

UDS/SQL V2.5

Design and Definition

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:

manuals@fujitsu-siemens.com

Certified documentation according to DIN EN ISO 9001:2000

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 9001:2000.

cognitas. Gesellschaft für Technik-Dokumentation mbH

www.cognitas.de

Copyright and Trademarks

Copyright © Fujitsu Siemens Computers GmbH 2007.

All rights reserved.

Delivery subject to availability; right of technical modifications reserved.

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

Contents

1	Preface	9
1.1	Brief product description	9
1.2	Target group	10
1.3	Summary of contents	11
1.4	Readme file	16
1.5	Changes since the last version	17
1.6	Notational conventions	22
1.6.1	Warnings and notes	22
1.6.2	Non-SDF notational conventions	22
2	General information	25
2.1	Modern data organization	25
2.2	Data models	28
2.2.1	The CODASYL model	28
2.2.2	Relational model	31
2.2.3	Relative merits of the data models	35
2.2.4	Coexistence of the CODASYL and relational models	36
2.3	Universal Database System UDS/SQL	41
3	Designing the database	43
3.1	Data modeling	43
3.2	Distributing the data	45

3.3	Technical implementation	50
3.3.1	Defining the logical structure of a UDS/SQL database	50
3.3.2	Defining the physical structure of a UDS/SQL database	51
3.3.3	Views	51
4	Schema DDL	53
4.1	Introduction	53
4.2	Defining an item	55
4.2.1	Defining an unpacked numeric item	56
4.2.2	Defining a packed numeric item	58
4.2.3	Defining a binary item	59
4.2.4	Defining an alphanumeric item of fixed length	60
4.2.5	Defining an alphanumeric item of variable length	61
4.2.6	Defining a national item (UTF-16)	63
4.2.7	Defining a database key item	64
4.3	Defining a vector	65
4.4	Defining a repeating group	66
4.5	Grouping record elements to form a record type	68
4.6	Linking the records of two record types to form a set	70
4.6.1	Defining a set	70
4.6.2	Defining the type of membership of records in a set	79
4.7	Access paths and record sequences	84
4.7.1	Direct and sequential access on record type level via database key value	85
4.7.2	Generating additional access paths for direct access on record type level	87
4.7.3	Determining the order of records within a set occurrence	94
4.7.4	Generating additional paths for direct access on set level	100
4.7.5	Determining set occurrence selection	103
4.8	Special sets	105
4.8.1	SYSTEM set	105
4.8.2	Dynamic set	106
4.9	Assigning names to hash areas and tables	107
4.10	The realm concept	108
4.10.1	Defining a realm	109
4.10.2	Defining allocation of records to realms	110
4.10.3	Temporary realms	110

4.11	Assigning name and privacy to a schema	111
4.12	Comprehensive example of DDL application	112
4.13	Reserved words of the DDL compiler	123
5	SSL	129
<hr/>		
5.1	Introduction	129
5.1.1	Methods of physical representation of the logical data structure	130
5.1.2	DBTT (Database Key Translation Table)	131
5.2	Declaring the population	136
5.2.1	Specifying the number of records in one record type	136
5.2.2	Specifying the size of the set occurrences of a set	140
5.2.3	Overview of the initial sizes for storage space reservations	143
5.3	Determining the linkage of records	144
5.3.1	Determining the storage mode for set occurrences	144
5.3.2	Assessing pointer array, list and chain	153
5.3.3	Preventing redundancy in SEARCH key tables	157
5.3.4	Adding a pointer to link a member to its owner	159
5.4	Defining the placement of member records, tables and hash areas	160
5.4.1	Defining the placement of member records, associated tables and hash areas for secondary keys	160
5.4.1.1	Placement at realm level	161
5.4.1.2	Placement within a realm	163
5.4.2	Defining the placement of record SEARCH key table, DBTT and record hash areas	168
5.4.3	Overview of placement statements	170
5.5	Defining the extent of table reorganization desired	172
5.6	Storing the records of a record type in compressed form	176
5.7	Formulas for calculating the storage space requirements for records and tables	177
5.8	Comprehensive example of SSL application	179
5.9	Reserved words of the SSL compiler	182

6	Definition of the user interface to the database	183
6.1	Subschema DDL	183
6.1.1	Introduction	183
6.1.2	Assigning name and privacy to a subschema	184
6.1.3	Unlocking a schema for creating a subschema	184
6.1.4	Copying entire record types from the schema into the subschema	185
6.1.5	Copying part of a record type from the schema into the subschema	185
6.1.6	Copying sets from the schema into the subschema	193
6.1.7	Copying realms from the schema into the subschema	194
6.1.8	Comprehensive example of subschema DDL	195
6.2	Relational schema	196
7	Structure of pages	197
7.1	Page container	199
7.2	Act-key-0 and act-key-N page	200
7.3	FPA page	202
7.4	DBTT pages	205
7.4.1	DBTT anchor page	205
7.4.2	DBTT page	207
7.5	Direct CALC page	210
7.6	Indirect CALC page	213
7.7	Data page	215
8	Structure of records and tables	219
8.1	Structure of records	219
8.2	Structure of tables	224

9	Reference section	233
<hr/>		
9.1	Schema DDL syntax	236
9.1.1	Schema entry	237
9.1.2	Realm entry	237
9.1.3	Record entry	238
9.1.4	Set entry	243
9.2	SSL syntax	248
9.2.1	Schema entry	248
9.2.2	Record entry	249
9.2.3	Set entry	254
9.3	Subschema DDL syntax	258
9.3.1	IDENTIFICATION DIVISION	259
9.3.2	AREA SECTION	259
9.3.3	RECORD SECTION	260
9.3.4	SET SECTION	261
	Glossary	263
<hr/>		
	Abbreviations	305
<hr/>		
	References	309
<hr/>		
	Index	315
<hr/>		

1 Preface

1.1 Brief product description

The **Universal Database System 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 or via an ODBC interface.

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 via ODBC servers.

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

1.2 Target group

This manual is intended to support database designers in designing the logical and physical structure of their database and describing it with DDL and SSL. Furthermore, the manual explains how to make database data available to users in the form they require by using the subschema DDL to create suitable subschemas.

The language descriptions are also intended for database application programmers and the database administrator.

The general section contains information which is of interest to any user intending to work with UDS/SQL.

1.3 Summary of contents

What does this manual contain?

The “General” chapter contains general information on maintaining data and using databases and explains the various database models and the differences between them. The chapter is concluded by a general overview of UDS/SQL.

The chapter “Designing a database” provides an overview of the phases of database design and presents options for distributing the data. This chapter concludes with a section that briefly describes the technical implementation of the conceptual schema in a UDS/SQL database.

The chapter “Schema DDL” deals with the data definition language (DDL) used to define the logical data structure.

The chapter “SSL” describes the storage structure language (SSL), which can be used to optimize the physical storage of the data.

The chapter “Defining the user interface to the database” describes the possibilities afforded by the subschema DDL and via the relational schema.

The chapters “Structure of pages” and “Structure of records and tables” provide information for users who are interested in special details.

The “Reference” chapter summarizes the syntax of schema DDL, SSL and subschema DDL.

Using the manuals

The “Guide through the manuals” section explains which manuals and which parts of the manuals contain the information you require. A glossary gives brief definitions of the technical terms used in the text.

In addition to using the table of contents, you can find answers to your 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 [page 15](#).

For a basic introduction you 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 your previous knowledge and tasks. [Table 1](#) serves as a guide to help you find your way through the manuals.

Examples

If your task is to write COBOL DML programs, you 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 you 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 programming	Creation and re-structuring	Administration and operation	Working with openUTM	Working with IQS	Working with UDS-D

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	–	–	–	–	–	–

Table 1: Guide through the manuals

(part 1 of 3)

Contents of the five main manuals	User task							
	Design and definition	COBOL/CALL DML programming	SQL programming	Creation and restructuring	Administration and operation	Working with openUTM	Working with IQS	Working with UDS-D

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	–	–	D
DBH load parameters	–	–	–	–	L	–	–	D
Administration	–	–	–	–	L	–	–	D
High availability	–	–	–	–	B	–	–	–
Extending realms during database operation	D	–	–	–	B	–	–	–
Saving and recovering a database in the event of errors	D	–	–	D	L	D	–	D
Optimizing performance	–	–	–	–	D	–	–	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	–	D
Function codes of DML statements	–	D	–	–	D	–	–	–

Table 1: Guide through the manuals

(part 2 of 3)

Contents of the five main manuals	User task							
	Design and definition	COBOL/ CALL DML programming	SQL programming	Creation and re-structuring	Administration and operation	Working with openUTM	Working with IQS	Working with UDS-D

Manual**UDS/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	–	–	–
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	D
UDS/SQL System Reference Guide	S	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

(part 3 of 3)

¹ only for COBOL-DML

B → provides basic information if you have no experience of UDS/SQL

L → helps you 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. The following distinction is made: (see “Application Programming” manual, CONNECT) advises you to look up CONNECT in the “Application Programming” manual.

The complete titles of the manuals can be found under References.

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.

For information on accessing databases with SQL statements, refer to the “ESQL COBOL” manual and the various “DRIVE” manuals.

1.4 Readme file

Information on functional changes and additions to the current product version described in this manual can be found in the product-specific README file. You will find the README file on your BS2000 computer under the file name *SYSRME.product.version.language*. The user ID under which the README file is cataloged can be obtained from your system support. You can view the README file using the /SHOW-FILE command or an editor, and print it out on a standard printer using the following command:

```
/PRINT-DOCUMENT filename, LINE-SPACING=*BY-EBCDIC-CONTROL
```

SPOOL version smaller than 3.0A:

```
/PRINT-FILE FILE-NAME=filename, LAYOUT-CONTROL=  
PARAMETERS(CONTROL-CHARACTERS=EBCDIC)
```


1.5 Changes since the last version

The main changes introduced in UDS/SQL V2.5 in comparison with Version 2.4 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
START-UDS commands - Syntax description		
Syntax description	DBO	2, 12
	CRE	2
Restriction of the pubset environment		
In the case of DBH startup	DBO	2
Information on compatibility with details of the load parameters PP LOG/LOG-2/RESERVE	DBO	3
Specifying a UDS/SQL pubset declaration	DBO	9
Information on compatibility with details of the BMEND statement START-LOG	DBO	9
New DAL command NEW PUBSETS (checks and notes new UDS/SQL pubset declaration)	DBO	4
DAL command DISPLAY PUBSETS (displays the UDS/SQL pubset declaration)	DBO	4
Assignment of the COSSD file for compiling a COBOL-DML program	APP	6
SM pubsets for logging files		
Extension of the START-LOG statement (BMEND utility routine) by operands for log files on SM pubsets	RIR	2
Changed load parameters PP LOG/LOG-2/RESERVE with vsn specification	DBO	3
Changed DAL commands MODIFY LOG/LOG-2/RESERVE with vsn specification	DBO	4

Table 2: Changes in Version 2.5 compared to Version 2.4

(part 1 of 5)

Topic	Manual	Chapter
Session job variable Additional information	DBO	9
Database job variable New job variable with information on the status of a database	DBO	9
Renaming database objects		
New utility routine BRENAME for renaming database objects	CRE	7
New statement RENAME in the BGSIA utility routine	CRE	3
Automatic realm extension by means of utility routines		
Concept	DBO	6
Information on preparations in the database structure	CRE	3
Unicode support		
Concept	DBO	9
Description of the new SQL data types	RIR	5
New item types for the LOOKC function	APP	8
Definition of national items	DES	4, 6, 9
Additional reserved words of the DDL compiler	DES	4
Modified load parameters		
PP LOG/LOG-2/RESERVE with vsn specification for pubsets	DBO	3
Modified DAL commands		
DISPLAY PUBSETS (displays the UDS/SQL pubset declaration)	DBO	4
MODIFY LOG/LOG-2/RESERVE with vsn specification for pubsets	DBO	4
New DAL commands		
NEW PUBSETS - checks and notes new UDS/SQL pubset declaration	DBO	4
BALTER utility routine		
Information on automatic realm extension and UDS/SQL pubset declaration	CRE	6, 7
Use in the renaming cycle	CRE	7
Additional format of the FILLING statement: WITH POPULATION	CRE	6
BCALLSI utility routine		
Information on UDS/SQL pubset declaration	CRE	3

Table 2: Changes in Version 2.5 compared to Version 2.4

(part 2 of 5)

Topic	Manual	Chapter
BCHANGE utility routine		
Delimitation from renaming of database objects	CRE	6
Information on automatic realm extension and UDS/SQL pubset declaration	CRE	6
BCHECK utility routine		
Information on UDS/SQL pubset declaration	RIR	3
Modified message texts because of the new error classification (MINOR, STRUCTURAL, SERIOUS)	RIR	3
BCREATE utility routine		
Information on automatic realm extension and UDS/SQL pubset declaration	CRE	3
BFORMAT utility routine		
Information on UDS/SQL pubset declaration	CRE	3
BGSIA utility routine		
Information on automatic realm extension and UDS/SQL pubset declaration	CRE	3
New statement RENAME for renaming database objects	CRE	3
BGSSIA utility routine		
Information on automatic realm extension and UDS/SQL pubset declaration	CRE	3
BINILOAD utility routine		
Information on UDS/SQL pubset declaration	CRE	5
BMEND utility routine		
Information on UDS/SQL pubset declaration	RIR	2
Extension of the START-LOG statment by operands for log files on SM pubsets	RIR	2
BMODTT utility routine		
Information on UDS/SQL pubset declaration	RIR	9
BOUTLOAD utility routine	CRE	
Information on UDS/SQL pubset declaration	CRE	5
BGPSIZE utility routine		
Information on UDS/SQL pubset declaration	CRE	8

Table 2: Changes in Version 2.5 compared to Version 2.4

(part 3 of 5)

Topic	Manual	Chapter
BPRECORD utility routine		
Information on UDS/SQL pubset declaration	RIR	7
BPRIVACY utility routine		
Information on automatic realm extension and UDS/SQL pubset declaration	CRE	4
BPSIA utility routine		
Information on UDS/SQL pubset declaration	RIR	4
BPSQLSIA utility routine		
Information on UDS/SQL pubset declaration	RIR	5
Description of all SQL data types, including the new one for supporting Unicode	RIR	5
BRENAME utility routine		
New utility routine for initiating a renaming cycle	CRE	7
Use of DDL, SSL, BGSIA, BALTER and BGSSIA in the renaming cycle	CRE	7
Information on automatic realm extension and UDS/SQL pubset declaration	CRE	7
BREORG utility routine		
Information on automatic realm extension and UDS/SQL pubset declaration	RIR	8
BSTATUS utility routine		
Information on UDS/SQL pubset declaration	RIR	6
BTRANS24 utility routine		
Changed requirements for executing the utility routine	CRE	9
ONLINE-PRIVACY utility routine		
Information on automatic realm extension	CRE	4
DDL compiler		
Information on automatic realm extension and UDS/SQL pubset declaration	CRE	3
SSL compiler		
Information on automatic realm extension and UDS/SQL pubset declaration	CRE	3

Table 2: Changes in Version 2.5 compared to Version 2.4

(part 4 of 5)

Topic	Manual	Chapter
Messages		
New messages UDS0746 - UDS0755	MSG	2
Modified message UDS0728	MSG	2
Additional inserts in the case of UDS0209, UDS0353, UDS0700, UDS0711, UDS0720	MSG	2
New messages of the utility routines: 0045 - 0052, 0055, 0064, 0065, 0084, 1004 - 1007, 1034, 1059, 1060, 1063, 1064, 1091 - 1096, 1317, 2006, 2018, 2037 - 2039, 2926, 2927, 2933, 2934, 2979, 3026, 3608, 4049	MSG	3
Modified messages of the utility routines: 0073, 0074, 1309, 2040, 2507, 3607, 3630, 3639, 3643, 3651, 3652, 3655, 3661 - 3667	MSG	3
New DDL syntax error messages: 116, 193, 208, 296	MSG	4
Modified DDL syntax error messages: 166, 194, 196	MSG	4
New status code for the LOOK function: 786	MSG	5
Omitted status code for the LOOK function: 806	MSG	5



Table 2: Changes in Version 2.5 compared to Version 2.4

(part 5 of 5)

1.6 Notational conventions

This section provides an explanation of the notational conventions used to describe syntax rules.

1.6.1 Warnings and notes

	Points out particularly important information
 CAUTION!	Warnings

1.6.2 Non-SDF notational conventions

Language element	Explanation	Example
<u>KEYWORD</u>	Keywords are shown in underlined uppercase letters. You may enter either the underlined parts of the keyword exactly as shown or the complete keyword.	DATABASE- <u>KEY</u> <u>MANUAL</u>
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
<i>variable</i>	Variables are shown in italic lowercase letters. In a format which contains variables, a current value must be entered in place of each variable.	<i>item-name</i> <i>literal-3</i> <i>integer</i>
{ 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.	{ <u>CALC</u> } { <u>INDEX</u> } { <u>VALUE IS</u> } { <u>VALUES ARE</u> }
[optional]	The expression in square brackets can be omitted. UDS/SQL then uses the default value The brackets themselves must not be specified.	[IS <i>integer</i>] [<u>WITHIN</u> <i>realm-name</i>]

Table 3: Notational conventions

(part 1 of 2)

Language element	Explanation	Example
. . . 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.	<i>item-name</i> , . . . { <u>SEARCH</u> KEY } . . .
. or . .	Indicates where entries have been omitted for reasons of clarity. When the formats are used, these omissions are not allowed.	<u>SEARCH</u> KEY IS <u>RECORD</u> NAME . .
.	The period must be specified and must be followed by at least one blank. The underline must not be specified.	<u>SET</u> <u>SECTION</u> . 03 <i>item-name</i>
Space	Means that at least one blank has to be specified.	<u>USING</u> <u>CALC</u>

Table 3: Notational conventions

(part 2 of 2)

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

2 General information

This chapter provides general information on maintaining data and using databases. It also explains the various database models and the differences between them. It concludes with a general overview of UDS/SQL.

2.1 Modern data organization

Now that data processing is used at all levels in an organization, the qualitative and quantitative demands on data storage and organization have increased considerably. Moreover, they are subject to constant change.

The reasons for this change are:

- rapid growth of data volumes
- client/server applications
- transition from data to information processing
- influence of new communication media and technology
- increasing tendencies towards decentralization.

It is now widely recognized that data is as an independent factor in production alongside traditional factors such as accounting, personnel and infrastructure.

How and by whom data is used at the different levels in an organization company, is shown in the [figure 1](#) in form of an information pyramid.

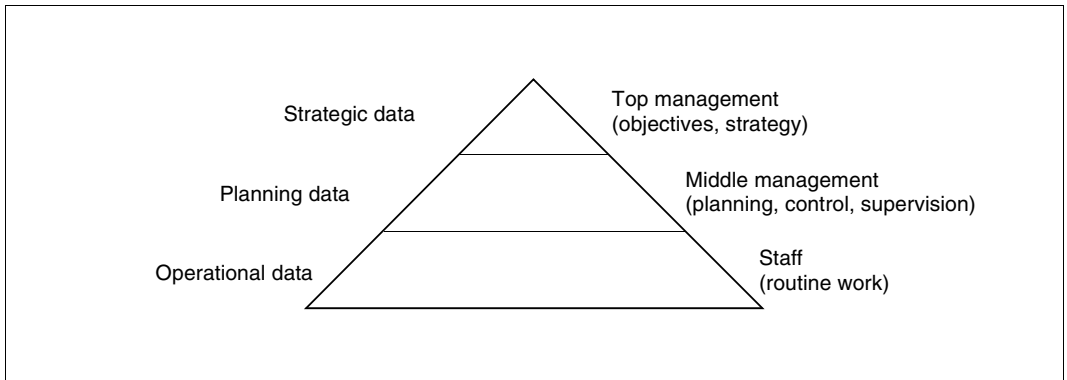


Figure 1: Flow of data and information in an organization

The basis for physical data storage and logical data organization within a company is the database, which is the focus of the company's informational processes.

Reasons for using databases

Often, people who are considering using a database system are motivated to do so because they find that is no longer possible to process extensive quantities of data and an increasing number of applications with conventional files.

Mutually isolated, frequently overlapping (redundant) data resources that are divided into different files are awkward to handle and can be kept up to date only with considerable effort and expense.

This is where the functionality offered by database systems can help. These systems provide a stable, non-overlapping and expandable data organization, as well as incorporating convenient, efficient functions for retrieving and manipulating data.

Combining the different sets of data and administering them together in a database system also ensures consistency. At any given time, all the data has the same update status - even when a large number of applications access the data at the same time.

Database systems also protect their data against unauthorized access. They make it possible to have data resources constantly available and to reconstruct data that has been corrupted or destroyed.

Database systems are also capable of satisfying demanding performance requirements through the use of suitable techniques, for example for optimal operation of the data processing system's input and output devices.

The use of a database system results in a noticeable reduction of costs, particularly in the development of applications.

The cost savings are achieved due to the following reasons:

- All the people involved in development can use a uniform, non-application-specific database.
- Preprogrammed functions are provided.
- A data organization which remains stable over the long term solves lots of problems.

Follow-up costs for service and maintenance can be reduced due to the following reasons:

- Data and programs are clearly separated and independent from one another.
- Logical data organization and physical storage form are mutually independent.

2.2 Data models

The UDS/SQL database system supports both the network model (CODASYL model) and the relational model. It encompasses the principles and capabilities of both the CODASYL and the relational models in a **single** system. UDS/SQL can be regarded as the implementation of a coexistence model of a database. The following sections briefly describe the CODASYL model, the relational model, the relative merits of the two models, and the coexistence model.

2.2.1 The CODASYL model

A major user demand is that database systems are compatible. The standardization of database systems has thus been the goal of powerful user associations for many years. The Conference on Data System Languages (CODASYL) has developed widely accepted recommendations for the standardization of database systems. This association is already well-known for its achievements in application portability thanks to its development of the programming language COBOL.

CODASYL was constituted in 1959 by US producers and users, and, notably, with the participation of the US Administration.

Since 1965, this association has concentrated partly on data organization and databases. When the basic results of the different study groups were published, they described a database concept which has been continually improved.

The CODASYL model provided the basis for the implementation of the UDS/SQL database system.

In the CODASYL model, a database contains not only records, but also the relationships between the records. This is why it is also referred to as the network model.

The following diagram ([figure 2](#)) shows an example of a networked data structure, which is represented by boxes and arrows. A box symbolizes a type of record. In this example, the records that describe the suppliers are grouped together in the record type SUPPLIER. All records of the given record type have the same structure. For each supplier, the SUPPLIER record type contains a record which defines the name and the city of that supplier. NAME and CITY are record elements names.

Another record type shown in [figure 2](#) is the record type ARTICLE. The record types ARTICLE and SUPPLIER are connected by an arrow, which indicates that a relationship exists between the record types. Such relationships are referred to as set relationships or simply sets.

The set between the SUPPLIER and ARTICLE record types has the name SUPPLIES. In the SUPPLIES set, SUPPLIER is the owner record type, and ARTICLE the member record type. As a rule, two or more records of the member record type are assigned to a given record of the owner record type. In this example, the supplier Schmidt supplies the articles gingerbread cookies and marzipan.

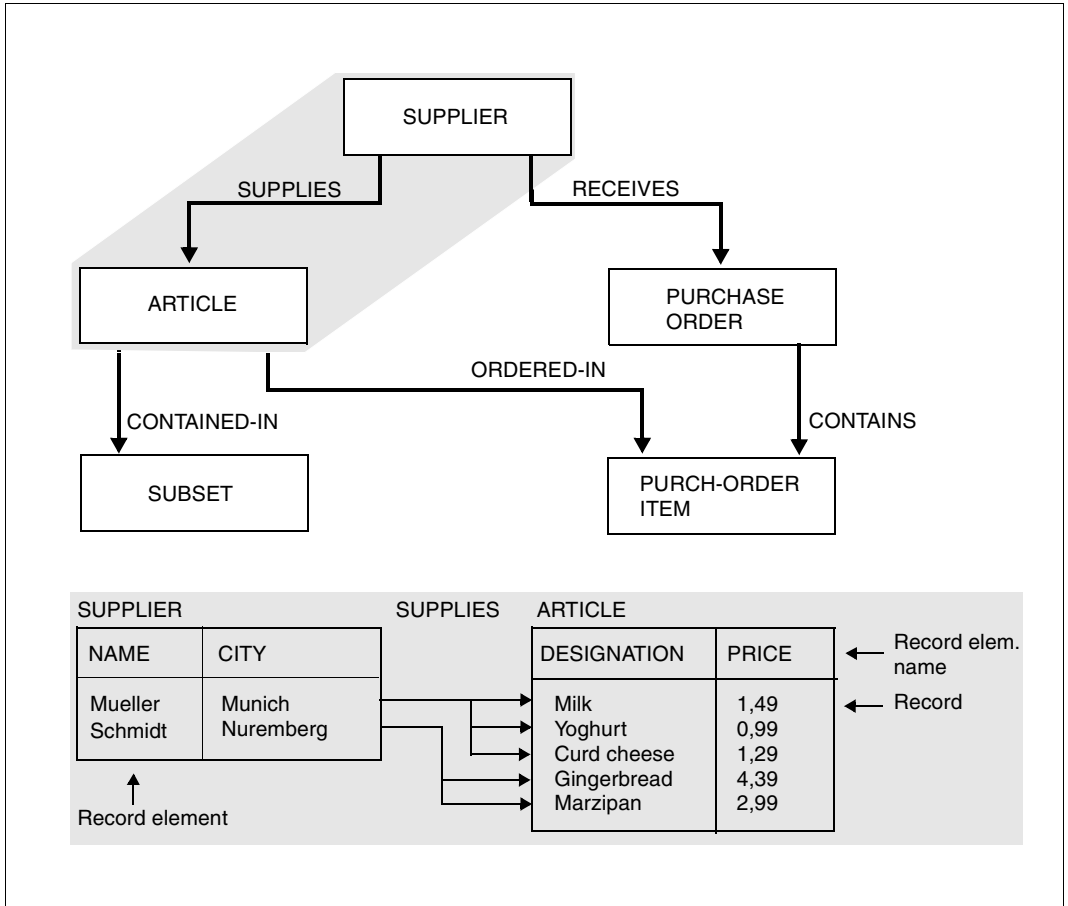


Figure 2: Network-like structure of data

In a network structure, a record type may be a member in two or more different set relationships and thus may also have more than one owner. A record can have only one owner per set relationship.

Relationships between record types and referential integrity

In [figure 2](#), the SUPPLIES set indicates that a record of the ARTICLE record type is not an isolated record, but is assigned to a record of the SUPPLIER record type. In UDS/SQL, it is possible to declare in a set definition that no article may be entered without a supplier being assigned to that article. A relationship of this type is an example of referential integrity. Referential integrity also ensures, for example, that the relationships between order and customer remain consistent; in this case, a customer must exist for each order. This means that no order can be entered without the customer who placed it also being entered, and that a given customer cannot be deleted as long as an order from that customer exists.

Language components of the CODSASYL report

The CODASYL report describes three basic language components:

1. The schema DDL (Data Description Language) defines the logical structure of the data in the database. It allows any type of network structure to be defined.
2. The subschema DDL describes user-specific views of the database.

A subschema is a part of a schema. Two or more subschemas may exist for each schema. These subschemas may overlap one another, i.e. a given record type may exist in two or more subschemas. A subschema can include the entire database or contain only a single record type.

The schema/subschema concept is a major part of data protection. Each user is allowed to perform operations only within his or her own subschema, with no access to the rest of the database.

3. The COBOL-DML (Data Manipulation Language) is a comprehensive language for accessing databases. Both its range of functions and the way it is embedded in the programming language COBOL are defined precisely. Its basic language components are used for navigation, reading and updating in the database as well as to control processing.

2.2.2 Relational model

The relational model is based on the theoretical work of E.F. Codd, who described the organization and manipulation of data in database systems in terms of relational algebra. He used precisely defined terms to represent the mathematical model of his relational theory.

The terms and concepts used in the relational model are explained below, taking as an example the following structure of a small database:

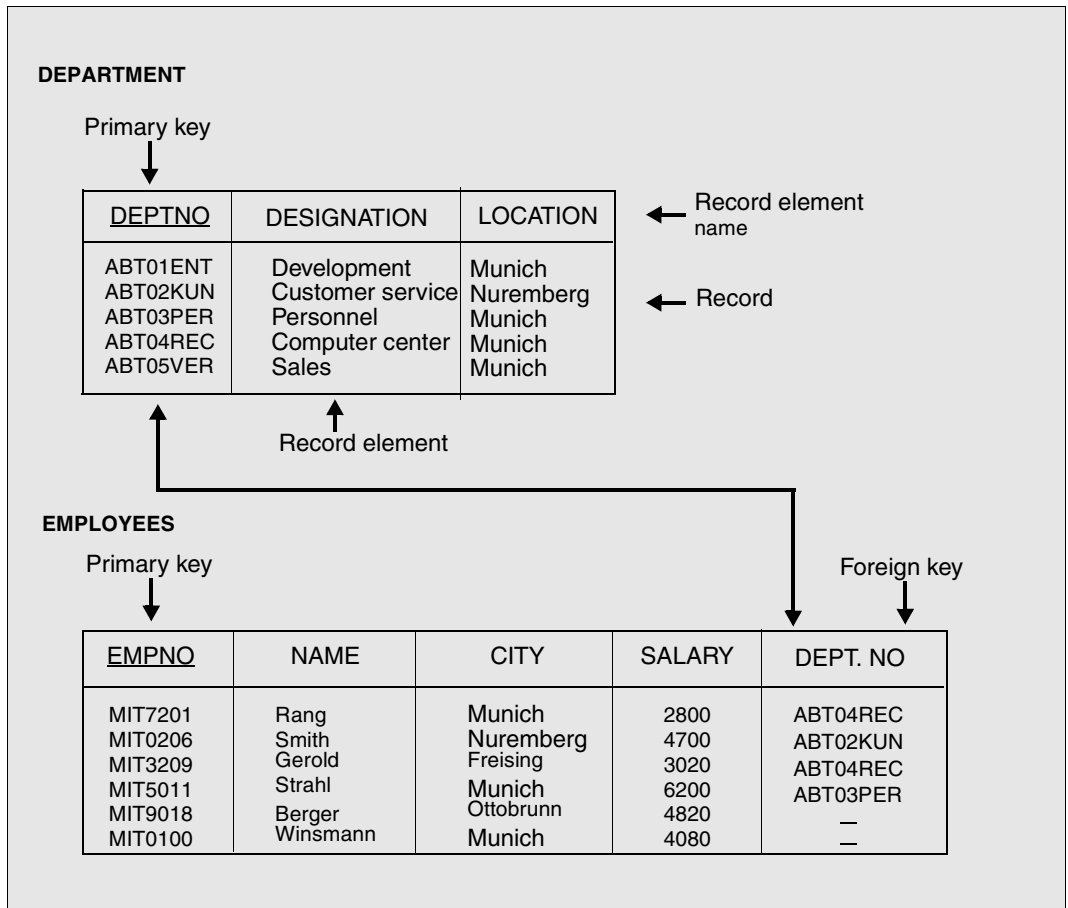


Figure 3: Terminology used in the relational model

The following list shows the terms used in this manual and the corresponding formal terms defined by Codd:

Term used in manual	Formal term (Codd)
table	relation
record (row)	tuple
column, record element	attribute
value	attribute value
value range	domain

In the relational model, the data is managed and processed in the form of tables. Different types of tables exist:

- base tables
- result tables
- views

Base tables

Base tables are tables that have a fixed definition within the database. In [figure 3](#), the database contains two base tables with the table names DEPARTMENT and EMPLOYEES. A table is composed of lines and columns. The base table in the relational model is comparable to the record type in the CODASYL model. A line of a table is called a record. A record of the EMPLOYEES table contains all the information on a given employee. Instead of the term “column”, this manual uses the term “record element”.

Result tables

Querying the database yields a new table, a result table, which contains the records returned in response to the query.

Views

A view is a defined section of a database. A view may contain record elements and records from one or more base tables. Views can be addressed with SQL statements in the same way as a base table can. The use of views can afford a vast simplification of the database interface and provide a large degree of logical data independence.

Relationships between tables

In the case of relational data organization, record types are linked by means of certain record elements with matching contents. A table in a relational schema thus corresponds to a record type in a CODASYL schema. For example, to link a table with another table by means of record element contents, the identifying primary key of the first table must also be included as a (redundant) foreign key in the second table (see [figure 3](#), DEPARTMENT and EMPLOYEE tables).

A foreign key must match the values of another table's primary key. Foreign keys create logical links between tables. Foreign keys are defined by the users themselves.

Tables are the only fixed data structures in the relational model. Most relational database systems neither monitor nor maintain the logical relationships between tables. Users create these relationships by means of values when processing the database, i.e. the tables are linked logically on the basis of their contents. Users are thus not restricted by structures prescribed by the system.

Data manipulation and retrieval in relational databases

The relational model defines basic database functions which are analogous to set operations. Thus, in addition to the principle of processing data in the form of tables, another major feature characterizing the relational model is a defined set of basic operations which can be used to retrieve information from relational databases.

The relational model does not define a language for data manipulation and retrieval.

The main features of data manipulation languages (DML) in relational systems are the set-oriented operations *projection*, *selection* and *join* and the absence of any predefined structures. In other words, relational data manipulation languages describe the desired result itself, instead of describing how that result is to be obtained, i.e. users no longer need to specify **how** the desired result is to be ascertained from the database - they need merely state **what** they want to see.

The relational model is supported in UDS/SQL through the inclusion of SQL (Structured Query Language), a language for relational database systems.

SQL - a uniform language for relational database systems

Development of the theory of relational databases by Codd and others was paralleled by work on the user interface for such systems. The initial results of this work were first presented in the "System R" prototype and were continually revised and supplemented with further results in the years following. The first commercial implementations of this language interface have been on the market since the early eighties under the name of SQL. Major standardization organizations such as ANSI and ISO have been working on standardizing SQL since 1982.

With the active support of Siemens, the ISO standard for SQL (ISO 9075:1989(E)) was established in 1987.

The power of the SQL-DML enables users to perform all of the essential operations on the database. New data can be inserted in the tables or data already in the tables can be queried, updated or deleted. Tables can be linked with one another, and the result sets of two or more queries combined. All processing by the DML is set-oriented. For example, a selection yields a table which the user can then process further record by record, and an update statement changes all the records that satisfy the specified conditions. It is precisely this set-oriented processing that distinguishes SQL from database languages for non-relational database systems.

