normal database operation is possible between these steps.

Two typical conversion examples reflecting different application scenarios are outlined at the end of this chapter, including the case that occurs most frequently in practice.

8.1 Criteria for conversion

A database may need to be converted to a larger page format for the following reasons:

- to prevent capacity bottlenecks
- to use database key values in the extended value range
- to utilize the full functionality of UDS/SQL

Converting a database for capacity reasons

The individual capacity reasons that could make it necessary to convert a database from a 2-Kbyte to a 4-Kbyte or 8-Kbyte format are as follows (see the “Design and Definition” manual):

- More than 16777 215 (=2^{24}-1) records need to be stored for at least one record type of the database.
- The record length for at least one record type is greater than 1700 bytes (guideline: die maximum record length in a database with a 2-Kbyte format equals 2020 bytes).
- The database uses more than 200 record types (guideline: a maximum of 253 record types can be defined in a database with a 2-Kbyte format).
- The database is expected to contain more than 123 realms in the future.
- A multi-DB configuration was created with the sole purpose of distributing records of a record type across multiple databases (due to capacity limits).

The conversion of a database from a 4-Kbyte to an 8-Kbyte format is required if the maximum record length permitted for the 4-Kbyte format, i.e. 3968 bytes, is no longer sufficient.

The relevant information on whether and to what extent the above criteria are satisfied for a database can be obtained from the SIA report of the BPSIA utility routine and the record type statistics output by the BSTATUS utility routine (see the “Recovery, Information and Reorganization” manual).

Converting a 2-Kbyte database due to the use of extended database key values

If a database key of a 2- Kbyte database (i.e. one with a 2-Kbyte format) is supplied by an application program with database key values from the extended value range (i.e. with a REC-REF > 254 and/or an RSQ > 2^{24}-1; see the “Design and Definition” manual), the database will need to be converted to the 4-Kbyte or 8-Kbyte format even if there are initially no capacity shortages.

Consequently, in the case of multi-DB programs, i.e. application programs which access multiple databases, it is first necessary to check whether these programs also use database key values across multiple databases. If such an application program copies database key values from a 4-Kbyte or 8-Kbyte database to a database key of a 2-Kbyte database, this 2-Kbyte database will also need to be converted to the 4-Kbyte or 8-Kbyte format and restructured accordingly (see section “Restructuring the converted database”).

Databases that are affected by the use of extended database key values can be essentially converted in any order; however, the following approach is recommended:

1. First convert, and possibly restructure, any databases that are to be supplied with extended database key values in the future.
2. Then convert, and possibly restructure, the databases from which extended database key values my be copied to other databases by application programs.

This approach will eliminate the possibility of errors arising from copying extended database key values to databases that haven’t been converted.

Converting a 2-Kbyte database in order to use the full UDS/SQL functionality

Databases with a 4-Kbyte or 8-Kbyte format are also required in order to use the following functions:

- databases on NK4 pubsets
- BS2000 access method FASTPAM for files and realms of the database

8.2 Converting databases with BPGSIZE

The BPGSIZE utility can be used to convert a database to a format with the same or a higher page length. The database to be converted may be the original database or any shadow database (i.e. a database copy) with a copy name other than NEW (suffix). The database to be converted must be consistent and is not altered by BPGSIZE. The consistency of a database to be converted can be verified with the BCHECK utility routine (see the “Recovery, Information and Reorganization” manual).

BPGSIZE creates the converted database as a shadow database with the copy name (suffix) NEW. During the conversion with BPGSIZE, other UDS/SQL utility routines and UDS/SQL sessions are allowed read access to the database.

The BPGSIZE utility routine operates without AFIM logging. Consequently, if AFIM logging has been turned on for the database to be converted, BPGSIZE reports the fact during conversion and also issues a warning at the end of processing. Both AFIM logging and online backup capability are always turned off for the converted database.

Following the BPGSIZE run, a number of additional measures are required (see section “Preparing the converted database for DB operation”) to prepare the converted database for use in the running UDS/SQL mode.

When it starts BPGSIZE takes into account any UDS/SQL pubset declaration which is assigned (see the “Database Operation” manual, Pubset declaration job variable). Faulty assignment results in the program aborting.

8.2.1 BPGSIZE functions

The BPGSIZE utility routine offers the following functions:

- Conversion of a 2-Kbyte database to a database (copy name NEW) with a 4-Kbyte or8-Kbyte format.
  When BPGSIZE converts a database to pages with a larger format, it extends, among other things, the system data structures such as the SCD and page index and thus prepares the database to accept and manage database key values with a REC-REF > 254 and/or an RSQ > 2^{24}-1 (see the "Design and Definition" manual).
- Conversion of a 4-Kbyte database to a database (copy name NEW) with a 8-Kbyte format.
- Conversion to a database (copy name NEW) that has the same page length, but requires less storage space than the original database due to the compression of realms.
  Compression is enabled by the fact that neither the gaps in the pages of the original database nor the empty pages are transferred to the converted database by BPGSIZE during conversion.
- Deletion of FPA extents.
  For each realm, BPGSIZE combines the base free space administration table (FPA base) and any existing FPA extents to form an FPA that consists solely of the FPA base. FPA extents may be created in the context of an automatic realm extension, an online realm extension or a realm extension by the BREORG utility routine (see the “Database Operation“ manual, Online realm extension).
- Deletion of DBTT extents
  BPGSIZE combines each DBTT into a DBTT that consists solely of the DBTT base. Any existing DBTT extents are eliminated.

Realms for which the online extensibility has been activated with the DAL command ACT INCR retain this activation during a BPGSIZE run.

8.2.2 Realms and files

The realms of the converted database are created by BPGSIZE on the basis of those in the original database. This is achieved by means of work files.

8.2.2.1 Realms of the converted database

By default, BPGSIZE creates a converted realm on a public volume with the same size as the original realm. If desired, you can use the BS2000 CREATE-FILE command to have the file for a converted realm selectively created under a specific catalog ID or on a private disk.

File name of the converted realm

The file name of the new converted realm is as follows: *dbname*.*realm-name*.NEW

dbname

Name of the converted database

realm-name

Name of the realm that is converted.
realm-name stands for the DBDIR or DBCOM, depending on whether the database directory or database compiler realm is involved.

Calculating storage space for the converted realm

The approach followed by BPGSIZE during conversion is outlined below to assist you in determining whether the converted realm will require more space than the original realm and how the primary allocation for the converted realm should be selected.

Factors due to which additional storage space may be required for the converted realm in some circumstances:

- When a 2-Kbyte database is converted to the 4-Kbyte or 8-Kbyte format, the 2048 byte page length of the original realm is not exactly doubled, but quadrupled. Furthermore, since each record must be fully stored within *one* page, a new 4000-byte or 8096-byte page is generally not enough for the contents of two or four full 2048-byte pages.
- When converting a 2-Kbyte database to the 4-Kbyte or 8-Kbyte format, BPGSIZE converts hash areas by mapping each direct or indirect CALC page with a length of 2048 bytes to a new CALC page of 4000 or 8096 bytes on a one-to-one basis. When a 4-Kbyte database is converted to the 8-Kbyte format, a CALC page with a length of 4000 bytes is mapped on a one-to-one basis to new CALC page of 8096 bytes. The initial storage space required in the converted realm for hash areas is thus double or four times the original space.
- If the value specified for TABLE-FILLING in the CONVERT-DATABASE statement is too low (see "Statements for BPGSIZE"), the tables in the new realm will not be fully filled by BPGSIZE. If the occupancy level of the tables is reduced as a result, more space is required.

Factors that may reduce the required storage space for the converted realm in some circumstances:

- Only the pages in the original realm that already contain records or tables are transferred by BPGSIZE to a page of the converted realm; empty pages are ignored.
- BPGSIZE fills pages of the converted realm with records and tables to the maximum extend possible. The storage space required for the converted realm is thus reduced if the original realm contains several partially-filled page.
- BPGSIZE reduces the number of overflow pages for hash areas and duplicates tables wherever possible.

Automatic creation of the converted realm by BPGSIZE

If the realm *dbname*.*realm-name*.NEW is created by BPGSIZE, an FPA area of the same size as that of original realm *dbname*.*realm-name* is created by BPGSIZE for *dbname*.*realm-name*.NEW.

Note that the new realm *dbname*.*realm-name*.NEW may become too large for the FPA area calculated by BPGSIZE due to file extensions and thus cause the conversion to be aborted by BPGSIZE with an error message. If this occurs, you will need to use the BS2000 CREATE-FILE command to create the file for the realm *dbname*.*realm-name*.NEW with a larger primary allocation and then start the conversion of the realm with BPGSIZE again.

Creating the file for the converted realm

If the converted realm requires more space than the original realm, you will need to create the file *dbname*.*realm-name*.NEW for the converted realm yourself. This can be done with the BS2000 CREATE-FILE command.
(see "Maximum size of UDS/SQL files" in chapter "Files and realms of a UDS/SQL database").

In this case, BPGSIZE calculates the required size for the FPA area from the primary allocation specified in the CREATE-FILE command. BPGSIZE then creates the FPA area for the converted realm using this calculated value, provided it is greater than the one for the original realm.
If you specify a value of less than 576 PAM pages as the secondary allocation, the value for the secondary allocation is automatically set to 576 PAM pages by BPGSIZE.

Converting realms with distributable lists

When realms which participate in a distributable list are converted (and must therefore be converted in a single BPGSIZE run), you must bear in mind that the conversion attempts to distribute the records of the member record type in the distributable list evenly over the realms.

If the distribution over the realms was previously very uneven, before conversion takes place you should explicitly create the new realms with a size which takes into consideration even distribution (for this purpose you can use BSTATUS to obtain information on the location and distribution of the records over the various realms). If you do not create the realms explicitly, they will be created with the current size without taking the intended even distribution into consideration. This might then lead to abnormal termination while attempting even distribution as BPGSIZE does not extend the new realms.

8.2.2.2 Required work files

In order to update the DBTTs of the database, one work file is required by BPGSIZE for each record type contained in the realm being converted.
These work files have the following names:

UTI.BPGSIZE.*dbname*.*realmref*.*recref*

dbname

Name of the database being converted

realmref

Three-digit internal number of the realm being converted

recref

Five-digit internal number of the record type

A work file with *recref* 0 is also required for each realm for the handling of SYSTEM sets. A work file with *realmref* 0 and *recref* 0 that is shared by all realms is required for completion processing.

These files are created by BPGSIZE during conversion on a public volume by default and with a primary allocation of 129 PAM pages and a secondary allocation of 25 PAM pages.

If desired, you can use the BS2000 CREATE-FILE command before executing BPGSIZE to create the work files independently under a specific catalog ID or on private disk. Note, however, that you must always use the standard file names. The internal realm and record reference numbers and their corresponding assignments can be determined with the BPSIA utility routine (see the “Recovery, Information and Reorganization” manual). The data population for buffering is calculated for the record-specific work files using the following formula:

```
Number of records of the record type in the relevant realm *
Number of DBTT columns of the record type * 9 Bytes
```

If the DBTT of a record type lies in the same realm as all records of that type, BPGSIZE processed the associated work file together with the realm and then deletes the work file. Otherwise, BPGSIZE deletes the work file only after all realms (including the DBDIR and DBCOM) and possibly the COBOL subschema directory (COSSD; see next section) have been successfully converted.

The LMS library SIPLIB.UDS-SQL-T.024 contains the example procedure P.GEN-FILE-BPGSIZE which uses CSV output data from the BPSIA and BSTATUS utility routines to create the record-specific work files.

8.2.2.3 COBOL subschema directory (COSSD) of the converted database

BPGSIZE generates the converted COBOL subschema directory (COSSD) only if the page length of the database is extended during conversion. The converted COSSD that is automatically created by BPGSIZE in such cases is placed on a public volume and has the same size as the original COSSD.
The file name of the converted COSSD is:

dbname.COSSD.NEW

dbname

Name of the database being converted

If desired, the file *dbname*.COSSD.NEW for the converted COBOL subschema directory can also be created independently under a particular catalog ID or on private disk by using the BS2000 CREATE-FILE command.

If the page length of the database is not modified by BPGSIZE during conversion, the contents of the original COSSD will also apply to the converted database. The association between the COSSD and the converted database can then be created by copying the file or recataloging it with the BS2000 command COPY-FILE or MODIFY-FILE-ATTRIBUTES, respectively.

8.2.2.4 Module library for hash routines (HASHLIB) of the converted database

The module library for hash routines (HASHLIB) is not taken into account by BPGSIZE during database conversion, since the contents of the HASHLIB for the original database are also applicable to the converted database. The association between the HASHLIB and the converted database can be established by copying or recataloging it with the BS2000 command COPY-FILE or MODIFY-FILE-ATTRIBUTES.

8.2.3 Conversion phases

You can optionally convert the entire database in *one* BPGSIZE run or split the database conversion process over a multiple BPGSIZE runs.

The following three phases must always be differentiated, regardless of how many BPGSIZE runs are used to convert the database:

1. Conversion of the database directory (*dbname*.DBDIR)
2. Conversion of the database compiler realm (*dbname*.DBCOM) and all user realms (*dbname.realm-name*)
3. Update run: among other things, BPGSIZE reads in auxiliary files that have not been processed and updates the remaining DBTTs; if conversion to a larger page format is involved, the COBOL subschema directory (*dbname*.COSSD) is also converted here.

The realm being currently processed can be determined from the runtime messages of BPGSIZE.

When realms which do not participate in distributable lists are converted, only the realm which is to be converted plus *dbname*.DBDIR and *dbname*.DBDIR.NEW are required. You can therefore temporarily swap out all other original and converted realms in the database (to create space for converting the realm which is currently being processed).

You cannot convert a realm which is being used by a distributable list on its own. Instead, you must convert all realms which belong to a distributable list *S* in the same BPGSIZE run. The set of realms which can be converted in a BPGSIZE run must be closed with respect to distributable lists.

A set of realms is closed with respect to distributable lists when it is assigned the characteristic that for every realm of the set which is used by any distributable list *S* every other realm which is used by the same list *S* is also contained in the set.
If, for example, *SX* is a distributable list with the realms *R1*, *R2*, *R3* and *SY* is a distributable list with the realms *R2*, *R4*, *R5*, the realm set *R1*, *R2*, *R3*, *R4*, *R5* is closed with respect to distributable lists. The set *R1*, *R2*, *R3*, on the other hand, is not closed with respect to distributable lists as *R2* is contained in the set, but realms *R4* and *R5* which belong to the same list *SY* are not.

Required realms

The following overview shows which realms are required by BPGSIZE to convert the database directory (*dbname*.DBDIR), the database compiler realm (*dbname*.DBCOM), and the individual user realms. The required realms must be made available on magnetic disk during the period required for conversion.