This is also a major advantage for users in that it enables them to have a variety of actions performed on the database by issuing a single statement. In the case of record-oriented processing, all of these actions would have to be initiated separately.

In summary, SQL has the following main features:

- SQL is based on the relational data model developed by E.F. Codd and incorporates all of its advantages: powerful data manipulation by means of descriptive set-processing statements and simple data organization, which does not affect programs when altered.
- The use of the standardized SQL interface promotes the portability of DB applications and provides a high degree of independence from special database interfaces (facilitates the procurement of know-how).
- SQL is a uniform, easy-to-learn language.

Each of these features results in increased productivity in application programming and improves the cost-effectiveness of database use overall. The obvious advantages of SQL have prompted nearly all manufacturers to offer the SQL interface in their present products or to announce its incorporation in their future ones.

A separate manual (see the "SQL for UDS/SQL" manual) provides detailed information on the basic concepts and terminology of SQL and describes the SQL language set supported by UDS/SQL.

2.2.3 Relative merits of the data models

Comparing the data models with one another in terms of quality is difficult and possible only from the perspective of a specific field of application. Especially an attempt to weight the relative advantages and disadvantages of the models can be made only for a specific application.

Basically, all data structures can be implemented with either of the two models.

Advantages of the relational model

The clear advantage of the relational model is its greater **flexibility** in regard to data structures.

Since the application programs are not dependent on the data structures, the data structures can in many cases be changed without affecting the applications. Through the use of views, the underlying data structures can be hidden completely from the users.

Advantages of the CODASYL model

A major advantage of the CODASYL model lies in the **performance** of the application.

The CODASYL model explicitly defines not only the record types and tables, but also their relationships to one another. This means that the relationships between the record types must also be known to the applications. Since the applications build on the defined data structures, the applications are optimally adjusted to them. This adaptation of the applications to the data structures known to the database system has a very favorable effect on the applications' performance.

Of course, this favorable influence of the data structures on the applications is achieved at the cost of flexibility. In many cases, alterations to the data structures have an effect on the applications.

A further advantage of the CODASYL model lies in its **monitoring of referential integrity**.

A database system based on the CODASYL model automatically ensures that the defined logical relationships between the record types are not violated.

2.2.4 Coexistence of the CODASYL and relational models

A decision to use UDS/SQL is not a decision in favor of the CODASYL model and against the relational model. UDS/SQL supports **both** models within a single database system, which is consequently referred to as the coexistence model. Both SQL and CODASYL applications can work with the same data resources at the same time.

The coexistence model also provides users with the advantages of both data models:

- a high degree of flexibility in regard to data structures for SQL applications, for example through the use of views,
- optimal performance for CODASYL applications and
- monitoring of referential integrity for CODASYL **and** SQL applications, provided set relationships were defined.

UDS/SQL supports two forms of logical data organization:

- CODASYL data organization with set relationships between the record types
- relational data organization, in which record types are linked only via the contents of specific record elements

The SQL interface to CODASYL data structures is supported by the BPSQLSIA utility, which generates a relational view for practically all of the system's CODASYL structures (see the "[Recovery, Information and Reorganization](#)" manual). This is necessary because SQL statements require the use of explicit data elements which do not exist in the CODASYL database description; in the relational database description, primary keys are added to all owner record types, and foreign keys are added to member record types.

The generation of the relational data description by the BPSQLSIA utility does not physically change the database. The additional data elements (primary keys and foreign keys) exist only **logically**. As the result of the generation, SQL application programmers receive a printout containing all the information needed to process a CODASYL database with SQL (e.g. table names, record element names, record element descriptions, etc.). SQL application programmers can work exclusively with the relational schema, yet the CODASYL schema remains unchanged and available for use by CODASYL applications.

This results in two different user views of a UDS/SQL database, as is shown in [figure 4](#) below:

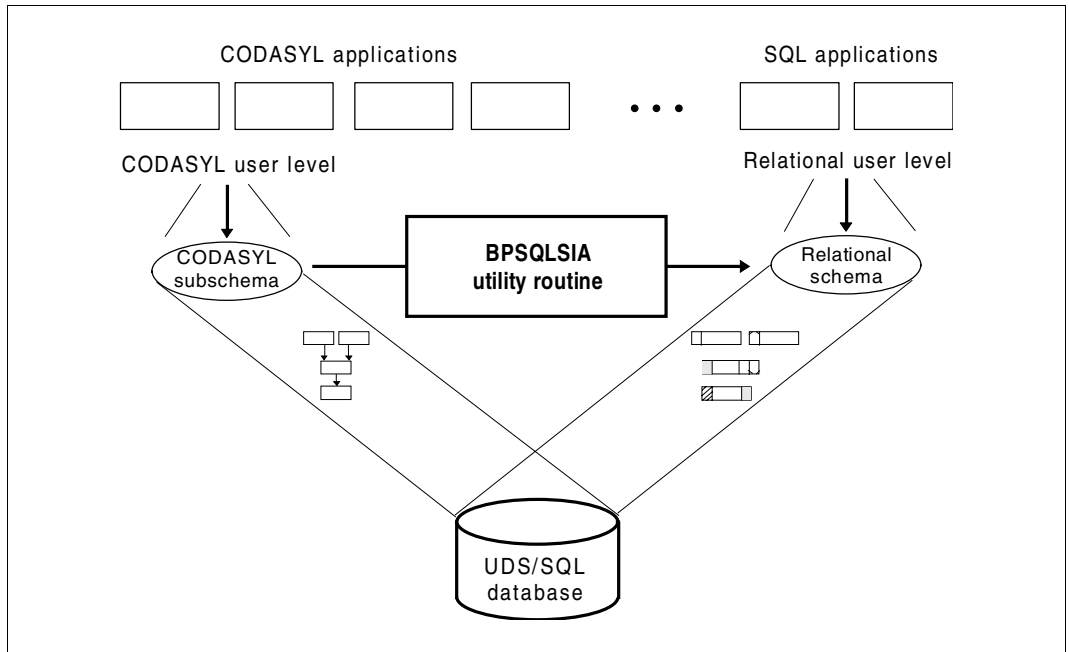


Figure 4: Two user views of the same UDS/SQL database

In summary, UDS/SQL offers the following options for combining program interfaces and data organizations:

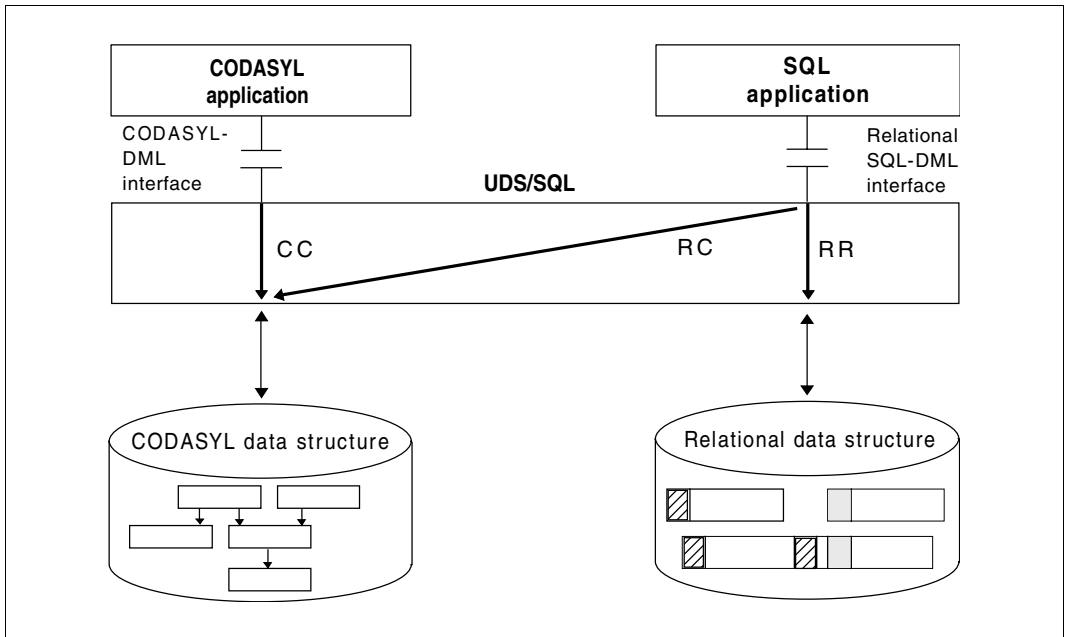


Figure 5: Coexistence of interfaces and data models in UDS/SQL

Relational SQL program interface → CODASYL data organization (RC)

Via the relational SQL program interface, applications access a CODASYL data organization for which a relational view has been generated as described above. This combination of program interface and data organization will be referred to in the following as an RC combination.

An RC combination makes it possible to take advantage of the implicit set relations without requiring SQL programmers to be acquainted with the CODASYL structures. By taking advantage of these set relationships, UDS/SQL ensures the integrity of the data relationships (referential integrity).

Relational SQL program interface → relational data organization (RR)

Via the relational SQL program interface, applications access a purely relational data organization. This combination of program interface and data organization will be referred to in the following as an RR combination.

An RR combination is suitable for compatible programs since in this case the primary and foreign keys are administered by the user, as prescribed by the standard, which corresponds to their handling in SESAM/SQL and INFORMIX. Since the data structures are not based on set relations, the system cannot ensure referential integrity.

CODASYL program interface → CODASYL data organization (CC)

Via the CODASYL program interface (COBOL-DML or CALL-DML of UDS/SQL), applications access the CODASYL data structures. This combination of program interface and data organization will be referred to in the following as a CC combination.

Coexistence of interfaces and data models

UDS/SQL offers the options just described **coexistently**, which means that a database handler can perform the following tasks simultaneously:

- run SQL applications and CODASYL applications
(SQL and CODASYL language elements can even be mixed in one and the same application, but not within a transaction),
- process relational data structures and CODASYL data structures, and
- support all three of the program-interface/data-organization combinations described above (RR, RC and CC).

In other words, the UDS/SQL data model incorporates the relational and the CODASYL data models in such a way that each model can exist independently and that SQL statements can be applied to CODASYL data structures. This is referred to as the **coexistence model**, a model that was realized for the first time in UDS/SQL and which provides an application environment combining the flexibility and simplicity of the relational data model with the efficiency and performance of network database systems.

Interface suitability

The COBOL-DML or CALL-DML interface is typically used for

- high-end OLTP applications and extremely performance-critical applications (Online Transaction Processing) and
- special applications that run especially efficiently with network structures, e.g. parts list processing.

The SQL interface is generally used in the following application areas:

- object-oriented database systems
- client/server applications
- data warehousing
- applications that are to be portable and comply with standards,
- 4GL programming with DRIVE (4th Generation Language) and
- COBOL programs that are to access databases.

2.3 Universal Database System UDS/SQL

The **Universal Database System UDS/SQL** offers users a wide range of structuring options and an abundance of capabilities for setting up, using and maintaining databases:

- Structuring options

The structure of a company's database is a map reflecting the organizational, business and technical aspects of the company. UDS/SQL allows you to map relational, hierarchical and/or networked structures, i.e. the database schema can be adapted to existing company structures without any problem.

The data structure can be adapted to new conditions if, for example, aspects of the company change or the database is expanded to include new things.

- Redundancy-free data storage

In UDS/SQL, data is stored non-redundantly, thus minimizing the storage space required and simplifying updates. An update is made only once, at a single location. The data is thus always available in the same form and with the same update status to all applications.

- Data independence of programs

In UDS/SQL, the records of the entire database are written only once at the logical level. The data structure is defined during the creation of the database and is binding for all application programs.

Application programs operate at the logical level. Alterations to the physical organization of the data have no effect on the programs. Physical aspects (for instance of storage) are thus decoupled or hidden from the users, who get to know only the logical structure of the data required for their respective work.

Program creation and maintenance are simplified and made uniform.

- Flexible data evaluation

Users can evaluate data from different points of view since UDS/SQL maintains the data resources centrally. Records and record elements can be selected not only via defined keys, but also by means of any desired complex conditions or on the basis of any desired record-element contents.

- Central data protection measures

UDS/SQL incorporates effective, flexibly usable protective mechanisms to ensure that each user group can access only precisely defined parts and sections of the database. UDS/SQL checks the user's access rights before requests to the database are actually executed.
- Simultaneous data access by many users

UDS/SQL permits application-independent data to be accessed by large numbers of users simultaneously. Their requests are "time interleaved" for processing to optimize overall throughput.

UDS/SQL also contains functions that prevent users from hindering each other when they contend for access to the same record.
- Central security concept

The security concept of UDS/SQL includes precautions to protect the data resources against destruction or loss. Data resources which have been destroyed can be restored to the most recent consistent status.
- Support for mirrored disks

UDS/SQL supports DRV (Dual Recording by Volume). When DRV is used, the operating system writes data to two disks simultaneously. This ensures increased availability and reliability since, even if one of the disks fails, all the data is still consistent, up-to-date and available to the applications.
- Support for the BS2000 access method FASTPAM

UDS/SQL supports the BS2000 access method FASTPAM, which provides a highly efficient means of accessing files and realms of the database.
- Support for uninterrupted time conversions

Support for uninterrupted time conversions ensures the continuous and uninterrupted operation of UDS/SQL during seasonal conversions of the local time, e.g. when converting from Central European Daylight Saving Time (MESZ) to Central European Time (MEZ).

3 Designing the database

In order to design a database with product-specific language resources, it is first necessary to make a precise and detailed analysis of the data items, their interrelationships, their interdependencies and the specific user requirements. This analysis should be performed as thoroughly and as carefully as possible because it is of crucial importance for all subsequent work. Design errors are known to entail the most far-reaching consequences because correcting them at a later time involves a lot of effort and expense.

3.1 Data modeling

For data modeling, there are a wide range of models and aids available, the most important of which include the Entity Relationship Model (ERM) and Structured Analysis (SA).

For example, a data dictionary can be used to acquire and administer the data collected. Literature and training programs provide substantially more detailed information on data analysis and design than can practicably be included in a product-specific manual.

Basically, the analysis includes the following steps:

- Delimiting the microcosm from the real world

The first step is to define the microcosm, i.e. the section of the real world on which the data model is to be based, so that the requirements can be implemented in data-processing procedures.

- Information analysis

The objective of information analysis is to study the data items and information of the microcosm, as well as the interrelationships of the data items.

- Function analysis

This analysis documents what data is required by the individual applications and in what order.

The analytical process yields a model which describes the designated section of the real world in such a way that the data can be administered with a database system. The data is complete, consistent and available in normalized form. The type of relationships existing between the data items is defined, e.g. a 1:n relationship between customer and order.

The logical structure of the data is also called the conceptual schema. The conceptual schema is an important basis for further work, both for database designers and for specialized departments. The conceptual schema represents the data items and their interrelationships without defining the data model in which the described structure is mapped.

After a conceptual schema has been developed, it is necessary to consider how the data is to be distributed in databases and on different systems.

3.2 Distributing the data

The data resources may be distributed over several databases, e.g., a personnel database for the accounting department, a customer database for the order-processing department, etc. The databases can be operated by one or more database handlers (DBHs). Databases may also be distributed over different systems.

In dividing data resources, it is advisable to take the following aspects into account:

- Links between the applications
- Organizational circumstances
- Requirements for availability and throughput

One DBH - multiple databases

This constellation is also referred to as multi-DB operation. Many application programs work with two or more databases simultaneously. An application program (AP) may access several databases within a single transaction. The DBH controls the processing of the databases and ensures the consistency of all the data resources.

The databases that are operated by a DBH are part of a database configuration.

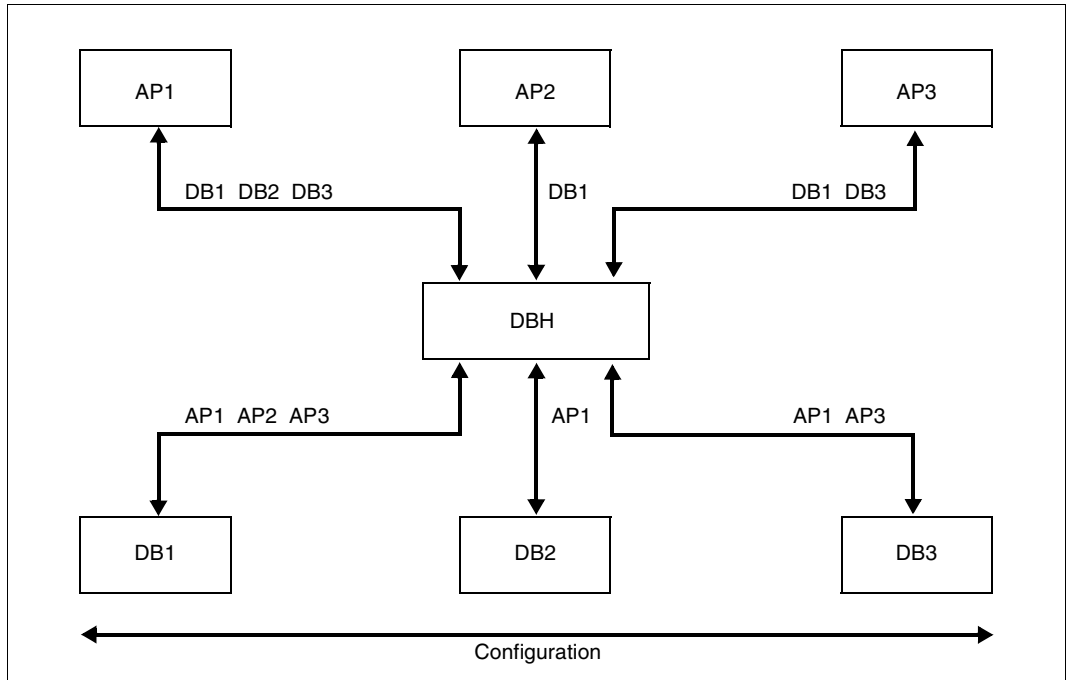


Figure 6: Multi-DB operation

Reasons for distributing data over multiple databases:

- Data that is used only at certain times or for certain tasks can be stored in a separate database, which need not be connected the whole time.
- A large-scale database project is more easily implemented by gradually developing additional databases.

Multiple DBHs - multiple databases

Multi-DB operation is also possible with databases belonging to other configurations. Other configurations may be located on another host in a homogeneous BS2000 network (see [figure 7](#)) or on the same host (see [figure 8](#)).

The UDS/SQL supplementary product UDS-D is required for operation with multiple database handlers (see the “[Database Operation](#)” manual). With UDS-D, a COBOL or CALL DML application program, but not an SQL program, can access databases from two or more different configurations within a single transaction, without requiring any information on the location of the databases accessed. This means that multi-DB operation can be extended to include UDS-D operation without any changes to existing application programs.

Inter-configuration consistency of all databases is maintained at all times during processing. UDS-D also provides inter-configuration deadlock detection and resolution.

The following diagrams show the additional options provided by UDS-D, which are to be understood as an addition to [figure 6](#).

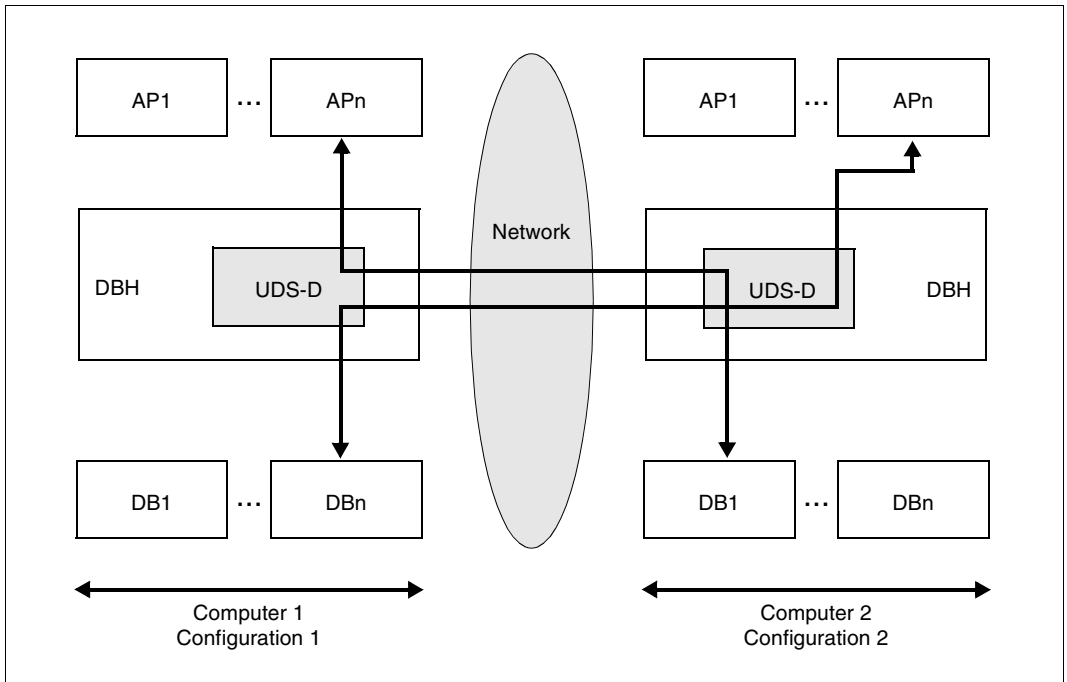


Figure 7: Accessing databases on remote computers

Reasons for distributing databases among several computers within a network:

- Adaptability

Work processes can be optimized for the local computer center, and data storage can be adapted even better to the organization of the company.

- Availability

When distributed databases are used, the data resources are more easily available since especially important and frequently used data can be stored in more than one system, and the copies coupled together. If the backups are stored at different locations and one of the systems fails, the data can still be accessed in other systems.

- Reduced cost

Costs are cut when fewer data stations require a continuous link to the central computer center. Most of the data required can be accessed locally; only occasionally is it necessary to access remote data.

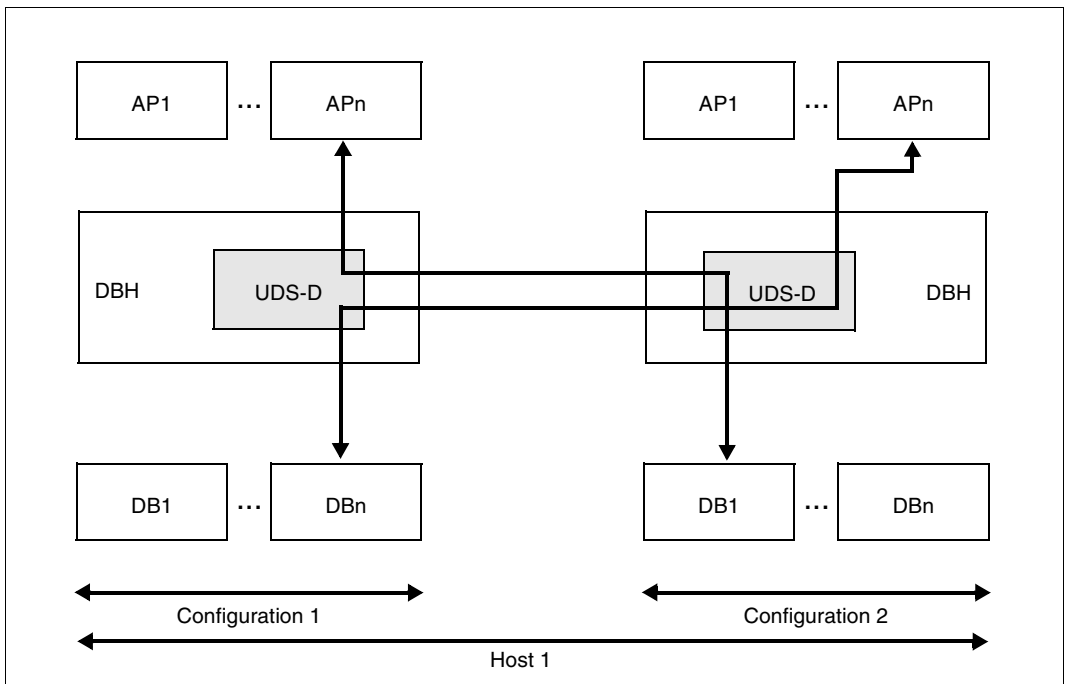


Figure 8: Accessing databases in another configuration within the same host

Reasons for having two or more configurations on a single host:

- Improved performance
- Greater mutual independence of applications
- Separate administration
- Separate accounting
- Separate spheres of responsibility

3.3 Technical implementation

3.3.1 Defining the logical structure of a UDS/SQL database

The logical structure of a UDS/SQL database, i.e. the UDS/SQL schema, can be defined on the basis of either the CODASYL concept or the relational concept.

CODASYL database design

In a CODASYL database, data relationships are defined via database structures. A major advantage of this concept is that the database handler automatically checks these relationships for consistency (referential integrity) at runtime.

The language used to define a database is the schema DDL (Data Description Language). Using the schema DDL to create a CODASYL database involves the following information objects:

- database realms
- types of records
- sets
- keys as optimized access paths
- sort procedures
- set membership
- set selection

For a precise description of the schema DDL, see [page 53](#).

Relational database design

In a relational database, the data relationships are defined by linking values at runtime.

The components of a relational structure, i.e. tables (record types) and columns (items), are defined by means of the schema DDL.

In order to represent the relationships between tables, it is also necessary to define the primary keys of the respective tables as foreign keys in the corresponding tables.

The users themselves must assign these primary and foreign keys and then check the relationships for uniqueness.

Users must also check these relationships at runtime in their respective programs.

For a detailed description of the methods used to implement a conceptual schema in relational structures, you can either refer to database literature or attend courses on the subject.

3.3.2 Defining the physical structure of a UDS/SQL database

To define the physical data structure, you use the SSL (Storage Structure Language).

The SSL is used to specify mainly:

- set information
- data placement
- optimization
- links: internal tables, chains, lists.

The data is essentially distributed over database areas via the SSL, taking the following aspects into account:

- depending on access frequency,
- according to physical storage options (disk capacity) and
- according to the backup procedures used.

3.3.3 Views

Subschema

With the subschema DDL, the user selects sections from the database which are required for a particular application.

For a detailed description of the subschema DDL, see [page 183](#).

Relational schema

A relational schema is used in an SQL application. A relational schema is the relational view of an actual subschema. Such a view can be derived from a subschema with the aid of the utility BPSQLSIA.

For further information on the relational schema, see [page 196](#).

4 Schema DDL

4.1 Introduction

Before you start designing a UDS/SQL schema, it is first necessary to thoroughly analyze the intended database applications and the information which is to be processed by them. The analysis must yield not only all the information required, but the relationships between the information as well.

The data structure thus derived is translated into a UDS/SQL schema by means of the schema DDL.

The UDS/SQL schema must define all data required for the tasks to be performed using the database. However, the UDS/SQL schema has no direct user interface; user-friendly editing of the data need not be provided for when designing the UDS/SQL schema.

The CODASYL model provides the following units for defining the logical data structure:

Item The smallest unit of named data within a record type. It is defined by item type, item length and item name.

Vector Item with repetition factor. The repetition factor must be greater than 1. It defines how many duplicates of the item the vector comprises.

Group item

Named grouping of record elements within a record type. A record element can in this case be either an item, a vector or a group item. Group items that are not repeating groups may only be defined for subschemas.

Repeating group

Group item with repetition factor. The repetition factor must be greater than 1. It defines how many duplicates of the group item the repeating group comprises.

Record type

Named collection of record elements and the smallest unit of data which is recognized by UDS/SQL via a unique identifier.

A record element may be an item, a vector or a repeating group.

Set Named relationship between two record types.

Realm Named physical subdivision of the database. Management unit for data privacy and security as well as for handling concurrent accesses.

The language elements of the DDL which are used to define the data units are described on [page 55](#) through [page 70](#).

The notational convention are explained on [page 22](#), and the general syntax rules are summarized on [page 233](#).

You compile the Schema DDL with the DDL compiler. For information on operating the DDL compiler, please refer to the “[Creation and Restructuring](#)” manual, Compiling the Schema DDL).

4.2 Defining an item

An “item” is the smallest unit of data within a record type that can be assigned a name by means of the schema DDL. “Item” also stands for all the values which can be assumed by the item. The particular value contained in an item is referred to as item content. The entirety of all possible item contents is known as the value range of the item.

Defining an item basically means defining an item's value range. The item description also determines the physical form of storage of an item content.

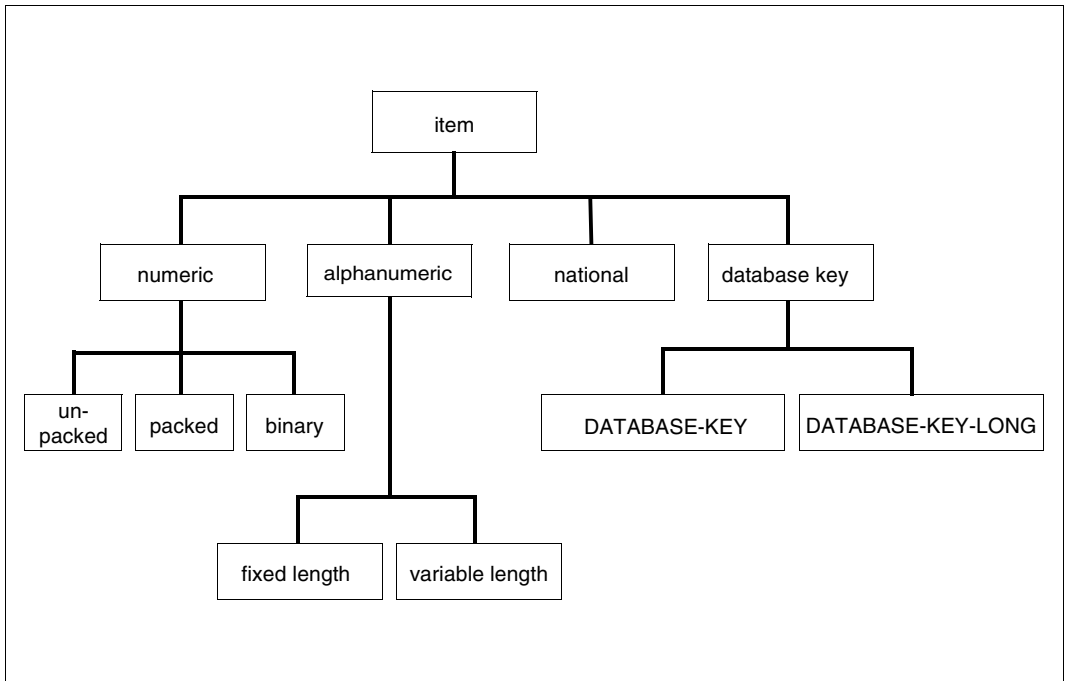


Figure 9: Item types

[figure 9](#) shows the various possibilities of physical storage for the contents of an item.

4.2.1 Defining an unpacked numeric item

```
[level-number ]item-name PICTURE IS mask-string;
```

Unpacked items can contain numeric values only. They can be used for arithmetic operations and can be printed out.

level-number specifies whether the item is part of a repeating group:

If the item is not to be part of a repeating group, the specified level number should be the smallest level number in the whole record type. This is especially important if the item is to be used as a key item.

If the item is to be part of a repeating group, proceed as described on [page 66](#).

The default value for *level-number* is 1.

item-name specifies the name assigned by the user.

mask-string defines the item value range, i.e. the symbolic representation of all possible item contents, by means of a mask.

mask-string may consist of the following symbols:

Symbol	Designation	Explanation
S	Sign symbol	Mandatory, if positive and negative item contents are to be distinguished. If S is not specified, UDS/SQL assumes the item content to be a positive numeric value. This symbol is specified once only at the beginning of the mask string.
9	Digit symbol	is the only symbol of the mask specifying a physical storage location. This storage location can be filled with a decimal digit. The digit symbol can be repeated any number of times. The frequency of repetition determines the item length

Table 4: Symbols of *mask-string* when defining an unpacked numeric item

(part 1 of 2)

Symbol	Designation	Explanation
V	Decimal point symbol	specifies the position of the decimal point for the mask defined by the digit symbols 9. Default value is V after the last digit symbol of the mask.
P		is specified if the decimal point is to be more than one position outside the mask defined by digit symbols 9 and cannot be specified by "V". P stands for a zero to be added by UDS/SQL between the mask defined by digit symbols 9 and the decimal point. P can be repeated any number of times.
(<i>integer</i>)	Repetition symbol	The digit symbol 9 and the decimal point symbol P can be repeated. In order to avoid repeating the symbols, the user can add a repetition factor <i>integer</i> after the symbol to denote the number of times the symbol is to be repeated.

Table 4: Symbols of *mask-string* when defining an unpacked numeric item

(part 2 of 2)

You can define the item to hold up to 18 digits.

If at least one of the following assertions applies to an unpacked numeric item, the item cannot be accessed using SQL, nor can any new records of the record type involved be inserted:

- number of storage locations > 15
- scale factor < 0
- scale factor > number of storage locations

A positive scale factor specifies the number of digits to the right of the decimal point, while a negative scale factor specifies how many zeroes UDS/SQL must append to the item contents when performing calculations.

4.2.2 Defining a packed numeric item

```
[ level-number ] item-name TYPE IS FIXED REAL DECIMAL  
[ integer-1 [ , integer-2 ] ] _
```

Packed items can contain numeric values only. They are exclusively used as computational items by the database programmer and cannot be printed without prior editing by a DML program. Packed items require less storage space and can be processed faster than unpacked items.

level-number denotes whether the item is part of a repeating group:

If the item is not to be part of a repeating group, the specified level number should be the smallest level number in the whole record type.

This is especially important if the item is to be used as a key item.

If the item is to be part of a repeating group, proceed as described on [page 66](#).

item-name specifies the name assigned by the user.

integer-1 and *integer-2* are used to describe the value range of an item, where

- *integer-1* indicates the number of storage locations the item contains, the maximum being 18. Each storage location can be filled with a decimal digit.
Default value for *integer-1*: 18
- *integer-2* specifies the position of the decimal point. This has no effect on the number of storage locations, however. If *integer-2* is positive, it denotes the number of digits to the right of the decimal point; if it is negative, it signals to UDS/SQL how many zeros it is to append to the item contents when performing calculations.
It follows that if *integer-2* is m , specification of *integer-2* means a multiplication of the item contents by 10^{-m} .
Default value for *integer-2*: 0

If at least one of the following assertions applies to an unpacked numeric item, the item cannot be accessed using SQL, nor can any new records of the record type involved be inserted:

- number of bytes of storage > 15
- scale factor < 0
- scale factor > number of bytes of storage

A positive scale factor specifies the number of digits to the right of the decimal point, while a negative scale factor specifies how many zeroes UDS/SQL must append to the item contents when performing calculations.

4.2.3 Defining a binary item

```
[level-number] item-name TYPE IS FIXED REAL BINARY [ { 15 }  

{ 31 } ] .
```

Binary items can contain integers only. They are used exclusively as computational items by the database programmer and cannot be printed without prior editing by a DML program. Binary items require less storage space and can be processed faster than packed or unpacked items.

If BINARY 15 items are not aligned to halfwords and BINARY 31 items are not aligned to words by the user, UDS/SQL aligns them automatically.

level-number denotes whether the item is part of a repeating group:

If the item is not to be part of a repeating group, the specified level number should be the smallest level number in the whole record type.

This is especially important if the item is to be used as a key item.

If the item is to be part of a repeating group, proceed as described on [page 66](#).

item-name specifies the name assigned by the user.

BINARY 15 is used to define a binary item with a value range of -2^{15} through $2^{15}-1$.

BINARY 31 is used to define a binary item with a value range of -2^{31} through $2^{31}-1$.

If you do not specify a number, the default value BINARY 15 is assumed.

4.2.4 Defining an alphanumeric item of fixed length

```
[level-number ]item-name { PICTURE IS mask-string }  

{ TYPE IS CHARACTER integer }+
```

Alphanumeric items can contain any type of character.

level-number denotes whether the item is part of a repeating group: If the item is not to be part of a repeating group, the specified level number should be the smallest level number in the whole record type. This is especially important if the item is to be used as a key item. If the item is to be part of a repeating group, proceed as described on [page 66](#).

item-name specifies the name assigned by the user.

mask-string may consist of the following symbols:

Symbol	Designation	Explanation
X		stands for a storage location containing any character of the character set.
A		stands for a storage location containing a letter or a blank. UDS/SQL does not distinguish between A and X however.
9	Digit symbol	stands for a storage location containing a digit. 9 may not stand to the left of A or X.
(<i>integer</i>)	Repetition symbol	Each of the X, A or 9 symbols can be repeated. To do this, the user can either write the symbols the desired number of times or add a repetition factor <i>integer</i> after the symbol to denote the number of times the symbol is to be repeated. The default value is 1; the maximum value is 18 for 9 and 255 for X and A.

Table 5: Symbols of *mask-string* when defining an alphanumeric item of fixed length

The first symbol in a mask string must be A or X. (A mask string beginning with 9 defines a numeric item.)

The item can be defined for a maximum of 255 characters. The digit symbol may not be repeated more than 18 times and the mask string may consist of no more than 30 characters.

integer specifies the number of storage locations the item contains, each storage location being able to contain one character of the character set.

4.2.5 Defining an alphanumeric item of variable length

```
[level-number ]item-name-1 { PICTURE IS LX(integer)
                                TYPE IS CHARACTER integer }
                                DEPENDING ON item-name-2.
```

Alphanumeric items of variable length can contain any type of character.

level-number denotes whether the item is part of a repeating group. As items of variable length may not be part of a repeating group, the specified level number must be the smallest level number in the whole record type.

item-name-1 specifies the name assigned by the user.

integer specifies the maximum length of the variable-length item, i.e. it denotes the maximum number of storage locations, where each storage location can contain any character of the character set. The value specified for *integer* must be > 0.

item-name-2 specifies the name of an item that must have been defined immediately before the variable item as a binary item with a value range of 0 through $2^{15}-1$. Before storing or updating a record containing a variable item, the database programmer must specify the current length of the variable item in the *item-name-2* item. This (current) length may also be 0, in which case, neither the variable item nor the record length item are stored in the database.

Depending on which page length has been defined for the database, the maximum length of a record with a variable item must not exceed the following values:

- 2012 bytes for a 2048-byte page length (2-Kbyte page format)
- 3960 bytes for a 4000-byte page length (4-Kbyte page format)
- 8056 bytes for a 8096-byte page length (8-Kbyte page format)

The maximum length of a record containing a variable item could, however, also be somewhat lower, depending on the connection data for the record (see “SCD” on [page 219](#)).

Since the record must also contain at least the record length item *item-name-2* in addition to the variable item, the maximum length of the variable item is equal to:

- 2010 bytes for a 2048-byte page length
- 3958 bytes for a 4000-byte page length
- 8054 bytes for a 8096-byte page length

If the record includes other items besides the variable item and the record length item, the maximum length of the variable item decreases accordingly.

Furthermore, the following restrictions apply for variable items:

- There may be only one variable item for each record type; it must also be the last item.
- A variable-length item may not be defined as a key item, nor used for direct access to records.
- A record type containing a variable item may not be restructured by means of subschema DDL. It must be taken over into the subschema as defined by the schema DDL.
- A record type containing an item of variable length cannot be addressed with SQL.

Example

```

RECORD NAME IS ARTICLE MASTER
  DATA-ITEM-1          TYPE IS
      .                .
      .                .
      .                .
  LENGTH-ITEM          TYPE IS BINARY 15.
  ARTICLE-INFO         PICTURE IS LX(500) DEPENDING ON LENGTH ITEM.

```

4.2.6 Defining a national item (UTF-16)

Detailed information is provided in the COBOL2000 “[Language Reference Manual](#)” under “Character representation by UTF-16”.

```
[ level-number ] item-name PICTURE IS mask-string.
```

National items can be filled with any characters.

level-number specifies whether the item is part of a repeating group:

If the item is not part of a repeating group, the specified level number must be the smallest level number in the whole record type.

This is especially important if the item is to be used as a key item.

If the item is part of a repeating group, proceed as described on [page 66](#).

item-name specifies the name assigned by the user.

mask-string may consist of the following symbols:

Symbol	Designation	Explanation
N		stands for a storage location containing any character of the character set.
(<i>integer</i>)	Repetition symbol	You can repeat the symbol N by writing it the desired number of times or by adding a repetition factor <i>integer</i> after the symbol to denote the number of times the symbol is to be repeated. The default value is 1; the maximum value is 127.

Table 6: Symbols of the *mask-string* when defining a national item

The first symbol in a mask string must be N. The mask string may consist of no more than 30 characters.

integer specifies the number of storage locations the item contains, each storage location being able to contain one character of the character set.

A national character occupies 2 bytes and is aligned to the byte boundary in data structures (see the COBOL2000 “[Language Reference Manual](#)” under “Character representation by UTF-16”).

4.2.7 Defining a database key item

```
[level-number] item-name TYPE IS { DATABASE-KEY.  
                                     DATABASE-KEY-LONG. }
```

Database key items are binary items that are intended for storing database key values. At the same time, they are the only items whose contents are interpreted by UDS/SQL as database key values.

Database key items must be defined by the database programmer if the database key values are not implicitly defined by UDS/SQL (see section “[Assignment of database key values by the user](#)” on page 85).

level-number denotes whether the item is part of a repeating group:

If the item is not to be part of a repeating group, the specified level number should be the smallest level number in the whole record type.

This is especially important if the item is to be used as a key item.

If the item is to be part of a repeating group, proceed as described on [page 66](#).

A database key item can be defined as an item of type DATABASE-KEY or DATABASE-KEY-LONG:

- A DATABASE-KEY item is a binary item of 4-byte length with a value range from 0 - $2^{31}-1$.
- A DATABASE-KEY-LONG item is a binary item of 8-byte length with a value range from 0 - $2^{63}-1$. Note that the bit positions 17 - 32 (from the left) are not evaluated by UDS/SQL.

The structure of database key values is described in detail on [page 132](#).



A database key item must be supplied with values by the database programmer.

This *allows* the database programmer to independently define the database key value used by UDS/SQL (see LOCATION MODE clause on [page 85](#)).

Note, however, that the content of the database key item *need not* always match the value used internally by UDS/SQL for the database key of the record.

4.3 Defining a vector

```
[ level-number ] vector-name { PICTURE..... } OCCURS integer TIMES.
```

A vector is an item with a repetition factor. The repetition factor must be greater than 1. It denotes how many duplicates of the item are grouped into the vector.

A vector is defined in the same way as an item as described on [page 55](#).

integer specifies the repetition factor.

Items of variable length and key items may not be declared vectors.

level-number denotes whether the vector is part of a repeating group:

If the vector is not to be part of a repeating group, the specified level number must be the smallest level number in the whole record type.

If you want to declare the vector to be part of a repeating group, proceed as described on [page 66](#).

The limit is the maximum record length.

Example

```
02 CUST-ADDRESS PICTURE IS X(20) OCCURS 2 TIMES.
```

4.4 Defining a repeating group

```
[ level-number-1 ] group-item-name OCCURS integer TIMES_
```

```
{ level-number-2 record-element-name [ { PICTURE..... } ] [ OCCURS..... ]_ } ...
```

A group item is a named grouping of record elements within a record type. A record element can in this case be either an item, a vector or even a group item.

A repeating group is a group item with repetition factor. The repetition factor must be greater than 1. It defines how many duplicates of a group item the repeating group comprises.

The definition of group items that are not repeating groups is only useful for subschemata and is therefore not possible using the schema DDL.

group-item-name specifies the user-assigned name of the repeating group.

integer denotes the repetition factor.

record-element-name specifies the record element that is to become part of the repeating group. It must be defined as described on [page 55](#) if it is an item, as described on [page 65](#) if it is a vector or as described on this page if it is a repeating group.

level-number-2 must be greater than *level-number-1*.

The following applies for all record elements that are to become part of a repeating group: Record elements must have the same level number if they have the next higher repeating group in common.

A hierarchy of repeating groups may not exceed three levels.

If one record element is a vector, only two more hierarchy levels are allowed.

The limit is the maximum record length.

If you want to use elementary items of a repeating group as key items, you must bear in mind that in each case the first variant of the higher-ranking repeating group is taken as a basis and that this may be continued recursively up to the outer repeating group.

Example

```

01 ADDRESSES OCCURS 2 TIMES.
02 CUST-ADDRESS PICTURE IS X(20) OCCURS 2 TIMES.
02 TEL          PICTURE IS X(12).
    
```

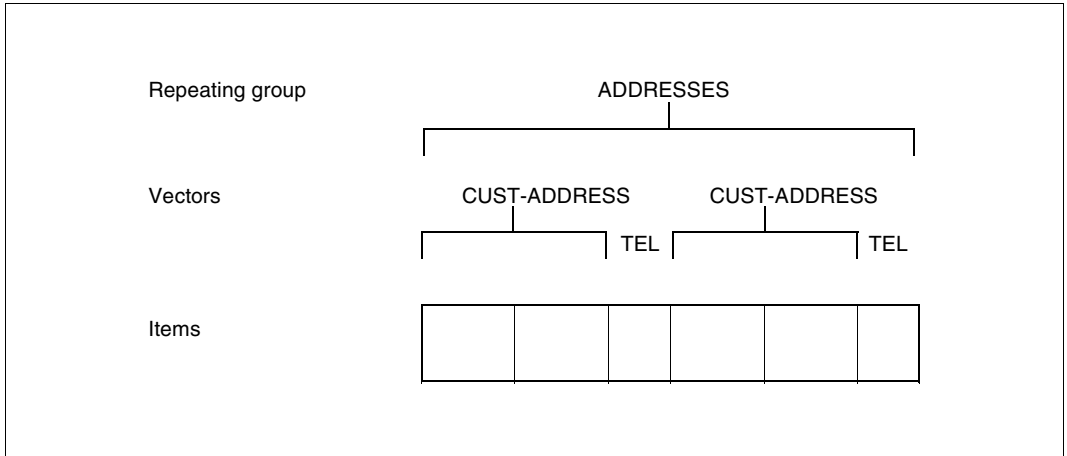


Figure 10: Items and vectors grouped to form a repeating group

4.5 Grouping record elements to form a record type

RECORD NAME IS *record-name*

.
 .
 .
 { [*level-number*] *record-element-name* [{ PICTURE..... }] [OCCURS.....] } ...
 { TYPE..... }

A record type is a named collection of record elements. A record element may be an item, a vector or a repeating group.

A single occurrence of a record type is a record. A record thus consists of one item content each of all the items represented in the record type.

A record type is also the smallest unit of data recognized by UDS/SQL by means of a unique identifier - the database key. All record elements therefore have to be defined as parts of record types.

record-name specifies the name of the record type assigned by the user.

record-element-name specifies the record element which is to become part of the record type. It must be defined as described on [page 55](#) through [page 65](#).

The total length of all record elements within a record type must not exceed the maximum record length.

Depending on which page length was defined for the database, the maximum record length may be:

- 2020 bytes for a 2048-byte page length (2-Kbyte page format)
- 3968 bytes for a 4000-byte page length (4-Kbyte page format)
- 8064 bytes for a 8096-byte page length (8-Kbyte page format)

The maximum record length could, however, also be somewhat lower, depending on the connection data for the record (see “SCD” on [page 219](#)).

The following applies to the maximum number of record types per database:

- A maximum of 253 record types can be defined in the schema of a database with a page length of 2048 bytes.
- A maximum of 32 766 record types can be defined in the schema of a database with a page length of 4000 or 8096 bytes.

Example

```

RECORD NAME IS CUSTOMER
.
.
.
01 C-NO          PICTURE IS 9(10).
01 C-NAME       PICTURE IS X(20).
01 ADDRESSES    OCCURS 2 TIMES.
02 CUST-ADDRESS PICTURE IS X(20) OCCURS 2 TIMES.
02 TEL         PICTURE IS X(12).
    
```

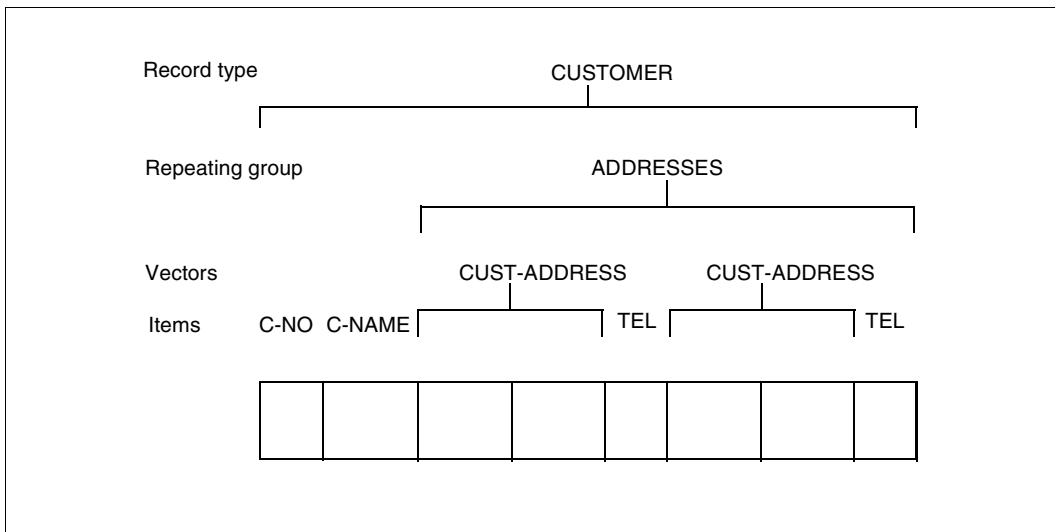


Figure 11: Grouping of items and a repeating group to form a record type

4.6 Linking the records of two record types to form a set

UDS/SQL depicts the relationships and interdependencies of information existing in a corporate organization and a planned database application as relationships between record types using the set concept.

In a relational application, the definition of set relationships causes foreign keys to be assigned appropriately. This is the prerequisite for linking tables (join).

A maximum of 32 766 sets can be defined per database.

For each record type which is owner of a set you can generate a maximum of 255 tables in these sets. A table is created when the set mode pointer array or list or chain is of the type sorted indexed, also for each secondary key in these sets.

Irrespective of this you may define up to 255 secondary keys per record type on record type level and per singular set on set level; hash routines are not counted here.

4.6.1 Defining a set

```

SET NAME IS set-name
.
.
.
OWNER IS record-name-1
MEMBER IS record-name-2.....
.
.
.

```

A set is a named relationship between two record types. It is a hierarchic relationship in which one record type is defined as higher-ranking, the other lower-ranking.

The higher-ranking record type is called the owner of the set.

record-name-1 specifies the name of the record type which is to be the owner.

The lower-ranking record type is called a member of the set.

record-name-2 specifies the name of the record type which is to be a member of the set.

set-name specifies the name of the set relationship between two record types and is assigned by the user.

An individual occurrence of a set is known as a set occurrence. A set occurrence consists of exactly one owner record and any number of lower-ranking member records. In other words, a set consists of as many set occurrences as the owner record type has records. An owner without a member is referred to as an empty set occurrence.

Sets and set occurrences are represented according to the following principle:

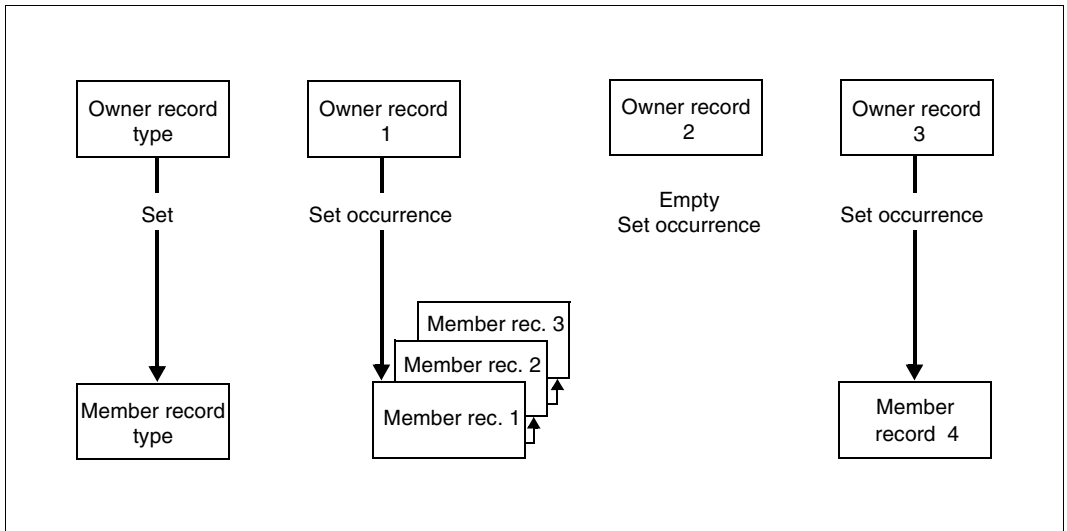


Figure 12: Representation of a set and its set occurrences

figure 12 shows all the owner record types, i.e. all set occurrences of a set. However only member records that are part of a set occurrence are shown.

The member records form a logical sequence within a set occurrence. The sequence is represented by arrows pointing from one member record to the corresponding successor.

Example

```

RECORD NAME IS SUPPLIER
.
.
.
RECORD NAME IS ARTICLE
.
.
.
SET NAME IS ARTICLES-AVAILABLE
.
.
.
OWNER IS SUPPLIER.
MEMBER IS ARTICLE.....
.
.
.
    
```

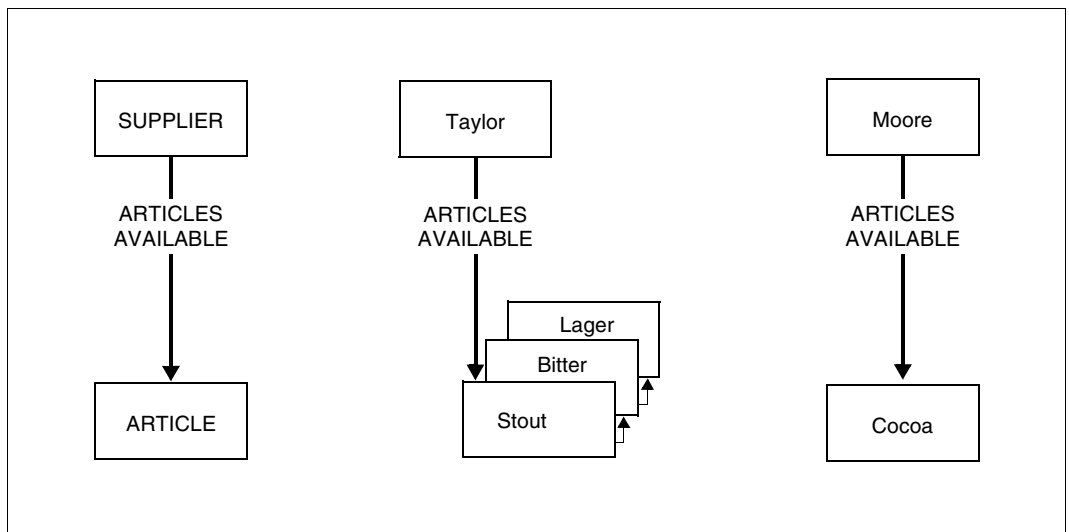


Figure 13: Set and set occurrences as defined above

Example of a 1:n relationship

The relationship between customers and their orders is a 1:n relationship. A customer can place several orders, but each order can only be placed by one customer.

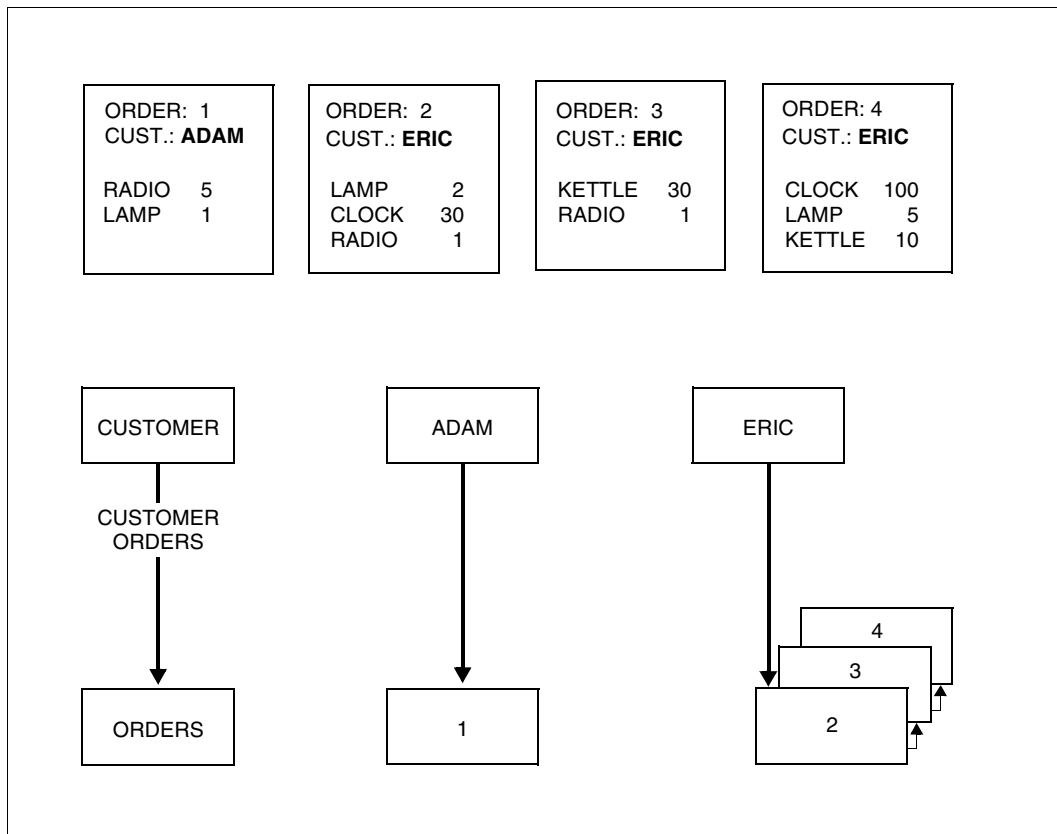


Figure 15: 1:n relationship between CUSTOMER and ORDERS

The logical relationship is created by defining the set CUSTOMER-ORDERS for the record types to be associated.

m:n relationship between two record types (many-to-many relationship)

An m:n relationship is a relationship in which a member record can be associated with more than one owner record.

If m:n relationships are to be represented, they have to be broken down into two 1:n relationships. To this purpose a new record type (auxiliary record type) has to be defined which acts as member record type. After the relationship has been resolved, each member record has an owner.

Example of an m:n relationship

An example of an m:n relationship is the relationship between orders and articles. An order can refer to several articles, an article can be referred to in several orders.

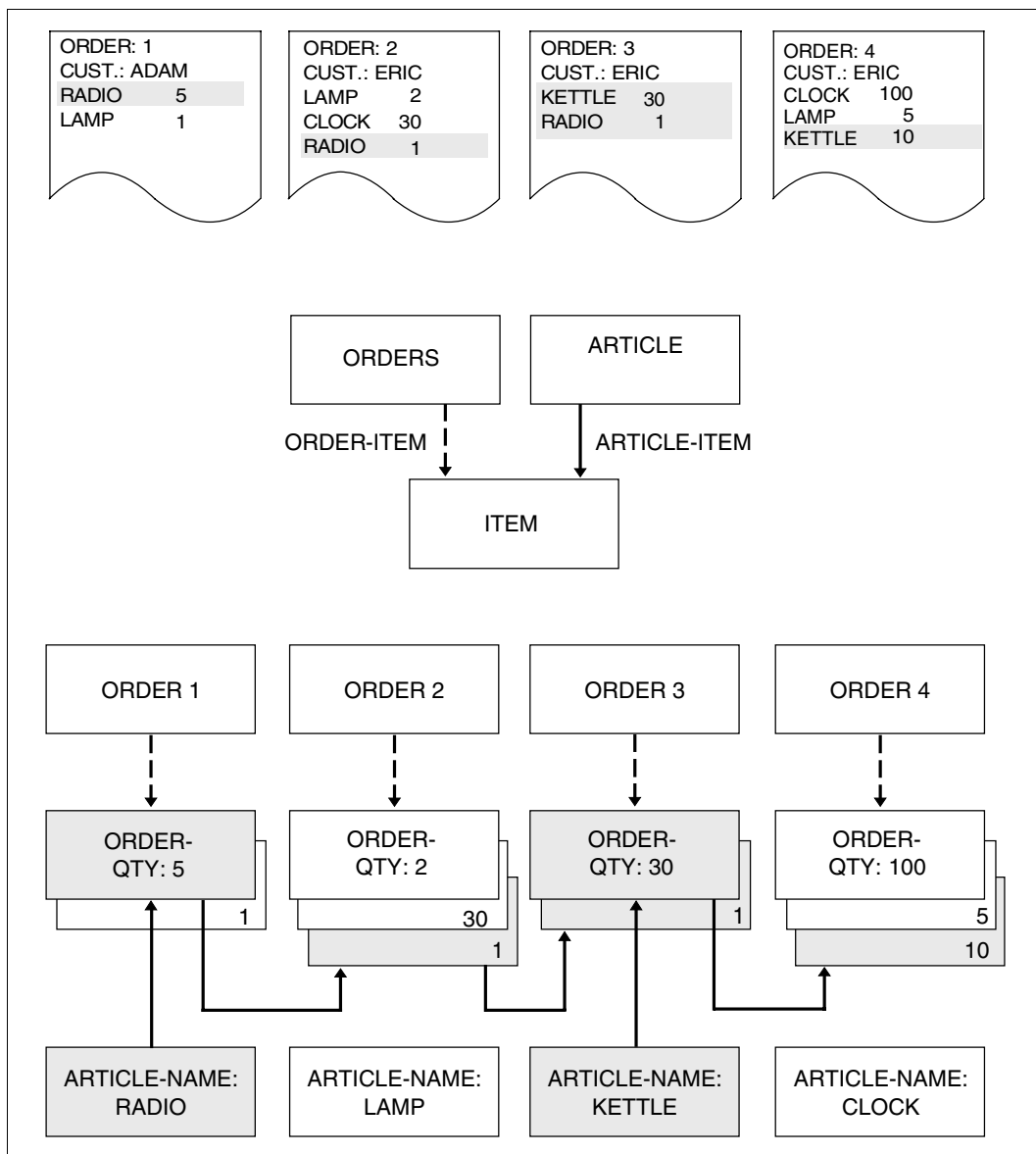


Figure 16: m:n relationship between ORDERS and ARTICLE

The m:n relationship between ORDERS and ARTICLE is resolved to form two 1:n relationships by creating a new record type (auxiliary record type) ITEM which is a member of both ORDERS and ARTICLE.

m:n relationship within one record type (parts list processing)

In this special case the m:n relationship is not between two record types, but exists within one record type.

This type of relationship is resolved by defining a new record type (auxiliary record type) which acts as a member, and defining a new set.

Example of an m:n relationship within one record type

An (assembled) part is made up of several subparts: → parts list

A subpart is used in several (assembled) parts: → where-used list

A bicycle is made up of several parts. Parts used for the bicycle are again in other parts of the bike.

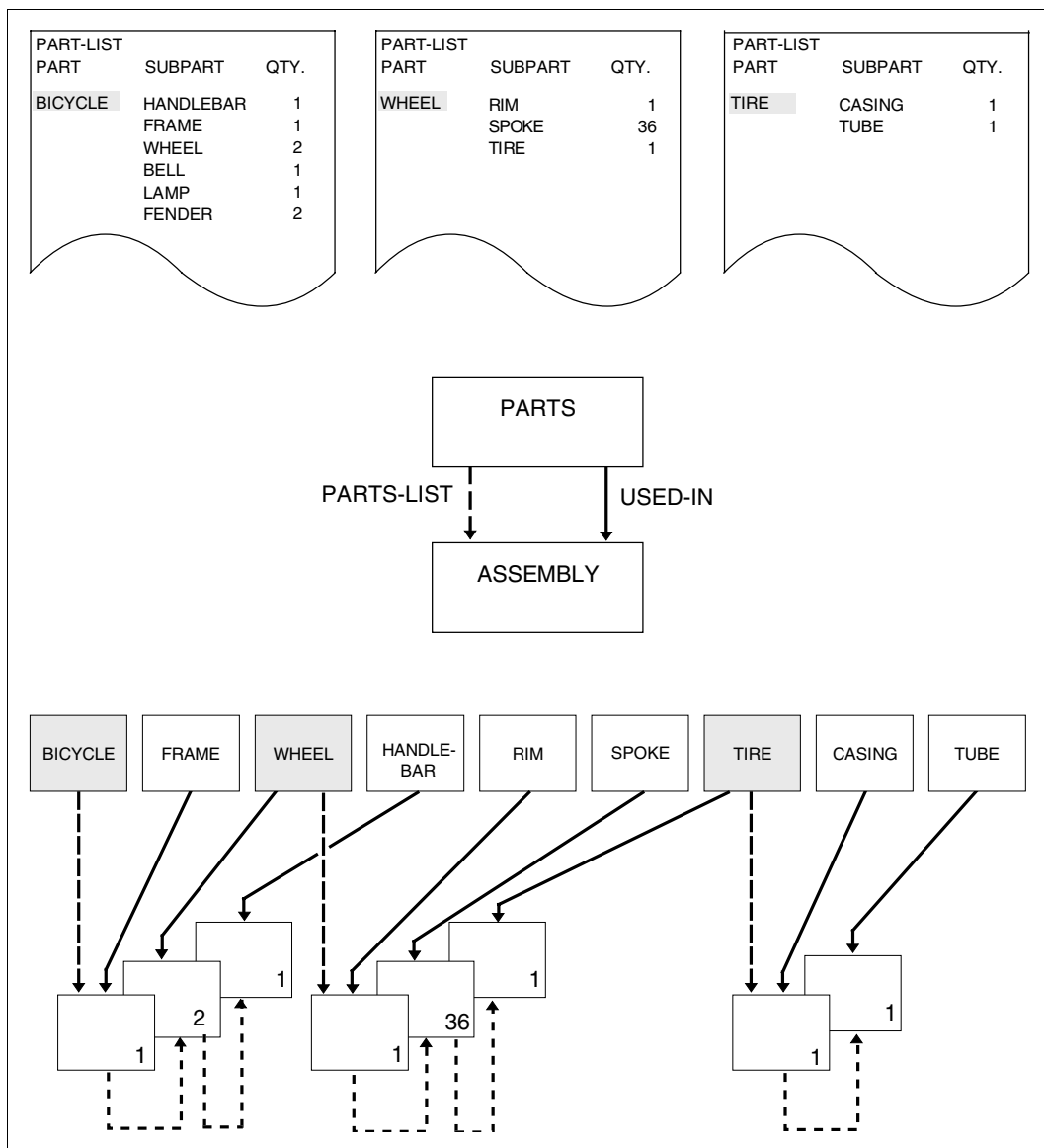


Figure 17: m:n relationship within the record type PARTS

4.6.2 Defining the type of membership of records in a set

`MEMBER IS record-name` $\left\{ \begin{array}{l} \text{MANDATORY} \\ \text{OPTIONAL} \end{array} \right\} \left\{ \begin{array}{l} \text{AUTOMATIC} \\ \text{MANUAL} \end{array} \right\}$

The records of a member record type do not automatically have to be members in a set occurrence. The type of membership of a record in a set can be defined under two aspects:

1. At which point is the record to be inserted in the set occurrence?

AUTOMATIC

The member record is automatically inserted in the set occurrence at storage time (see the “[Application Programming](#)” manual, STORE).

MANUAL

The member record is not automatically inserted in the set occurrence at storage time. Membership in the set occurrence only takes effect when a specific DML statement is entered (see the “[Application Programming](#)” manual, CONNECT).

2. What type of link is to exist between an existing member and an owner?

MANDATORY

The link is fixed. A member record can only exist in connection with an owner record. In this case, membership in a set occurrence can only be released by making the member record a member of another set occurrence of the same set or by deleting it from the database (see the “[Application Programming](#)” manual, MODIFY and ERASE).

OPTIONAL

The link can be established or released by the database programmer (see the manual “[Application Programming](#)”, CONNECT and DISCONNECT). A member can be released from a set occurrence without the member record being deleted.