- The following are required to convert the database directory:
  - the original database directory *dbname*.DBDIR
  - the file *dbname*.DBDIR.NEW for the converted database directory. This file is created by BPGSIZE by default.
- The following are required to convert the database compiler realm:
  - the original database compiler realm *dbname*.DBCOM
  - the file *dbname*.DBCOM.NEW for the converted database compiler realm. This file is created by BPGSIZE by default.
  - the original database directory *dbname*.DBDIR
  - the converted database directory *dbname*.DBDIR.NEW

- The following are required to convert a user realm *dbname*.*realm-name*:
  - the original user realm *dbname*.*realm-name*
  - the file *dbname*.*realm-name*.NEW for the converted user realm. This file is created by BPGSIZE by default.
  - the original database directory *dbname*.DBDIR
  - the converted database directory *dbname*.DBDIR.NEW

Apart from the original database directory (*dbname*.DBDIR), the original realm is no longer required by BPGSIZE for conversion as soon as that realm has been fully converted. The original database directory, by contrast, is no longer required only after the entire database has been converted.

System environment of BPGSIZE

The following diagrams show the system environment of BPGSIZE in the individual conversion phases.

System environment when converting the DBDIR

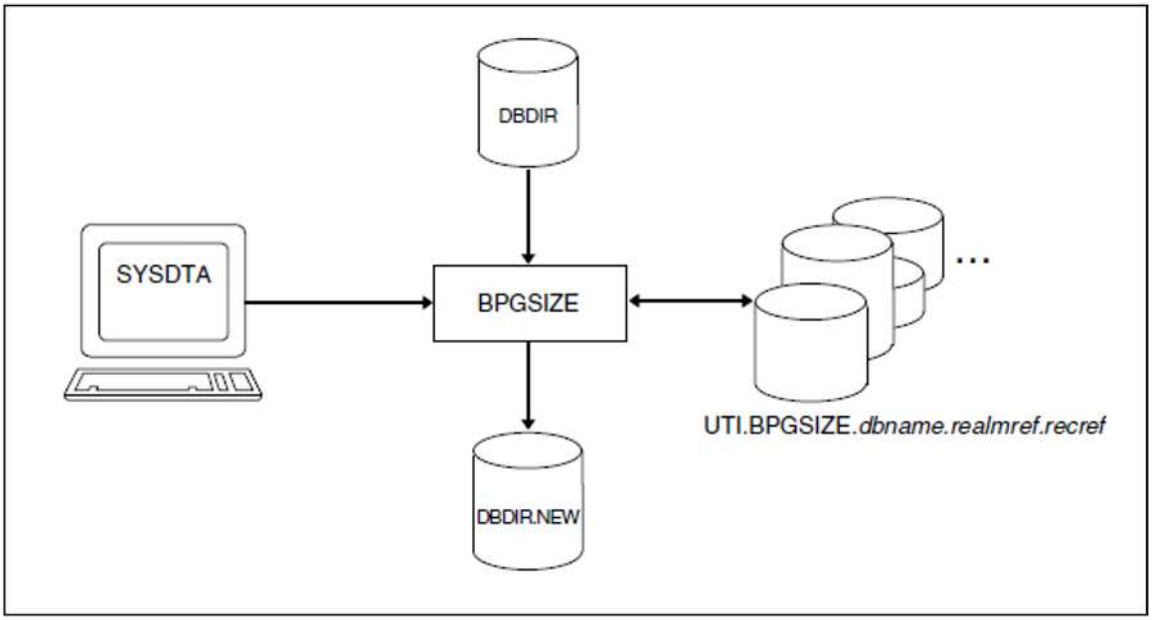


Figure 36: System environment of BPGSIZE when converting the DBDIR

System environment when converting the DBCOM and user realms

The following system environment is required by BPGSIZE in order to convert the database compiler realm (*dbname* .DBCOM) or a user realm (*dbname*.*realm-name*):

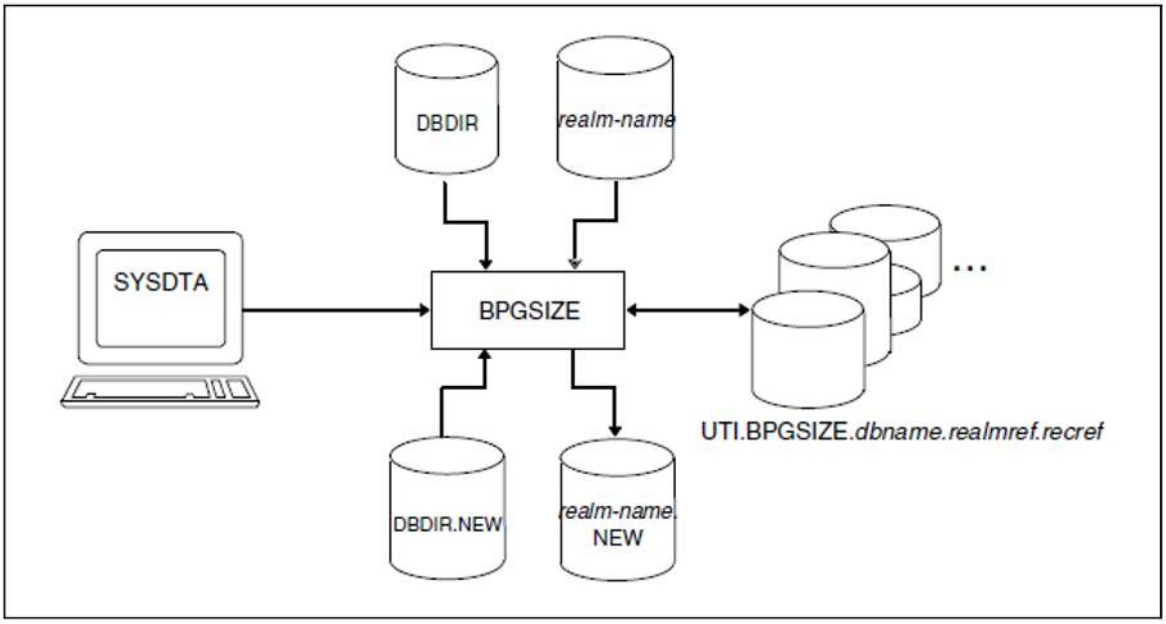


Figure 37: System environment of BPGSIZE when converting the DBCOM or a user realm

In the case of the database compiler realm, the *realm-name* in figure 37 *realm-name* refers to the DBCOM.

System environment during the update run

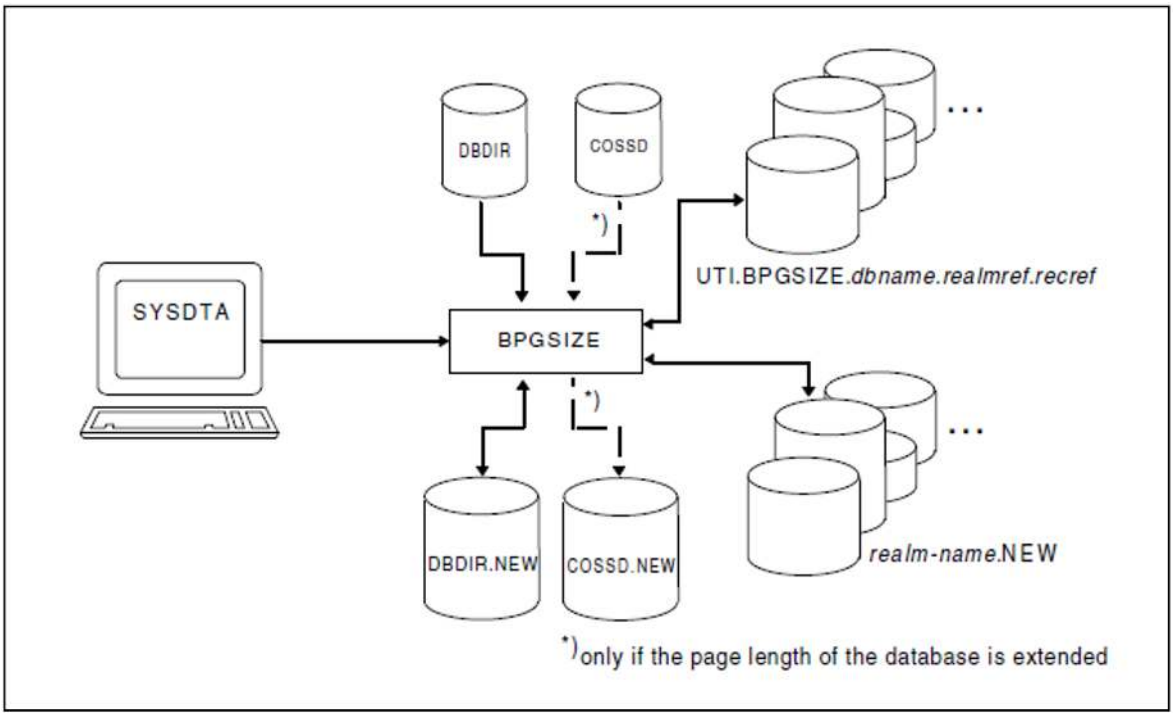


Figure 38: System environment of BPGSIZE during the update run

The COBOL subschema directory (COSSD) is converted only if the database is being converted to a larger page format.

Converting the database in one BPGSIZE run

If the entire database is converted in a single BPGSIZE run, the three phases of conversion are combined automatically by BPGSIZE in that run. This is typically achieved by calling BPGSIZE with REALM-NAME=*ALL in the CONVERT-DATABASE statement (see "Statements for BPGSIZE").
When a database is converted in *one* BPGSIZE run, the individual realms are converted sequentially. This could result in unacceptably long conversion times for large databases.It is therefore advisable to only convert small databases in *one* BPGSIZE run.

Converting the database in multiple BPGSIZE runs

If you want the database conversion to be distributed over multiple BPGSIZE runs, you will need to ensure the correct sequence for the conversion yourself.

To convert the database in multiple BPGSIZE runs, proceed as follows:

1. Create the converted database directory (*dbname*.DBDIR.NEW) in the first BPGSIZE run.
   This first BPGSIZE run could also be used to convert the database compiler realm (*dbname*.DBCOM) and/or user realms of the database.
2. You can now convert the remaining realms of the database in further BPGSIZE runs. Note that it is also possible to call BPGSIZE concurrently for individual realms. This may be meaningful in some cases, since the time required to convert a realm essentially depends on the size of that realm.

   > Exception:
   >
   > As only realm sets which are closed with respect to distributable lists can be converted in one BPGSIZE run (see "Conversion phases"), realms of a distributable list cannot be converted in parallel.

   The following factors should be taken into account when deciding whether to convert realms sequentially or in parallel:

   - Sequential realm conversion requires less memory, but generally results in longer runtimes.
   - Parallel realm conversion requires more memory in the short term, but speeds up the conversion of the database.
3. Now start the update run by calling the BPGSIZE with REALM- NAME=*ALL in the CONVERT-DATABASE statement (see "Statements for BPGSIZE"). If a conversion to a larger page format is involved, BPGSIZE will automatically convert the COBOL subschema directory (*dbname*.COSSD.NEW) as well.
4. All realms for a distributable list must be converted in the same BPGSIZE run.

Restarting an aborted BPGSIZE run

If a BPGSIZE run is aborted due to a small FPA area in the converted realm, for example, you will need to restart the aborted BPGSIZE run. This should be done by starting BPGSIZE again with the same statements so that BPGSIZE is prevented from processing fully converted realms again.

8.2.4 Statements for BPGSIZE

The following statements are available for the BPGSIZE utility routine:

Statement	Function
ALLOCATE-BUFFER-POOL	Define buffer size
CONVERT-DATABASE	Convert database
END	Terminate input of statements
OPEN-DATABASE	Open database
UNDO	Cancel effect of statement

Table 49: Statements for BPGSIZE

BPGSIZE statements can be applied on not only the user realms of the database, but also the database directory (DBDIR) and the database compiler realm (DBCOM). Temporary realms must not be specified.

Rules for statements

Incorrectly entered statements can be corrected, and every correctly entered statement can also be cancelled with the UNDO statement.

If conflicting entries are made with respect to the function or object, the last specification entered always applies.

The ALLOCATE-BUFFER-POOL, OPEN-DATABASE and UNDO statements are executed immediately by BPGSIZE. All valid CONVERT-DATABASE statements are executed by BPGSIZE after the END statement.

The statements must be entered in the following order:

1. ALLOCATE-BUFFER-POOL
2. OPEN-DATABASE
3. CONVERT-DATABASE (possibly more than once)
4. UNDO (if required)
5. END

Statement syntax

The individual statements are described in detail below in alphabetic order.

ALLOCATE-BUFFER-POOL (define buffer size)

The ALLOCATE-BUFFER-POOL statement is used to define the size of the used buffer pools in Mbytes (see the "Database Operation" manual).

The ALLOCATE-BUFFER-POOL statement must be specified as the first statement. It may be dropped if the default values are to be used for buffer initialization.

The ALLOCATE-BUFFER-POOL statement will then no longer be offered in the SDF mask.

The ALLOCATE-BUFFER-POOL statement cannot be canceled with the UNDO statement.

ALLOCATE-BUFFER-POOL
BUFFER-SIZE = **STD** / <integer 1..2000>

BUFFER-SIZE = STD

Sets the buffer pool to the default size of 2 Mbytes.

BUFFER-SIZE = <integer 1..2000>
Size of the buffer pool in Mbytes. The size of the buffer pool must lie within the specified limits. The maximum value depends on the operating system version, the main memory configuration of the system, and the ADDRESS-SPACE-LIMIT value set for the specific user ID.

CONVERT-DATABASE (control database conversion)

The CONVERT-DATABASE statement is used to control the conversion of realms. The realms of the converted database are created by BPGSIZE with the copy name NEW. For every database conversion, the DBDIR must be the first realm to be converted.

The CONVERT-DATABASE statement may be specified more than once in each BPGSIZE run. The statement is considered valid (i.e. can be executed) only if all the realms named in it are present. If a realm that has already been converted is specified in the CONVERT-DATABASE statement, the realm is *not* converted again.

CONVERT-DATABASE

REALM-NAME = **_*ALL_** / ***ALL-EXCEPT(...)** / list-poss(30): <realm-name>

***ALL-EXCEPT**(...)

| **NAME** = list-poss(30): <realm-name>

,**DATABASE-PAGE-LENGTH** = **_*UNCHANGED_** / **2KB** / **4KB** / **8KB**

,**TABLE-FILLING** = ***UNCHANGED** / **_*MAXIMUM_** **/** <integer 1..100>

REALM-NAME = _*ALL_

All realms of the database, including the DBDIR and DBCOM, are converted.Every database conversion involves the execution of a BPGSIZE run with only one CONVERT-DATABASE statement using REALM-NAME=*ALL. This must be

- the first and only BPGSIZE run if the entire database is to be converted at once and with the same occupancy level (see below) for all tables, or
- the final update run after all realms of the database have been converted in earlier BPGSIZE runs.