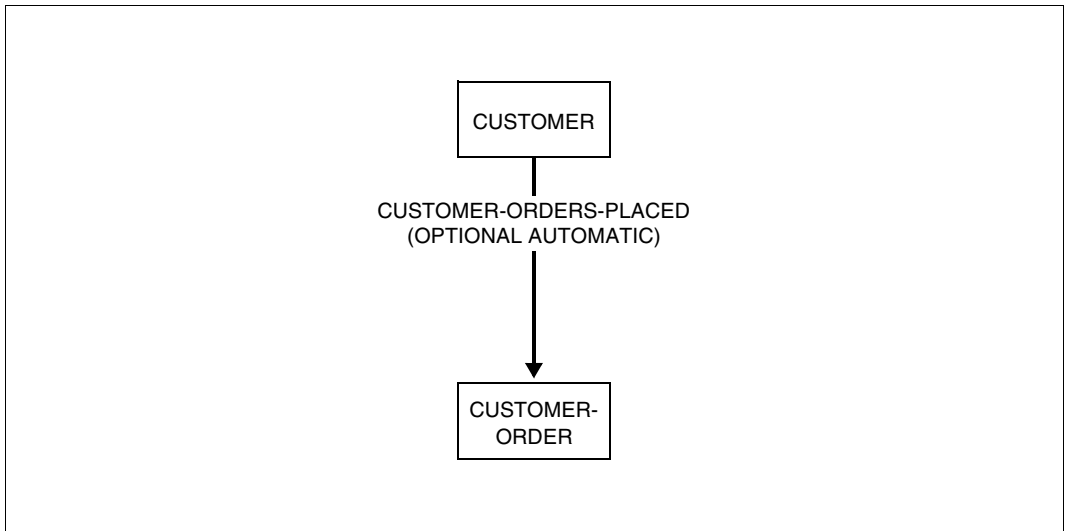
Example 1

Figure 18: Example of OPTIONAL AUTOMATIC

The link between a CUSTOMER-ORDER record and a CUSTOMER record is relatively stable. An order only exists if a customer has placed one. The link to the owner record is thus automatic at the time the CUSTOMER-ORDER record is stored. Filled orders are to remain in the database for statistical purposes even when the link to the CUSTOMER record no longer exists. This is why the set membership is defined as OPTIONAL.

Example 2

The MANUAL option is used if, for example, a record type is a member type of two sets and some of its member records are to belong to one set occurrence only.

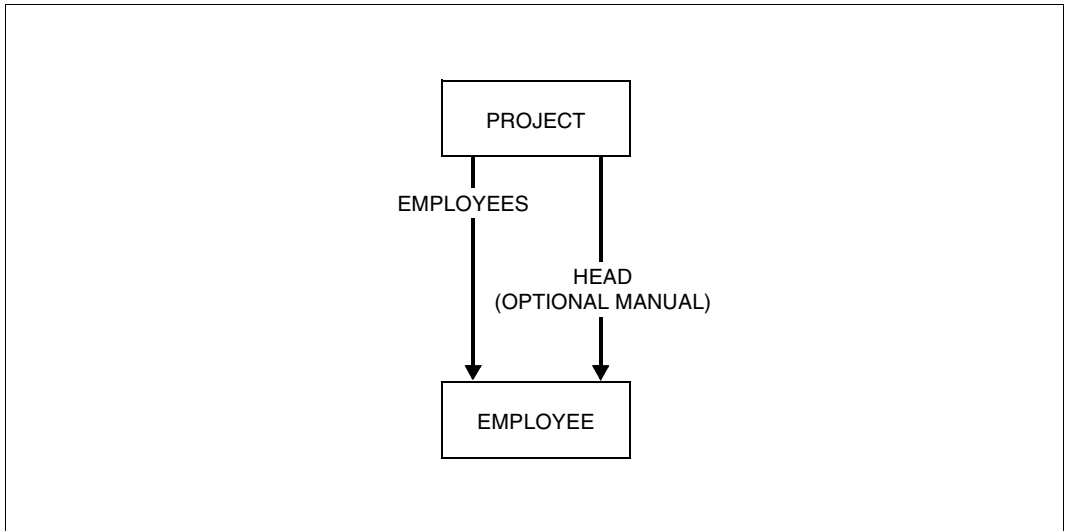


Figure 19: Example of membership in two parallel sets

This example illustrates how project heads can be selected from all employees working on a project by using a set. As not all employees working on a project can be project heads, this set must be defined as MANUAL. An employee is taken over into the corresponding set occurrence only when he or she becomes head of a project. If the set is defined as OPTIONAL, the employee can be deleted from the set once no longer head of the project.

Example 3

The **MANUAL** option is used in cyclic data structures, i.e. a number of record types are connected in such a way that each record type is at the same time owner of one set and member in another.

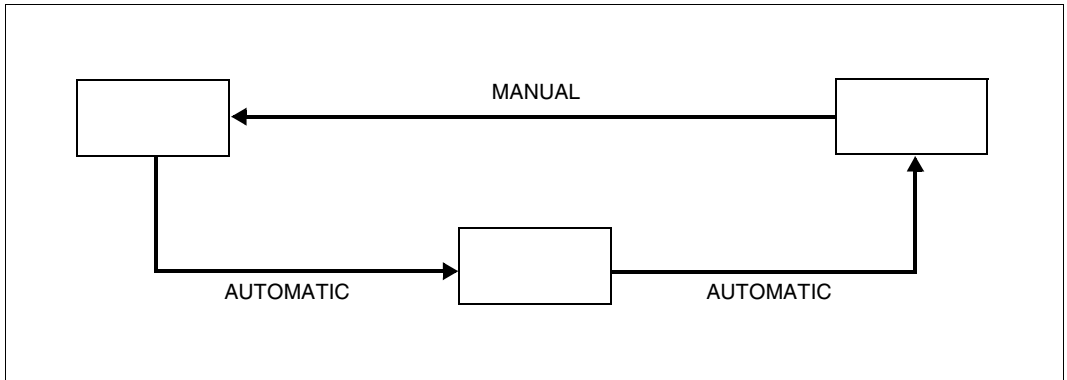


Figure 20: Type of set membership in a cycle

If all sets of a cycle were defined as **AUTOMATIC**, it would be impossible to store a record of this cycle in the databases, as the **AUTOMATIC** option requires that the owner of a record to be stored already exists in the database.

Example 4

In order to resolve a many-to-many relationship, it is necessary to define an auxiliary record type. This requires the following types of set membership:

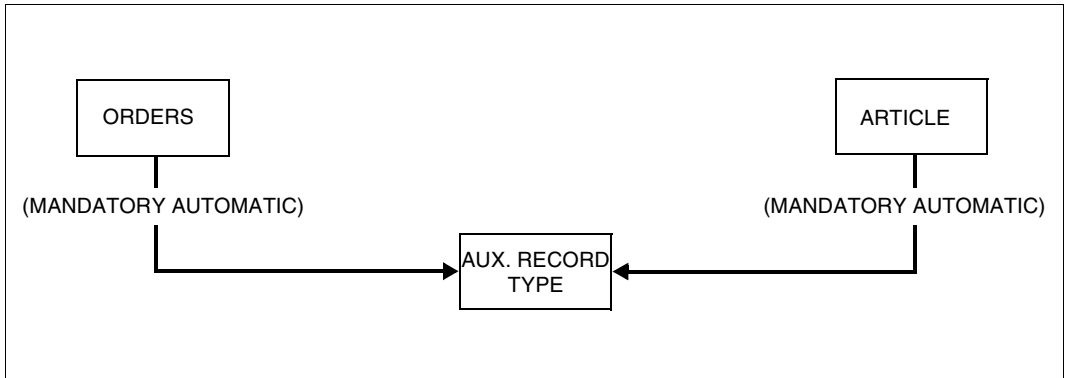


Figure 21: Types of set membership in the case of a many-to-many relationship

The purpose of an auxiliary record type is to link the ORDERS records with the ARTICLE records. The link is only effected if every auxiliary record is present in each set occurrence of the two sets. This is why membership is enforced by the AUTOMATIC option.

If the auxiliary records contain no valid information once they have been released from their owner records, the link to the owner records can be retained unless the records are to be deleted. Membership can thus be specified as MANDATORY.

4.7 Access paths and record sequences

The user can define the following access types using DDL:

- direct access on record type level

A record is selected from all records of one record type via the content of an item or a combination of items.

- sequential access on record type level

A record is selected on the basis of its position within the logical sequence of all records of the record type.

- direct access on set level

A record is selected from all records of a set occurrence via the content of an item or a combination of items.

- sequential access on set level

A record is selected on the basis of its position within the logical sequence of all records of one set occurrence.

The following describes:

- which access paths for direct accesses are a standard feature and which can be generated by the user.
- how to define the logical sequence of records if sequential processing is required.

4.7.1 Direct and sequential access on record type level via database key value

The database key value is a unique internal record key assigned at the time a record is stored, and retained for the entire life of the record.

The database key value is a combination of the record type identifier, the record reference number and a record sequence number (see the section “[Structure of a database key value](#)” on page 132). The order of records within a record type is according to ascending record sequence numbers.

For each record type, UDS/SQL automatically generates a table, the Database Key Translation Table (DBTT), which contains the physical addresses (page numbers) of all records of one record type (see section “[DBTT \(Database Key Translation Table\)](#)” on page 131). UDS/SQL obtains the physical address of a record in the DBTT by converting the record’s database key value; it does not have to sequentially scan the DBTT.

Thus the database key represents a means of directly and sequentially accessing data on record type level.

As a standard feature, the database key values are assigned by UDS/SQL. In this case, the order of records is not usually predictable when sequential processing is applied:

If the database programmer stores the database key values, he or she can determine the order in which the records are stored. This requires the following provisions:

Assignment of database key values by the user

```

LOCATION MODE IS { DIRECT
                  DIRECT-LONG } { item-name { IN
                                                OF } record-name
                                { identifier }
    
```

If this clause is specified, the database programmer is not only enabled to determine the order of the records, but it is also possible to select any associated set occurrences more conveniently (see the section “[Determining set occurrence selection](#)” on page 103). If you specify this clause, you cannot insert any new records of the specified record type with SQL.

item-name specifies an item which can contain database key values. It is at the same time defined as a key item for direct access.

If LOCATION MODE IS DIRECT is specified, *item-name* must be defined as a DATABASE-KEY item.

If LOCATION MODE IS DIRECT-LONG is specified, *item-name* must be defined as a DATABASE-KEY-LONG item.

record-name specifies the record type containing the database key item referred to by *item-name*.

identifier specifies the name of an item which is automatically generated by UDS/SQL for storing database key values. This item serves as a key item, but is not part of a record type. If you specify LOCATION MODE IS DIRECT, UDS/SQL generates the *identifier* item as a DATABASE-KEY item.

If you specify LOCATION MODE IS DIRECT-LONG, UDS/SQL generates the *identifier* item as a DATABASE-KEY-LONG item.

When entering a record, the database programmer can store a database key value signifying the position of the record within the record sequence in the item referred to by *item-name* or *identifier* (see the section [“DBTT \(Database Key Translation Table\)” on page 131](#)).

If the specified database key has already been assigned, another database key that is not in use is automatically assigned by UDS/SQL.

If the database programmer does not wish to assign the database key value independently, 0 may be entered instead. This resumes the automatic assignment of database key values by UDS/SQL.

4.7.2 Generating additional access paths for direct access on record type level

Defining a primary key for conversion by hash routine

```
LOCATION MODE IS CALC[ hash-routine]  
    USING item-name,... DUPLICATES ARE[ NOT] ALLOWED
```

With this clause, the user specifies the distributed storage of records on the basis of a hash routine. The storage area which you address with this routine is called the hash area. If the records are owners of a set, the corresponding set occurrences can also be conveniently selected (see the section [“Determining set occurrence selection” on page 103](#)).

A key declared by LOCATION MODE IS CALC is referred to as a CALC key. It is the primary key for the record type. It may comprise several items (compound key).

item-name specifies the item(s) representing the key; all named items must belong to the same associated record type and may be both numeric and alphanumeric.

With the DUPLICATES ARE[NOT] ALLOWED specification, the user determines (for a particular realm) whether UDS/SQL is to accept or reject duplicate key values.

Use *hash-routine* to specify the name of a module when you want to control the location of the data in the hash area yourself. If you make no entry for *hash-routine*, UDS/SQL uses its default hash routine.

If you have specified a *hash-routine*, it converts your primary key to a four-byte binary number. UDS/SQL then converts the binary number to a relative page number. The page with that number generally contains the associated record (see the section [“Direct CALC page” on page 210](#)).

If you want to define a user-specific hash routine, you must observe the following register conventions:

- Before and after running the routine, all UDS/SQL registers except Register 1 must have the same content.
The hash routine is branched to by BALR 14,15; return to the UDS/SQL hash routine is effected by BR 14.
- The following information is provided by the UDS/SQL DBH:
 - Register 1: The address of a word containing the address of the key item
 - Register 2: The address of a byte containing the length of the key item.
 - Register 3: The address of a word containing the number of pages of the hash area assigned to this key.
 - Register 13: Starting with the address (Register 13) + X'0C', 13 words are available to the user as a transaction-oriented save area for registers.
- The UDS/SQL DBH expects the result of the hash routine in Register 1. It converts the binary number encountered there into a relative page number as follows:
 - Bit 2³¹ in Register 1 is set to 0.
 - If the resulting value is less than the number of pages in the hash area (content of Register 3), this value is used as the relative page number.
 - If the resulting value is greater, UDS/SQL performs the following division:
content (Register 1) / content (Register 3)
The quotient of this division is used as the relative page number.

Example 1

The following exemplifies the conversion of a key value into a physical address using the UDS/SQL standard hash routine.

Let the key value be 9952333 and the number of pages in the hash area be 503. This results in the following arithmetic operations:

- 1) Beginning from the left, the key value is subdivided into words (units of four bytes length).

	9952/333
--	----------

- 2) An exclusive OR is performed with the corresponding binary representation.

	F9F9F5F2 → 11111001 11111001 11110101 11110010
	00F3F3F3 → 00000000 11110011 11110011 11110011
	11111001 00001010 00000110 00000001

- 3) The first bit is set to 0 (positive).

	01111001 00001010 00000110 00000001=790A0601
--	--

- 4) The result is divided by the number of pages in the hash area.

	790A0601 ₁₆ : 503 ₁₀
	=2030700033 ₁₀ : 503 ₁₀ = 4037117 Remainder= <u>2</u>

The remainder gives the physical address in the form of a relative page number (see the section [“Structure of a physical page address” on page 131](#)) within the hash area.

If the user employs a user-specific hash routine, the first two operations can be replaced. UDS/SQL always performs the last two operations.

Example 2

This example shows how a user can program a user-specific hash routine.

The program replaces the first two operations of the UDS/SQL standard hash routine by a division/remainder algorithm. The entire key value is considered a positive binary number which is divided by the number of available CALC pages.

The algorithm is the same as a normal-type division, the only difference being that three digits are brought down. An example of this type of operation follows (for the sake of simplicity, in decimal numbers):

```
1234567890 : 13
117
  6
 6456
  8
 8789
  1
 10
```

The resulting remainder is the relative page number. It is always smaller than the number of CALC pages, i.e. after returning to the standard hash routine, the page number remains unchanged. The DBH accepts the binary value stored in Register 1 as the final result.

This hash routine gives the same results as the standard hash routine if the key value is not longer than four bytes.

Viewed logically, the hash routine BYTEHASH generates a CALC page number from a CALC key in 2 steps:

1. The order of the bytes within the CALC key is reversed.
2. The entire byte string that results is treated as a positive integer and divided by the number of CALC pages. The remainder of the division is then the CALC page number.

Example in Assembler:

```

BYTEHASH CSECT READ
BYTEHASH AMODE ANY
BYTEHASH RMODE ANY
        USING *,15
        STM 4,8,12(13)
        LM 4,7 ALLZEROS
        L 8,0(0,1)
        LA 4,0(0,8)
        BCTR 4,0 _____ (1)
        IC 5,0(0,2) _____ (2)
        DR 6,5 _____ (3)
HASHBYTE SRDL 6,24
        IC 7,0(4,5)
        D 6,0(0,3)
        BCT 5,HASHBYTE
        LR 1,6
        LM 4,8,12(13)
        BR 14
ALLZEROS DC 4F'0'
        END

```

- (1) Register 4 always contains the address before the CALC key.
- (2) Register 5 contains the index to the CALC key; the initial value is the length of the CALC key.
- (3) If a zero is passed for the length of the CALC key, this division results in a P104.

Defining secondary keys for conversion by hash routine

```
SEARCH KEY IS item-name,... USING CALC[ hash-routine]
DUPLICATES ARE[ NOT] ALLOWED
```

A key declared by SEARCH KEY IS... is a SEARCH key or secondary key. It may consist of more than one item (compound key).

item-name specifies the item(s) comprising the key. All items must be part of the corresponding record type.

DUPLICATES ARE[NOT] ALLOWED specifies whether UDS/SQL is to accept or reject duplicate key values.

hash-routine denotes the name of a module which converts the secondary key to a 4-byte binary number. This binary number is subsequently converted into a relative page number by UDS/SQL. The corresponding page contains a pointer to the record (see the section [“Indirect CALC page” on page 213](#)).

The hash routine does not compute the record address directly, as it is up to the user to place the record either via the primary key or by means of SSL statements.

Note that hash areas placed with SSL statements must be assigned names (see [page 107](#)).

If *hash-routine* is omitted, UDS/SQL uses the same standard hash routine as for conversion of the primary key. (For programming a hash routine and running the standard hash routine, refer to [“Defining a primary key for conversion by hash routine” on page 87](#).)

More than one secondary key can be defined for one record type.

Example

```
RECORD NAME IS ARTICLE
      .
      .
      .
SEARCH KEY IS ART-NO-AVAIL, COLOR-NO-AVAIL, SIZE USING CALC.....
SEARCH KEY IS ARTICLE-NAME USING CALC.....
01 ART-NO-AVAIL          PICTURE IS 9(4).
01 COL-NO-AVAIL          PICTURE IS 99.
01 SIZE                  PICTURE IS 99.
01 ARTICLE-NAME          TYPE IS CHARACTER 40.
```

Defining a secondary key for direct access via table

```
SEARCH KEY IS item-name,... USING INDEX [NAME IS name]  
DUPLICATES ARE [NOT] ALLOWED
```

A key declared by SEARCH KEY IS... is a SEARCH key or secondary key. It may be made up of more than one item.

item-name specifies the item(s) comprising the key. All items must be part of the corresponding record type.

name specifies the name of the table. This name is referred to in the SSL statements concerning the table.

DUPLICATES ARE [NOT] ALLOWED specifies whether UDS/SQL is to accept or reject duplicate key values.

Based on this definition, UDS/SQL sets up a record SEARCH key table. This table contains the values of the secondary keys of all records of a record type, representing a unique reference between key value, database key value and physical address of the record. The physical address is not automatically updated, however, when the position of the record is changed.

A SEARCH key table is used only for direct access to the records of a record type. It always involves several levels in order to speed up access.

Several independent secondary keys can be defined for one record type.

4.7.3 Determining the order of records within a set occurrence

Two basic concepts in determining the logical order of the member records within a set occurrence can be distinguished:

- sorting without key values, and
- sorting according to primary key values

They are described in detail below.

Sorting without key values

ORDER IS { LAST
FIRST
NEXT
PRIOR
IMMATERIAL }

ORDER IS LAST

Specifies that the order of the member records in a set occurrence corresponds to the chronological sequence in which they are stored.

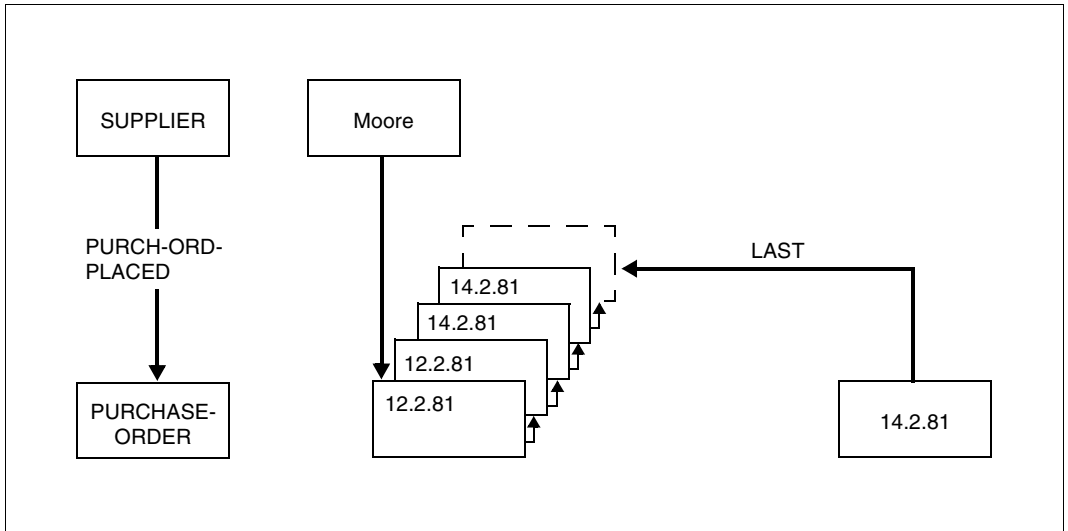


Figure 22: Record sequence for ORDER IS LAST

ORDER IS FIRST

Specifies the reverse order to that in which the member records were chronologically entered.

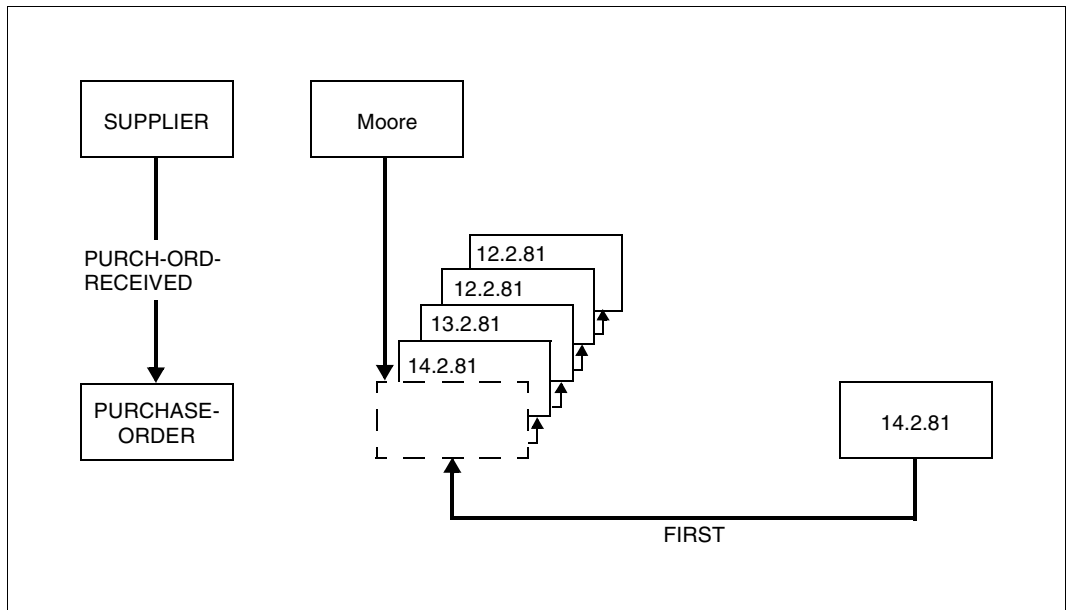


Figure 23: Record sequence for ORDER IS FIRST

ORDER IS NEXT/PRIOR

If this type of ordering is specified, the database programmer has the possibility of establishing a certain order when storing the member records.

The CRS (current record of set), i.e. the record of a set referred to last, is used as a reference point for positioning the record to be stored. Thus the CRS determines:

- the set occurrence, and
- the position within the set occurrence.

UDS/SQL places the record to be stored:

- immediately behind the CRS if ORDER IS NEXT, or
- immediately before the CRS if ORDER IS PRIOR.

If automatic selection of a set occurrence has been specified (see the [section “Determining set occurrence selection” on page 103](#)), UDS/SQL automatically makes the owner record of the set occurrence the current record of set (CRS). In this case, ORDER IS NEXT has the same effect as ORDER IS FIRST.

Sets defined with ORDER IS NEXT or ORDER IS PRIOR cannot be accessed by SQL with INSERT or UPDATE.

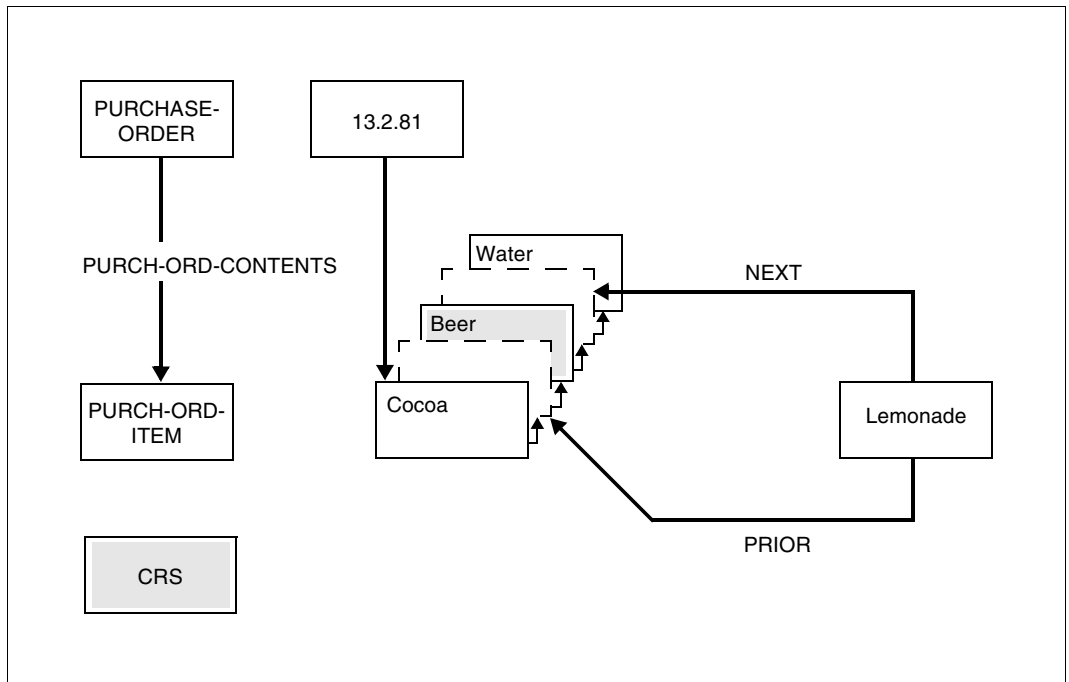


Figure 24: Record order if ORDER IS NEXT/PRIOR

ORDER IS IMMATERIAL

The user leaves ordering of the member records to UDS/SQL. Default values for ORDER IS IMMATERIAL are

- for the SSL specification MODE IS CHAIN: ORDER IS NEXT
- for the SSL specification MODE IS LIST/POINTER ARRAY: ORDER IS LAST

Sets defined with ORDER IS IMMATERIAL cannot be accessed by SQL with INSERT or UPDATE.

Sorting member records according to primary key values

```
ORDER IS SORTED BY { DATABASE-KEY
                    .
                    .
                    .
                    { ASCENDING }
                    { DESCENDING } } KEY IS item-name,...
```

ORDER IS SORTED BY DEFINED KEYS DUPLICATES ARE[NOT] ALLOWED

Items defined as keys with this ORDER clause are primary keys of a set. UDS/SQL sorts the member records of the set occurrences according to the values of the primary key (which may consist of one or several items of a member record type) either in ASCENDING or DESCENDING order. If the key consists of several items, it is a compound key.

item-name specifies the item(s) comprising the key. When inserting new member records in a set occurrence, UDS/SQL automatically selects the position corresponding to the key value. If the user changes key values in the database, UDS/SQL automatically updates the sequence of the associated member records.

DUPLICATES ARE [NOT] ALLOWED specifies whether UDS/SQL is to accept or reject duplicate key values.

If DUPLICATES ARE ALLOWED is specified, member records with identical key values are sorted by database key values in ascending order.

The primary key does not only determine the sequence of the member records within a set occurrence. If, by means of SSL, a storage mode is established which causes UDS/SQL to set up a table in order to sort a set occurrence (see [page 144](#); pointer array, list), UDS/SQL also uses the table as a direct access path.

In this case the above clause must be supplemented by INDEXED, which results in the creation of a multi-level table (see [page 100](#)).

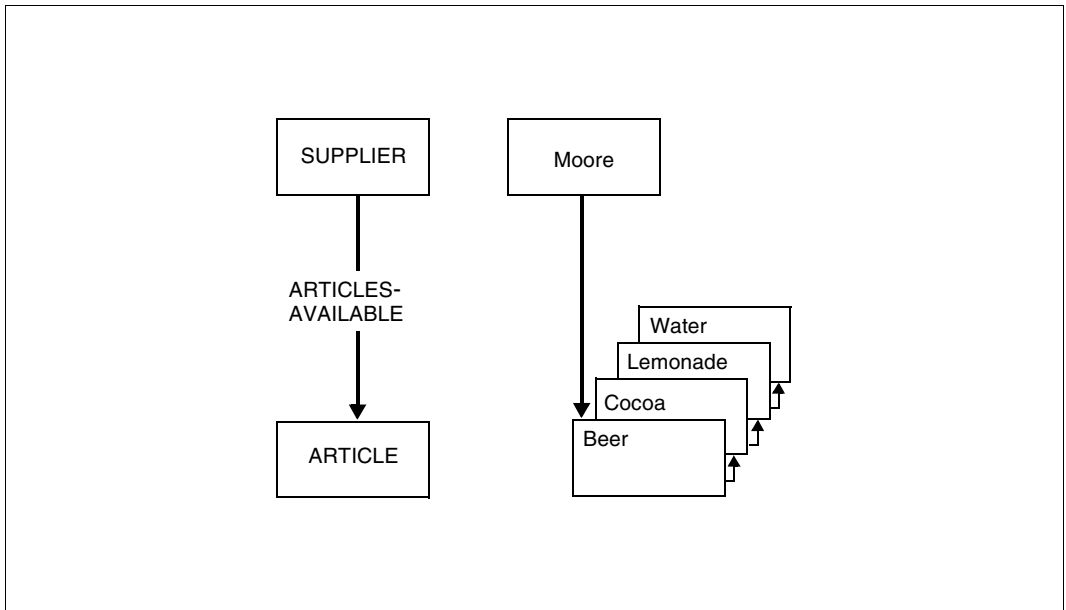


Figure 25: Sorting a set occurrence according to the key ART-NAME

ORDER IS SORTED BY DATABASE-KEY

UDS/SQL sorts the member records of the set occurrence in ascending order according to the values of the database key. In this case, the database key is also referred to as primary key of the set.

The values of the database key can be defined by the user (see the [section “Direct and sequential access on record type level via database key value” on page 85](#)) or left to UDS/SQL.

If, by means of SSL, a storage mode is established which causes UDS/SQL to set up a table in order to sort a set occurrence (see [page 144](#); pointer array, list), UDS/SQL also uses the table as direct access path.

In this case, the above clause must be supplemented by INDEXED, which results in the creation of a multi-level table (see [page 100](#)).

4.7.4 Generating additional paths for direct access on set level

Unlike on record level, on set level UDS/SQL supports direct access only via tables. Only SYSTEM sets (see [section “SYSTEM set” on page 105](#)) allow direct access via a hash routine. Two kinds of tables can be defined for direct access on set level:

- a table containing the primary key of the set, or
- one or more tables containing a secondary key (SEARCH key) of the set.

Generating an additional access path via primary key

```
ORDER IS SORTED INDEXED[ NAME IS name] BY
    { DATABASE-KEY
      { DEFINED KEYS DUPLICATES ARE [ NOT ] ALLOWED }
      .
      .
      .
    { ASCENDING }
    { DESCENDING } } KEY IS item-name,...
```

ORDER IS SORTED INDEXED BY DEFINED KEYS DUPLICATES ARE [NOT] ALLOWED
 ASCENDING/DESCENDING KEY IS *item-name*,...

Specifies a primary key determining the order of the member records within a set occurrences (see [section “Determining the order of records within a set occurrence” on page 98](#)). It also sets up a multi-level table for each set occurrence of the set. For each record belonging to the set occurrence, the table represents a unique reference between primary key value, database key value and physical record address. In the case of a change in the position of the record, its physical address is not automatically updated, however.

item-name specifies the item(s) of the member record type comprising the key.

DUPLICATES ARE [NOT] ALLOWED specifies whether UDS/SQL is to accept or reject duplicate key values.

Example

```
RECORD NAME IS SUPPLIER
.
.
.

RECORD NAME IS ARTICLE
.
.
.

01 ARTICLE-NAME PICTURE IS X(30).

SET NAME IS ARTICLES-AVAILABLE
  ORDER IS SORTED INDEXED BY DEFINED KEYS.....
  OWNER IS SUPPLIER
MEMBER IS ARTICLE.....
  ASCENDING KEY IS ARTICLE-NAME
.
.
.
```

ORDER IS SORTED INDEXED BY DATABASE-KEY

Specifies the order of the members of a set occurrence (see the [section “Determining the order of records within a set occurrence” on page 98](#)). It also sets up a multi-level table for each set occurrence of the set. The table represents a unique reference between database key value and physical address for each record belonging to the set occurrence. UDS/SQL makes use of this access path when inserting a record in or deleting a record from the set occurrence and in sequential reading, but not for direct accesses.

Generating an additional access path via secondary key

```
SEARCH KEY IS item-name,... USING INDEX [NAME IS name]  
DUPLICATES ARE[ NOT] ALLOWED
```

A key defined by means of SEARCH-KEY IS... is called a SEARCH key or secondary key. It may consist of more than one item.

item-name specifies the item(s) comprising the key. All items have to be part of the member.

name specifies the name of the table. This name is referred to in SSL statements concerning the table.

DUPLICATES ARE[NOT] ALLOWED specifies whether UDS/SQL is to accept or reject duplicate key values.

Based on this definition, UDS/SQL sets up a SEARCH key table on set level or a set SEARCH key table for each set occurrence of the set. This table represents a unique reference between secondary key value, database key value and physical address for each record belonging to the set occurrence. It must be a multi-level table. When the position of the record is changed, the physical address is not automatically updated.

Several independent secondary keys can be defined for each set.

4.7.5 Determining set occurrence selection

SET OCCURRENCE SELECTION IS THRU

$$\left\{ \begin{array}{l} \text{CURRENT OF SET} \\ \text{LOCATION MODE OF OWNER} [\text{ALIAS FOR } \left\{ \begin{array}{l} \textit{item-name} \\ \textit{identifier-1} \end{array} \right\} \text{ IS } \textit{identifier-2}] \dots \end{array} \right\}$$

When accessing records via sets, it is first necessary to select the desired set occurrences.

Two selection options are available:

SET OCCURRENCE SELECTION IS THRU CURRENT OF SET

In this case, the record last referenced within a set (CRS) identifies the set occurrence.

SET OCCURRENCE SELECTION IS THRU LOCATION MODE OF OWNER
[ALIAS FOR *item-name* / *identifier-1* IS *identifier-2*]...

$$[\text{ALIAS FOR } \left\{ \begin{array}{l} \textit{item-name} \\ \textit{identifier-1} \end{array} \right\} \text{ IS } \textit{identifier-2}] \dots$$

This option requires you to have defined a unique primary key for the owner record type with one of the following clauses:

- LOCATION MODE IS DIRECT or LOCATION MODE IS DIRECT-LONG
- LOCATION MODE IS CALC..... DUPLICATES ARE NOT ALLOWED

This key is either the database key or a CALC key.

UDS/SQL can use this option either to read a record from a set occurrence (see the manual “[Application Programming](#)”, FIND 7) or to insert a new record into a set occurrence (see STORE and MODIFY in the “[Application Programming](#)” manual). All the programmer has to do is provide UDS/SQL with the key value uniquely identifying the owner record of the set occurrence. UDS/SQL will automatically select the set occurrence on owner record type level using direct access via the database key or the hash routine, and make the selected owner record the CRS.

Simultaneous automatic selection of several owner records from one record type

With certain data structures, it may be necessary to select several set occurrences (i.e. several owner records) at the same time when inserting a new member record, where all owner records are part of the same record type.

Such a case applies when the following two conditions are met:

- Two record types are connected by more than one set.
- The member record type is defined as an AUTOMATIC member in at least two of these sets.

If the selection option LOCATION MODE OF OWNER has been defined, UDS/SQL automatically selects the set occurrences of the sets. In order to select several different owner records simultaneously, however, the user then requires an additional language resource, because it is not possible to provide the items defined as primary keys with LOCATION MODE IS... with multiple key values at the same time.

The following clause can be used to create an additional item for the owner selection in this set to hold the key values:

```
ALIAS FOR { item-name } IS identifier-2
           { identifier-1 }
```

item-name and *identifier-1* specify items that have been declared primary keys by LOCATION MODE IS... .

identifier-2 specifies the name of the additional item to be created. UDS/SQL automatically provides this item with the same item type and length as items referred to by *item-name* or *identifier-1*.

If a key consisting of several items has been defined by LOCATION MODE IS CALC, the ALIAS clause must be repeated an appropriate number of times in order to create one substitute item for every key item.

4.8 Special sets

4.8.1 SYSTEM set

```
OWNER IS SYSTEM.
```

A record type which is not related to a higher-ranking record type on the basis of its data structure can still be declared member of a set.

This applies in the following cases:

- For reasons of sequential processing, the records are to be in a different order than ascending order according to database key values (see [page 85](#) and [page 94](#) ff).
- Only a certain collection of records is to be processed (see [section “Defining the type of membership of records in a set” on page 79](#)).

In these cases, the record type is declared member of a set which has only a symbolic owner record type named SYSTEM. Accordingly such sets are called SYSTEM sets.

The symbolic owner record type contains exactly one record which is automatically generated by UDS/SQL. It is called an anchor record. A SYSTEM set thus consists of just one set occurrence.

In addition to the possibilities of a normal set, SYSTEM sets allow direct access via hash routines.

Defining secondary keys for conversion by hash routine

```
SEARCH KEY IS item-name,... USING CALC[ hash-routine]  
DUPLICATES ARE[ NOT] ALLOWED
```

A key declared by SEARCH KEY IS... is a SEARCH key or secondary key. It may be consist of more than one item.

item-name specifies the item(s) comprising the key. All items must be part of the corresponding member record type.

DUPLICATES ARE[NOT] ALLOWED specifies whether UDS/SQL is to accept or reject duplicate key values.

hash-routine denotes the name of a module converting the secondary key to a 4-byte binary number. This binary number is subsequently converted into a relative page number by UDS/SQL. The corresponding page contains the pointer to the record (see [page 213](#)).

If *hash-routine* is omitted, UDS/SQL uses the same standard hash routine as for conversion of the primary key on record type level (for programming a hash routine as well as using the standard hash routine, see [page 87](#)).

More than one secondary key may be defined.

4.8.2 Dynamic set

In general, DML statements process one record of the database at a time. One particular DML retrieval statement, however, selects several records from the database at the same time (see the “[Application Programming](#)“ manual, FIND 7). The selected records are buffered for further processing. UDS/SQL does this by automatically making the records member records in a dynamic set.

It is characteristic of a dynamic set that, during a transaction, it can accept records of various record types, identifying them as intermediate results of a search query, and discard them when the intermediate result is no longer required. Membership in a dynamic set is thus of the OPTIONAL MANUAL type and there is no defined member record type.

One set occurrence is sufficient for storing selected records in a dynamic set as connection data relating to certain owner records need not be stored, i.e. a dynamic set is declared as SYSTEM set.

The set is called dynamic because it holds member records for the course of one transaction only. There is no static set membership.

In order to be able to define a dynamic set, the user must include a temporary realm in the schema. The dynamic set is defined as follows:

```
SET NAME IS set-name
SET IS DYNAMIC
ORDER IS IMMATERIAL
OWNER IS SYSTEM.
```

If the Interactive Query Language IQL is to be used, the user must define eight dynamic sets with the names IQL-DYN1 through IQL-DYN8. More details can be found in the manual “[Interactive Query System IQS](#)“.

4.9 Assigning names to hash areas and tables

Names must be assigned to hash areas and tables for secondary keys and tables for primary keys if they are to be referenced by the SSL for the following purposes:

- determining the physical placement of the hash areas and tables
- preventing redundancy in tables
- defining the type of table reorganization desired

Names are assigned to hash areas or tables for secondary keys on record type and on set level by:

```
SEARCH KEY IS..... USING { CALC } NAME IS name.....  
                             { INDEX }
```

Names are assigned to tables for primary keys on set level by means of:

```
ORDER IS SORTED INDEXED NAME IS name.....
```

where *name* specifies the name of the hash area or the table.

4.10 The realm concept

In order to take the aspects of

- data privacy,
- data recovery,
- concurrent access, and
- the logical association of certain data

into account, it is often advisable to subdivide the database into subunits. These subdivisions are called “realms” or “areas”. They are generated as BS2000 files at database creation (see the “[Creation and Restructuring](#)” manual, Database creation). There are realms which contain UDS/SQL-internal information only and realms for user data. The latter are called user realms.

A maximum of 123 realms can be defined for a database with a page length of 2048 bytes (2-Kbyte page format).

A database with a page length of 4000 or 8096 bytes (4-Kbyte or 8-Kbyte page format), by contrast, may be subdivided into a maximum of 245 realms.

Data privacy

With the BPRIVACY utility, the database administrator can grant certain groups of users access privileges to database objects (realms, record types, sets).

When defining the subschema, a user can restrict the use of data to certain realms.

Data security

The subdivision into realms allows the effects of hardware errors or read/write errors to be restricted to a few realms or even to one realm. In such cases, only the realm or realms concerned needs to be recovered by means of realm copies or after-images, if used (see the “[Recovery, Information and Reorganization](#)” manual, BMEND). Data that is updated frequently should therefore be stored in a separate realm from data that is seldom updated or updated only at certain times.

Handling concurrent access

When a transaction is opened, the database programmer states the realms to be accessed within the transaction (see the “[Application Programming](#)” manual, READY). He can also define usage modes for realms, restricting or prohibiting concurrent access to these realms by other transactions. The subdivision of the database into realms helps keep mutual interference by transactions to a minimum.

Data which is often accessed concurrently can be stored in different realms, and the realms can be assigned different disk drives. This allows concurrent accesses by different transactions, which again reduces access time.

Logical association of data

Data can be distributed over the different realms so that programs need access only some of the existing realms. Realms can be connected or disconnected at the user’s discretion by means of the BMEND utility or, in a session, by means of the administration with DAL commands. There is no need to waste resources on realms which are rarely used.

4.10.1 Defining a realm

```
AREA NAME IS realm-name
```

realm-name specifies the name of the realm assigned by the user. No further specifications are required to define a realm. Its size is specified at database creation (see the “[Creation and Restructuring](#)” manual, Database creation).

4.10.2 Defining allocation of records to realms

```
RECORD NAME IS record-name  
WITHIN realm-name-1[,realm-name-2,... AREA-ID IS identifier]
```

The allocation of data to realms and the placement of data within realms is performed mainly when defining the physical storage structure by means of SSL (see the [section “Defining the placement of member records, tables and hash areas” on page 160](#)). The allocation of records to realms is however defined by means of the schema DDL.

realm-name-1, etc. specifies all the realms to contain records of the record type *record-name*. If you specify multiple realms, you cannot insert any new records of this record type with SQL.

identifier must be specified only if more than one realm name is specified.

identifier is specified by the user to denote the name of an item that is automatically generated by UDS/SQL to store one of the specified realm names in each case. Prior to storing a record, the database programmer must provide this item with the name of the realm that is to contain the record (see the [“Application Programming” manual, STORE](#)).

The time required for retrieving member records via their set relationship can be decreased by storing the member records in the same realm as the associated owner record. This allows further placement optimization for records by means of SSL (see the [section “Placement within a realm” on page 163](#)).

4.10.3 Temporary realms

```
AREA IS TEMPORARY
```

A temporary realm must be defined if the schema description contains dynamic sets or if UDS/SQL must automatically generate a dynamic set because a specific DML statement has been given (see the [“Application Programming” manual, FIND 7](#)). The temporary realm serves to store the table that represents the set occurrence of the dynamic set and that points to the associated member records.

A temporary realm must also be defined when you wish to access the database using SQL.

4.11 Assigning name and privacy to a schema

schema-name
[PRIVACY LOCK FOR COPY IS *literal-1* [OR *literal-2*]]_

The definition of a database with the schema DDL always begins with the assigning of a name to the schema.

schema-name specifies the name assigned to the schema by the user. This name is later referenced by the SSL, the subschema DDL and the utility routines.

How the data in the database can be accessed by means of the subschema depends on the individual user's access rights.

literal-1 and *literal-2* specify passwords which prevent the creation of unauthorized subschemas. In order to create a subschema, knowledge of at least one password is required.

4.12 Comprehensive example of DDL application

This example shows the schema of a mail order business. The schema supports the following functions:

- Management of master data relating to customers, articles, suppliers, customer orders and orders placed with suppliers
- Stock management
- Issuing of reminders
- Parts list processing for replacement parts

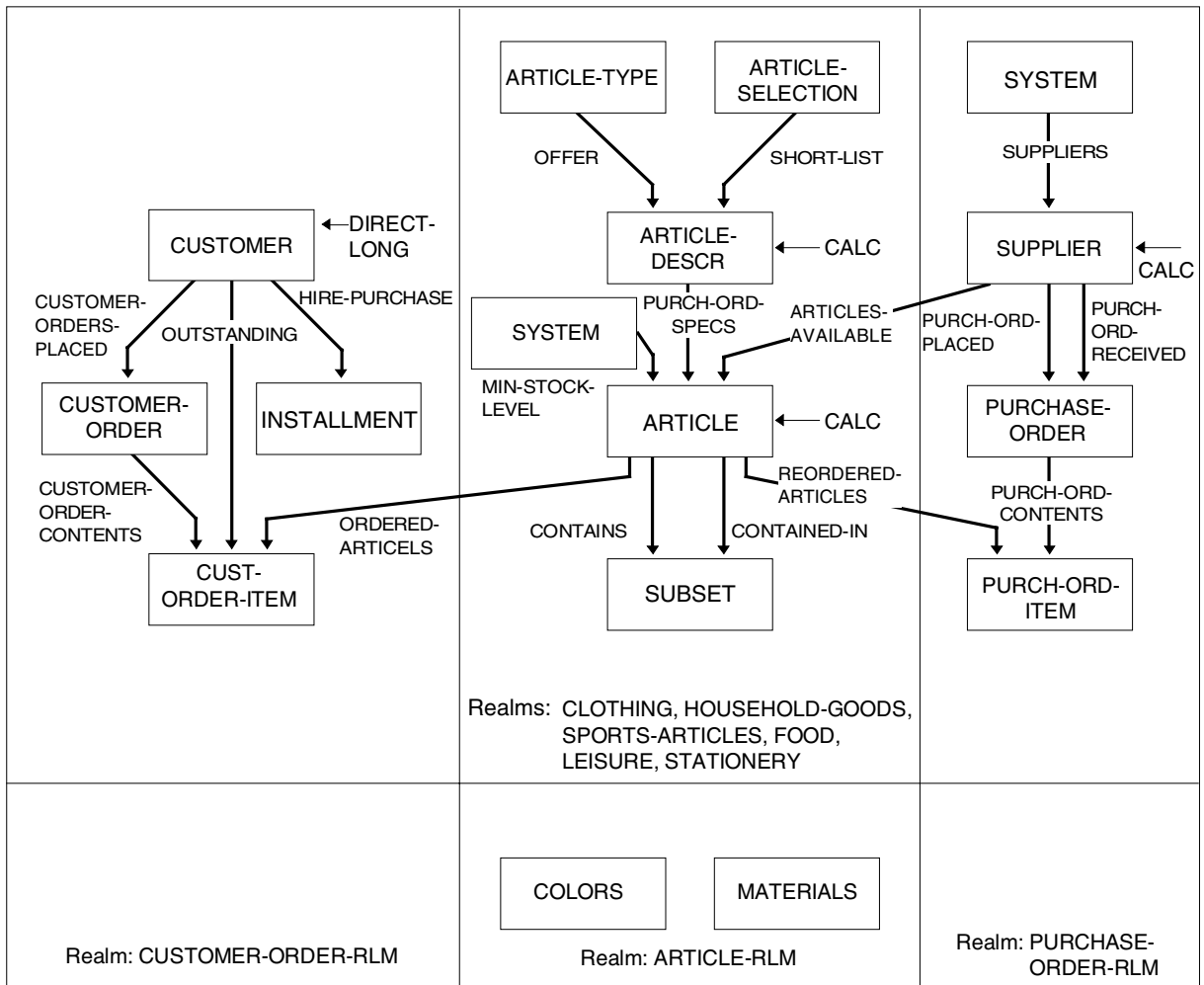


Figure 26: Schema of a mail order business

A rough selection of articles is possible by means of the criteria ARTICLE-TYPE and ARTICLE-SELECTION. This selection leads to a detailed article description. An article description can comprise several articles differing in color, size and price.

The sets CONTAINS and CONTAINED-IN comprise a parts list used for parts supply. In the ARTICLE records, colors are coded by numbers, materials in by abbreviations with percentages. The record types COLOR and MATERIALS contain the assignments of the codes to the associated meanings.

In the example, the schema is designed to provide as full an understanding of DDL clauses as possible.

```

1          SCHEMA NAME IS          MAIL-ORDERS
2          PRIVACY LOCK FOR COPY IS "SHIP-KEY" .
3          *
4          *
5          *
6          AREA NAME IS            CUSTOMER-ORDER-RLM.
7          AREA NAME IS            PURCHASE-ORDER-RLM.
8          AREA NAME IS            CLOTHING.
9          AREA NAME IS            HOUSEHOLD-GOODS.
10         AREA NAME IS            SPORTS-ARTICLES.
11         AREA NAME IS            FOOD.
12         AREA NAME IS            LEISURE.
13         AREA NAME IS            STATIONERY.
14         AREA NAME IS            ARTICLE-RLM.
15         AREA NAME IS            SEARCH-RLM
16         AREA IS TEMPORARY.
17         *
18         *
19         *
20         RECORD NAME IS          CUSTOMER
21         LOCATION MODE IS DIRECT-LONG CUST-NO OF CUSTOMER
22         WITHIN CUSTOMER-ORDER-RLM.
23         *
24         01 CUST-NAME             TYPE IS CHARACTER 30.
25         01 CUST-F-NAME           TYPE IS CHARACTER 30.
26         01 CUST-NO               TYPE IS DATABASE-KEY-LONG.
27         *
28         *
29         RECORD NAME IS          CST-ORDERS
30         WITHIN CUSTOMER-ORDER-RLM.
31         *
32         01 ORD-NO                PICTURE IS 9(4).
33         01 ORD-YEAR              PICTURE IS 99.
34         01 ORD-MONTH             PICTURE IS 99.
35         01 ORD-DAY               PICTURE IS 99.
36         01 ORD-STATUS            PICTURE IS X.

```

```

37      *
38      *
39      RECORD NAME IS                ORD-ITEM
40      WITHIN CUSTOMER-ORDER-RLM.
41      *
42      01 ORD-NO-ITEM                PICTURE IS 99.
43      01 ORD-QTY                    TYPE IS DECIMAL 6.
44      01 PAY-INSTAL-CODE            PICTURE IS X.
45      01 ORD-STATUS-ITEM            PICTURE IS X.
46      *
47      *
48      RECORD NAME IS                INSTALMENT
49      WITHIN CUSTOMER-ORDER-RLM
50      SEARCH KEY IS YEAR-NEXT-INSTAL, MONTH-NEXT-INSTAL,
51      DAY-NEXT-INSTAL
52      USING INDEX NAME IS SEARCH-TAB-INSTALMENT
53      DUPLICATES ARE ALLOWED.
54      *
55      01 ORD-NO                    PICTURE IS 9(4).
56      01 ORD-NO-ITEM                PICTURE IS 99.
57      01 TOT-PRICE-INSTAL           TYPE IS DECIMAL 9,2.
58      01 SINGLE-INSTAL              TYPE IS DECIMAL 7,2.
59      01 BALANCE                    TYPE IS DECIMAL 9,2.
60      01 YEAR-NEXT-INSTAL           PICTURE IS 99.
61      01 MONTH-NEXT-INSTAL          PICTURE IS 99.
62      01 DAY-NEXT-INSTAL            PICTURE IS 99.
63      *
64      *
65      RECORD NAME IS                ART-TYPE
66      WITHIN CLOTHING, HOUSEHOLD-GOODS, SPORTS-ARTICLES, FOOD,
67      LEISURE, STATIONERY AREA-ID IS RLM-SELECTION-1
68      SEARCH KEY IS ART-NAME USING CALC
69      NAME IS SEARCH-TAB-ART-TYPE DUPLICATES ARE ALLOWED.
70      *
71      01 ART-NAME                    TYPE IS CHARACTER 25.
72      *
73      *
74      RECORD NAME IS                ART-SELECTION
75      WITHIN CLOTHING, HOUSEHOLD-GOODS, SPORTS-ARTICLES, FOOD,
76      LEISURE, STATIONERY AREA-ID IS RLM-SELECTION-2
77      SEARCH KEY IS SEL-CRIT USING INDEX
78      NAME IS SEARCH-TAB-ARTICLE-SELECTION
79      DUPLICATES ARE ALLOWED.
80      *
81      01 SEL-CRIT                    TYPE IS CHARACTER 25.
82      *
83      *
84      RECORD NAME IS                ART-DESCR

```

```

85          LOCATION MODE IS CALC USING ARTICLE-NAME
86          DUPPLICATES ARE ALLOWED
87          WITHIN CLOTHING, HOUSEHOLD-GOODS, SPORTS-ARTICLES, FOOD,
88          LEISURE, STATIONERY AREA-ID IS RLM-SELECTION-3.
89      *
90      01 ART-NO          PICTURE IS 9(6).
91      01 ARTICLE-NAME   TYPE IS CHARACTER 40.
92      01 MATERIAL       OCCURS 4 TIMES.
93          02 PERZENT    PICTURE IS 99.
94          02 MAT-CODE   PICTURE IS X.
95      01 LENGTH-FIELD   TYPE IS BINARY 15.
96      01 ART-INFO       PICTURE IS LX(500)
97                          DEPENDING ON LENGTH-FIELD.
98      *
99      *
100     RECORD NAME IS          ARTICLE
101     LOCATION MODE IS CALC USING ART-NO, COL-NO, ART-SIZE
102     DUPPLICATES ARE NOT ALLOWED
103     WITHIN CLOTHING, HOUSEHOLD-GOODS, SPORTS-ARTICLES, FOOD,
104     LEISURE, STATIONERY AREA-ID IS RLM-SELECTION-4
105     SEARCH KEY IS ART-NO-AVAIL, COL-NO-AVAIL, ART-SIZE
106     USING CALC NAME IS SEARCH-TAB-ARTICLE-1
107     DUPPLICATES ARE NOT ALLOWED
108     SEARCH KEY IS ARTICLE-NAME USING CALC
109     NAME IS SEARCH-TAB-ARTICLE-2 DUPPLICATES ARE ALLOWED.
110     *
111     01 ART-NO          PICTURE IS 9(6).
112     01 COL-NO          PICTURE IS 99.
113     01 ARTICLE-NAME   TYPE IS CHARACTER 40.
114     01 ART-NO-AVAIL   PICTURE IS 9(4).
115     01 COL-NO-AVAIL   PICTURE IS 99.
116     01 ART-SIZE       PICTURE IS 99.
117     01 PRICE          TYPE IS DECIMAL 7,2.
118     01 INSTALMENT-PRICE TYPE IS DECIMAL 7,2.
119     01 MAX-STOCK      TYPE IS DECIMAL 10.
120     01 MIN-STOCK      TYPE IS DECIMAL 3.
121     01 CURR-STOCK     TYPE IS DECIMAL 10.
122     01 STATISTICS     TYPE IS DECIMAL 15.
123     01 NOT-AVAIL-CODE PICTURE IS X.
124     *
125     *
126     RECORD NAME IS          SUBSET
127     WITHIN HOUSEHOLD-GOODS, SPORTS-ARTICLES
128     AREA-ID IS RLM-SELECTION-5.
129     *
130     01 QUANTITY       PICTURE IS 99.
131     *
132     *

```

```

133         RECORD NAME IS                COLORS
134         WITHIN ARTICLE-RLM
135         SEARCH KEY IS COL-NAME USING CALC DUPLICATES ARE NOT ALLOWED
136         SEARCH KEY IS COL-NO USING CALC DUPLICATES ARE NOT ALLOWED.
137     *
138     01 COL-NO                            PICTURE IS 99.
139     01 COL-NAME                          TYPE IS CHARACTER 20.
140 *
141 *
142     RECORD NAME IS                MATERIALS
143     WITHIN ARTICLE-RLM
144     SEARCH KEY IS MAT-CODE USING INDEX
145     NAME IS SEARCH-TAB-MATERIAL-1 DUPLICATES ARE NOT ALLOWED
146     SEARCH KEY IS MAT-NAME USING INDEX
147     NAME IS SEARCH-TAB-MATERIAL-2 DUPLICATES ARE NOT ALLOWED.
148 *
149     01 MAT-CODE                          TYPE IS CHARACTER 1.
150     01 MAT-NAME                          TYPE IS CHARACTER 20.
151 *
152 *
153     RECORD NAME IS                SUPPLIER
154     LOCATION MODE IS CALC USING SUPPL-NO, SUPPL-NAME
155     DUPLICATES ARE NOT ALLOWED
156     WITHIN PURCHASE-ORDER-RLM.
157 *
158     01 SUPPL-NO                          PICTURE IS 9(5).
159     01 SUPPL-NAME                        TYPE IS CHARACTER 30.
160     01 SUPPL-PCODE                       TYPE IS CHARACTER 4.
161     01 SUPPL-TOWN                        TYPE IS CHARACTER 30.
162     01 SUPPL-STREET                     TYPE IS CHARACTER 30.
163     01 SUPP-STREET-NO                   TYPE IS CHARACTER 3.
164     01 SUPPL-TEL                         PICTURE IS 9(12).
165     01 SUPPL-POBOX                       PIC 9(4).
166     01 SUPP-TELEX                        PIC 9(12).
167 *
168 *
169     RECORD NAME IS                PURCHASE-ORDER
170     WITHIN PURCHASE-ORDER-RLM.
171 *
172     01 P-ORD-NO                          PICTURE IS 9(4).
173     01 P-ORD-YEAR                        PICTURE IS 99.
174     01 P-ORD-MONTH                      PICTURE IS 99.
175     01 P-ORD-DAY                        PICTURE IS 99.
176 *
177 *
178     RECORD NAME IS                P-ORD-ITEM
179     WITHIN PURCHASE-ORDER-RLM.
180 *

```

```
181          01 P-ORD-NO-ITEM          PICTURE IS 99.
182          01 P-ORD-QTY              TYPE IS DECIMAL 10.
183          *
184          *
185          *
186          SET NAME IS                CST-ORD-PLACED
187          ORDER IS SORTED INDEXED BY DEFINED KEYS
188          DUPLICATES ARE NOT ALLOWED
189          OWNER IS CUSTOMER.
190          MEMBER IS CST-ORDERS OPTIONAL AUTOMATIC
191          ASCENDING KEY IS ORD-NO
192          SEARCH KEY IS ORD-YEAR, ORD-MONTH, ORD-DAY USING INDEX
193          NAME IS SEARCH-TAB-C-O-PLCD DUPLICATES ARE ALLOWED
194          SET OCCURRENCE SELECTION IS THRU LOCATION MODE OF OWNER.
195          *
196          *
197          SET NAME IS                CST-O-CONTENTS
198          ORDER IS SORTED INDEXED BY DEFINED KEYS
199          DUPLICATES ARE NOT ALLOWED
200          OWNER IS CST-ORDERS.
201          MEMBER IS ORD-ITEM MANDATORY AUTOMATIC
202          ASCENDING KEY IS ORD-NO-ITEM
203          SET OCCURRENCE SELECTION IS THRU CURRENT OF SET.
204          *
205          *
206          SET NAME IS                OUTSTANDING
207          ORDER IS LAST
208          OWNER IS CUSTOMER.
209          MEMBER IS ORD-ITEM OPTIONAL AUTOMATIC
210          SET OCCURRENCE SELECTION IS THRU LOCATION MODE OF OWNER.
211          *
212          *
213          SET NAME IS                HIRE-PURCHASE
214          ORDER IS LAST
215          OWNER IS CUSTOMER.
216          MEMBER IS INSTALMENT MANDATORY AUTOMATIC
217          SET OCCURRENCE SELECTION IS THRU LOCATION MODE OF OWNER.
218          *
219          *
220          SET NAME IS                OFFER
221          ORDER IS SORTED INDEXED BY DEFINED KEYS
222          DUPLICATES ARE ALLOWED
223          OWNER IS ART-TYPE.
224          MEMBER IS ART-DESCR MANDATORY AUTOMATIC
225          ASCENDING KEY IS ARTICLE-NAME
226          SET OCCURRENCE SELECTION IS THRU CURRENT OF SET.
227          *
228          *
```

```

229         SET NAME IS                SHORT-LIST
230         ORDER IS SORTED INDEXED BY DEFINED KEYS
231         DUPLICATES ARE ALLOWED
232         OWNER IS ART-SELECTION.
233     MEMBER IS ART-DESCR MANDATORY AUTOMATIC
234         ASCENDING KEY IS ARTICLE-NAME
235         SET OCCURRENCE SELECTION IS THRU CURRENT OF SET.
236     *
237     *
238     SET NAME IS                P-ORD-SPEC
239     ORDER IS SORTED INDEXED BY DEFINED KEYS
240     DUPLICATES ARE NOT ALLOWED
241     OWNER IS ART-DESCR.
242     MEMBER IS ARTICLE MANDATORY AUTOMATIC
243     ASCENDING KEY IS COL-NO, ART-SIZE
244     SET OCCURRENCE SELECTION IS THRU CURRENT OF SET.
245     *
246     *
247     SET NAME IS                MIN-STOCK-LEVEL
248     ORDER IS SORTED INDEXED BY DEFINED KEYS
249     DUPLICATES ARE NOT ALLOWED
250     OWNER IS SYSTEM.
251     MEMBER IS ARTICLE OPTIONAL MANUAL
252     ASCENDING KEY IS ART-NO, COL-NO, ART-SIZE.
253     *
254     *
255     SET NAME IS                CONTAINING
256     ORDER IS NEXT
257     OWNER IS ARTICLE.
258     MEMBER IS SUBSET MANDATORY AUTOMATIC
259     SET OCCURRENCE SELECTION IS THRU LOCATION MODE OF OWNER.
260     *
261     *
262     SET NAME IS                CONTAINED-IN
263     ORDER IS NEXT
264     OWNER IS ARTICLE.
265     MEMBER IS SUBSET MANDATORY AUTOMATIC
266     SET OCCURRENCE SELECTION IS THRU LOCATION MODE OF OWNER
267         ALIAS FOR  ART-NO    IS  SUBST-ART-NO
268         ALIAS FOR  COL-NO    IS  SUBST-COL-NO
269         ALIAS FOR  ART-SIZE  IS  SUBST-SIZE.
270     *
271     *
272     SET NAME IS                SUPPLIERS
273     ORDER IS SORTED INDEXED BY DEFINED KEYS
274     DUPLICATES ARE NOT ALLOWED
275     OWNER IS SYSTEM.
276     MEMBER IS SUPPLIER MANDATORY AUTOMATIC

```

```

277             ASCENDING KEY IS SUPPL-NAME, SUPPL-NO.
278 *
279 *
280 SET NAME IS                ARTICLES-AVAILABLE
281 ORDER IS SORTED INDEXED BY DEFINED KEYS
282     DUPLICATES ARE ALLOWED
283 OWNER IS SUPPLIER.
284 MEMBER IS ARTICLE MANDATORY AUTOMATIC
285     ASCENDING KEY IS ARTICLE-NAME
286     SEARCH KEY IS NOT-AVAIL-CODE USING INDEX
287     NAME IS SEARCH-TAB-ART-AVAIL DUPLICATES ARE ALLOWED
288     SET OCCURRENCE SELECTION IS THRU CURRENT OF SET.
289 *
290 *
291 SET NAME IS                ORDERED-ARTICLES
292 ORDER IS LAST
293 OWNER IS ARTICLE.
294 MEMBER IS ORD-ITEM MANDATORY AUTOMATIC
295     SET OCCURRENCE SELECTION IS THRU LOCATION MODE OF OWNER.
296 *
297 *
298 SET NAME IS                REORDERED-ARTICLES
299 ORDER IS LAST
300 OWNER IS ARTICLE.
301 MEMBER IS P-ORD-ITEM MANDATORY AUTOMATIC
302     SET OCCURRENCE SELECTION IS THRU LOCATION MODE OF OWNER.
303 *
304 *
305 SET NAME IS                P-ORD-PLACED
306 ORDER IS LAST
307 OWNER IS SUPPLIER.
308 MEMBER IS PURCHASE-ORDER MANDATORY AUTOMATIC
309     SET OCCURRENCE SELECTION IS THRU CURRENT OF SET.
310 *
311 *
312 SET NAME IS                P-ORD-RECEIVED
313 ORDER IS FIRST
314 OWNER IS SUPPLIER.
315 MEMBER IS PURCHASE-ORDER MANDATORY MANUAL
316     SET OCCURRENCE SELECTION IS THRU CURRENT OF SET.
317 *
318 *
319 SET NAME IS                P-ORD-CONTENTS
320 ORDER IS NEXT
321 OWNER IS PURCHASE-ORDER.
322 MEMBER IS P-ORD-ITEM MANDATORY AUTOMATIC
323     SET OCCURRENCE SELECTION IS THRU CURRENT OF SET.
324 *

```



```
325      *
326      *
327      SET NAME IS                                RESULT-SET
328          SET IS DYNAMIC
329          ORDER IS IMMATERIAL
330          OWNER IS SYSTEM.
331      *
332      SET NAME IS                                LIMITED-SET
333          SET IS DYNAMIC
334          ORDER IS IMMATERIAL
335          OWNER IS SYSTEM.
336      *
337      *
338      SET NAME IS                                IQL-DYN1
339          SET IS DYNAMIC
340          ORDER IS IMMATERIAL
341          OWNER IS SYSTEM.
342      *
343      SET NAME IS                                IQL-DYN2
344          SET IS DYNAMIC
345          ORDER IS IMMATERIAL
346          OWNER IS SYSTEM.
347      *
348      SET NAME IS                                IQL-DYN3
349          SET IS DYNAMIC
350          ORDER IS IMMATERIAL
351          OWNER IS SYSTEM.
352      *
353      SET NAME IS                                IQL-DYN4
354          SET IS DYNAMIC
355          ORDER IS IMMATERIAL
356          OWNER IS SYSTEM.
357      *
358      SET NAME IS                                IQL-DYN5
359          SET IS DYNAMIC
360          ORDER IS IMMATERIAL
361          OWNER IS SYSTEM.
362      *
363      SET NAME IS                                IQL-DYN6
364          SET IS DYNAMIC
365          ORDER IS IMMATERIAL
366          OWNER IS SYSTEM.
367      *
368      SET NAME IS                                IQL-DYN7
369          SET IS DYNAMIC
370          ORDER IS IMMATERIAL
371          OWNER IS SYSTEM.
372      *
```

```
373          SET NAME IS          IQL-DYN8
374          SET IS DYNAMIC
375          ORDER IS IMMATERIAL
376          OWNER IS SYSTEM.
```

4.13 Reserved words of the DDL compiler

ACCEPT	ACCESS	ACTUAL
AD	ADD	ADVANCING
AFTER	ALIAS	ALL
ALLOWED	ALPHABETIC	ALPHANUMERIC
ALSO	ALTER	ALTERNATE
AN	AND	ANY
APPLY	ARE	AREA
AREA-ID	AREAS	ASC
ASCENDING	ASSIGN	ASSIGNED
AT	ATTACHED	AUTHOR
AUTO	AUTOMATIC	BEFORE
BEGINNING	BETWEEN	BIN
BINARY	BLANK	BLOCK
BLOCK-DENSITY	BOTTOM	BY
C01	C02	C03
C04	C05	C06
C07	C08	C09
C10	C11	C12
CALC	CALL	CANCEL
CARD-PUNCH	CARD-READER	CBL-CTR
CD	CH	CHAIN
CHANGED	CHAR	CHARACTER
CHARACTERS	CHECK	CHECKING
CHECKPOINT	CLASS	CLOCK-UNITS
CLOSE	COBOL	CODE
CODE-SET	COLLATING	COLUMN
COMMA	COMMUNICATION	COMP
COMP-1	COMP-2	COMP-3
COMPILE	COMPRESSION	COMPUTATIONAL
COMPUTATIONAL-1	COMPUTATIONAL-2	COMPUTATIONAL-3