REALM-NAME = *ALL-EXCEPT(...)
All realms except for those specified are converted.

NAME = list-poss(30): <realmname>
Names of realms to be excluded from conversion. You can also specify the DBDIR (database directory) and DBCOM (database compiler realm) here.

REALM-NAME = list-poss(30): <realmname>
All specified realms are converted. You can also specify the DBDIR (database directory) and DBCOM (database compiler realm) here.

BPGSIZE does not create a converted realm for a realm specified with REALM-NAME=... in the following cases:

- The realm does not exist.
- The realm is detached.
- The realm is inconsistent.
- The realm does not match the original DBDIR (*dbname*.DBDIR) or the converted DBDIR (*dbname*.DBDIR.NEW).

DATABASE-PAGE-LENGTH = *UNCHANGED

The database page length is not changed during conversion. If the original realm contain empty or only partially filled pages, this option can be used to reduce the storage space requirements for the realm.

DATABASE-PAGE-LENGTH = 2KB
Sets the database page length in the converted database to 2048 bytes. This entry is only allowed for 2-Kbyte databases and has the same effect there as the value *UNCHANGED. If a 4-Kbyte or 8-Kbyte database is involved, the statement is rejected.

DATABASE-PAGE-LENGTH = 4KB
Sets the database page length in the converted database to 4000 bytes. This entry is only allowed for 2-Kbyte and 4-Kbyte database. If an 8-Kbyte database is involved, the statement is rejected.

DATABASE-PAGE-LENGTH = 8KB
Sets the database page length in the converted database to 8096 bytes. In the case of an 8-Kbyte database, the entry has the same effect as the value *UNCHANGED.

The database page length in the converted database must be uniform.

Consequently, the following must be observed when specifying the DATABASE-PAGE-LENGTH:

- In the first BPGSIZE run for the database:
  If multiple CONVERT-DATABASE statements are specified with different DATABASE-PAGE-LENGTH values, BPGSIZE will always use the DATABASE-PAGE-LENGTH value entered in the last valid CONVERT-DATABASE statement. This entry thus determines the page length in the converted database. This is reported by means of a runtime message of BPGSIZE.
- In all other BPGSIZE runs for the database:
  The page length specified in the first BPGSIZE run must be specified for all DATABASE-PAGE-LENGTH values. Otherwise, BPGSIZE will terminate the run with an error message.

TABLE-FILLING = *UNCHANGED

The level to which tables are filled in the converted realms remain (virtually) unchanged.

TABLE-FILLING = *MAXIMUM
Every table in the converted realms is filled as follows:

- One table entry remains free at level 0 (base level).
- Up to 95 % of the table is filled on level 1.

- One table entry remains free on every higher level that follows.

TABLE-FILLING = <integer 1..100>

Every table in the converted realms is filled as follows:

- On level 0 (base level), the table is filled in accordance with the specified occupancy level (%).
- Up to 95 % of the table is filled on level 1.
- One table entry remains free on every higher level that follows.

If the same realm is addressed in multiple CONVERT-DATABASE statements with different TABLE-FILLING values, the entry found in the last valid CONVERT-DATABASE statement in which that realm is addressed will apply to the associated converted realm.

Since all tables of the converted realm are filled up to the specified percentage, a converted realms may require more storage space than the associated original realm in some cases. It is therefore generally advisable to use a high occupancy level (i.e. 99, 100 or *MAXIMUM). More details on selecting a meaningful occupancy level can be found in the “SSL” chapter of the "Design and Definition" manual.

END (terminate input of statements)

The END statement terminates the input of statements and starts execution.

END

This statement has no operands.

OPEN-DATABASE (open database)

The OPEN-DATABASE statement defines the database to be processed by subsequent statements.

This statement is not allowed if the database has already been assigned using SET-FILE-LINK LINK-NAME=DATABASE.

OPEN-DATABASE
DATABASE-NAME = <dbname> ,**COPY-NAME** = ***NONE** / <copyname> ,**USER-IDENTIFICATION** = ***OWN** / <userid>

DATABASE-NAME = <dbname>

Name of the database. You can only process a database that is available under your own user ID. A database from another user ID can only be processed from the TSOS ID of the system administrator.

COPY-NAME = *NONE

The original database is processed.

COPY-NAME = <copyname>
The database copy (shadow database) with the specified copy name is processed.

USER-IDENTIFICATION = *OWN

The database is located under the user‘s own ID.

USER-IDENTIFICATION = <userid>
The specification of a foreign database ID is only allowed under the TSOS ID.

UNDO (cancel statement)

The UNDO statement cancels the last correctly entered statement (other than UNDO itself).

The UNDO statement cannot be used to cancel the ALLOCATE-BUFFER-POOL statement.

A sequence of two UNDO statements cancels the two statements that immediately precede the UNDO sequence; a sequence of three UNDO statements cancels the three statements that precede the sequence, and so on.

UNDO

This statement has no operands.

8.2.5 Command sequence to start BPGSIZE

It is assumed for the command sequence described here that UDS/SQL was installed with IMON (see section “START commands for the UDS/SQL programs”).

```
[01 [/ADD-FILE-LINK LINK-NAME=DATABASE
          ,FILE-NAME=[:catid:][$userid.].DBDIR.[copyname] ]

02 [/CREATE-FILE FILE-NAME=[:catid:][$userid.]dbname.realm-name.NEW
        [,SUPPORT=*PUBLIC-DISK(SPACE=*RELATIVE(PRIMARY-ALLOCATION=primary
         ,SECONDARY-ALLOCATION=576)) /
         ,SUPPORT=*PRIVATE-DISK(VOLUME=priv-vsn -
          DEVICE-TYPE=device[,SPACE=...])] ]

03 ... Further CREATE-FILE statements for files of the converted realm

04 [/CREATE-FILE FILE-NAME= -
    [:catid:][$userid.]UTI.BPGSIZE.dbname.realm-nummer.recref-number
    [,SUPPORT=*PUBLIC-DISK(SPACE=*RELATIVE(PRIMARY-ALLOCATION=primary
     ,SECONDARY-ALLOCATION=secondary)) /
     ,SUPPORT=*PRIVATE-DISK(VOLUME=priv-vsn -
      DEVICE-TYPE=device[,SPACE=...])] ]

05 ... Further CREATE-FILE statements for work files of BPGSIZE

06 /SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,VERSION=version,
SCOPE=*TASK

07 /START-UDS-BPGSIZE

08 [//OPEN-DATABASE DATABASE-NAME = ...]

09 //BPGSIZE-statements

10 //END
```

01,08 You must specify only one of these two statements.

02 This CREATE-FILE command may be optionally used to create a file for the converted realm (DBDIR, DBCOM or user realm); see "Realms of the converted database".

04 This CREATE-FILE command may be optionally used to create a work file for BPGSIZE (see "Required work files").

06 The specified version of BPGSIZE is selected.
It is generally recommended that you specify the version since it is possible for several UDS/SQL versions to be installed in parallel.

07 The UDS/SQL utility routine can also be started using the aliases BPGSIZE and START-UDS-PAGE-RESIZING.

8.2.6 Example for BPGSIZE

The following example is based on the assumption that the TRAVEL database is initially available in a 2-Kbyte format. The example shows the conversion of this 2-Kbyte database to the 4-Kbyte format. Since a small database is involved, it makes sense to convert this database in a single BPGSIZE run. The work files are created automatically by BPGSIZE.

```
/ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=TRAVEL.DBDIR
/CREATE-FILE FILE-NAME=TRAVEL.DBDIR.NEW,SUPPORT=PUBLIC-DISK(SPACE=RELATIVE -
/    (PRIMARY-ALLOCATION=200,SECONDARY-ALLOCATION=50))
/CREATE-FILE FILE-NAME=TRAVEL.DBCOM.NEW,SUPPORT=PUBLIC-DISK(SPACE=RELATIVE -
/    (PRIMARY-ALLOCATION=550,SECONDARY-ALLOCATION=50))
/CREATE-FILE FILE-NAME=TRAVEL.TRAVEL-RLM.NEW,SUPPORT=PUBLIC-DISK( -
/    SPACE=RELATIVE(PRIMARY-ALLOCATION=250,SECONDARY-ALLOCATION=50))
/ADD-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=TRAVEL.DBDIR
/SELECT-PRODUCT-VERSION PRODUCT-NAME=UDS-SQL,    VERSION=02.9C00
/START-UDS-BPGSIZE
```

```
***** START        BPGSIZE      (UDS/SQL V2.9  1801 )     2019-01-29   09:27:01
```

```
//CONVERT-DATABASE REALM-NAME=*ALL,DATABASE-PAGE-LENGTH=4KB
//END
```

```
***** BEGIN        FUNCTION CONVERT DATABASE AT 2019-01-29 09:27:01
***** CONVERSION OF REALM DBDIR STARTED
                  CALC FOR RECORD USERGROUP-RECORD CONVERTED
                  CALC FOR RECORD SUBSCHEMA-RECORD CONVERTED
                  CALC FOR RECORD ERROR-MESSAGE CONVERTED
***** CONVERSION OF REALM DBDIR FINISHED
***** CONVERSION OF REALM DBCOM STARTED
***** CONVERSION OF REALM DBCOM FINISHED
***** CONVERSION OF REALM TRAVEL-RLM STARTED
                  CALC FOR RECORD TRANSPORTATION CONVERTED
                  CALC FOR RECORD ARRANGEMENT CONVERTED
                  CALC FOR RECORD HOTEL CONVERTED
***** CONVERSION OF REALM TRAVEL-RLM FINISHED
***** NORMAL   END FUNCTION CONVERT DATABASE AT 2019-01-29 09:27:02

***** DIAGNOSTIC SUMMARY OF BPGSIZE

              NO WARNINGS
              NO ERRORS
              NO SYSTEM-ERRORS

***** END OF DIAGNOSTIC SUMMARY
***** NR OF DATABASE ACCESSES  :         345
***** NORMAL END   BPGSIZE      (UDS/SQL V2.9  1801 )     2019-01-29   09:27:02
```

8.3 Preparing the converted database for DB operation

The following section describes the steps required to create a practically usable UDS/SQL database from the database copy (copy name NEW) generated by BPGSIZE:

1. Declare the converted database as the original database.
2. Check the consistency of the database if required and output database information.
3. Reorganize and adapt the converted database:
   - Reorganize hash areas of the converted database.
   - Adjust the size of realms in the converted database (if required).
   - Update the probable position pointers (PPP).
   - Extend the record population for record types where more than 16 777 215 records are to be stored in the future.
4. Check the consistency of the database if required and output database information.
5. Turn on AFIM logging (if desired) and/or online backup capability, and back up the usable database to tape.

In order to perform the above activities, you will need, among other things, the BMEND, BCHECK, BPSIA, BSTATUS, BPRECORD and BREORG utility routines (described in the “Recovery, Information and Reorganization” manual) and possibly also the BCHANGE and BALTER utility routines described in chapter “Restructuring the database (BCHANGE, BALTER)” of this manual.

Declaring the converted database as the original database

Following the conversion with BPGSIZE, you will need to declare the database copy (copy name NEW) that was created by BPGSIZE as the original database. This can be optionally done by using the BS2000 command MODIFY-FILE-ATTRIBUTES (to recatalog the database) or the BS2000 command COPY-FILE (to copy the database). You can then process the converted database.

If you want to be able to extend the realms of the database online, then you can fulfill the requirements for this now: A realm that is to be extendable online must have enough space available at the time of the extension. This requirement is fulfilled when at the time of the extension the realm either has a sufficient number of file pages available or the option of online realm extension is available (see the “Database Operation” manual, Online realm extension). Storage space is assigned when the realm file is created with the BS2000 command CREATE-FILE (see section “Preparing database creation”). You can change the storage space assignment after creation with the MODIFY-FILE-ATTRIBUTES command.

Now you can work with the converted database.

Checking the database consistency and displaying database information

Before you start reorganizing and adapting the converted database, you may use the BCHECK utility routine to check its consistency, but make sure that you take the runtimes for BCHECK into account.
You can then use the BPSIA, BSTATUS and BPRECORD utility routines to obtain relevant database information as a guideline for the required reorganization and adaptation measures.

After you have completed the reorganization and adaptation measures, you can repeat the BCHECK run and also the BPSIA, BSTATUS, and BPRECORD runs if desired.
Since the BCHECK run may take some time, this can also be done independently of the measures to adapt the database at a later stage.

Reorganizing and adapting the converted database

Reorganizing hash areas of the converted database

BPGSIZE fills pages of the converted realm with records and tables to the maximum possible extent without transferring any empty pages to it. This means that a converted realm will initially have no free space for the addition of new data. On the other hand, when the hash areas are converted by BPGSIZE, each direct and indirect CALC page of an original realm is mapped on a one-to-one basis to a CALC page in the converted realm that is approximately double or four times its original size (if conversion to a larger page format is involved).

The BREORG utility routine (REORGANIZE-CALC statement) can be used to reduce the storage space required for hash areas in the converted realm and to thus create space for the addition of new data.

Example

```
/REMARK **** REORGANIZING HASH AREAS ****
/START-UDS-BREORG
//SPECIFY-SCHEMA SCHEMA-NAME=schema name
//REORGANIZE-CALC RECORD-NAME=rec-name-1, -
                  CALC-RECORD=*WITHIN-POPULATION(REALM=*ALL, -
                  POPULATION=1000), -
                  CALC-SEARCHKEY=*KEY-POPULATION(KEY-REF=*ALL, -
                  POPULATION=1000)
//REORGANIZE-CALC RECORD-NAME=rec-name-2, -
                  CALC-RECORD=*WITHIN-POPULATION(REALM=*ALL, -
                  POPULATION=15000), -
                  CALC-SEARCHKEY=*KEY-POPULATION(KEY-REF=*ALL, -
                  POPULATION=150000)
.
.
.
//END
```

Adjusting the size of converted realms

If the additional space regained from reorganizing the hash areas is not sufficient, you will need to extend the realm in question by using the MODIFY-REALM-SIZE statement of the BREORG utility routine.

Note that if the space required to store new data in the converted realm is less than the already available amount, the MODIFY-REALM-SIZE statement could also be used to reduce the realm after reorganizing the hash areas.

When adjusting the realm size, the DBDIR (database directory) and DBCOM (database compiler realm) realms, in particular, should also be taken into account.