COMPUTE	CONFIGURATION	CONNECT
CONSOLE	CONTAINS	CONTROL
CONTROLS	COPY	CORR
CORRESPONDING	COUNT	CREATING
CSP	CURRENCY	CURRENT
CURRENT-DATE	CYCLES	CYLINDER-OFLO
DATA	DATABASE-EXCEPTION	DATABASE-KEY
DATABASE-KEY-LIST	DATABASE-KEY-LONG	DATABASE-KEY-NAME
DATABASE-KEY-RANGE	DATABASE-KEY-TRANSLATION-TABLE	DATABASE-PRIVACY-KEY
DATABASE-REALM-NAME	DATABASE-RECORD-NAME	DATABASE-SET-NAME
DATABASE-STATUS	DATE	DATE-COMPILED
DATE-WRITTEN	DAY	DB
DBKEY	DBKEY-LONG	DBKEY-TRANSLATION-TABLE
DBTT	DCB-NAME	DE
DEBUGGING	DEC	DECIMAL
DECIMAL-POINT	DECLARATIVES	DEFINED
DELETE	DELIMITED	DELIMETER
DEPENDING	DESC	DESCENDING
DESTINATION	DETACHED	DETAIL
DIGITS	DIRECT	DIRECT-LONG
DISABLE	DISC	DISC64
DISC80	DISC90	DISCONNECT
DISPLAY	DISPLAY-ST	DIVIDE
DIVISION	DOWN	DUP
DUPLICATE	DUPLICATES	DYNAMIC
EGI	ELSE	EMI
EMPTY	ENABLE	END
END-OF-PAGE	ENDING	ENTER
ENTRY	ENVIRONMENT	EOP
EQUAL	ERASE	ERROR
ESI	EVERY	EXAMINE

EXCEPTION	EXCL	EXCLUSIVE
EXHIBIT	EXIT	EXTEND
EXTENDED	FD	FETCH
FILE	FILE-CONTROL	FILE-LIMIT
FILE-LIMITS	FILES	FILLER
FINAL	FIND	FINISH
FIRST	FIXED	FOOTING
FOR	FORM-OVERFLOW	FREE
FROM	GENERATE	GET
GIVING	GO	GREATER
GREATEST	GROUP	GROUP-USAGE
HEADING	HIGH-VALUE	HIGH-VALUES
HOLD	I-O	I-O-CONTROL
ID	IDENTIFICATION	IF
IMMATERIAL	IN	INCLUDING
INCREASE	INDEX	INDEXED
INDICATE	INDICATOR	INITIAL
INITIATE	INPUT	INPUT-OUTPUT
INSPECT	INSTALLATION	INTO
INVALID	IS	ITEMS
JUST	JUSTIFIED	KEEP
KEY	KEYS	LABEL
LABELS	LAST	LEADING
LEFT	LENGTH	LESS
LIBRARY	LIMIT	LIMITED
LIMITS	LINAGE	LINE
LINE-COUNTER	LINES	LINK
LINKAGE	LINKED	LIST
LOC	LOCATION	LOCK
LOCKS	LOG	LOW-VALUE
LOW-VALUES	MAND	MANDATORY
MANUAL	MASK	MEMBER

MEMBERS	MEMBERSHIP	MEMORY
MERGE	MESSAGE	MINUS
MIXED	MODE	MODIFY
MODULES	MORE-LABELS	MOVE
MULTIPLE	MULTIPLY	NAME
NAMED	NATIONAL	NATIVE
NEGATIVE	NEXT	NO
NOT	NOTE	NUMBER
NUMERIC	OBJECT-COMPUTER	OCCURRENCE
OCCURS	OF	OFF
OH	OMITTED	ON
ONES	ONLY	OPEN
OPT	OPTIMIZATION	OPTIONAL
OR	ORDER	ORGANIZATION
OTHER	OTHERWISE	OUTPUT
OV	OVERFLOW	OWNER
PA	PAGE	PAGE-COUNTER
PAGES	PERCENT	PERFORM
PERMANENT	PF	PH
PHYSICAL	PHYSICALLY	PIC
PICTURE	PLACEMENT	PLACING
PLUS	POINTER	POINTER-ARRAY
POPULATION	POSITION	POSITIONING
POSITIVE	PRESELECTION	PRINT-SWITCH
PRINTER	PRINTING	PRIOR
PRIVACY	PRIVACY-GROUP	PRIVACY-IDENTIFICATION
PRIVACY-NAME	PRIVACY-RECORD	PROCEDURE
PROCEDURES	PROCEED	PROCESS
PROCESSING	PROGRAM	PROGRAM-ID
PROT	PROTECTED	PROTECTION
PUNCH4	PUNCH6	QUEUE
QUOTE	QUOTES	RANDOM

RD	READ	READER
READY	REAL	REALM
REALM-NAME	REALMS	RECEIVE
RECORD	RECORDING	RECORDS
REDEFINES	REEL	REFERENCE-YEAR
REFERENCES	RELATIVE	RELEASE
REMAINDER	REMARKS	REMOVAL
RENAMES	REORGANIZATION	REPEATED-KEY
REPLACING	REPORT	REPORTING
REPORTS	RERUN	RESERVE
RESET	RESTRICTED	RESULT
RETAINING	RETR	RETRIEVAL
RETURN	REVERSED	REWIND
REWRITE	RF	RH
RIGHT	ROUNDED	RUN
SA	SAME	SCHEMA
SD	SEARCH	SECTION
SECURITY	SEEK	SEGMENT
SEGMENT-LIMIT	SELECT	SELECTION
SELECTIVE	SEND	SENTENCE
SEPARATE	SEQUENCE	SEQUENTIAL
SET	SETS	SIEMENS-4004
SIGN	SIGNED	SIZE
SMALLEST	SORT	SORT-MERGE
SORT-TAPE	SORT-TAPES	SORTED
SOURCE	SOURCE-COMPUTER	SPACE
SPACES	SPANS	SPECIAL-NAMES
STANDARD	STANDARD-1	START
STATUS	STOP	STORAGE
STORE	STRING	STRUCTURE
SUB-QUEUE-1	SUB-QUEUE-2	SUB-QUEUE-3
SUB-SCHEMA	SUBTRACT	SUM

SUPPRESS	SYMBOLIC	SYNC
SYNCHRONIZED	SYSIN	SYSIPT
SYSLST	SYSOPT	SYSOPT-234
SYSOUT	SYSPUNCH	SYSRDR
SYSTEM	TABLE	TALLY
TALLYING	TAPE	TAPES
TEMP	TEMPORARY	TENANT
TERMINAL	TERMINATE	TEXT
THAN	THEN	THROUGH
THRU	TIME	TIMES
TO	TODAYS-DATE	TOP
TRACE	TRACK-AREA	TRACKS
TRAILING	TRANSFORM	TRY
TYPE	UNDER	UNIT
UNITS	UNSTRING	UNTIL
UP	UPDATE	UPON
USAGE	USAGE-MODE	USE
USING	VALUE	VALUES
VARYING	VIA	WHEN
WITH	WITHIN	WITHOUT
WORDS	WORKING-STORAGE	WRITE
WRITE-ONLY	YEAR	ZERO
ZEROES	ZEROS	

5 SSL

5.1 Introduction

The schema DDL is used to describe the logical structure of the data; the Storage Structure Language SSL is used to specify the physical storage structure when storing data and logical relationships between data. The definition of the physical storage structure is optional. If it is omitted, UDS/SQL applies the default values. The user can influence the storage space requirements for the data and especially the system run-time behavior in future applications when defining the storage structure.

Designing the storage structure involves the following steps:

1. Declaring the population
2. Defining the linkage of records
3. Determining the placement of member records, tables and hash areas
4. Determining the extent of table reorganization

Each of these steps is dealt with below in a separate section, which also explains the default cases. These sections are preceded by a description of the methods UDS/SQL uses for the physical representation of the logical data structure.

The metalanguage used is described on [page 22](#), and the general syntax rules are summarized on [page 234](#).

You compile the SSL using the SSL compiler. For information on operating the SSL compiler, please refer to the "[Creation and Restructuring](#)" manual, Compiling SSL).

5.1.1 Methods of physical representation of the logical data structure

The physical representation of the entirety of

- all records of a record type:

DBTT
(Database Key Translation Table)

Table representing all records of a record type and linking all owner records to the tables of their set occurrences.

Record SEARCH key table

Table representing all records of a record type.

- all records of a set occurrence:

Pointer array

Sort key table

Set SEARCH key table

List

Chain

Tables representing all member records of a set occurrence

Table containing all member records of a set occurrence

Contains owner record and all member records of a set occurrence

Using the SSL, the user determines if and how a pointer array, list, chain, sort key table or SEARCH key table is to be set up. A description of these elements can be found on [page 144](#).

UDS/SQL automatically sets up a DBTT without user intervention. A description of the DBTT follows.

5.1.2 DBTT (Database Key Translation Table)

The DBTT establishes the link between the database key value of a database record and the physical page address of that record.

Structure of a physical page address

The total space available for storing data in the database is divided into realms. A realm consists of a number of pages defined at database creation (see the “[Creation and Restructuring](#)” manual, “Database creation”). The pages in each realm are consecutively numbered, as are the realms. UDS/SQL can thus locate data in the database if the number of the page containing the data and the number of the realm containing the page are known. The realm reference and page number are therefore combined to form a physical page address, which UDS/SQL can use as a pointer to the physical position of the data.

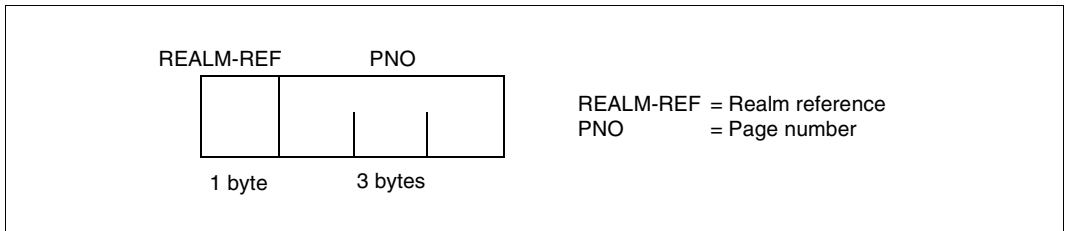


Figure 27: Structure of a physical page address

For the main part, data in the database comprises the records and tables defined by the user. The physical placement of these records and tables can change, but UDS/SQL does not update all the associated pointers when this happens.

Pointers that must be current at all times are referred to as actual keys (act-key), the remaining pointers are called Probable Position Pointers (PPP).

PPPs can be updated to the most recent update level by means of the BREORG utility routine (see the “[Recovery, Information and Reorganization](#)” manual, BREORG).

Structure of a database key value

To find the physical address of records and associated tables, UDS/SQL can always make use of an additional identifier, the database key value, which never changes during the life of a record in the database.

A record type consists of a number of records which are consecutively numbered. The record types are also numbered. The record sequence number (RSQ) and record type reference (REC-REF) are combined to form the database key value of the record. The database key is thus an identifier that uniquely identifies each record contained in the database.

The structure of the database key value varies, depending on whether the database page length is 2048 bytes or 4000/8096 bytes.

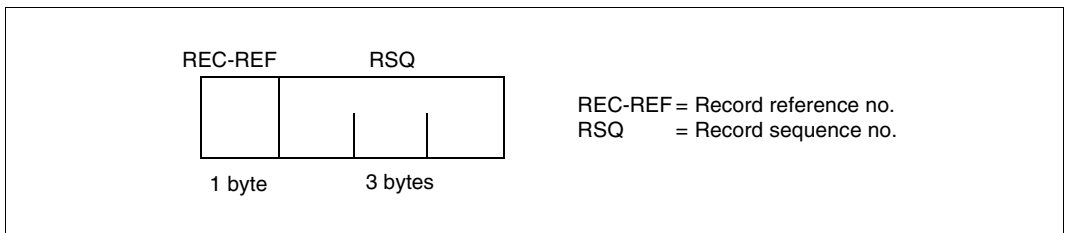


Figure 28: Structure of a database key value if the page length is 2048 bytes

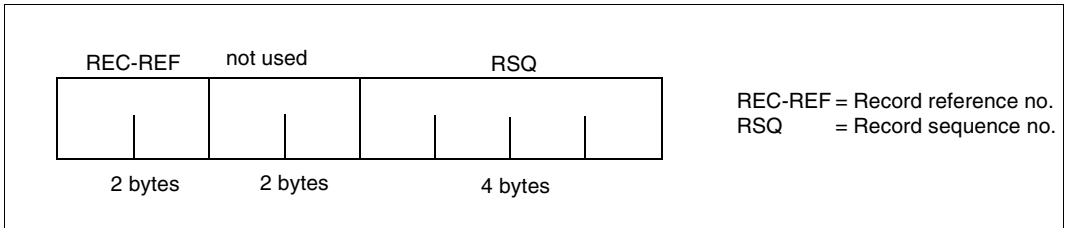


Figure 29: Structure of a database key value if the page length is 4000 or 8096 bytes

The following applies to the value range for a record reference number (REC-REF):

- $1 \leq \text{REC-REF} \leq 254$ for databases with a 2048-byte page length
- $1 \leq \text{REC-REF} \leq 2^{15}-1$ for databases with a 4000/8096-byte page length

The following applies to the value range for a record sequence number (RSQ):

- $1 \leq \text{RSQ} \leq 2^{24} - 1$ for databases with a 2048-byte page length if the record type is not an owner in any set.
- $1 \leq \text{RSQ} \leq 2^{24} - 2^{16} - 1$ for databases with a 2048-byte if the record type is an owner in a set.
- $1 \leq \text{RSQ} \leq 2^{31} - 1$ for databases with a 4000/8096-byte page length

Structure of the DBTT

In order to locate the physical placement of a record via the database key value, UDS/SQL uses the database key translation table or DBTT.

UDS/SQL automatically sets up a DBTT for every record type. The DBTT contains exactly one line for every record of a record type. A DBTT line is divided into columns. Column 0 contains the page address of the record. The DBTT of a record type which is not an owner consists of column 0 only.

The database key value is contained only implicitly in the DBTT. It is symbolized by the place where the associated physical page address is entered. For example, the record with RSQ=3 is stored in line 3 of the DBTT; the record with RSQ=9 is stored in line 9, etc.

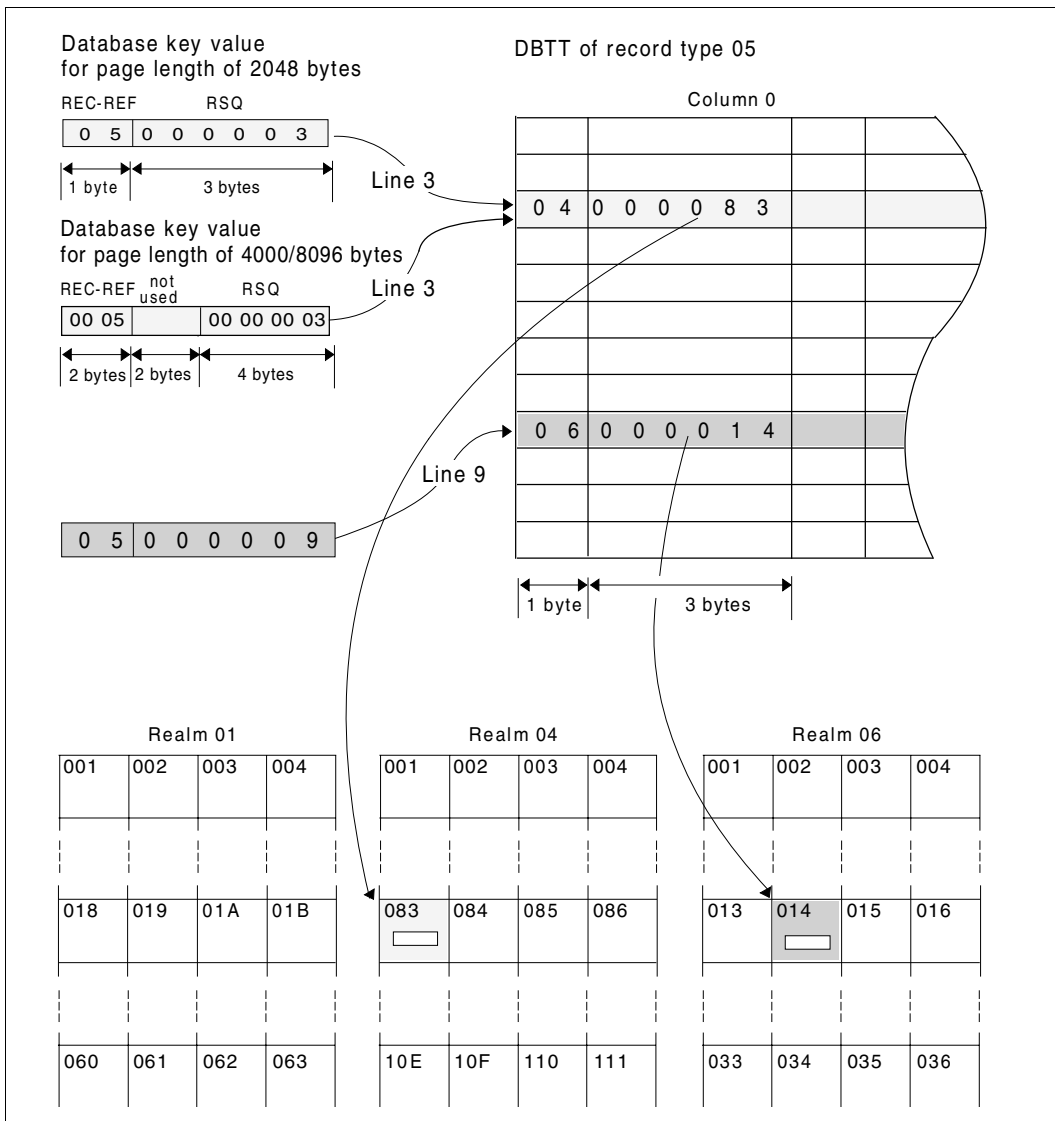


Figure 30: Linkage between the database key value and the record address via the DBTT

If the placement of records changes, updating of the page addresses can be limited to the DBTT.

In addition to the page addresses of the owner records, the DBTT of an owner record type contains the page addresses of all tables pertaining to the set occurrences of the owner records. To identify these tables, the database key value of the corresponding owner record is combined with the DBTT column number, forming a unique identifier which never changes.

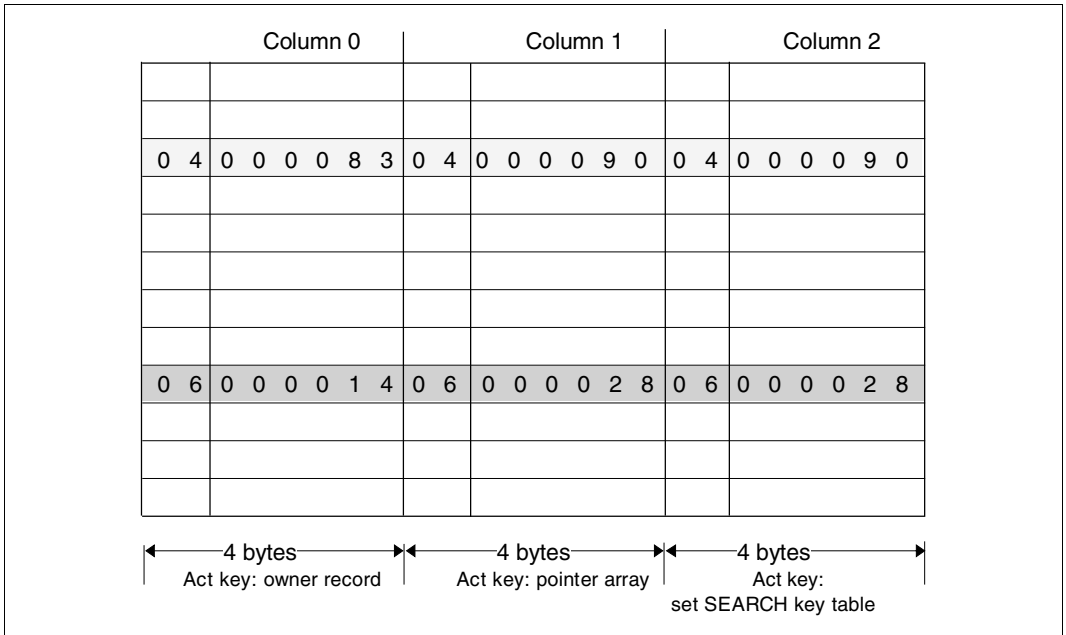


Figure 31: Complete example of a DBTT

General conclusions:

- Page address updating for records and tables can be limited to the DBTT.
- All page addresses listed in the DBTT are current.

5.2 Declaring the population

5.2.1 Specifying the number of records in one record type

The number of records included in one record type is defined in the DBTT and in the record POPULATION clause. Using this number UDS/SQL computes:

- the storage space requirement for the DBTT,
- the hash area size for the primary key,
- the hash area sizes for secondary keys.

A record SEARCH key table is always initially set up in a single database page and can be subsequently extended page by page as required for additional records.

Storage space requirements for the DBTT

```
DATABASE-KEY-TRANSLATION-TABLE IS integer [WITHIN realm-name]
```

For each record, there must be one line available in the DBTT of the associated record type. *integer* specifies the number of expected records of the record type, and thus the number of DBTT lines.

- If the database has a page length of 2048 bytes, you may specify a maximum value of 16 777 215 ($=2^{24}-1$) for *integer*. If the record type in question is the owner in a set, the maximum possible value for *integer* is 16 711 679.
- If the database has a page length of 4000 or 8096 bytes, the maximum value that can be specified for *integer* is $2^{31}-1$.

If the record type is not “owner”, the DBTT consists of exactly one column. Otherwise, one column is added for each reference to

- a pointer array (MODE IS POINTER-ARRAY),
- a list (MODE IS LIST),
- a sort key table (MODE IS CHAIN, ORDER IS SORTED INDEXED), or
- a set SEARCH key table.

UDS/SQL calculates the storage space requirement of the DBTT from the number of lines and columns, and reserves the required number of pages. Since only whole pages or DBTT extents can be reserved for a DBTT, the table can accommodate more lines than specified by *integer*.

Note, however, that UDS/SQL always creates only as many DBTT pages as required for the DBTT and that, depending on the database page length, a maximum of $2^{24}-1$ or $2^{24} - 2^{16} - 1$ (=16711679) or $2^{31}-1$ entries per DBTT can be administered. Since a DBTT base can only be created in whole pages and a DBTT extent only in DBTT extent size (128 PAM pages), an excess amount is usually created if the maximum values are specified. This excess amount is not used.

The DBTT can be extended by means of the BREORG utility routine (see the "[Recovery, Information and Reorganization](#)" manual, BREORG).

By default, UDS/SQL reserves one page for a DBTT.

If a secondary key has been defined for a record type for conversion by a hash routine, the size specification for the DBTT also affects the initial size of the hash area (see the section "[Size of a hash area for the primary key](#)" on page 138).

Size of a hash area for the primary key

`POPULATION IS {integer WITHIN realm-name},...`

The hash area for the primary key of a record type is distributed over several realms if the records of the record type are allocated to several realms by means of the schema DDL (see the section [“Defining allocation of records to realms” on page 110](#)).

integer specifies the number of records to be stored in the realm referenced by *realm-name*. Using this number, UDS/SQL calculates the minimum number of pages required for storing the records (see the section [“Direct CALC page” on page 210](#)) or the record addresses (see the section [“Indirect CALC page” on page 213](#)) in the respective realm. In each realm, UDS/SQL then reserves a number of pages corresponding to the next higher prime number.

A hash area is extended by means of overflow pages if the data assigned to a page by a hash routine exceeds the maximum capacity of the page. The overflowing page is linked to the overflow page by means of act-keys.

The efficiency of a hash routine can be judged by how evenly it distributes data over the hash area, i.e. how many overflow pages are generated.

A hash area can be subsequently extended by means of the BREORG utility routine (see the [“Recovery, Information and Reorganization”](#) manual, BREORG).

By default, UDS/SQL sets up a hash area of one page.

Example

DDL:

```
RECORD NAME IS ARTICLE
  LOCATION MODE IS CALC
  WITHIN CLOTHING, HOUSEHOLD-GOODS, ..... AREA-ID .....
  .
  .
  .
```

SSL:

```
RECORD NAME IS ARTICLE
  POPULATION IS 400 WITHIN CLOTHING, 100 WITHIN HOUSEHOLD-GOODS, .....
```

Size of the hash area for a secondary key

`DATABASE-KEY-TRANSLATION-TABLE` IS *integer*

integer specifies the number of records of a record type.

- If the database has a page length of 2048 bytes, you may specify a maximum value of $2^{24}-1$ for *integer*. If the record type in question is the owner in a set, the maximum possible value for *integer* is 16 711 679.
- If the database has a page length of 4000 or 8096 bytes, the maximum value that can be specified for *integer* is $2^{31}-1$.

Using this number, UDS/SQL not only calculates the size of the DBTT, but also the minimum number of pages required for the distributed storage of the record addresses (see the section "[Indirect CALC page](#)" on page 213).

UDS/SQL then reserves a number of pages corresponding to the next higher prime number.

A hash area is extended by means of overflow pages if the data assigned to a page by a hash routine exceeds the maximum capacity of the page. The overflowing page is linked to the overflow page by means of act-keys.

A hash area can be extended at a later time by means of the BREORG utility routine (see the "[Recovery, Information and Reorganization](#)" manual, BREORG).

By default, UDS/SQL sets up a hash area of one page for each secondary key.

5.2.2 Specifying the size of the set occurrences of a set

`POPULATION IS integer`

The size of the set occurrences must be specified in the following cases:

- if you want to store tables and member records in the proximity of the associated owner records,
- if you want to facilitate subsequent table extensions,
- if you want to specify the size of a hash area for secondary keys in a SYSTEM set in order to avoid overflow pages.

integer specifies an average set occurrence population. It should be the value which applies to the majority of the set occurrences.

Based on this specification, UDS/SQL calculates:

- the storage space requirements for pointer arrays, lists, sort key tables and set SEARCH key tables, provided the set is not a SYSTEM set,
- the storage space requirements for the member records of the set occurrences, provided the set is not a SYSTEM set,
- the size of hash areas for secondary keys in a SYSTEM set.

UDS/SQL reserves the storage space calculated for a set occurrence in the next free space following the owner record when storing the owner record.

The default value for *integer* is 0. In this case, no space is reserved for tables and member records, and one page is reserved for each hash area.

Storage space requirements for pointer array, list, sort key table and set SEARCH key table

Each of the above tables represents or contains all member records of a set occurrence. Each member record corresponds to one line in the associated tables. This means that *integer* specifies the number of lines in each table of the set occurrence.

Based on the number and the length of lines, UDS/SQL calculates the storage space requirements for the tables.

When storing an owner record, UDS/SQL reserves:

- the calculated storage space, if it is smaller than one page,
- one page, if the calculated storage space is greater than one page.

for each table belonging to the associated set occurrence.

In the case of SYSTEM sets, UDS/SQL reserves exactly one database page for each table, regardless of the POPULATION clause.

Table extensions

UDS/SQL automatically extends pointer arrays, lists, sort key tables and set SEARCH key tables if the need arises when member records are stored. In order to avoid frequent table extensions, the user can control table size using the following specification in the set POPULATION clause:

POPULATION IS *integer-1* INCREASE IS *integer-2*

integer-2 specifies the number of table lines to be added with the first two table extensions. Following the second extension, $integer-2 * 2^n$ ($n \hat{=}$ number of extensions) table lines are added until there is no more space for a further table line. If data other than that belonging to the table is contained in the page, preventing the table from being extended, a subsequent table extension includes the relocation of the table. In this case, UDS/SQL makes available a page large enough to accommodate the table and its desired extension.

For table extensions beyond the first table page, *integer-2* no longer has any effect (see the section [“Defining the extent of table reorganization desired” on page 172](#)).

The default value for *integer-2* is 1.

Storage space requirements for member records

This only applies to member records which are not included in a list. When the owner is stored, UDS/SQL reserves space for its member records if the PLACEMENT OPTIMIZATION clause is defined for the member record type (see the section [“PLACEMENT OPTIMIZATION” on page 165](#)). In this case, UDS/SQL uses the entry in the set occurrence population clause to calculate the storage space requirement for the member records.

Size of the hash area for a secondary key

Only in the case of SYSTEM sets can a secondary key be used for conversion by a hash routine. UDS/SQL calculates the minimum number of pages required for distributed storage of record addresses on the basis of the set occurrence population entry (see the section ["Indirect CALC page" on page 213](#)) and then reserves the number of pages corresponding to the next higher prime number.

If some pages overflow due to uneven data distribution by the hash routine, UDS/SQL automatically sets up overflow pages.

A hash area can be extended at a later time by means of the BREORG utility routine (see the ["Recovery, Information and Reorganization"](#) manual, BREORG).

By default, UDS/SQL sets up a hash area with a size of one page.

5.2.3 Overview of the initial sizes for storage space reservations

Type of data	Set type	Reserved storage space
Records with PLACEMENT OPTIMIZATION	not	According to <i>integer</i> in the set POPULATION clause
List Pointer array Sort key table	SYSTEM	According to <i>integer</i> in the set POPULATION clause; maximum one page
Set SEARCH key table	SYSTEM	One page
Record SEARCH key table	-	
Hash area for set SEARCH key	SYSTEM	According to <i>integer</i> in the set POPULATION clause; minimum one page
Hash area for record SEARCH key	-	According to <i>integer</i> in the DBTT clause; minimum one page
Hash area for primary key		According to <i>integer</i> in the record POPULATION clause; minimum one page
DBTT		According to <i>integer</i> in the DBTT clause; minimum one page

Table 7: Initial sizes for storage space reservations

5.3 Determining the linkage of records

5.3.1 Determining the storage mode for set occurrences

UDS/SQL offers three different storage modes for linking member records to form a set occurrence. These are:

- pointer array (POINTER-ARRAY)
- list (LIST), and
- chain (CHAIN)

The following example is used in this section to explain the three storage modes:

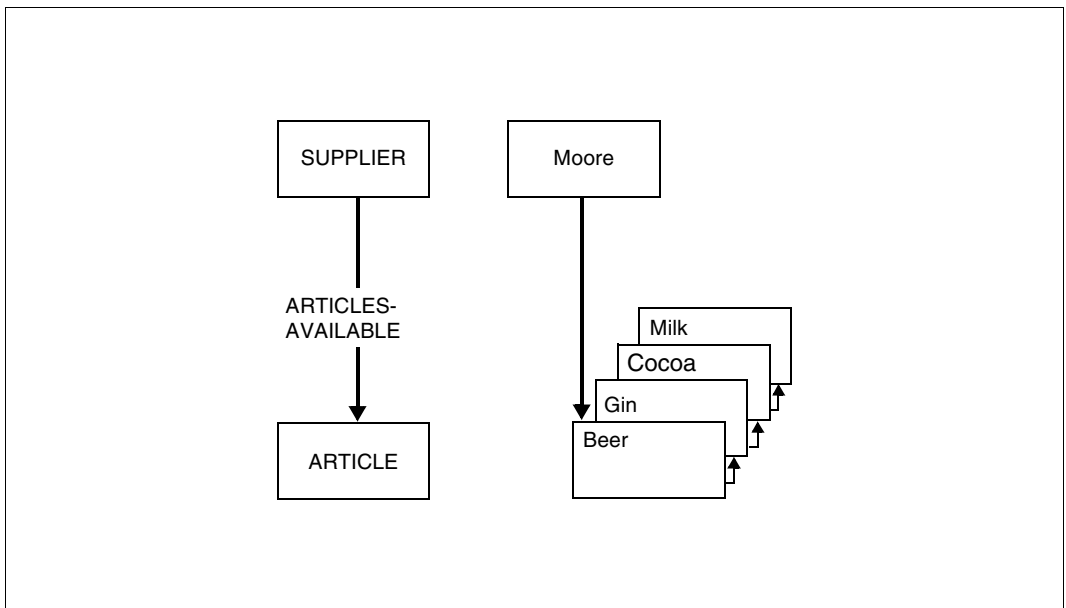


Figure 32: Example of storage modes for set occurrences

Storing a set occurrence as a pointer array

MODE IS POINTER-ARRAY

If `MODE IS POINTER-ARRAY` is defined for a set, UDS/SQL links the member records of its set occurrences via a table called a pointer array.

This table contains pointers to each member record of a set occurrence. A pointer consists of the RSQ of the member record and the PPP of the page containing the member record (see [page 224](#), Table line).

The further contents of the pointer array depend on which `ORDER` clause was defined for the set in the schema DDL.

- `ORDER IS LAST/FIRST/NEXT/PRIOR/IMMATERIAL or SORTED INDEXED BY DATABASE-KEY`

The pointer array contains only the pointers to the member records of the set occurrence.

- `ORDER IS SORTED INDEXED BY DEFINED KEYS`

In addition to the pointers to the member records of the set occurrence, the pointer array contains the associated values of the primary key.

- `ORDER IS SORTED`

This entry is not allowed for a pointer array.

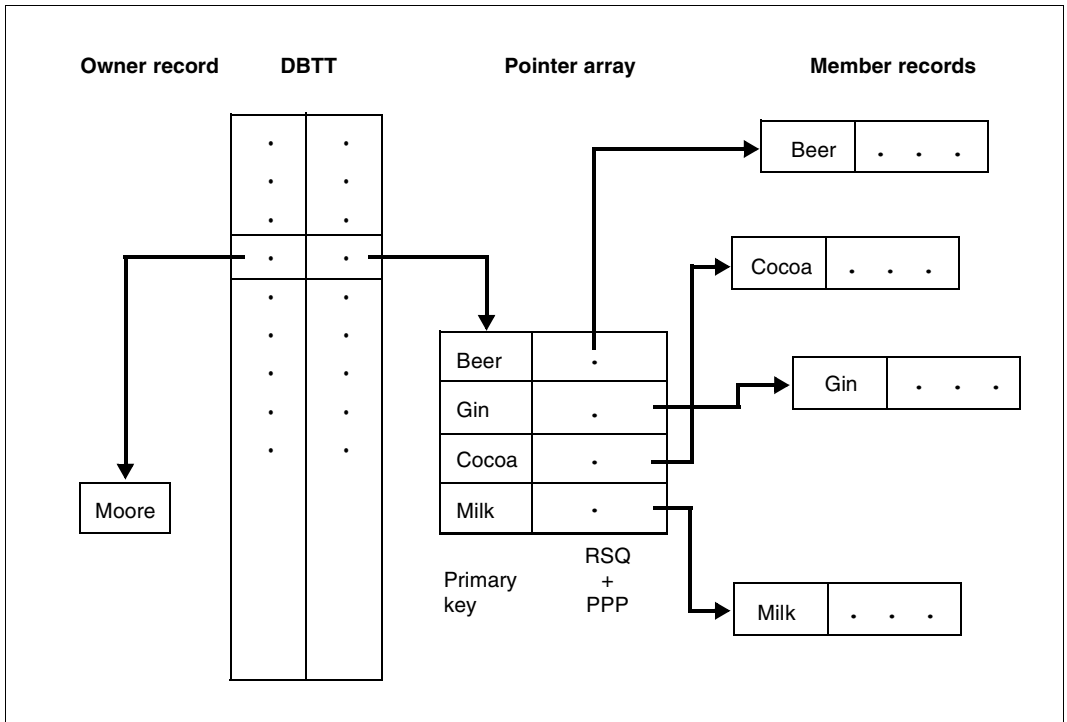


Figure 33: Set occurrence stored as a pointer array

If a pointer array occupies more than one page, each page is connected by act-keys twice. If **ORDER IS SORTED INDEXED**, the pointer array is provided with additional higher-ranking table levels. Each higher-ranking level has the same structure as the lowest level, but contains only the last table entry of the pages containing the next lower level. The PPPs pointing to the member records are replaced by the act-key of the page where the table entry originates (see [page 228](#), [figure 59](#)).

Additional pointer from owner to its pointer array

`MODE IS POINTER-ARRAY.....WITH PHYSICAL LINK`

The standard UDS/SQL link between an owner record and the pointer array of its member records is via the DBTT. The user can bypass the DBTT and save one page access by specifying `WITH PHYSICAL LINK`, which generates a pointer to the pointer array in the owner record.

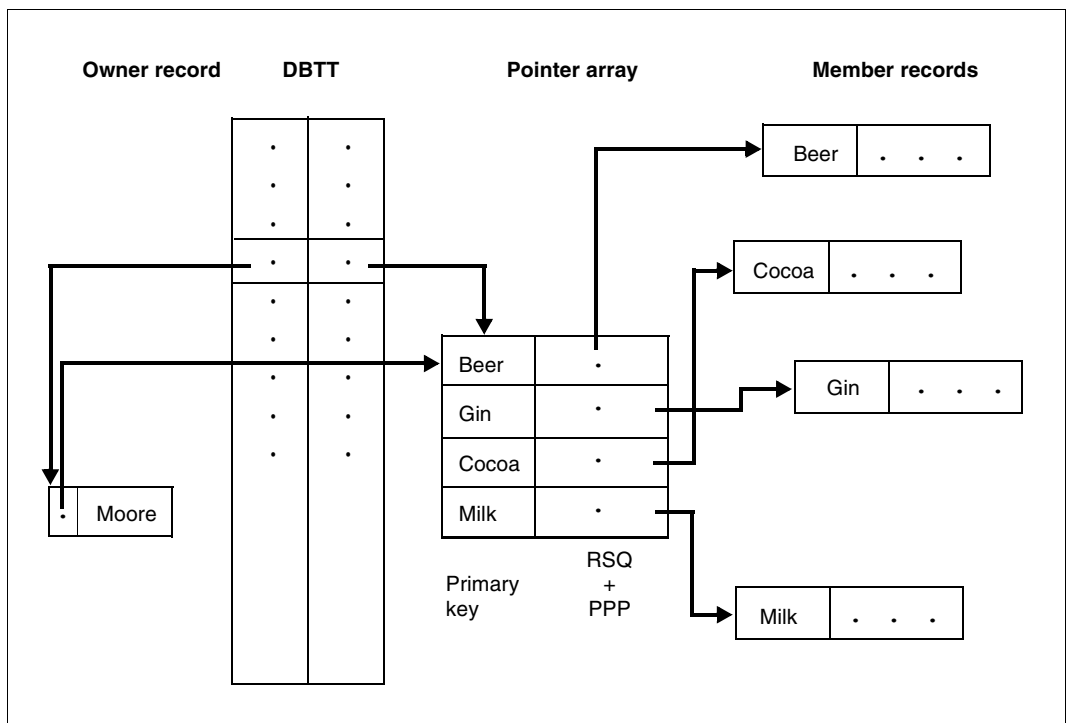


Figure 34: Additional pointer from owner → pointer array

This entry is not permitted in `SYSTEM` sets, where both the function of the owner record and of the DBTT is assumed by an anchor record.

Storing a set occurrence as a list

MODE IS LIST

If a set is defined with `MODE IS LIST`, UDS/SQL stores the member records of a set occurrence in a table called a list. The physical sequence of the records corresponds to the logical sequence defined in the `ORDER` clause.

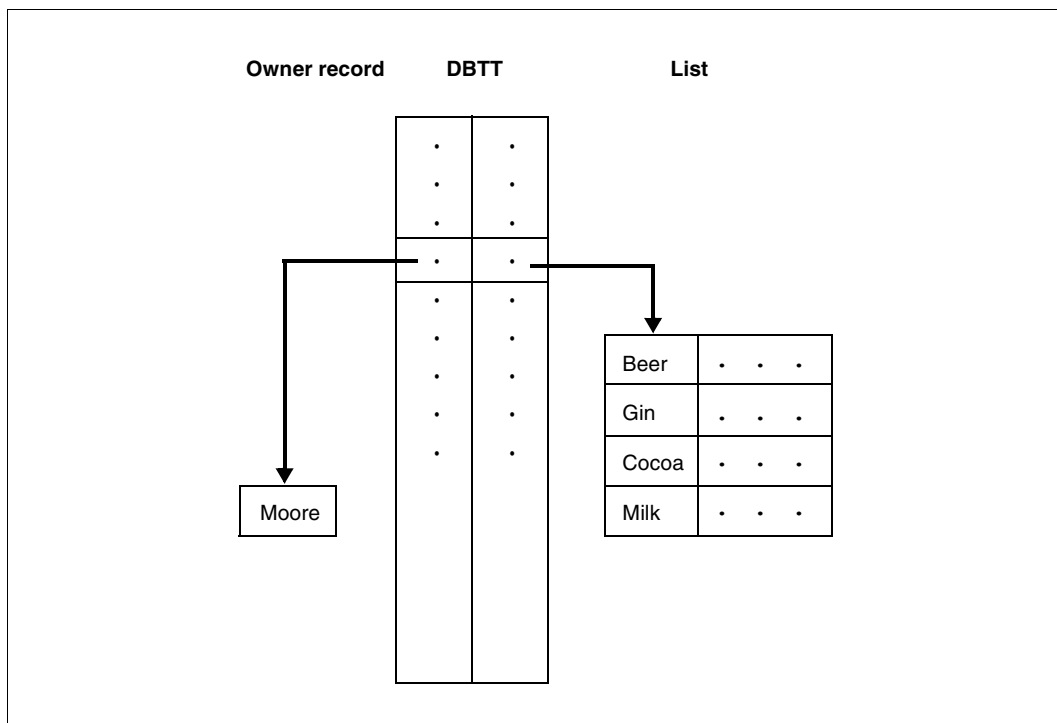


Figure 35: Set occurrence stored as a list

If a list occupies more than one page, each page is linked twice by act-keys.

If `ORDER IS SORTED INDEXED`, the list is provided with additional higher-ranking table levels. The higher-ranking levels have the same structure as that of a pointer array.

The physical placement of records can be defined only once:

- If a record type is a member of several sets, only one of the sets can be defined with `MODE IS LIST`.
- It is not possible to define records with `MODE IS LIST` and `PLACEMENT OPTIMIZATION` at the same time (see the section [“PLACEMENT OPTIMIZATION” on page 165](#)).
- If the records of a record type are placed both by `LOCATION MODE IS CALC` and `MODE IS LIST`, they are stored in lists, and the hash area consists of indirect `CALC` pages.

The following restrictions also apply:

`MODE IS LIST` may only be specified if the set membership has been defined as `MANDATORY AUTOMATIC` in the schema DDL (see the section [“Defining the type of membership of records in a set” on page 79](#)).

`MODE IS LIST` may not be specified in the following cases:

- if the record lengths (including pointers; see [page 220](#), SCD) exceed
 - 993 bytes in a database with a 2048-byte page length (2-Kbyte page format)
 - 1963 bytes in a database with a 4000-byte page length (4-Kbyte page format)
 - 4011 bytes in a database with a 8096-byte page length (8-Kbyte page format)(At least two records must fit in a list, but this would not be possible in the above cases.)
- if a member record type contains an item of variable length;
- if the order of the records within the set occurrence is defined with `ORDER IS SORTED` (without `INDEXED`) or
- if member records are to be stored in compressed form (see the section [“Storing the records of a record type in compressed form” on page 176](#)).

Additional pointer from owner to its list

`MODE IS LIST.....WITH PHYSICAL LINK`

The UDS/SQL standard connection between an owner record and the list of its member records is via the DBTT. The user can bypass the DBTT and save one page access by specifying WITH PHYSICAL LINK, which generates a pointer to the list in the owner record.

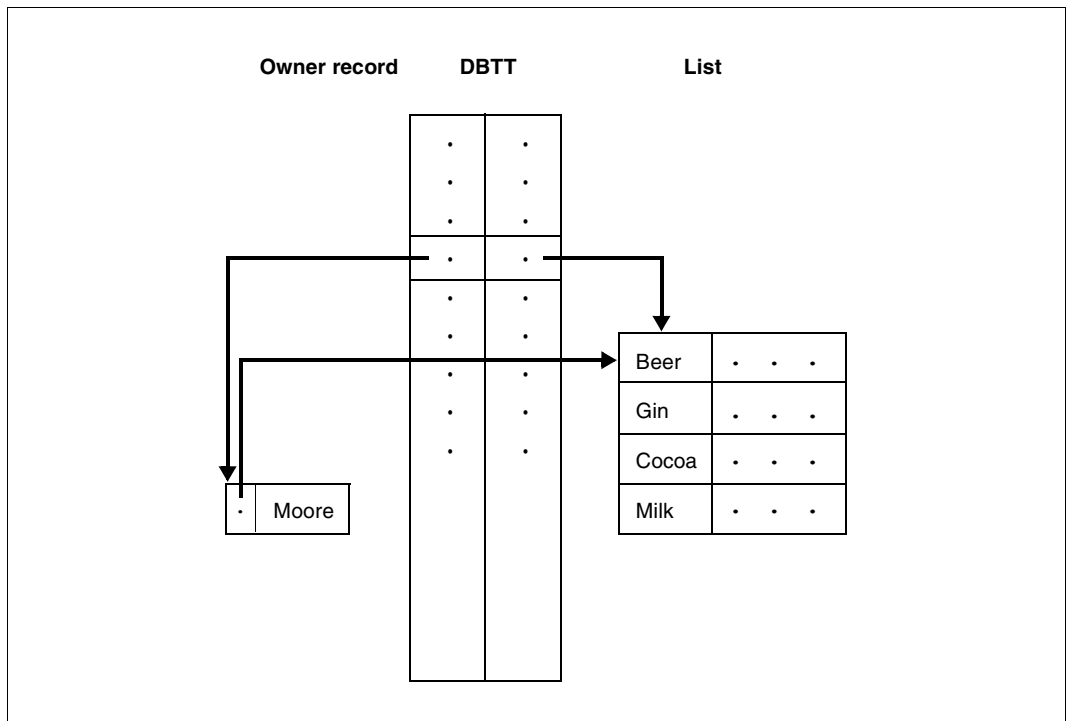


Figure 36: Additional pointer from owner → list

This entry is ignored by the compiler in SYSTEM sets, where both the function of the owner record and the DBTT are assumed by an anchor record. The user is informed by a warning.

Storing a set occurrence as a chain

MODE IS CHAIN

If `MODE IS CHAIN` is defined for a set, each set occurrence of the set is stored as a closed chain of records. The chain comprises the owner record and all member records of the set occurrence. The member records are chained by means of pointers in the order defined in the `ORDER` clause for the set in the schema DDL. The position of the owner record is between the last and the first member record.

The pointer is composed of the database key value of the subsequent record in the chain and the PPP of the page containing this record.

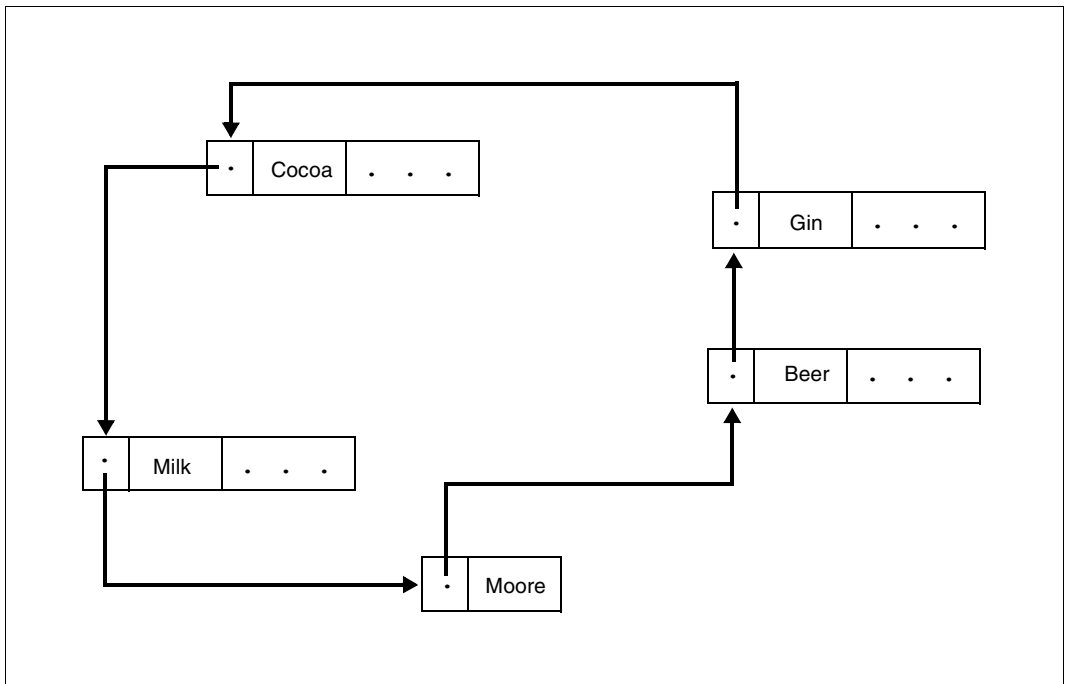


Figure 37: Set occurrence stored as a chain

The chain is the only storage mode applicable if the set has been defined with `ORDER IS SORTED` (without `INDEXED`) in the schema.

If `ORDER IS SORTED INDEXED`, a table is set up as an additional path for direct access. This table is referred to as the sort key table. Its structure is the same as that of a multi-level pointer array.

Additional backward chaining for chain

MODE IS CHAIN LINKED TO PRIOR

In addition to standard forward chaining, the records of a chain can be concatenated in reverse order. If LINKED TO PRIOR is specified, a further pointer is added to each record pointing to the logically preceding record. In the same way as forward chaining, the pointer is composed of the database key value of the preceding record and the PPP of the page containing this record.

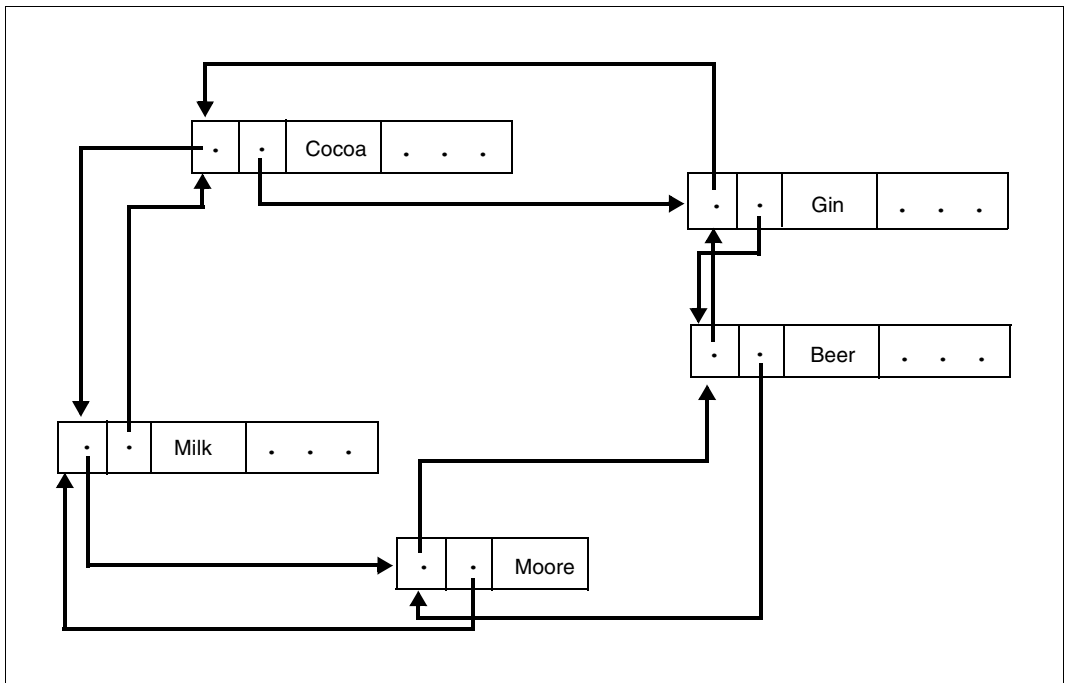


Figure 38: Additional backward chaining for chain

Backward chaining is important for large set occurrences where prior records are frequently the object of a search.

5.3.2 Assessing pointer array, list and chain

The time required for the execution of a program depends on the storage modes defined for the set occurrences.

This section compares retrieval and updating operations when using pointer arrays, lists and chains. The retrieval operations are subdivided into sequential and direct accesses. Updating operations dealt with are insertion and deletion.

Pointer arrays

- Sequential access

The pointer array must be present in memory. This is almost always the case whenever a member record of the set occurrence has already been accessed via the set. One entry in the pointer array must then be read before the associated record can be read. The number of disk accesses depends on how records are distributed over the pages and can be reduced by optimizing the placement of the records (see the sections [“Natural optimization” on page 164](#) and [“PLACEMENT OPTIMIZATION” on page 165](#)).

- Direct access

- ORDER IS SORTED INDEXED: All levels of the pointer array must be in memory. The higher levels are used to select the number of the page containing the entry of the record to be found. One disk access is required for each level of the table.
- ORDER IS LAST/FIRST/NEXT/PRIOR: UDS/SQL must search the set occurrence sequentially until the desired record is found. This may require a considerable number of disk accesses.

- Insertion

In order to insert an entry, the place of insertion in the pointer array must be determined. If ORDER IS SORTED INDEXED, this is possible via direct access. If ORDER IS LAST/FIRST, UDS/SQL can directly access the last or first table page respectively. If ORDER IS NEXT/PRIOR, the place of insertion is found via the currency information or by sequentially reading the pointer array. Writing the record usually requires a further disk access.

- Deletion

The record and its associated entry in the pointer array must be deleted.

- Result

MODE IS POINTER-ARRAY allows speedy sequential and direct access and updating operations. The response times are largely independent of the order of the sets and the size of the set occurrences.

List

- Sequential access

If MODE IS LIST, the records are grouped together in a contiguous storage area. This storage mode offers fastest sequential processing. The number of accesses when processing large numbers of records depends on the record length.

- Direct access

If ORDER IS SORTED INDEXED, all levels of the list must be in memory. The higher levels are used to select the number of the page containing the record. One disk access is required for each level of the table. The number of levels depends on the record length.

If ORDER IS SORTED INDEXED has not been defined, UDS/SQL must search the set occurrence from the beginning. The number of disk accesses required for this depends to a large degree on the record length and the size of the set occurrence.

- Insertion

If ORDER IS SORTED INDEXED, UDS/SQL finds the place where the record is to be inserted by means of direct access. Otherwise UDS/SQL requires a maximum of two disk accesses.

Insertion of a record may necessitate re-storage of a number of records by UDS/SQL.

- Deletion

In general, UDS/SQL need modify only the page from which the record is to be deleted.

- Result

MODE IS LIST offers the fastest sequential access mode. If an updating operation causes a change in the record sequence, this may result in the physical relocation of records. The PPPs in SEARCH key tables pointing to these records are then no longer valid.

The access behavior depends to a large degree on the order and size of the set occurrences.

Chain

- Sequential access

When records are processed in their logical order, a maximum of one disk access for each member record is required. The number of accesses can be considerably reduced by optimizing the placement of the records (see [page 165](#)). If the logical order is not adhered to, UDS/SQL will normally have to search large parts of the set occurrence which may involve a large number of disk accesses. It should be noted that for reading a preceding record UDS/SQL must search the set occurrence from the beginning unless backward chaining has been specified.

If the physical placement of the records changes frequently (e.g. if member records for a different set are stored as a list), it is advisable to perform occasional reorganization runs in order to keep the PPPs current (see the "[Recovery, Information and Reorganization](#)" manual, BREORG).

- Direct access

- ORDER IS SORTED INDEXED:

Analogous to direct access in pointer array.

- ORDER IS LAST/FIRST/NEXT/PRIOR/SORTED:

On average, UDS/SQL must search half a set occurrence to find a record. A maximum of one disk access per record is required.

- Insertion

When a record is inserted, it is linked to the preceding and to the subsequent record by means of pointers.

For this reason, UDS/SQL updates the records preceding and, in the case of backward chaining, following the record inserted.

- ORDER IS SORTED [INDEXED]:

UDS/SQL finds the place where the record is inserted by direct access.

- ORDER IS LAST/FIRST/NEXT:

A pointer points from the owner record to the preceding record, or the owner record is the preceding record, or the preceding record is immediately obtainable by means of the currency information. Insertion of the record requires a maximum of two disk accesses.

- ORDER IS PRIOR:

UDS/SQL must search the set occurrence from the beginning unless backward chaining has been specified. If backward chaining has been specified, insertion of a record requires a maximum of three disk accesses.

- Deletion

UDS/SQL must find the record to be deleted and also the record preceding it. This requires less time if backward chaining has been specified. In the case of backward chaining, UDS/SQL must also update the pointer in the subsequent record. If ORDER IS SORTED INDEXED, the entry made for this record in the sort key table must be deleted.

- Result

MODE IS CHAIN allows fast sequential access if the records are processed in the sequence defined in the ORDER clause.

If the ORDER clause specifies SORTED INDEXED, fast direct access is possible even in large set occurrences. If ORDER IS SORTED is specified, insertions are slower and depend on the number of member records in the set occurrence. Modifications of the primary key are relatively time consuming.

If the order of the member records is immaterial, the fastest mode of storing records is achieved by specifying MODE IS CHAIN and ORDER IS NEXT or ORDER IS FIRST.

5.3.3 Preventing redundancy in SEARCH key tables

TYPE IS DATABASE-KEY-LIST

In the description of the logical data structure drawn up using the schema DDL, the user decides if UDS/SQL is to set up a SEARCH key table (SEARCH KEY IS... USING INDEX). The form of this table can be influenced by means of the TYPE clause.

The UDS/SQL standard form of SEARCH key table is that of a multi-level pointer array: It consists of one line for every record of the associated record type or for each member record of the associated set occurrence. This line contains the key value of the record and the pointer to the record.

(This standard form corresponds to the specification: TYPE IS REPEATED-KEY.)

Different records can often have the same key values (DUPLICATES ARE ALLOWED). In such cases, TYPE IS DATABASE-KEY-LIST can be specified; this ensures that a key value with duplicates is stored only once.

This type of SEARCH key table is referred to as a duplicates table. Duplicates tables are useful

- in the case of long keys,
- if several key values have between 5 and 2000 duplicates,
- when processing large numbers of records (see the "[Application Programming](#)" manual, FIND 7).

A third overflow page must be generated by UDS/SQL in the following cases:

- if there are more than approximately 2000 duplicates in a 2-Kbyte or 4-Kbyte database
- if there are more than approximately 4000 duplicates in an 8-Kbyte database.

This has an adverse effect on the access behavior (one additional access per overflow page), which should be weighed against the saving in space achieved.

Duplicates tables are set up by UDS/SQL on the lowest level only. Higher table levels are comparable to those of a standard SEARCH key table.

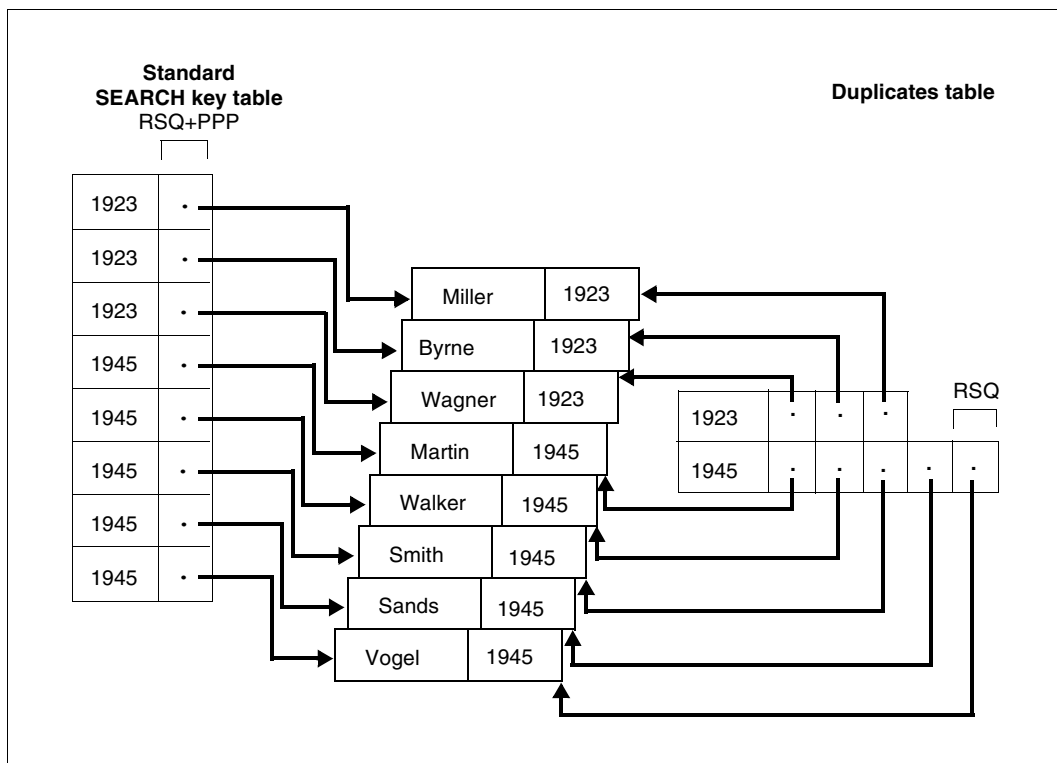


Figure 39: Comparison of standard SEARCH key table with duplicates table

The pointers in the duplicates table are the RSQs of the associated records. The pointers associated with the same key value are sorted in ascending order according to RSQs. Thus an additional access to the DBTT is required to find the pages containing the records.

5.3.4 Adding a pointer to link a member to its owner

MEMBER IS PHYSICALLY LINKED TO OWNER

This entry specifies that each member record of a set is provided with a pointer to the associated owner record. This pointer is a PPP. It optimizes access to the owner record if one of its member records has already been selected (see the "[Application Programming](#)" manual, FIND 6). Such accesses are often required in parts list processing, for example.

This must not be specified for a SYSTEM set.

5.4 Defining the placement of member records, tables and hash areas

The SSL provides options to define the placement of the following objects:

- member records
- lists
- pointer arrays
- sort key tables
- SEARCH key tables
- DBTTs
- hash areas.

These options include, in particular:

- For member records, lists, pointer arrays, sort key tables and set SEARCH key tables, you can define not only the realm that is to contain the records but also the concentration of data in one page or in contiguous pages within this realm.
- In the case of hash areas, DBTTs and record SEARCH key tables, it is only possible to define the realm in which they are to be stored.

5.4.1 Defining the placement of member records, associated tables and hash areas for secondary keys

This section covers the possibilities of defining the placement of data belonging to one set occurrence. Such data may be:

- an owner record
- member records or list
- a pointer array
- a sort key table
- a set SEARCH key table
- a hash area for the set secondary key

5.4.1.1 Placement at realm level

The following clauses are used to allocate data to certain realms without specifying its position within the realm.

Owner record, member records

WITHIN clause of the schema DDL (see the section [“Defining allocation of records to realms” on page 110](#)).

List

```
MODE IS LIST DETACHED [WITHIN realm-name]
```

Pointer array

```
MODE IS POINTER-ARRAY DETACHED [WITHIN realm-name]
```

Sort key table, set SEARCH key table, hash area for set secondary key

```
INDEX NAME IS name PLACING IS DETACHED [WITHIN realm-name]
```

realm-name specifies the name of the realm that is to contain the list, pointer array, sort key table, set SEARCH key table or hash area.

If this entry is omitted, UDS/SQL places the lists, pointer arrays or sort key tables in the realm of the associated owner record, unless the set is a SYSTEM set. In the case of a non-dynamic SYSTEM set, these tables or hash areas are automatically placed in the first realm referred to in the DDL WITHIN clause for the member record type. A dynamic set is stored in the temporary realm by UDS/SQL.

The placement of a list is determined without a DETACHED WITHIN clause by the placement of the owner.

If no entry is made for a set SEARCH key table or hash area, UDS/SQL selects the realm according to the following principle:

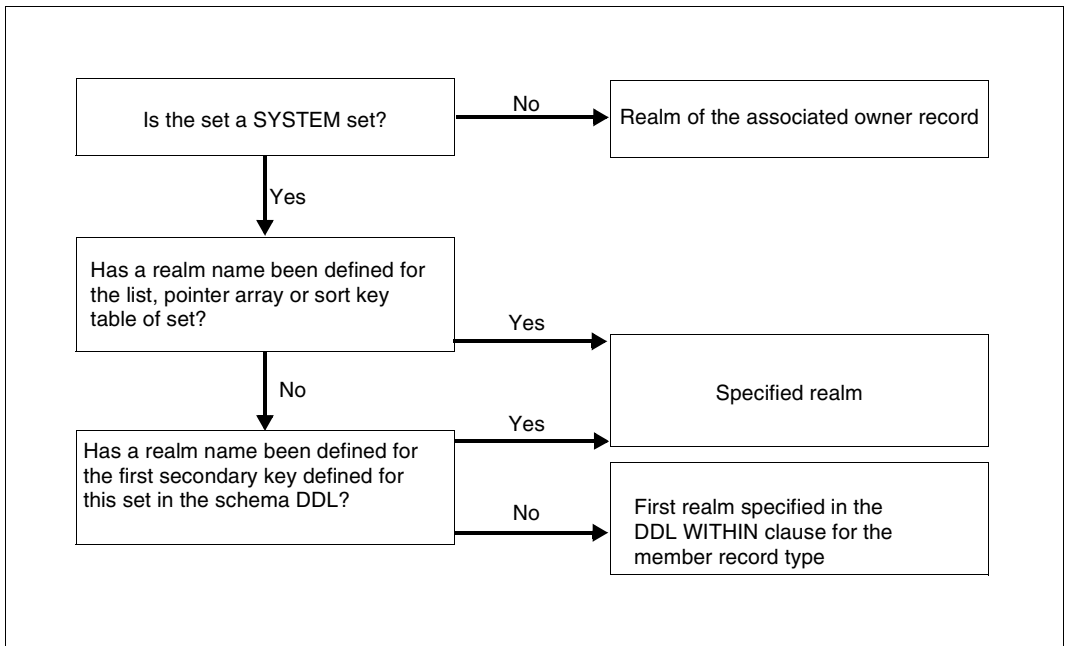


Figure 40: Default value for *realm-name*

name specifies the name of the sort or SEARCH key table to be placed. This name must have been assigned in the schema DDL (see the section [“Assigning names to hash areas and tables” on page 107](#)).

5.4.1.2 Placement within a realm

Within a realm, data belonging to one set occurrence can be stored contiguously. If, for example, member records are stored as list, they are physically stored in one page until the page is completely filled. The other possibilities relate to the storage of member records and the associated tables in the proximity of the owner record.

These are:

- Natural optimization
Owner proximity for lists, pointer arrays, sort key tables, set SEARCH key tables and member records
- PLACEMENT OPTIMIZATION
Owner proximity for member records
- MODE clause
Owner page contains pointer array or list
- INDEX clause
Owner page contains sort key table or set SEARCH key table

Natural optimization

If the user does not influence the placement of data within a realm by means of the DDL and SSL, UDS/SQL physically stores the data in the chronological order in which it is entered. Thus, at initial load time or later on when running unload or load programs, the user can select the load sequence in such a manner that a complete set occurrence with all associated tables is stored in contiguous pages. Unlike other optimization methods, natural optimization is effective over several levels of hierarchy. This method is especially useful if the user recognizes critical access paths. In this case, data must be stored in the same sequence as in the critical access path.

Example

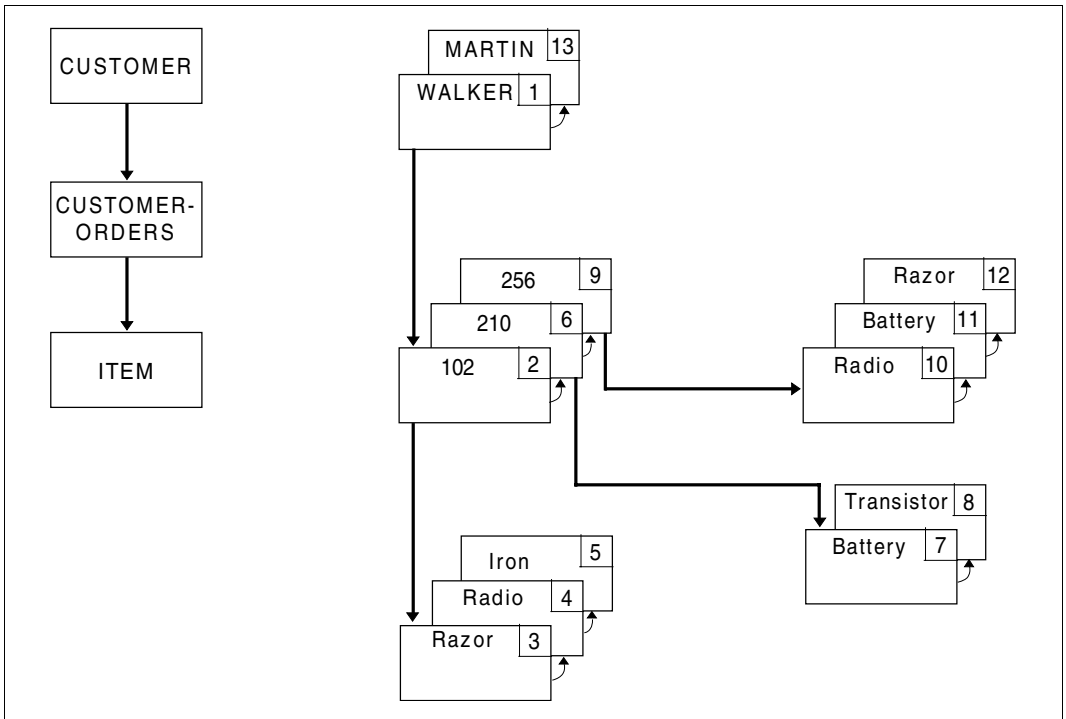


Figure 41: Load sequence for natural optimization

In this example, the access behavior on the critical access path CUSTOMER, CUSTOMER-ORDER, ITEM is improved by the specified load sequence. The emphasis has been placed on the optimization of the CUSTOMER-ORDER, ITEM path.