Updating probable position pointers (PPP)

Pointer arrays that are created as probable position pointers (PPP) are not updated by BPGSIZE when converting the database. After the conversion, you can use the SIA report of the BPSIA utility routine to determine which probable position pointers (PPP) are inaccurate (see PPP-BITS in the output of CALC/KEY/CALC-SEARCH-KEY INFORMATION) and then correct them selectively for each access path with the REORGANIZE-SET statement of the BREORG utility routine. Sets in which records are relocated must be reorganized first. Such sets are defined in the SSL with MODE IS LIST (see the "Design and Definition" manual).
Note that since the BREORG runs to update the probable position pointers may take a relatively long time, this BREORG runs could also be completed later independently of the steps required to prepare the database for DB operation.

Example

```
/REMARK **** UPDATING PPPs ****
/START-UDS-BREORG
/SPECIFY-SCHEMA SCHEMA-NAME=schema-name
//REORGANIZE-SET SET-NAME=set-name-1,OWNER-SELECTION=ALL,FILLING=UNCHANGED
//REORGANIZE-SET SET-NAME=set-name-2,OWNER-SELECTION=ALL,FILLING=UNCHANGED
.
.
.
//REORGANIZE-SET SET-NAME=set-name-n,OWNER-SELECTION=ALL,FILLING=UNCHANGED
//END
```

Updating the probable position pointers is faster if you use the REORGANIZE-POINTERS statement (see the section entitled 'Reorganizing the database with BREORG' in the "Recovery, Information and Reorganization" manual).

```
/START-UDS-BREORG
//SPECIFY-SCHEMA SCHEMA-NAME=schema-name
//REORGANIZE-POINTERS REALM-NAME=realm-name-1
//REORGANIZE-POINTERS REALM-NAME=realm-name-2
.
.
.
//REORGANIZE-POINTERS REALM-NAME=realm-name-n,
//END
```

The following schema names and set names are used to update the probable position pointers (PPP) of the DBDIR and DBCOM:

	Schema name	**Set name**
DBDIR	PRIVACY-AND-IQF-SCHEMA	USERGROUP-USERID
DBCOM	COMPILER-SCHEMA	NAME-TABLE

Example

```
/REMARK **** REORGANIZING THE DBDIR ****
/START-UDS-BREORG
//SPECIFY-SCHEMA SCHEMA-NAME=PRIVACY-AND-IQF-SCHEMA
//REORGANIZE-SET SET-NAME=USERGROUP-USERID,OWNER-SELECTION=ALL
//END
/REMARK **** REORGANISATION VON DBCOM ****
/START-UDS-BREORG
//SPECIFY-SCHEMA SCHEMA-NAME=COMPILER-SCHEMA
//REORGANIZE-SET SET-NAME=NAME-TABLE,OWNER-SELECTION=ALL
//END
```

Extending the record set population

If the converted database was created from an original database with a 2-Kbyte format, you will need to extend the record population for record types that are expected to have more than the maximum number of 16 777 215 records allowed for 2-Kbyte databases. The record population can be extended by using the MODIFY-RECORD-POPULATION statement of the BREORG utility routine.

Turning on AFIM logging and online backup capability, saving the prepared database

If you want to operate the converted database with AFIM logging, you can turn on AFIM logging for the converted database with the BMEND utility routine.

If you want to be able to back up the converted database online in the future, use the BMEND utility routine to turn on online backup capability.

Make sure that you create a consistent backup copy of the converted database before restarting any database operations.

8.4 Restructuring the converted database

When a database has been converted from the 2-Kbyte format to the 4-Kbyte or 8-Kbyte format, you can use the extended value range for the maximum number or records per record type without changing the database schema, unless a field of type DATABASE-KEY is defined in the record description or in the LOCATION-MODE in the schema DDL - if you want to use the extended value range for records then you must convert this field to DATABASE-KEY-LONG for each record type.
The main advantage of this is that application programs can be adapted independently of the database conversion. Changes in the schema are required if you want to use the extended value range without restrictions even for record types with an explicitly defined database key item.

This section only illustrates the changes required in the Schema DDL and Subschema DDL If the extended value range is to be used. The restructuring of a database is described in detail in chapter “Restructuring the database (BCHANGE, BALTER)”.

If DB application programs use database key values of a record type for which no more than

16 777 215 records are to be stored even in the future (RSQ < 2^{24}-1), you should ensure that the record type involved receives a record reference (REC-REF) < 255 when restructuring the database. This helps reduce the overhead for adapting and possibly recompiling the application programs, since the corresponding USAGE IS DATABASE-KEY items need not be adapted in this case (see section “Adapting COBOL and CALL DML statements”).

Existing database key values are converted to the extended format on restructuring the database when the data repository is adapted with the BALTER utility routine (see section “Analyzing schema modifications and adapting stored data with BALTER”).

In order to optimize your database, you should adjust the values specified for the population in the SSL to the extended value range (see the "Design and Definition" manual).

Adapting the database schema

The following language options of the Schema DDL must be changed in order to use extended database key values:

- TYPE IS DATABASE-KEY changed to TYPE IS DATABASE-KEY-LONG
- LOCATION MODE IS DIRECT changed to LOCATION MODE IS DIRECT-LONG

In the case of record types that contain a database key item (TYPE IS DATABASE-KEY), you should check whether this database key item will need to accept extended database key values in the future. Note, however, that a database key item need not always be used for the database key of the record in which it is defined. It could, for example, also be used for the database key of some other record type. You should therefore also examine all possible cross-links when deciding whether to convert an item of type DATABASE-KEY to a DATABASE-KEY-LONG item.
The conversion of a DATABASE-KEY item to a DATABASE-KEY-LONG item also requires all LOCATION MODE IS DIRECT clauses in which that DATABASE-KEY-LONG item is addressed to be changed to LOCATION MODE IS DIRECT-LONG (see the "Design and Definition" manual).

Adapting subschemas

For each database key item of the database schema that has been changed to DATABASE-KEY-LONG, the associated USAGE IS DATABASE-KEY clauses must be changed to USAGE IS DATABASE-KEY-LONG in the subschema definitions involved (see the "Design and Definition" manual). These subschemas and the application programs that access them must then be recompiled.

If all record types of a subschema containing a DATABASE-KEY-LONG item are to be copied with the Subschema DDL statement COPY, this subschema will only need to be recompiled, but not adapted.

The subschemas that need to be adapted can be determined following the BALTER run to adapt stored data (see section "Schema entry", "Modifying the Schema DDL") by means of the DDL compiler run, which copies the subschemas and also filters out those that are incompatible (see section “Copying compatible subschemas”).

8.5 Adapting COBOL and CALL DML statements

The conversion of a 2-Kbyte database to a 4-Kbyte or 8-Kbyte format with the BPGSIZE utility routine has no impact on connected database applications and other databases so long as the database contains only database key values with a REC-REF <= 254 and an RSQ <= 2^{24}-1.

In most cases, however, a database will be converted so that the extended value range for database key values (REC REF > 254 and/or RSQ > 2^{24}-1) can be used in it. The use of extended database key values usually affects the associated application programs as well as other databases in some circumstances and therefore mandates appropriate adaptation measures.

In order to enable the use of extended database key values in the database, an application program may need to be adapted in the following cases:

- The Schema DDL of the converted database contains at least one of the following clauses:
  - TYPE IS DATABASE-KEY-LONG
  - LOCATION MODE IS DIRECT-LONG
  - SET SELECTION THRU LOCATION MODE OF OWNER if the primary key is defined with LOCATION MODE IS DIRECT-LONG
- The application program uses DML statements that access the database via database key items.
- The application program uses at least one of the following COBOL definitions:
  - definition of a COBOL item of type USAGE IS DATABASE-KEY
  - redefinition with REDEFINES for a USAGE IS DATABASE-KEY item
  - reference via a LINKAGE to a data structure containing items of type USAGE IS DATABASE-KEY-LONG (e.g. LINKAGE to a subschema).
- The application program uses database key values in areas that are not directly detectable (see section “Adapting additional locations in the application program”).

Following the adaptation, the application program must be recompiled and linked again.

8.5.1 DDL clauses that indicate the use of extended database key values

An analysis of the schema and Subschema DDL indicates if and where database key values of the extended value range are used in an application program.

The use of extended database key values in an application program can be deduced from the following DDL clauses:

- Definition of an item with TYPE IS DATABASE-KEY-LONG in the Schema DDL entry for a record type that is copied into a subschema used by the application program with COPY.
- Specification of the LOCATION MODE IS DIRECT-LONG clause in the Schema DDL entry for a record type that is copied into a subschema used by the application program with COPY.
- Specification of the SET SELECTION THRU LOCATION clause in the DDL set entry for an owner record type with the following attributes:
  - The record type is defined with LOCATION MODE IS DIRECT-LONG.
  - The record type is copied to a subschema used by the application program with COPY.
- Definition of a USAGE IS DATABASE-KEY-LONG item in a subschema used by the application program.

A description of which COBOL DML and CALL DML statements are affected by these DDL clauses can be found in section "Adapting DML statements", below.

Subschemas that are affected by data type changes (DATABASE-KEY-LONG) in the Schema DDL are subject to other validation criteria. Application programs that use such subschemas must therefore be adapted, recompiled, and linked again.

8.5.2 Adapting DML statements

DML statements that use a database key value provide an indication of where adaptations to database key values of the extended value range (REC-REF > 254 and/or RSQ > 2^{24}-1) may be required in the application program.

The following COBOL DML statements use a database key value:

- ACCEPT
- FIND (format 1) or FETCH (format 1)
- FIND (format 7) or FETCH (format 7)
- MODIFY
- SET
- STORE

The following CALL DML statements use a database key value:

- ACCEPTC
- FIND1 or FTCH1
- STORE1 or STORE2

DML statements are described in detail in the “Application Programming” manual.

8.5.2.1 Overview

The table 50 contains a general decision framework indicating when adaptations to database key values of the extended value range are required for individual DML statements. This table only takes the functionality of DML statements into account; no distinction is made between COBOL DML and CALL DML statements.

DML statement	... must be adapted if:		
ACCEPT FIND/FETCH-1	there are database key values with a REC-REF > 254 and/or RSQ > 2^{24}-1 for a record type used by the statement	–	–
STORE	–	the DDL entry of the used record type contains the clause LOCATION MODE IS DIRECT-LONG	–
STORE, MODIFY, FIND/FETCH-7 (without CURRENT)	–	the DDL entry of the owner record type contains the clause LOCATION MODE IS DIRECT-LONG	the DDL entry of the affected set in which the record type is “owner” contains the clause: SET SELECTION THRU LOC MODE OF OWNER

Table 50: Criteria for adapting DML statements to extended database key values

8.5.2.2 Adapting COBOL DML statements

ACCEPT

The ACCEPT statement has two variants:

- Format 1 of ACCEPT determines the database key value of the CRR, CRS, CRA or CRU and places it in a COBOL item of type USAGE IS DATABASE-KEY. This item must be adapted (USAGE IS DATABASE-KEY-LONG) if it is not possible to guarantee that the database key values of the CRR, CRS, CRA or CRU lies in the extended value range.
  If the determined database key value is too large for the result field, UDS/SQL reports status code 15102.
- Format 2 of ACCEPT determines the realm containing the record whose database key value is specified in the COBOL item of type USAGE IS DATABASE-KEY. This item must be adapted
  (USAGE IS DATABASE KEY LONG) if database key values from the extended value range are to be passed in it.

FIND (format 1) and FETCH (format 1)

Format 1 of the FIND and FETCH statements locate a database record via the value of a COBOL item of type USAGE IS DATABASE-KEY. This item must be adapted (USAGE IS DATABASE-KEY-LONG) if a record whose database key value lies in the extended value range is to be found.

FIND (format 7) and FETCH (format 7)

Format 7 of FIND/FETCH can be used by the application program to select the desired set occurrence via the database key of the owner record if the Schema DDL contains the following two clauses:

- LOCATION MODE IS DIRECT or LOCATION MODE IS DIRECT-LONG in the record entry of the owner record type
- SET OCCURRENCE SELECTION THRU LOCATION MODE OF OWNER in the corresponding set entry

In order to select the set occurrence, the application program passes the database key value to the database key item named in the LOCATION MODE clause (see the "Design and Definition" manual).

In the case of LOCATION MODE IS DIRECT-LONG, the type of the associated database key item is DATABASE-KEY-LONG. In order to enable the application program to pass database key values of the extended value range, the type of the COBOL item that passes the database key value must be changed to USAGE IS DATABASE-KEY-LONG.

MODIFY

MODIFY can be used by the application program to move the Current Record of Run-unit (CRU) to another set occurrence if the Schema DDL contains the following two clauses:

- LOCATION MODE IS DIRECT or LOCATION MODE IS DIRECT-LONG in the record entry of the owner record type and
- SET OCCURRENCE SELECTION THRU LOCATION MODE OF OWNER in the corresponding set entry

The application program can select the desired set occurrence with MODIFY via the database key value of the owner record by passing the database key value to the database key item named in the LOCATION MODE clause (see the "Design and Definition" manual).

In the case of LOCATION MODE IS DIRECT-LONG, the type of the associated database key item is DATABASE-KEY-LONG. In order to enable the application program to pass database key values of the extended value range, the type of the COBOL item that passes the database key value must be changed to USAGE IS DATABASE-KEY-LONG.

SET

The SET statement transfers the contents of a database key item to one or more other database key items.

The target items must be adapted to the extended value range (USAGE IS DATABASE-KEY-LONG) if the following applies:

- The item whose contents are to be transferred has already been changed to the type USAGE IS DATABASE-KEY-LONG or is a redefinition of an item of type USAGE IS DATABASE-KEY-LONG.
- Database key values of the extended value range are to be transferred.

The behavior of the COBOL runtime system when assigning database key values with the SET statement is reflected in the table below (see the “Application Programming” manual):

Transfer to from	DATABASE-KEY	DATABASE-KEY-LONG
DATABASE-KEY	a	b
DATABASE-KEY-LONG	c	a

a. The values are mapped on a 1:1 basis

b. The values are extended internally

c. The values are internally truncated. If the database key value to be transferred is too large for a target item (REC-REF > 254 and/or RSQ > 2^{24}-1), a status code of 00102 is reported, and the value of the target item is set to 0; however, the execution of the statement is not aborted.

STORE

The STORE statement adds a new record to the database. If the schema entry for the record type involved contains the LOCATION MODE IS DIRECT or LOCATION MODE IS DIRECT-LONG clause, the database key value for the record to be stored can be specified by the application program. This is done by passing the database key value to the database key item named in the LOCATION MODE clause.

UDS/SQL then proceeds as described below:

- Database key assignment:
  UDS/SQL stores the record in the database under the database key value provided by the application program.
- Set occurrence selection:
  If the record type involved is a member of a set that is defined with
  SET OCCURRENCE SELECTION THRU LOCATION MODE OF OWNER, UDS/SQL stores the record as a member record in the set occurrence of the owner record whose database key value matches the one provided by the application program.

In the case of LOCATION MODE IS DIRECT-LONG, the type of the associated database key item is DATABASE-KEY-LONG. In order to enable the application program to pass database key values of the extended value range, the type of the COBOL item that passes the value must be changed to USAGE IS DATABASE-KEY-LONG.

8.5.2.3 Adapting CALL DML statements

ACCPTC

The ACCPTC statement has two variants:

- ACCPTC with the function option DB-KEY, DBKREC, DBKRLM or DBKSET determines the database key value of the CRR, CRS, CRA or CRU and makes it available in the user information area (UINF). The ACCPTC statement must be replaced by the ACCPTL statement if it is not possible to guarantee that the database key values of the CRR, CRS, CRA or CRU are not in the extended value range .
  UDS/SQL reports status code 15102 for ACCPTC if the value of the determined database key lies in the extended value range.
- ACCPTC with the function option RLMDBK determines the realm containing the record with the database key value specified in the user information area (UINF). The ACCPTC statement must be replaced by ACCPTL if database key values from the extended value range are to be passed in the user information area.

FIND1 and FTCH1

The FIND1 and FTCH1 statements locate a database record via a database key value that is specified by the database programmer in the user information area (UINF). These statements must be replaced by the corresponding FIND1L or FTCH1L statement if a record with a database key value in the extended value range is to be found.

STORE1 and STORE2

The STORE1 statement stores a complete record in the database. The STORE2 statement stores individual items in the database and also compressed them if the corresponding record type is defined with the COMPRESSION ALL clause in the SSL.
If the record type in question is defined in the Schema DDL with the LOCATION MODE IS DIRECT or LOCATION MODE IS DIRECT-LONG clause, UDS/SQL stores the record under the database key value specified by the database programmer in special parameter 2 (SPP2).
Database key values from the extended value range cannot be specified in the SPP2 of the STORE1 and STORE2 statements. In order to ensure that such values can be specified in combination with LOCATION MODE IS DIRECT-LONG, the STORE1 or STORE2 statement must be replaced by STOR1L or STOR2L, respectively.

8.5.3 Adapting COBOL definitions

COBOL definitions that refer to the database key provide an indication of where adaptations to database key values of the extended value range (REC-REF > 254 and/or RSQ > 2^{24}-1) may be required in the application program.

The following COBOL definitions refer to the database key:

- Definition of a COBOL item with type USAGE IS DATABASE-KEY
- Redefinition with REDEFINES of a COBOL item or subschema item of type USAGE IS DATABASE KEY
- LINKAGE SECTION describing COBOL or subschema items of type USAGE IS DATABASE-KEY

These definitions thus indicate where adaptations for extended database key values may be required in the application program.

Adapting definitions of COBOL items with USAGE IS DATABASE-KEY

The definition of COBOL items of type USAGE IS DATABASE-KEY or USAGE IS DATABASE-KEY-LONG in the WORKING-STORAGE SECTION of an COBOLD program corresponds to the definition of a database key item in the subschema.

A COBOL item that is defined with USAGE IS DATABASE-KEY indicates that the application program uses database key values. If this COBOL item is also to accept database key values of the extended value range (e.g. with ACCEPT), the type of the COBOL item must be changed to USAGE IS DATABASE-KEY-LONG.

Any COBOL program that only uses database key items (USAGE IS DATABASE-KEY) which are defined within its WORKING STORAGE SECTION can be prepared to use extended database key values (USAGE IS DATABASE-KEY-LONG) even before the database conversion has been performed.

Adapting redefinitions of database key items

Individual components of database key values can be selectively addressed and processed in a COBOL program by redefining them with REDEFINES in the WORKING-STORAGE SECTION.
The following database key items can be redefined:

- COBOL items of type USAGE IS DATABASE-KEY or USAGE IS DATABASE-KEY-LONG
- Items of type USAGE IS DATABASE-KEY or USAGE IS DATABASE-KEY-LONG that are defined in a subschema (SUB-SCHEMA SECTION) used by the application program.

All redefinitions for COBOL or subschema items of type USAGE IS DATABASE-KEY-LONG must be adapted accordingly.

Adapting the LINKAGE SECTION

The LINKAGE SECTION in a called COBOL program describes and uses a data structure that is defined outside the program in another called program.

If a subschema or COBOL item of type USAGE IS DATABASE-KEY is addressed in the LINKAGE SECTION of an application program, the following must be checked:

- whether and how (if relevant) this item is supplied with values,
- whether this item is supplied with values originating from records of other databases,
- whether a type conversion is performed when assigning the value.

If the item defined in the called program was changed to the type USAGE IS DATABASE-KEY-LONG, and if the need to pass database key values from the extended value range cannot be excluded in the future, the local definition must also be changed to DATABASE-KEY-LONG.

As a rule, the LINKAGE-SECTION refers to a data item defined in the calling program by means of the COPY statement. It is therefore sufficient to recompile the called program in order to have the data type changes in it copied to the calling program.

8.5.4 Adapting additional locations in the application program

The application program must be adapted for the use of extended database key values even in areas where it is not immediately obvious that such database key values are being used. This includes the following cases:

- The application program gets database key values from user data.
- The application program transfers database key values between two or more record types.
- The application program uses database key values across databases. In this case, changes may be required in the databases involved (see "Criteria for conversion") in addition to those needed in the application program.
- The application program manages set connections via database key values.
- The application program uses database key values
  - to directly influence the management of records of this record type
  - to remember links and dependencies.
- The application program uses a COBOL item that is not of type USAGE IS DATABASE-KEY or USAGE IS DATABASE-KEY-LONG in order to temporarily store or process database key values.
- The application program uses a COBOL item in order to temporarily store or process database key values of different record types.
- The application program transfers database key values to database items that are not of type DATABASE-KEY or DATABASE-KEY-LONG. In this case, adaptations may be required in the database involved (see "Criteria for conversion") in addition to the changes in the application program.

8.6 Adapting SQL, IQS and KDBS applications

Adapting SQL applications

Application programs can access UDS/SQL databases using SQL.

When an SQL program processes database key values, a check must be performed to determine if the database keys involved are of type DATABASE-KEY-LONG. If this is the case, the fields of the host language in which the database key values are to be accepted must be of type CHARACTER(8).

Primary and foreign key attributes are unaffected by conversions and retain their INTEGER type.

Adapting IQS and KDBS applications

As in the case of CALL DML and COBOL DML applications, even applications of the interactive system IQS and the compatible database interface UDS-KDBS must be checked to determine whether database key items are directly accessed within them.

If an application accesses a database key item of type DATABASE-KEY-LONG, the related data fields in the application must be adapted with respect to the data type and the supplied values.

8.7 Examples of database conversions

The following examples outline the conversion process for two different application scenarios:

- Database key extension for cross-transactional use of database key values within *one* database
- Database key extensions for the use of database key values across databases in a multi-DB configuration

The examples refer to the databases TRAVEL, CUSTOMER and SHIPPING. They are based on the assumption that the TRAVEL, CUSTOMER and SHIPPING databases are initially available in 2-Kbyte format. Following the conversion procedure outlined in the examples, the TRAVEL, CUSTOMER and SHIPPING databases will be identical to the 4-Kbyte databases described in section “Sample databases”. The AP1, AP2 and AP3 application programs listed in the examples are COBOL programs.

Cross-transactional use of extended database key values

The use of database key values across transactions within *one* database represents the typical situation in which database key values are used in application programs:

1. The application program TA1 determines the database key value of the CRR of record type CSTMR by means of the DML statement ACCEPT in a transaction TA1 and stores this value in a COBOL item DBK of type USAGE IS DATABASE-KEY:

   ```
   ACCEPT DBK FROM CSTMR CURRENCY
   ```

   The record type CSTMR is defined in the schema TRAVEL-AGENCY of the TRAVEL database.
2. In a subsequent transaction TA2, the application program AP1 uses the DML statement FIND (format 1) to immediately reposition to the CSTMR record containing the database key value stored in the COBOL item DBK:

   ```
   FIND CSTMR DATABASE-KEY IS DBK
   ```

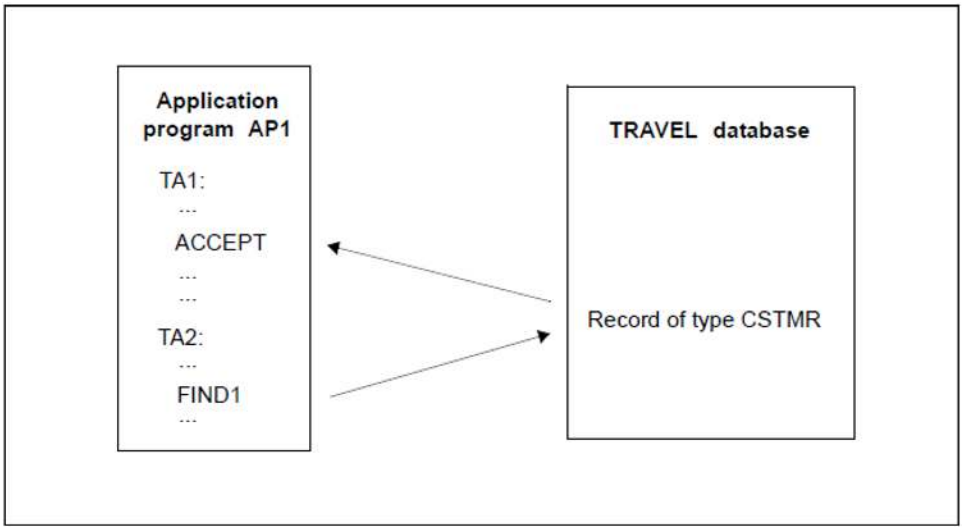


Figure 39: Cross-transactional use of database key values by an application program

A maximum of 16 777 215 ($=2^{24}-1$) records can be stored for the record type CSTMR in the 2-Kbyte TRAVEL database. In this case, however, more than 16 777 215 records will need to be stored for the record type CSTMR in the future, and these extended database key values will need to be processed by the application program AP1 accordingly. No other capacity bottlenecks are present in the TRAVEL database.

The required conversion and adaptation measures can be performed in the following steps:

1. Use the BPGSIZE utility routine to convert the TRAVEL database to the 4-Kbyte format. Then declare the converted database as the original.
2. The next step is to prepare the TRAVEL database for database operation. This means, in particular, that the BREORG utility routine must be used to extend the population for the record type CSTMR in the schema TRAVEL-AGENCY to the required value. It is now possible to store more than 16 777 215 records for the record type CSTMR in the TRAVEL database.
3. Since database key values for the record type CSTMR are not assigned by the application program (no LOCATION MODE IS DIRECT clause in the DDL record entry for the record type CSTMR), the Schema DDL (and thus the Subschema DDL) for the TRAVEL-AGENCY schema of the TRAVEL database need not be adapted, i.e. the TRAVEL database need not be restructured.
4. Existing application programs are still executable without changes so long as they do not access CSTMR records whose database key values contain an RSQ > 2^{24}-1 in the TRAVEL database. In order to enable the application program AP1 to access CSTMR records in the TRAVEL database even in cases where their database key values have an RSQ > 2^{24}-1, the declaration of the COBOL item DBK in the application program AP1 must be adapted to the data type USAGE IS DATABASE-KEY-LONG. If required, redefinitions of the DBK item must also be taken into account.
   The application program AP1 must now be recompiled and linked again.The adaptation of the application program is not chronologically dependent on the database conversion.

Database key extension in a multi-DB configuration

This example outlines a multi-DB configuration in which an application program AP2 accesses the CUSTOMER and SHIPPING databases via *one* UDS/SQL DBH. The application program AP2 contains two modules, MODULE1 and MODULE2, which exchange database key values via a globally defined COBOL item DBK of type USAGE IS DATABASE-KEY. MODULE1 accesses the CUSTOMER database; MODULE2 accesses the SHIPPING database (see figure 40):

1. MODULE1 uses the DML statement ACCEPT on the CUSTOMER database to determine the database key value of the CRR of record type CSTMR and stores this value in a globally defined COBOL item DBK of type USAGE IS DATABASE-KEY:

   ```
   ACCEPT DBK FROM CSTMR CURRENCY
   ```

   The record type CSTMR is defined in the schema CUSTOMER-CARDS of the CUSTOMER database. It is also contained in a subschema of CUSTOMER-CARDS that is used by MODULE2 (transfer to the Subschema DDL with COPY CSTMR).
2. MODULE2 supplies the CSTMR-NO item of record type CSTMR with the database key value stored in the COBOL item DBK in the record area of the UWA (User Work Area) for the SHIPPING database:

   ```
   SET CSTMR-NO TO DBK
   ```

   The record type CSTMR is defined in the MAIL-ORDERS schema of the 2-Kbyte database SHIPPING, and the item CSTMR-NO is of type DATABASE-KEY. In addition, the record type CSTMR is also contained in a subschema of MAIL-ORDERS that is used by MODULE2 (transfer to the Subschema DDL with COPY CSTMR).
3. After the remaining items of the CSTMR record have also been supplied with values in the UWA by MODULE2, this record is copied by MODULE2 to the SHIPPING database.

   ```
   STORE CSTMR [...]
   ```

   Since the entry for the record type CSTMR in the Schema DDL for the 2-Kbyte SHIPPING database contains the clause LOCATION MODE IS DIRECT CSTMR-NO OF CSTMR, the value stored in the CSTMR-NO item also serves as the database key value of the stored record.

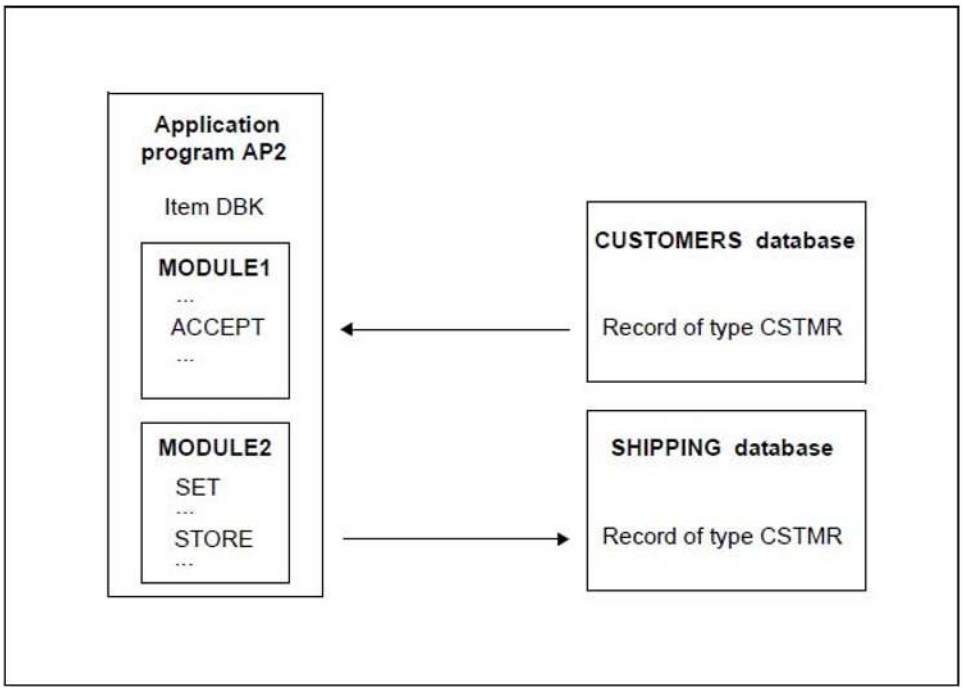


Figure 40: Use of database key values by an application program across databases

To begin with, only a maximum of 16 777 215 (=2^{24}-1) records can be stored for both the record type CSTMR in the CUSTOMER-CARDS schema of the 2-Kbyte CUSTOMER database and for the record type CSTMR in the MAIL-ORDERS schema of the 2-Kbyte SHIPPING database.

In this case, however, more than 16 777 215 records are expected in the future for the record type CSTMR in the CUSTOMER-CARDS schema of the CUSTOMER database. No other capacity bottlenecks are present in the CUSTOMER database.

The CUSTOMER and SHIPPING databases can be converted and adapted independently of one another. The following sequence is recommended:

Converting and restructuring the SHIPPING database

1. Use the BPGSIZE utility routine to convert the 2-Kbyte SHIPPING database to the 4-Kbyte_format. This conversion is required, since database key values with an RSQ > 2^{24}-1 are to be copied from the CUSTOMER database to the record type CSTMR of the SHIPPING database in the future.
   Then declare the converted database as the original.
2. You can now prepare the SHIPPING database for database operation. This means, in particular, that the BREORG utility routine must be used to extend the population for the record type CSTMR in the MAIL-ORDERS schema to the required value.
   It is now possible to store more than 2^{24}-1 records for the record type CSTMR of the MAIL-ORDERS schema in the SHIPPING database.
3. The next step is to restructure the SHIPPING database:

   - Change the LOCATION MODE IS DIRECT CSTMR-NO OF CSTMR clause in the record entry of the record type CSTMR in the MAIL-ORDERS schema to LOCATION MODE IS DIRECT-LONG CSTMR-NO OF CSTMR. The data type of the CSTMR-NO item must be change to type DATABASE-KEY-LONG.

- The subschema required by MODULE2 need not be modified, since the record type CSTMR from the MAIL-ORDERS schema is transferred to it with COPY. The subschema only needs to be recompiled. In addition, all application programs that use this subschema must also be recompiled and linked again.
- Further restructuring measures can then be performed as described in chapter “Restructuring the database (BCHANGE, BALTER)”.

Converting the CUSTOMER database

1. Use the BPGSIZE utility routine to convert the 2-Kbyte CUSTOMER database to the 4-Kbyte format. Then declare the converted database as the original.
2. You can now prepare the CUSTOMER database for database operation. This means, in particular, that the BREORG utility routine must be used to extend the population for the record type CSTMR in the CUSTOMER-CARDS schema to the required value.
   It is now possible to store more than 2^{24}-1 records for the record type CSTMR of the CUSTOMER-CARDS schema in the CUSTOMER database.
3. No changes are required in the CUSTOMER-CARDS schema of the CUSTOMER database, i.e. the CUSTOMER database need not be restructured.

Adapting the AWP2 application program

In the application program AWP2, the declaration of the COBOL item DBK must be changed to type USAGE IS DATABASE-KEY-LONG, while also taking redefinitions of the DBK items into account. The AWP2 application program must then be recompiled and linked again.

Alternative method: converting and restructuring independently

An alternative method to the one above is to convert the databases with BPGSIZE and to restructure them independently at some other time, i.e.:

1. convert the SHIPPING database with BPGSIZE, but differ restructuring to a later stage
2. convert the CUSTOMER database with BPGSIZE, but differ restructuring to a later stage
3. restructure the SHIPPING database and adapt application programs
4. restructure the CUSTOMER database and adapt application programs

Normal database operation is possible between the individual steps outlined above. As far as possible, steps 3.) and 4.) should be combined and performed at the same time, since at least some of the application programs will need to be adapted and compiled for both 3.) and 4.) application program.

9 Migrating databases to DB Layout Version 4 (BTRANS24)

To permit version migration from UDS/SQL V2.0 up to and including V2.3 to a version V2.4 or higher the databases must be converted to Database Layout Version '004.00' using the BTRANS24 utility routine.

Databases which were last modified using UDS/SQL versions up to and including V2.3 must also be migrated to Database Layout Version '004.00' in this way before they are processed - for example for auditing purposes using UDS/SQL version V2.4 or higher. When migration takes place, the Realm Layout Version of all realms is converted to '004.00'.

The BTRANS24 utility routine is a component part of the UDS-SQL-T package and is by default contained in the SIPPRG.UDS-SQL-T.029 library.

The BTRANS24 utility routine provides two functions for migrating the DB Layout Version:

- It checks prerequisites for database migration (see section “Checking the prerequisites for migration”)
- It transforms the databases (see section “Performing a database transformation with BTRANS24”)

For a description of the BTRANS24 statements, see section “BTRANS24 statements”.

For information on calling BTRANS24, see section “Calling BTRANS24”.

The messages output by BTRANS24 can be found in the “Messages” manual.

9.1 Checking the prerequisites for migration

You can use the utility routine BTRANS24 to check whether the prerequisites for transforming a database to the version '004.00' are fulfilled.

You can explicitly ask for a check to be performed using the CHECK-DATABASE statement. In this case, the check can be performed in parallel with the live DBH session. In addition, a check run is always performed implicitly before the database is transformed (TRANSFORM-DATABASE statement).

The following prerequisites must be fulfilled before migration takes place:

- The databases to be transformed must be original databases.
- The databases to be transformed must have database layout version '002.00' or '003.00', i.e. they must at least be usable in UDS/SQL V2.0.
  If this is not the case then the databases must first be transformed using the conversion utility routines from the corresponding UDS/SQL versions (BTRANS20 for use in UDS/SQL V2.0; BTRANSDB for use in UDS/SQL V1.2).
- An empty database page must be available in the corresponding realm for each DBTT. This is used to store the initial state of the DBTT anchor table.
  If this condition is not satisfied, you must provide sufficient space in the realm by performing offline realm extension using BREORG **Version 2.3**.
  The BREORG utility routine Version 2.3 is also available in the current UDS/SQL version. It is a component part of the UDS-SQL-T package and is by default contained in the SIPPRG.UDS-SQL-T.029 library. There is no start command for BREORG V2.3; the utility routine can be started using the following command:

  `/START-EXECUTABLE-PROGRAM FROM-FILE=(LIB=`*`UDS/SQL-T-module-library`*`,ELEM=BREORG)`

  Free pages must also be available in the DBDIR (8 pages) and DBCOM (29 pages) for the DBTTs of PRIVACY-AND-IQF schema and compiler schema.
- All the realms in the database must be accessible and attached - in particular DBCOM.
- COSSD and HASHLIB are not needed for the transformation.

9.2 Performing a database transformation with BTRANS24

The actual database transformation using BTRANS24 is performed offline on a consistent database state. Transformation must be performed under the user ID of the database. You start database transformation with the TRANSFORM-DATABASE statement.

Check run

A check is implicitly conducted before transformation. Only if this check indicates that all the prerequisites for database transformation are fulfilled are the modifications made. If any prerequisites are not satisfied then the database is not transformed. The database continues to be consistent. The database is also consistent if BTRANS24 aborts for any reason whatsoever up to this point.

Logging

To facilitate version migration from UDS/SQL V2.3 and earlier versions, all the changes made by BTRANS24 are performed with Alogging provided that this is active. Like all modifying utility routines that support Alogging, BTRANS24 first generates a checkpoint for AFIM logging. If recovery is necessary, for example because of a disk crash in a subsequent session with a version higher than UDS/SQL V2.3, then the database can be restored to the state of the last save copy using the following tools and resources:

- the BMEND in UDS/SQL V2.3 or the version previously used by means of UPDATE-DATABASE...DEADLINE=BREAKPOINT
- the ALOG files created before conversion
- the BTRANS24 ALOG file
- the ALOG files created during subsequent operation with a version higher than UDS/SQL V2.3

Transformation

BTRANS24 makes the following changes:

- A new DB layout version that is incompatible with earlier versions is entered in the DBDIR.
- The required DBTT anchor pages are created in the realms.
- For each record type, the ACTKEY of the first new DBTT- anchor table page is entered in the SIA.
- The new DB_LAYOUT_VERSION is entered in the ACT_KEY0 and ACT_KEYN of each realm.
- If there is a realm present with realm layout version '002.00' then the administration tables for FPA extents are created in ACT_KEY0. Any unused data areas belonging to ACT_KEY0 are deleted.
- Data areas are initialized in ACT_KEY0 for the restartable monitoring of online DBTT extensions

Overall, only very few changes are required in the database. As a result, the utility routine BTRANS24 executes quickly.

The utility routine cannot be restarted during the modification phase. In the event of a crash, it is necessary to recommence from a backup state possibly updated on the basis of ALOG files.

The COSSD and the SSIAs in the DBDIR are not modified. The criterion for subschema modification does not change. It is therefore not necessary to recompile or relink applications.

9.3 BTRANS24 statements

The following SDF statements exist for the utility routine BTRANS24

Statement	Meaning
CHECK-DATABASE	Start check run
TRANSFORM-DATABASE	Transform database
END	End statement input and start execution

Table 51: Statements for BTRANS24

You must specify precisely one of the two statements CHECK-DATABASE or TRANSFORM-DATABASE. If you specify more than one correctly formulated statement then the last statement issued is effective.

CHECK-DATABASE (start check run)

You use the CHECK-DATABASE statement to start an explicit check run. The check run must be performed under the user ID of the database. It can be performed in parallel with the DBH session. The realms are opened for read access only.

CHECK-DATABASE
DATABASE-NAME=<dbname>

DATABASE-NAME=<dbname>
Name of the database.

CHECK-DATABASE assesses the prerequisites for database transformation and outputs the corresponding messages. Please note that this assessment may become invalid due to changes that only take effect in the database after the check has been completed.

TRANSFORM-DATABASE (transform the database)

This statement starts database transformation. Transformation must be performed under the user ID.

TRANSFORM-DATABASE
DATABASE-NAME=<dbname>

DATABASE-NAME=<dbname>
Name of the database.

The realms are opened for modification only. An implicit check run is performed first. If this reveals that the conditions for transformation are not fulfilled then transformation is aborted. Up to this point, no changes are made to the database which therefore remains consistent.

END (terminate statement input)

You use the END statement to terminate statement input. Execution is then started.

END

This statement has no operands.

9.4 Calling BTRANS24

You call BTRANS24 with the following command

```
/START-EXECUTABLE-PROGRAM FROM-FILE=(LIB=UDS/SQL-T-modlib,ELEM=BTRANS24)
```

BTRANS24 is a component part of the UDS-SQL-T package and is by default contained in the SIPPRG.UDS-SQL-T.029 library.

You can then enter the BTRANS24 statements, e.g.

```
//TRANSFORM-DATABASE DATABASE-NAME=dbname
//END
```

10 Glossary

This Glossary contains the definitions of some of the important terms and concepts used in the UDS/SQL manuals.
Terms that appear in *italics* within a particular definition have also been defined in this Glossary.
In cases where two or more terms are used synonymously, a “See” reference points to the more commonly used term in these manuals.

A

access, contending

See *contending access*.

access, direct

See *direct access*.

access, sequential

See *sequential access*.

access authorization

The rights of a specified user group with regard to access to the *database*.
Access rights are defined during live database operation using ONLINE-PRIVACY utility routine or, in offline mode, using the BPRIVACY utility routine.

access path

Means of finding a certain subset of all *records* qualified by a search query, without having to carry out a sequential search of the whole *database*.

access rights

Right of access to a *database* as defined in the BPRIVACY utility routine.

access type

Type of access, e.g. read, update etc.

act-key

(actual key) Actual address of a *page*, consisting of *realm number* and *page number*.

act-key-0 page

First *page* of a *realm*; contains general information on the realm such as

- when the realm was created,
- when the realm was last updated,
- *internal version number* of the realm,
- *system break information*
- if applicable, *warm start* information.

B

backup database

See *shadow database*.

base interface block (BIB)

(Base Interface Block) Standard interface between UDS/SQL and each individual user; it contains, among other things, the *RECORD AREA* (user records as defined in the *subschema*).

before-image

Copy of a *page* taken before its contents are updated.
The *DBH* writes before-images to the *RLOG files* during database operation before the updates are applied to the *database*. A prerequisite is that the RLOG files exist.

BFIM

See *before-image*.

BIB

See *base interface block*.

buffer pool

See *system buffer pools* and *exclusive buffer pool*.

C

CALC key

Key whose value is converted into a relative *page number* by means of a *hash routine*.

CALC page

Page of a *hash area*.

CALC SEARCH key

Secondary key. Used as *access path* for *direct access* via *hash routine*.

CALC table

Table in the direct/indirect *CALC page* whose entries point to the stored records. Each line contains:

- the *CALC key*,
- the *record sequence number*
- the displacement to the related *page index entry* (direct CALC page) or the *probable position pointer* (indirect CALC page).

CALL DML

DML that is called by various programming languages (Assembler, COBOL, FORTRAN, PASCAL, PL/1) via the CALL interface.

catalog identifier

Name of the public volume set (PVS) under which the BS2000 UDS/SQL files are stored. The catalog identifier is part of the database or file name and must be enclosed in colons: ": *catid*:".

chain

Storage mode for a *set occurrence* in which every *record* contains a pointer to the subsequent record.

Character Separated Values (CSV)

Output format in which the values are separated by a predefined character.

checkpoint

Consistency point, at which the ALOG file was changed and to which it is possible to return at any time using BMEND utility routine

check records

Elements which provide information for checking the database. They vary in length from 20 to 271 bytes.

CHECK-TABLE

Check table produced by the *DDL* compiler during *Subschema DDL* compilation, and used by the COBOL compiler and *CALL DML* to check whether the *DML* statements specified in the *application program* are permitted. It is part of the *COSSD* or *SSITAB module*.

clone pair, clone pubset, clone session, clone unit

A clone unit is the copy of an (original) unit (logical disk in BS2000) at a particular time ("Point-in-Time copy"). The TimeFinder/Clone component creates this copy optionally as a complete copy or as a "snapshot".
After they have been activated, the unit and clone unit are split; applications can access both.
The unit and clone unit together form a clone pair. TimeFinder/Clone manages this pair in what is known as a clone session.
If clone units exist for all units of a pubset, these clone units together form the clone pubset.
Details of this are provided in the manual "Introduction to System Administration" (see chapter "Related publications").

COBOL DML

DML integrated in the COBOL language.

COBOL runtime system

Runtime system; sharable routines selected by the COBOL compiler (COBOL2000 or COBOL85) for the execution of complex statements.

COBOL Subschema Directory (COSSD)

Provides the COBOL compiler with subschema information for compilation of the DB *application programs*.

common memory

Shareable memory area used by several different tasks. In UDS/SQL, it always consists of the *common pool* and the *communication pool* and, depending on the application, the SSITAB pool (see *SSITAB module*) if *CALL DML* is used.
If UDS-D is used, it also consists of the *distribution pool* and the *transfer pool*.

common pool

Communication area of the *independent DBH*. Enables *DBH* modules to communicate with each other. Contains, among other things, an input/output buffer for *pages* (*buffer pools*).

communication partners

Tasks or data display terminals.

communication pool

Communication area of the *independent DBH* for *application programs*. One of its functions is to store base interface blocks (*BIB*).

compatible database interface (KDBS)

see *KDBS*

compiler database

The *realms* and files of the *database* which are required by the UDS/SQL compiler. They are

- *DBDIR* (*Database Directory*)
- *DBCOM* (*Database Compiler Realm*)
- *COSSD* (*COBOL Subschema Directory*).

COMPILER-SCHEMA

UDS/SQL-internal *schema* of the *compiler database*.

COMPILER-SUBSCHEMA

UDS/SQL-internal *subschema* of the *compiler database*.

D

DAL

(Database Administrator Language) Comprises the commands which monitor and control a *session*.

data backup

Protection against loss of data as a result of hardware or software failure.

data deadlock

See *deadlock.*

data protection (privacy)

Protection against unauthorized access to data. Implemented in UDS/SQL by means of the schema /subschema concept and access authorization. *Access rights* are granted by means of the BPRIVACY utility routine.

database (DB)

Related data resources that are evaluated, processed and administered with the help of a *database system*.
A database is identified by the database name. An UDS/SQL database consists of the *user database* and the *compiler database*.
To prevent the loss of data, a *shadow database* may be operated together with (i.e. parallel to) the original database.

database administrator

Person who manages and controls *database* operation. The DB administrator is responsible for the utility routines and the Database Administrator Language (*DAL*).

database copy

Copy of a consistent *database*; may be taken at a freely selectable point in time.

database compiler realm (DBCOM)

Stores information on the *realms*, *records* and *sets* defined by the user in the *Schema DDL* and *Subschema DDL*.

database copy update

Updating of a *database copy* to the status of a *checkpoint* by applying the appropriate *after-images*.

database directory (DBDIR)

Contains, among other things, the *SIA*, all the *SSIAs* and information on *access rights*.

database job variable

Job variable in which UDS/SQL stores information on the status of a *database*.

database key (DB key)

Key whose value represents a unique identifier of a *record* in the *database*. It consists of the *record reference number* and the *record sequence number*. The database key values are either defined by the database programmer or automatically assigned by UDS/SQL.

database key item

Item of type DATABASE-KEY or DATABASE-KEY-LONG that is used to accommodate *database key* values.
Items of type DATABASE-KEY and DATABASE-KEY-LONG differ in terms of the item length (4 bytes / 8 bytes) and value range.

DATABASE-KEY item

See *database key item*.

DATABASE-KEY-LONG item

See *database key item*.

database page

See *page*.

DATABASE-STATUS

Five-byte item indicating the database status and consisting of the *statement code* and the *status code*.

database system

Software system that supports all tasks in connection with managing and controlling large data resources. The database system provides mechanisms for stable and expandable data organization without redundancies. They allow many users to access *databases* concurrently and guarantee a consistent data repository.

DB status file

(database status file) Contains information on the most recently reset *transactions*.
openUTM-S or, in the case of distributed processing, UDS-D/openUTM-D needs this information for a *session restart*.

DB configuration

(database configuration) The *databases* attached to a *DBH* at any one point during *session* runtime. As the result of *DAL* commands or DBH error handling, the database configuration can change in the course of a session.
At the *session start*, the DB configuration may be empty. Databases can be attached with DAL commands after the start of the session. They can also be detached during the session with DAL commands.

DBCOM

See *database compiler realm*.

DBDIR

See *database directory*.

DBH

Database Handler: program (or group of programs) which controls access to the *database(s)* of a *session* and assumes all the attendant administrative functions.

DBH end

End of the *DBH* program run. DBH end can be either a *session end* or a *session abort*.

DBH, independent

See *independent DBH*.

DB key

See *database key*.

DBH, linked-in

See *linked-in DBH*.

DBH load parameters

See *load parameters* (*DBH*).

DBH start

Start of the *DBH* program run. DBH start can be either a *session start* or a *session restart*.

DBTT

(Database Key Translation Table) Table from which UDS/SQL can obtain the *page address* (*act-key*) of a *record* and associated tables by means of the database key value.
The DBTT for the SSIA-RECORD consists only of the DBTT base. For all other record types, the DBTT consists of a base table (DBTT base) and possibly of one or more extension tables (DBTT extents) resulting from an online DBTT extension or created by BREORG.

DBTT anchor page

Page lying within the realm of the associated DBTT in which the DBTT base and DBTT extents are administered. Depending on the number of DBTT extents multiple chained DBTT anchor pages may be required for their administration.

DBTT base

see *DBTT*

DBTT extent

see *DBTT*

DBTT page

Page containing the *DBTT* or part of the *DBTT* for a particular *record type*.

DCAM

Component of the TRANSDATA data communication program.

DCAM application

Communication application using the *DCAM* communication method. A DCAM application enables communication between

- a DCAM application and terminals.
- different DCAM applications within the same or different hosts, and with *remote configurations*.
- a DCAM and a openUTM application.

DDL

(Data Description Language) Formalized language for defining the logical data structure.

deadlock

Mutual blocking of *transactions*. A deadlock can occur in the following situations:

- Data deadlock: This occurs when *transactions* block each other with *contending access*.
- Task deadlock: This occurs when a *transaction* that is holding a lock cannot release it, since no openUTM task is free. This deadlock situation can only occur with UDS/SQL-openUTM interoperation .

descending key (DESC key)

Primary key of a set. Determines the sequence of *member records* in the *set occurrences* to reflect descending key values.

direct access

Access to a *record* via an item content. UDS/SQL supports direct access via the *database key, hash routines* and *multi-level tables*.

direct hash area

See *hash area*.

distributed database

A logically connected set of data resources that is distributed over more than one UDS/SQL configuration.

distributed transaction

Transaction that addresses at least one *remote configuration*. A transaction can be distributed over:

- UDS-D,
- openUTM-D,
- UDS-D and openUTM-D.

distribution pool

Area in the *independent DBH* used for communication between *UDSCT*, *server tasks*, *user tasks* and the *master task* with regard to UDS-D-specific data. The distribution pool contains the *distribution table* and the UDS-D-specific system tables.

distribution table

Table created by UDS-D using the input file assigned in the *distribution pool*. With the aid of the distribution table, the distribution component in the *user task* decides whether a *processing chain* should be processed locally or remotely.
Assigned in the distribution table are:
subschema - database
database - configuration
configuration - host computer.

DML

Data Manipulation Language: language for accessing a UDS/SQL *database*.

dummy subtransaction

A primary *subtransaction* is created by UDS-D when the first *READY* statement in a *transaction* addresses a *remote database*.
A dummy subtransaction is used to inform the *local configuration* of the transaction so that the *database* can be recovered following an error.

duplicates header

Contains general information on a *duplicates table* or a *page* of a duplicates table, i.e.

- chaining reference to the next and previous *overflow page*
- the number of free bytes in the page of the duplicates table.

duplicates table

Special *SEARCH-KEY table* in which a key value which occurs more than once is stored only once.
For each key value, the duplicates table contains:

- a table index entry with the key value and a pointer to the associated table entry
- a table entry (DB key list), which can extend over several pages, containing the *record sequence numbers* of the *records* which contain this key value.

duplicates table, main level

Main level, Level 0. Contains a table index entry and the beginning of the associated table entry (DB key list).

dynamic set

Set which exists only for the life of a *transaction* and which stores *member records* retrieved as result of search queries.

E

ESTIMATE-REPORT

Report produced after BGSIA run. Used to estimate the size of the *user realms*.

event name

Identification used in eventing.

F

foreign key

Record element whose value matches the primary key values of another table (UDS/SQL *record type*). Foreign keys in the sense of UDS/SQL are qualified as "REFERENCES owner record type" in the member record type of a set relationship in the BPSQLSIA protocol.

FPA

See *free place administration*.

FPA base

See *free place administration*.

FPA extent

See *free place administration*.

FPA page

Free place administration page.

free place administration (FPA)

Free space is managed both at realm level (*FPA pages*) and at page and table level. Free place administration of the pages is carried out in a base table (FPA base) and possibly in one or more extension tables (FPA extents) created by means of an online realm extension or BREORG.

function code

Coding of a *DML* statement; included in information output by means of the *DAL* command DISPLAY or by UDSMON.

G

group item

Nameable grouping of *record elements*.

H

hash area

Storage area in which UDS/SQL stores data and from which it retrieves data on the basis of key values which are converted into relative *page numbers*. A hash area may contain the *record* addresses as well as the records themselves.
A *direct hash area* contains the records themselves; an *indirect hash area*, by contrast, contains the addresses of records stored at some other location.

hash routine

Module which performs *hashing*.

hashing

Method of converting a key value into a *page address*.

HASHLIB

Module library for the storage of *hash routines* for one *database*.

I

identifier

Name allocated by the database designer to an *item* that UDS/SQL creates automatically. UDS/SQL adapts item type and length to the specified item usage.

implicit set

SYSTEM set created by UDS/SQL when a *SEARCH key* is defined at record type level.

inconsistency

State of the database in which the data values contained in it are inconsistent.

independent DBH

Independent program system enabling more than one user to access a single *database* (*mono-DB operation*) or several databases (*multi-DB operation*) simultaneously. The independent DBH is designed as a task family, consisting of

- a *master task* (*UDSSQL*)
- one or more *server tasks* (*UDSSUB*)
- an *administrator task* (*UDSADM*)

index level

Hierarchy level of an *index page*.

index page

Page in which the highest (lowest) key values of the next-lower level of an indexed table are stored.

INDEX search key

Secondary key. Used as *access path* for *direct access* via a *multi-level table*.

indirect hash area

See *hash area*.

integrity

State of the database in which the data contained in it is complete and free of errors.

- entity integrity
- *referential integrity*
- user integrity

interconfiguration

Concerning at least one *remote configuration*.

interconfiguration consistency

A *distributed transaction* that has caused updates in at least one *remote configuration* is terminated in such a way that the updates are either executed on the *databases* in each participating *DB configuration* or on none at all.
Interconfiguration consistency is assured by the *two-phase commit protocol*.

interconfiguration deadlock

Situation where *distributed transactions* are mutually locked due to *contending accesses*.

interface

In software: memory area used by several different programs for the transfer of data.

internal version number

Each *realm* of the *database*, including *DBDIR* and *DBCOM*, has an internal version number which the utility routines (e.g. BREORG, BALTER) increment by one whenever a realm is updated. This internal version number is kept in the *act-key-0 page* of the realm itself and also in the PHYS VERSION RECORD in the DBDIR.

item

Smallest nameable unit of data within a *record type*. It is defined by item type and item length.

K

KDBS

Compatible database interface. Enables programs to be applied to applications of *DB systems* by different manufacturers.

key

Item used by the database programmer for *direct access* to records; an optimized *access path* is provided for the key by UDS/SQL in accordance with the *schema* definition.

key, compound

Key consisting of several *key items*.

key item

Item defined as a *key* in the *schema*.

key reference number

Keys are numbered consecutively in ascending order, beginning at 1.

L

linked-in control system

UDS/SQL component for *linked-in DBH*, responsible for control functions (corresponds to the *subcontrol system* of the *independent DBH*).

M

main reference

In the *DBH* the main reference is used to manage the resources required for processing a transaction's requests, including those for transferring the requests from the application program to the DBH and back.

mainref number

Number assigned to the *transaction* at *READY*. This number is unique only at a given time; at the end of the transaction, it is assigned to another transaction.

master task

Task of the *independent DBH* in which the *UDSQL* module executes. Controls the start and end of a *session* and communicates with the *database administrator* directly or via the *administrator task*.

member

See *member record* or *member record type*.

member, AUTOMATIC

Record is inserted at storage time.

member, MANDATORY

Record cannot be removed.

member, MANUAL

Record is not inserted automatically at storage time.

member, OPTIONAL

Record can be removed.

member record

Lower-ranking *record* in a *set occurrence*.

member record type

Lower-ranking *record type* in a *set*.

mono-DB configuration

Type of configuration where only one *database* takes part in a *session*.

mono-DB operation

Mode of *database* operation where the *DBH* uses only one database of a *configuration*.

multi-DB configuration

Type of configuration where several *databases* take part in a *session*.

multi-DB operation

Mode of database operation where the *DBH* uses several *databases* of a *configuration*.

multi-DB program

Application program that addresses more than one *database*. The databases may be part of one or more *mono-DB* or *multi-DB configurations*.

multi-level table

SEARCH KEY table which contains a line for each *record* of the associated *record type* or each *member record* of the *set occurrence*, as appropriate. Each line comprises the key value of the record and the record pointer. It is also referred to as an indexed table.

multithreading

A mechanism that enables the *DBH* to fully exploit the CPU.
Multithreading means that the DBH processes several jobs concurrently by using so-called threads. Each thread has information on the current status of a particular job stored in it. When a job needs to wait for the completion of an I/O operation, DBH uses the CPU to process some other job.

N

network

All computers linked via TRANSDATA.

O

OLTP

(Online Transaction Processing) In an OLTP application, a very large numberof users access the same programs and data. This usually occurs under the control of a transaction monitor (TP monitor).

online backup

If AFIM logging is active, the *database* can be saved during a session. The ability to save a database online is determined with the BMEND utility routine.

online DBTT extension

Extension during ongoing database operation of the number of possible records of a record type. The DAL commands ACT DBTT-INCR, DEACT DBTT-INCR, DISPLAY DBTT-INCR and EXTEND DBTT can be used to administer the online extension of DBTTs.
See also *automatic DBTT extension*.

online realm extension

Extension of *user realms* and *DBDIR* in ongoing database operation. The DAL commands ACT INCR, DEACT INCR, DISPLAY INCR, EXTEND REALM and REACT INCR are provided for administering the online extensibility of realms.
See also *automatic realm extension*.

open transaction

Transaction which has not been closed with FINISH or FINISH WITH CANCEL, or with COMMIT or ROLLBACK.

openUTM

(universal transaction monitor) Facilitates the creation and operation of transaction-oriented applications.

operator task (OT)

See *master task*

original database

The term “original database” refers solely to the naming of the database files (*dbname.dbfile*), not to the status of the database content (see also *shadow database*).

overflow page

Page in *hash areas* and *duplicates tables* for storing data that does not fit in the primary page. Their structure is the same as that of the pages of the hash area or duplicates table in question.

owner

See *owner record* or *owner record type*.

owner record

Higher-ranking *record* in a *set occurrence*.

owner record type

Higher-ranking *record type* in a *set*.

P

page

Physical subunit of a *realm*. UDS/SQL identifies pages by means of unique keys (*act-key*).
The length of a page may be optionally 2048, 4000 or 8096 bytes. All pages within a database must have the same length. Pages with a length of 4000 or 8096 bytes are embedded in a *page container*.

page address

In a page address, a distinction is made between the current address of a *page*, i.e. the *act-key*, and the probable address of a page, the *probable position pointer (PPP)*.

page container

Pages with a length of 4000 or 8096 bytes are embedded in a so-called page container, which consists of a 64-byte header that precedes the page and a 32-byte trailer at the end of the page.

page header (page info)

The first 20 bytes of a database *page* (except for the *FPA* and *DBTT pages* with a length of 2048 bytes). They contain:

- the *act-key* of the *page* itself,
- the number of *page index entries*
- the length and displacement of the bytes which are still vacant in this page.
- the page type (*ACT-Key-0 page*, *FPA page*, *DBTT page*, *DBTT anchor page*, normal data page or *CALC page*)

page index entry

Indicates the position of a *record* within a *page*.

page number

In each *realm* the *pages* are numbered consecutively in ascending order starting starting from 0. The page number is part of the *page address*.
Page number = PAM page number -1 for databases with a page length of 2048 bytes
Page number = (PAM page number-1) / 2 for databases with a page length of 4000 bytes
Page number = (PAM page number-1) / 4 for databases with a page length of 8096 bytes.

password for UDS/SQL files

Password serving to protect the files created by UDS/SQL (default: C'UDS'BLANK'').
The *DB administrator* can define other passwords with PP CATPASS or MODIFY-FILE-ATTRIBUTES.

pattern

Symbolic representation of all possible *item* contents, used at item definition.

pattern string

String defining a *pattern*.

PETA

Preliminary end of transaction: UDS-D or openUTM-D statement that causes a preliminary transaction end.
The PETA statement belongs to the first phase of the *two-phase commit protocol* which terminates a *distributed transaction*.
The PETA statement stores the following information failproof in the *RLOG file* of the local *DBH*:

- each updated *page*
- rollback and locking information
- the names of all participating *configurations*.

This information is required for any future *warm start*.

pointer array

Table of pointers to the *member records* of a *set occurrence*. Used for *sequential* and *direct access* to member records.

PPP

See *probable position pointer (PPP)*.

prepared to commit (PTC)

Part of the *two-phase commit protocol*:
State of a *subtransaction* after execution of a *PETA* statement and before receipt of the message that the complete *transaction* is to be terminated with FINISH or FINISH WITH CANCEL.

primary key

Distinguished from *secondary keys* for reasons of efficiency. Usually a unique identifier for a *record*.

R

READY

Start of a *transaction* or a *processing chain* in *COBOL DML* programs.

READYC

Start of a *transaction* or a *processing chain* in *CALL DML* programs.

realm

Nameable physical subunit of the *database*. Equivalent to a file. Apart from the *user realms* for user data there are also the realms *DBDIR* and *DBCOM*, which are required by UDS/SQL.

realm configuration

Comprises all the database *realms* taking part in a *session*.

realm copy

See *database copy*.

realm reference number

Realms are numbered consecutively in ascending order, starting with 1. The realm reference number (area reference) is part of the *page address*.

reconfiguration

Regrouping of databases in a *DB configuration* after a *session abort*. A prerequisite for reconfiguration is that the *SLF* has been deleted or that its contents have been marked as invalid.

record

Single occurrence of a *record type*; consists of one item content for each of the *items* defined for the record type and is the smallest unit of data managed by UDS/SQL via a unique identifier, the *database key*.
The reserved word RECORD is used in DDL and SSL syntax to declare a record type.

record address

Address of the page containing the *record*. See *page address*.

RECORD AREA

Area in the *USER WORK AREA* (*UWA*) which can be referenced by the user. The record area contains the *record types* and the implicitly defined items (IMPLICITLY-DEFINED-DATA-NAMES) of the database such as the AREA-ID items of the WITHIN clauses of the schema. The length of the record area is essentially defined by the record types contained in it.

record element

Item, *vector* or *group item*.

S

schema

Formalized description of all data structures permitted in the *database*. A UDS/SQL schema is defined by means of the *Schema DDL*.

Schema DDL

Formalized language for defining a *schema*.

T

table, multi-level

See *multi-level table*.

table (SQL)

A table in the context of *SQL* corresponds to a UDS/SQL *record type*.

table header

Contains general information on a table or *table page*:

- the table type and the level number of the table page,
- the number of reserved and current entries in this table page,
- the chaining reference to other table pages on the same level,
- the pointer to the associated table page on the next higher level,
- the pointer to the page containing the last table on the main level (for the highest-level table only).

table page

Page containing a table or part of a table. If a *table* which does not extend over several pages or the highest level of a multi-level *table* is concerned, "table page" only refers to the object involved, not the entire *page*.

TANGRAM

(Task and Group Affinity Management) Subsystem of BS2000 that plans the allocation of processors for task groups which access large quantities of shared data in multi-task applications.

task attribute TP

There are 4 task attributes in BS2000: SYS, TP, DIALOG and BATCH.
Special runtime parameters that are significant for task scheduling are assigned to each of these task attributes.
In contrast to the other task attributes, the TP attribute is characterized by optimized main memory management that is specially tailored to transaction processing requirements.

task communication

Communication between the *DBH modules*. See also *common pool*.

task deadlock

See *deadlock*.

task priority

In BS2000, it is possible to define a priority for a task. This priority is taken into account when initiating and activating the task.
Priorities may be fixed or variable. Variable priorities are adapted dynamically; fixed priorities do not change.
Note that UDS/SQL server tasks should be started with a fixed priority in order to ensure consistent performance.

TCUA

See *Transaction Currency Area.*

time acknowledgment

Message sent by the *UDS-D task* to the remote *application program* to indicate that there is still a *DML* statement being processed.

transaction (TA)

Related sequence of *DML* statements which is processed by UDS/SQL either as a whole or not at all. This method ensures that the *database(s)* is/are always in a consistent state.
For UDS-D:
The total set of *subtransactions* active at a given time.

U

UDSADM

Module of the *independent DBH*; executes in the *administrator task*.

UDSHASH

Module generated by the BGSIA utility routine. It contains the names of all the *hash routines* defined in the *Schema DDL*.

UDSNET

Distribution component in the *user task*.

UDSSQL

Start module of the *independent DBH*; executes in the *master task*.

UDSSUB

Start module of the *independent DBH*; executes in the *server task*.

UDS-D task UDSCT

Task started for each *configuration* by UDS/SQL so that it can participate in distributed processing with UDS-D.

UDS/SQL / openUTM-D consistency

A *transaction* that has updated both openUTM data and UDS/SQL *databases* is terminated in such a way that the openUTM data and the UDS/SQL databases are either updated together or not at all.

UDS/SQL pubset declaration

Declaration in a *pubset declaration job variable* for restricting the UDS/SQL pubset environment. This reduces or prevents the risk of file names being ambiguous.

unique throughout the network

Unique in all the computers that are included in the *network*.

user database

The *realms* and files of the *database* required by the user in order to be able to store data in, and to retrieve data from a database are:

- the *Database Directory* (*DBDIR*),
- the *user realms*
- the module library for *hash routines* (*HASHLIB*).

user realm

A *realm* defined in the realm entry of the *Schema DDL*. It contains, among other things, the user records.

user task

Execution of an *application* program or openUTM program, including the parts linked by the system.

USER-WORK-AREA (UWA)

Transfer area for communication between the *application program* and the *DBH*.

UTM

See openUTM.

UWA

See *USER-WORK-AREA (UWA)*.

V

vector

Item with repetition factor. The repetition factor must be greater than 1. It specifies how many duplicates of the item are combined in the vector.

version number, internal

See *internal version number*.

W

warm start

A warm start is performed by UDS/SQL if an inconsistent *database* is attached to a *session*.
For UDS/SQL this involves applying all updates of completed *transactions* to the database which have not yet been applied, *rolling back* all database transactions that are open, and making the database consistent. The related *RLOG file* and the *DB status file* are required for a warm start.

11 Abbreviations

ACS	Alias Catalog Service
Act-Key	ACTual KEY
AFIM	AFter-IMage
AP	Application Program
ASC	ASCending
BIB	Base Interface Block
BFIM	BeFore-IMage
COBOL	COmmon Business Oriented Language
CODASYL	COnference on DAta SYstem Languages
CRA	CuRrent of Area
CRR	CuRrent of Record
CRS	CuRrent of Set
CRU	Current of RunUnit
COSSD	COBOL SubSchema Directory
DAL	Database Administration Language
DB	DataBase
DBCOM	DataBase COmpiler Realm
DBDIR	DataBase DIRectory
DBH	DataBase Handler
DB-Key	DataBase Key
DBTT	DataBase key Translation Table
DDL	Data Description Language
DESC	DESCending
DML	Data Manipulation Language
DRV	Dual Recording by Volume
DSA	Database System Access
DSSM	Dynamic SubSystem Management
FC	Function Code
FPA	Free Place Administration
GS	Global Storage
HSMS	Hierarchic Storage Management System

ID	IDentification
IQL	Interactive Query Language
IQS	Interactive Query System
KDBS	Kompatible Datenbank-Schnittstelle (= compatible database interface)
KDCS	Kompatible Datenkommunikationsschnittstelle (= compatible data communications interface)
LM	Lock Manager
LMS	Library Maintenance System
MPVS	Multiple Public Volume Set
MR-NR	MainRef NumbeR
MT	Master task
OLTP	OnLine transaction processing
openUTM	Universal Transaction Monitor
OT	Operator Task
PETA	Preliminary End of TrAnsaction
PPP	Probable Position Pointer
PTC	Prepared To Commit
PTT	Primäre Teiltransaktion (= primary subtransaction)
PVS	Public Volume Set
REC-REF	RECord REFerence number
RSQ	Record Sequence Number
SC	SubControl
SCD	Set Connection Data
SCI	Software Configuration Inventory
SECOLTP	SECure OnLine Transaction Processing
SECOS	SEcurity COntrol System
SET-REF	SET-REFerence
SIA	Schema Information Area
SIB	SQL Interface Block
SLF	Session Log File
SQL	Structured Query Language
SSD	Solid State Disk
SSIA	SubSchema Information Area

SSITAB	SubSchema Information TABle
SSL	Storage Structure Language
ST	ServerTask
STT	Sekundäre Teiltransaktion (= secondary subtransaction)
TA	TrAnsaction
TA-ID	TrAnsaction IDentification
TANGRAM	TAsk aNd GRoup Affinity Management
TCUA	Transaction CUrrency Area
UDS/SQL	Universal Database System/Structured Query Language
UWA	User Work Area

12 Related publications

You will find the manuals on the internet at http://bs2manuals.ts.fujitsu.com. You can order printed copies of those manuals which are displayed with an order number.

UDS/SQL (BS2000)
Application Programming
User Guide

UDS/SQL (BS2000)
Database Operation
User Guide

UDS/SQL (BS2000)
Design and Definition
User Guide

UDS/SQL (BS2000)
Messages
User Guide

UDS/SQL (BS2000)
Recovery, Information and Reorganization
User Guide

UDS/SQL (BS2000)
Ready Reference

UDS (BS2000)
Interactive Query System IQS
User’s Guide

UDS-KDBS (BS2000)
Compatible Database Interface
User Guide

SQL for UDS/SQL
Language Reference Manual

BS2000 OSD/BC Commands
User Guide

BS2000 OSD/BC
Introduction to System Administration
User Guide

BS2000 OSD/BC
Executive Macros
User Guide

BS2000 OSD/BC
Introductory Guide to DMS
User Guide

SDF (BS2000)
SDF Dialog Interface
User Guide

SORT (BS2000)
User Guide

SPACEOPT (BS2000)
Disk Optimization and Reorganization
User Guide

LMS (BS2000)
SDF Format
User Guide

DSSM/SSCM Subsystem Management in BS2000
User Guide

ARCHIVE (BS2000)
User Guide

DRV (BS2000)
Dual Recording by Volume
User Guide

HSMS / HSMS-SV (BS2000)
Hierarchical Storage Management System
Volume 1: Functions, Management and Installation
User Guide

SECOS (BS2000)
Security Control System
User Guide

openNet Server (BS2000)
BCAM
Reference Manual

DCAM (BS2000)
Program Interfaces
Reference Manual

DCAM (BS2000)
Macros
User Guide

OMNIS/OMNIS-MENU (BS2000)
Functions and Commands
User Guide

OMNIS/OMNIS-MENU (BS2000)
Administration and Programming
User Guide

openUTM
Concepts and Functions
User Guide

openUTM
Programming Applications with KDCS for COBOL, C and C++
User Guide

openUTM
Generating Applications
User Guide

openUTM
Administering Applications
User Guide

openUTM
Using openUTM Applications under BS2000
User Guide

openUTM
Messages, Debugging and Diagnostics in BS2000
User Guide

COBOL2000 (BS2000)
COBOL Compiler
Reference Manual

COBOL2000 (BS2000)
COBOL Compiler
User’s Guide

COBOL85 (BS2000)
COBOL Compiler
Reference Manual

COBOL85 (BS2000)
COBOL Compiler
User’s Guide

CRTE (BS2000)
Common Runtime Environment
User Guide

DRIVE/WINDOWS (BS2000)
Programming System
User Guide

DRIVE/WINDOWS (BS2000)
Programming Language
Reference Guide

DRIVE/WINDOWS (BS2000)
System Directory of DRIVE Statements
Reference Manual

DRIVE/WINDOWS (BS2000/SINIX)
Directory of DRIVE SQL Statements for UDS
Reference Manual

DAB (BS2000)
Disk Access Buffer
User Guide

Unicode in BS2000
Introduction

XHCS (BS2000)
8-Bit Code and Unicode Processing in BS2000
User Guide

BS2000 OSD/BC
Softbooks English
DVD

openSM2 (BS2000)
Software Monitor
User Guide

SNMP Management (BS2000)
User Guide