PLACEMENT OPTIMIZATION

PLACEMENT OPTIMIZATION FOR SET *set-name*

set-name specifies the name of the set to be optimized as an access path. It must not be a SYSTEM set. This entry is to be added to the member record type of the set.

As a result of this entry, UDS/SQL reserves sufficient storage space immediately after the owner record (when storing it) to hold the following information:

- Number of member records as specified in the POPULATION clause (see the section [“Storage space requirements for member records” on page 141](#)).
- If member records are owners in other sets:
all tables to be stored in the pages of these records using ATTACHED (see the sections [“MODE clause” on page 166](#) and [“INDEX clause” on page 167](#)).

A prerequisite is that placement has not been specified for the owner record type of the specified set using:

- LOCATION MODE IS CALC
- MODE IS LIST or
- PLACEMENT OPTIMIZATION

In the case of a record type defined in the schema DDL with LOCATION MODE IS CALC, PLACEMENT OPTIMIZATION generates indirect CALC pages.

If the set is defined with MODE IS LIST then PLACEMENT OPTIMIZATION is ignored.

MODE clause

`MODE IS` $\left. \begin{array}{l} \text{POINTER-ARRAY} \\ \text{LIST} \end{array} \right\}$ `ATTACHED TO OWNER`

This entry is not permitted in SYSTEM sets.

If a set occurrence population greater than zero has been specified for the set (see the section [“Specifying the size of the set occurrences of a set” on page 140](#)), UDS/SQL sets up all the tables of a set occurrence subsequent to the owner record when the owner record is stored.

A table for which ATTACHED has been specified takes precedence over other tables in that it is stored in the space immediately following the owner.

Even if the owner is a member of a set for which space has been reserved with the PLACEMENT OPTIMIZATION clause, UDS/SQL takes into account the storage space requirements for an associated ATTACHED table so that the owner record can be stored contiguously with the table (see the section [“PLACEMENT OPTIMIZATION” on page 165](#)).

If zero has been specified as set occurrence population, UDS/SQL sets up a table when the first member record is stored. In this case, ATTACHED results in UDS/SQL setting up the table as close as possible to the owner record.

ATTACHED is thus a means of making the owner record and its associated table available in one disk access, provided there is enough space for the table in the owner page.

A prerequisite is that the owner record type has not been defined with LOCATION MODE IS CALC, as space reservation for member records and tables is not possible in hash areas.

INDEX clause

INDEX NAME IS *name* PLACING IS ATTACHED TO OWNER

This entry is not permitted for SYSTEM sets. It is used to place tables of primary and secondary keys in the proximity of the associated owner.

For further details on the ATTACHED entry see the MODE clause.

name specifies the name of the table to be placed. It must have been assigned in the schema DDL (see the section [“Assigning names to hash areas and tables” on page 107](#)).

5.4.2 Defining the placement of record SEARCH key table, DBTT and record hash areas

For this data, the user can specify only the realm in which it is to be stored.

The following clauses are used to allocate data to specific realms:

Record SEARCH key table

```
NDEX NAME IS name PLACING IS WITHIN realm-name
```

DBTT

```
DATABASE-KEY-TRANSLATION-TABLE WITHIN realm-name
```

Hash area for primary key

```
POPULATION IS { integer WITHIN realm-name }, ...
```

Hash area for record secondary key

```
INDEX NAME IS name PLACING IS WITHIN realm-name
```

name specifies the name of the table or hash area to be placed. This name must have been assigned in the schema DDL (see the section [“Assigning names to hash areas and tables” on page 107](#)).

realm-name specifies the realm in which the table or the hash area is to be stored. If no realm name is specified, UDS/SQL stores the DBTT and the secondary keys in the realm specified first in the DDL WITHIN clause for this record type.

The hash area for primary keys is always set up according to the DDL WITHIN specification.

Example

```

DDL: SET NAME IS CUSTOMER-ORDERS-PLACED
      .
      .
      .
      OWNER IS CUSTOMER.
      MEMBER IS CUSTOMER-ORDER
      .
      .

SSL: RECORD NAME IS CUSTOMER-ORDER
      PLACEMENT OPTIMIZATION FOR SET CUSTOMER-ORDERS-PLACED.

SET NAME IS CUSTOMER-ORDERS-PLACED
POPULATION IS 10
MODE IS POINTER-ARRAY ATTACHED TO OWNER
INDEX NAME IS SEARCH-TAB-C-ORD-PLCD
PLACING IS DETACHED.

```

In this example, UDS/SQL would arrange the records and table of the set as follows:

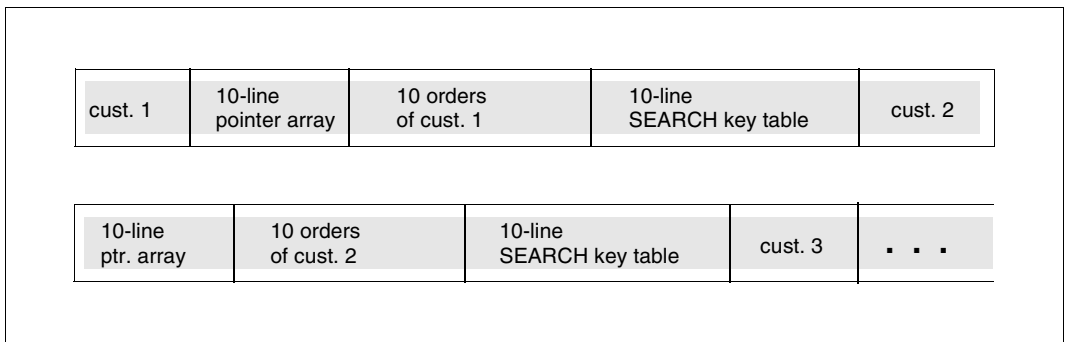


Figure 42: Placement of data based on the above definition

5.4.3 Overview of placement statements

Type of data	WITHIN realm-name,...	Member records in set with			LOCATI-ON MODE IS CALC	Placement of the records	
		OWNER IS SYSTEM	PLACEMENT OPTIMIZA-TION FOR SET	MODE IS ...			
records	manda-tory	yes	yes	not permitted			
			no	LIST	ATTACHED	not permitted	
					DETACHED	-	list in owner realm
					DETACHED WITHIN	-	list in DETACHED-WITHIN realm
				POINTER-ARRAY	ATTACHED	not permitted	
					DETACHED	yes	hash area in WITHIN realm
					DETACHED WITHIN	yes	hash area in WITHIN realm
			CHAIN	-	yes	in WITHIN realm	
				-	no	in WITHIN realm	
			no	yes	LIST	ATTACHED	-
		DETACHED				-	list of owner
		DETACHED WITHIN				not permitted	
		POINTER-ARRAY/ CHAIN			-	yes	hash area (indirect CALC) in WITHIN realm; record of owner
					-	no	with owner
		no			LIST	ATTACHED	-
				DETACHED		-	list in owner realm
				DETACHED WITHIN		-	list in DETACHED-WITHIN realm
				POINTER-ARRAY/ CHAIN	-	yes	hash area in WITHIN realm
		no		-	no	in WITHIN realm	

Table 8: Placement statements for records

Type of data	Type of Set	MODE, INDEX or DBTT clause specifications	Placement	
			Realm	within realm
List Pointer array Sort key table Set SEARCH key table	not SYSTEM	ATTACHED TO OWNER	Owner realm	next to owner
	SYSTEM	DETACHED ¹⁾	first realm in WITHIN clause for member record type	according to time of storage
	not SYSTEM or SYSTEM	DETACHED WITHIN <i>realm-name</i>	realm specified	
Pointer array	dynamic		temporary realm	
Hash area for set SEARCH key	SYSTEM	DETACHED	first realm in WITHIN clause for member record type	-
		DETACHED WITHIN <i>realm-name</i>	realm specified	
Record SEARCH key table	-	WITHIN <i>realm-name</i>		according to time of storage
		default	first realm in WITHIN clause	
Hash area for record SEARCH key	-	WITHIN <i>realm-name</i>	realm specified	
		default	first realm in WITHIN clause	
Hash area for primary key	-	-	realms in WITHIN clause	-
DBTT		DBTT WITHIN <i>realm-name</i>	realm specified	
		default	first realm in WITHIN clause	

Table 9: Placement statements for tables and hash areas

¹ For set SEARCH key tables for a SYSTEM set, refer to the default values for *realm-name* in [figure 40](#).

5.5 Defining the extent of table reorganization desired

DYNAMIC REORGANIZATION SPANS *integer* PAGES

If, when records are stored, the storage space requirements are formed to exceed those initially calculated on the basis of the POPULATION clause, UDS/SQL automatically performs a table extension. A table that is not quite the size of a page but that cannot be extended by at least two table lines within its page is continued in a further page by UDS/SQL. If a table spans several pages, however, and the part to be extended requires an entire page, UDS/SQL can search a certain number of the pages containing the logically adjacent parts of the table for free space.

integer denotes the number of table pages to be searched. In the case of pointer arrays, lists, sort key tables and set SEARCH key tables, this specification only takes effect if the set has not been defined with ORDER IS LAST/FIRST.

The pages to be searched are, at the same time, the pages submitted to UDS/SQL for dynamic reorganization.

If UDS/SQL finds free space for a new table entry within these pages, this space is relocated to the place where the table entry is to be inserted.

If, however, all pages searched are completely filled with table entries, UDS/SQL extends the table by one empty page after the searched pages. Further reorganization depends on the place at which the table entry is inserted.

1. The table entry is to be inserted in the table but not at the beginning or at the end: UDS/SQL evenly distributes the contents of the searched pages and the new table entry over the storage space which has been extended by one page.
2. The table entry is to be appended at the end of the table. The new page contains the last table entry and the new table entry.
3. The table entry is to be inserted at the beginning of the table: UDS/SQL moves the contents of the first table page, with the exception of the first entry, to the new page. The table entry to be inserted is stored in the first page.

By this method, a high occupancy level is achieved for the table pages. In cases 2) and 3), the occupancy level is obtained using the following formula:

$$\text{Occupancy level [\%]} = \frac{n-1}{n} \times 100$$

where *n* is the maximum number of table entries per page.

In case 1), the occupancy level is dependent on the *integer* specified. It must not fall below the following value:

$$\text{Minimum occupancy level [\%]} = \frac{\text{integer}}{\text{integer}+1} \times 100.$$

This means that high occupancy levels are most easily obtained if records are stored in sorted order. High occupancy levels reduce storage space requirements for tables and result in shorter access paths.

In order to obtain high occupancy levels by dynamic reorganization, the following must be taken into consideration:

High values for *integer*

- are more acceptable for pointer arrays, sort and SEARCH key tables than for lists;
- are more acceptable for records that remain unchanged than for frequently changing records.

Dynamic reorganization cannot be used for dynamic sets; for multi-level tables it can only be applied on the lowest table level. Duplicates tables cannot be dynamically reorganized.

The default value for *integer* is 2.

Examples

DYNAMIC REORGANIZATION SPANS 1 PAGES

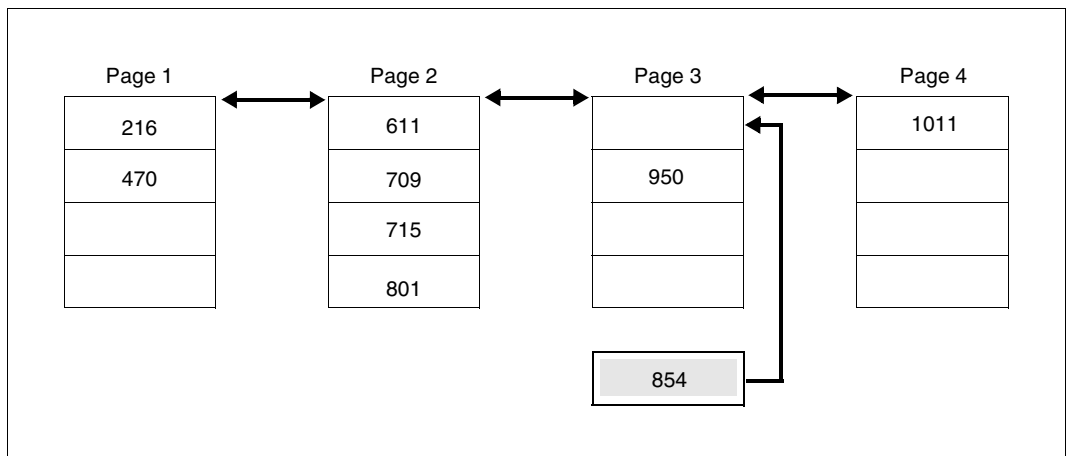


Figure 43: Inserting a record without reorganization

Record 854 can be inserted without reorganization.

In order to insert record 650 (see below), UDS/SQL must set up a new page, since page 2 is completely occupied. The new page accepts as many entries from page 2 as is necessary to ensure even distribution of records over the two pages.

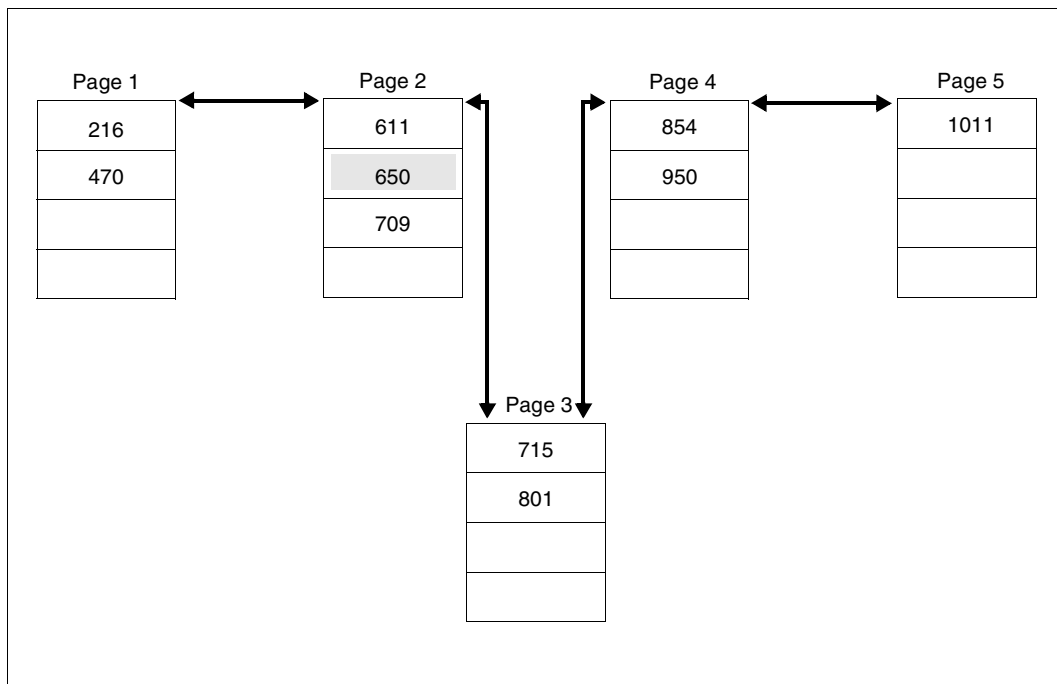


Figure 44: Inserting a record with table extension and subsequent reorganization

DYNAMIC REORGANIZATION SPANS 3 PAGES

Based on [figure 43](#), the following situation results from inserting record 650.

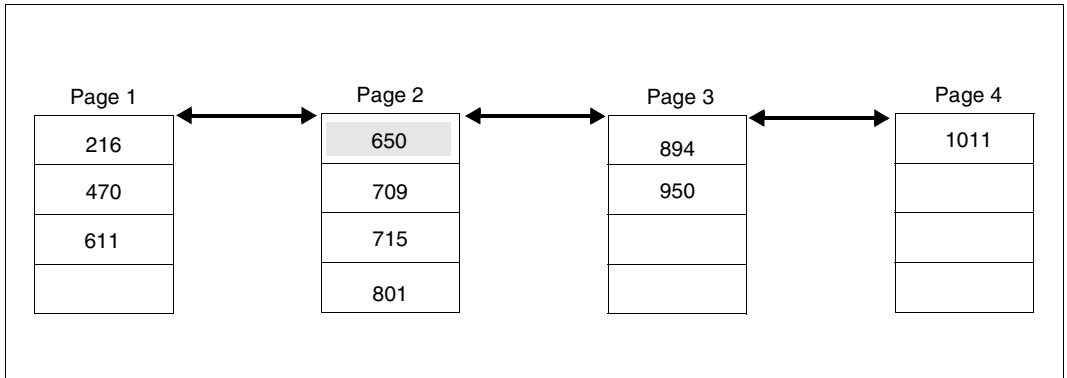


Figure 45: Inserting a record by shifting free space

5.6 Storing the records of a record type in compressed form

COMPRESSION FOR ALL ITEMS

A record type containing an item of variable length may not be compressed. Under CALL DML, records can be compressed by storing only part of the items belonging to the record type (see the "[Application Programming](#)" manual, STORE 2). The COMPRESSION clause is used if UDS/SQL is not to fill the omitted items with standard values, but is to store records in the compressed form in which they exist at the CALL interface.

For record types defined with LOCATION MODE IS CALC in the schema DDL compression generates indirect CALC pages.

If this clause is specified, no records of the record type can be updated with SQL.

5.7 Formulas for calculating the storage space requirements for records and tables

The storage space requirement for records varies depending on whether they are stored in a direct hash area, whether an indirect hash area is set up for them and which type of connection has been specified for them. [table 10](#) and [table 11](#) contain formulas to calculate the storage space requirement. These formulas differ, depending on whether the page length of the database is 2048 bytes, 4000 bytes or 8096 bytes.

Calculation formulas for a database with a 2048-byte page length

Number of records	in the data page	$\frac{2028}{\text{record length}^{1+8}}$
	in the direct CALC page	$\frac{2018}{\text{record length}+\text{key length}+15}$
Number of entries in the indirect CALC page		$\frac{2018}{\text{key length}+7}$
Number of table entries per page in a:	List	$\frac{2002}{\text{key length}+8}$
	pointer array ² sort key table ³	
	SEARCH key table (TYPE IS REPEATED-KEY)	$\frac{2002}{\text{key length}+7}$

Table 10: Calculation formulas for a database with a 2048-byte page length

Calculation formulas for a database with a 4000 or 8096-byte page length

Number of records	in the data page	$\frac{\text{page length}-20}{\text{record length}^1 +12}$
	in the direct CALC page	$\frac{\text{page length}-30}{\text{record length}+\text{key length}+22}$
Number of entries in the indirect CALC page		$\frac{\text{page length}-30}{\text{key length}+10}$
Number of table entries per page in a:	List	$\frac{\text{page length}-50}{\text{record length}+12}$
	pointer array ² sort key table ³	$\frac{\text{page length}-50}{\text{key length}+10}$
	SEARCH key table (TYPE IS REPEATED-KEY)	$\frac{\text{page length}-50}{\text{key length}+10}$

Table 11: Calculation formulas for a database with a 4000 or 8096-byte page length

1

The record length is the length of a record according to the schema DDL plus the length of its pointers (see [page 224](#), SCD)

2

If ORDER IS $\left. \begin{array}{l} \text{LAST} \\ \text{FIRST} \\ \text{NEXT} \end{array} \right\}$ key length = 0

$\left. \begin{array}{l} \text{PRIOR} \\ \text{IMMATERIAL} \\ \text{SORTED INDEXED BY DATABASE-KEY} \end{array} \right\}$

3

If ORDER IS SORTED INDEXED BY DATABASE-KEY key length = 0

5.8 Comprehensive example of SSL application

```
STORAGE STRUCTURE OF SCHEMA MAIL-ORDERS.
*
*
RECORD NAME IS                CUSTOMER
    DATABASE-KEY-TRANSLATION-TABLE IS 100.
*
RECORD NAME IS                CST-ORDERS
    DATABASE-KEY-TRANSLATION-TABLE IS 400
    PLACEMENT OPTIMIZATION FOR SET CST-ORD-PLACED.
*
RECORD NAME IS                ORD-ITEM
    DATABASE-KEY-TRANSLATION-TABLE IS 1000.
*
RECORD NAME IS                INSTALMENT
    DATABASE-KEY-TRANSLATION-TABLE IS 50
    INDEX NAME IS SEARCH-TAB-INSTALMENT
    TYPE IS DATABASE-KEY-LIST.
*
RECORD NAME IS                ART-TYPE
    DATABASE-KEY-TRANSLATION-TABLE WITHIN ARTICLE-RLM
    INDEX NAME IS SEARCH-TAB-ART-TYPE
    PLACING IS WITHIN ARTICLE-RLM.
*
RECORD NAME IS                ART-SELECTION
    DATABASE-KEY-TRANSLATION-TABLE WITHIN ARTICLE-RLM
    INDEX NAME IS SEARCH-TAB-ARTICLE-SELECTION
    PLACING IS WITHIN ARTICLE-RLM
    DYNAMIC REORGANIZATION SPANS 5 PAGES.
*
RECORD NAME IS                ART-DESCR
    DATABASE-KEY-TRANSLATION-TABLE IS 300 WITHIN ARTICLE-RLM
    POPULATION IS    200 WITHIN CLOTHING,
    100 WITHIN HOUSEHOLD-GOODS,
    200 WITHIN SPORTS-ARTICLES,
    70 WITHIN FOOD,
    200 WITHIN LEISURE,
    100 WITHIN STATIONERY.
*
RECORD NAME IS                ARTICLE
    DATABASE-KEY-TRANSLATION-TABLE IS 600 WITHIN ARTICLE-RLM
    POPULATION IS    400 WITHIN CLOTHING,
    100 WITHIN HOUSEHOLD-GOODS,
    400 WITHIN SPORTS-ARTICLES,
    150 WITHIN FOOD,
    400 WITHIN LEISURE,
```

```
250 WITHIN STATIONERY
INDEX NAME IS SEARCH-TAB-ARTICLE-1
PLACING IS WITHIN ARTICLE-RLM
INDEX NAME IS SEARCH-TAB-ARTICLE-2
PLACING IS WITHIN ARTICLE-RLM.
*
RECORD NAME IS                MATERIALS
INDEX NAME IS SEARCH-TAB-MATERIAL-1
DYNAMIC REORGANIZATION SPANS 5 PAGES
INDEX NAME IS SEARCH-TAB-MATERIAL-2
DYNAMIC REORGANIZATION SPANS 5 PAGES.
*
RECORD NAME IS                SUPPLIER
DATABASE-KEY-TRANSLATION-TABLE IS 500
POPULATION IS 200 WITHIN PURCHASE-ORDER-RLM.
*
RECORD NAME IS                PURCHASE-ORDER
DATABASE-KEY-TRANSLATION-TABLE IS 200.
*
RECORD NAME IS                P-ORD-ITEM
DATABASE-KEY-TRANSLATION-TABLE IS 500.
*
*
SET NAME IS                    CST-ORD-PLACED
MODE IS POINTER-ARRAY ATTACHED TO OWNER
POPULATION IS 10
INDEX NAME IS SEARCH-TAB-C-O-PLCD
PLACING IS DETACHED.
*
SET NAME IS                    OFFER
MODE IS POINTER-ARRAY DETACHED WITHIN ARTICLE-RLM
WITH PHYSICAL LINK
POPULATION IS 100 INCREASE IS 5
DYNAMIC REORGANIZATION SPANS 5 PAGES.
*
SET NAME IS                    SHORT-LIST
MODE IS POINTER-ARRAY DETACHED WITHIN ARTICLE-RLM
POPULATION IS 100 INCREASE IS 20
DYNAMIC REORGANIZATION SPANS 5 PAGES.
*
SET NAME IS                    P-ORD-SPEC
MODE IS LIST DETACHED WITH PHYSICAL LINK
POPULATION IS 15
MEMBER IS PHYSICALLY LINKED TO OWNER.
*
SET NAME IS                    MIN-STOCK-LEVEL
MODE IS CHAIN LINKED TO PRIOR.
*
```

SET NAME IS CONTAINING
POPULATION IS 10
MEMBER IS PHYSICALLY LINKED TO OWNER.
*
SET NAME IS CONTAINED-IN
MODE IS CHAIN LINKED TO PRIOR
MEMBER IS PHYSICALLY LINKED TO OWNER.
*
SET NAME IS ARTICLES-AVAILABLE
POPULATION IS 500
DYNAMIC REORGANIZATION SPANS 5 PAGES
INDEX NAME IS SEARCH-TAB-ART-AVAIL
PLACING IS DETACHED
TYPE IS DATABASE-KEY-LIST
MEMBER IS PHYSICALLY LINKED TO OWNER.
*
SET NAME IS ORDERED-ARTICLES
MEMBER IS PHYSICALLY LINKED TO OWNER.
*
SET NAME IS REORDERED-ARTICLES
MEMBER IS PHYSICALLY LINKED TO OWNER.
*
SET NAME IS P-ORD-CONTENTS
MODE IS LIST DETACHED WITH PHYSICAL LINK
POPULATION IS 20.

5.9 Reserved words of the SSL compiler

ALL	AREA
ASSIGNED	ATTACHED
CHAIN	COMPRESSION
DATABASE-KEY-LIST	DATABASE-KEY-TRANSLATION-TABLE
DBKEY-TRANSLATION-TABLE	DBTT
DCB-NAME	DETACHED
DYNAMIC	FOR
INCREASE	INDEX
INDICATOR	IS
ITEMS	LINK
LINKED	LIST
MEMBER	MODE
NAME	OF
OPTIMIZATION	OWNER
PAGES	PHYSICAL
PHYSICALLY	PLACEMENT
PLACING	POINTER-ARRAY
POPULATION	PRIOR
RECORD	REORGANIZATION
REPEATED-KEY	SCHEMA
SET	SPANS
STORAGE	STRUCTURE
TO	TYPE
WITH	WITHIN

6 Definition of the user interface to the database

6.1 Subschema DDL

6.1.1 Introduction

All data the database requires to perform its tasks must be defined in the UDS/SQL schema. The UDS/SQL schema has no direct interface with the user, however; aspects of user-oriented data editing must not be considered when designing the UDS/SQL schema.

The user interface is created when the subschema is defined. The subschema describes that section of the schema which is adapted to the requirements of a particular application.

This is advantageous in so far as the database programmer need know only that part of the database required for a particular application. The subschema plays its part in data protection as it prevents the user from becoming acquainted with the entire data resources contained in the database.

The following functions are available for changing the data structure of the schema to produce a subschema data structure designed for a particular application:

- exclusion of record types,
- exclusion of sets,
- exclusion of realms,
- exclusion of record elements from a record type,
- reduction of repetition factor of vectors and repeating groups,
- grouping of record elements into new group items,
- definition of conditions.

This chapter describes how these functions are implemented by the definition of a subschema.

The metalanguage used is described on [page 22](#), and the general syntax rules are summarized on [page 234](#).

6.1.2 Assigning name and privacy to a subschema

```
SUB-SCHEMA NAME IS subschema-name OF SCHEMA NAME schema-name  
[PRIVACY LOCK FOR COMPILE IS literal-1 [OR literal-2]]
```

subschema-name specifies the name of the subschema and is assigned by the user. Within one DB configuration, *subschema-name* must be unique in the first 6 characters.

schema-name specifies the schema from which the subschema is derived.

Access to the data in a database by means of a subschema is dependent on a user's access rights. Data protection can be enhanced by the assignment of passwords.

With *literal-1* and *literal-2*, the user can assign passwords that prevent the unauthorized compilation of a DML program by means of this subschema. Compilation is only possible if at least one password is known.

6.1.3 Unlocking a schema for creating a subschema

```
PRIVACY KEY FOR COPY IS literal_
```

If a schema has been provided with password protection to prevent the unauthorized creation of a subschema, users must prove their access authorization.

For *literal*, the user must specify one of the passwords defined to protect the schema (see the section [“Assigning name and privacy to a schema” on page 111](#)).

6.1.4 Copying entire record types from the schema into the subschema

Format 1:

```
COPY ALL RECORDS.
```

Format 2:

```
COPY record-name,... ┘
```

Format 1 is used if all record types in the schema are to be copied in their entirety into the subschema.

Format 2 is used if only part of the record types in the schema are to be copied into the subschema.

record-name denotes the record types to be copied in their entirety into the subschema. All record types belonging to sets contained in the subschema must be copied either in their entirety or in part.

6.1.5 Copying part of a record type from the schema into the subschema

```
01 record-name┘
   {level-number record-element-name[ PICTURE.....][ USAGE.....]
   [ OCCURS.....]┘}...
```

record-name specifies the name of the record type that is to be copied in part into the subschema. This must not be a record type containing an alphanumeric item of variable length. Such record types can only be copied in their entirety into the subschema (see above).

record-element-name denotes:

- a record element of the record type to be copied into the subschema. It is to be copied as described on pages 186 and 188 if it is an item, as described on page 189 if it is a vector or as described on page 190 if it is a repeating group.
- a group item defined as described on page 191.

All items comprising the record type as it is contained in the subschema must be specified in the same order as they appear in the schema description.

Copying a numeric item, an alphanumeric item of fixed length or a national item

level-number *item-name* PICTURE IS *mask-string*

[USAGE IS { DISPLAY
COMPUTATIONAL=3
COMPUTATIONAL
NATIONAL }]₊

In *level-number*, the user specifies if an item is to belong to a group item.

If the item is not to belong to a group item, the level number specified must be the lowest of any record element in the record type. It may not be lower than 02.

If the item is already part of a repeating group in the schema, it must be defined as part of the same group item in the subschema. For selection of the appropriate level number, refer to [page 190](#).

If the item is to be included in a newly defined group item, refer to [page 191](#).

item-name is the name assigned to the item in the schema.

The item definition with PICTURE and USAGE clauses can be derived from the following table 12, where n , m and l denote integers.

Item type	Item definition in schema	Item definition in subschema
numeric, unpacked	PICTURE IS <i>mask-string</i>	PICTURE IS <i>mask-string</i> [USAGE IS DISPLAY]
alphanumeric, fixed length		<i>mask-string</i> identical with <i>mask-string</i> in schema
	TYPE IS CHARACTER m	PICTURE IS $X(m)$ [USAGE IS DISPLAY]
national	PICTURE IS $N(m)$	PICTURE IS $N(m)$ [USAGE IS NATIONAL]
numeric, packed	TYPE IS FIXED REAL DECIMAL n,m for $n > m, m > 0$	PICTURE IS $S9(l)V9(m)$ USAGE IS COMPUTATIONAL-3 $l := n - m$
	for $n > 0, m < 0$	PICTURE IS $S9(n)P(-m)$ USAGE IS COMPUTATIONAL-3
	for $n < m$	PICTURE IS $SP(l)9(n)$ USAGE IS COMPUTATIONAL-3 $l := m - n$
	for $n = m$	PICTURE IS $SV9(n)$ USAGE IS COMPUTATIONAL-3
	for $m = 0$	PICTURE IS $S9(n)$ USAGE IS COMPUTATIONAL-3
binary	TYPE IS FIXED REAL BINARY 15	PICTURE IS $S9(l)$ USAGE IS COMPUTATIONAL $l := 1,2,3,4$
	BINARY 31	PICTURE IS $S9(l)$ USAGE IS COMPUTATIONAL $l := 5,6,7,8,9$

Table 12: Item definition with PICTURE and USAGE clauses

If no entry is specified for USAGE and the mask string **does not** contain the symbol N, DISPLAY is assumed by default. If the mask string contains the symbol N, NATIONAL is assumed if the USAGE clause is missing.

Copying a database key item

$$level-number \ item-name \ \underline{USAGE} \ IS \ \left\{ \begin{array}{l} \underline{DATABASE-KEY} \\ \underline{DATABASE-KEY-LONG} \end{array} \right\}^+$$

In *level-number*, the user specifies if an item is to belong to a group item.

If the item is not to belong to a group item, the level number specified must be the lowest of any record element in the record type. It may not be lower than 02.

If the item is already part of a repeating group in the schema, it must be defined as part of the same group item in the subschema. To select the appropriate level number, proceed as described on [page 190](#).

If the item is to be included in a newly defined group item, proceed as described on [page 191](#).

item-name is the name assigned to the database key item in the schema.

An item that is copied into the subschema with USAGE IS DATABASE-KEY-LONG must be defined in the schema as a DATABASE-KEY-LONG item.

Copying a vector and reducing it if required

level-number *vector-name* PICTURE IS *mask-string*

```

[USAGE IS {
  DISPLAY
  COMPUTATIONAL_3
  COMPUTATIONAL
  NATIONAL
  DATABASE-KEY
  DATABASE-KEY-LONG
}]
[OCCURS integer TIMES]_

```

A vector is an item with a repetition factor. The repetition factor indicates how many duplicates of the item are grouped into the vector.

Copying a vector into a subschema requires the same steps as copying an item (see section [“Copying a numeric item, an alphanumeric item of fixed length or a national item” on page 186](#) and section [“Copying a database key item” on page 188](#)).

integer specifies the repetition factor, which must be at least 1, but no greater than that specified in the schema. A vector can thus be reduced to any number of items, even down to one item, when copying it into the subschema.

The default value for *integer* is 1.

Copying a repeating group and reducing it if required

```

level-number-1 group-item-name[ GROUP-USAGE IS NATIONAL]
                               [ OCCURS integer TIMES]_
{ level-number-2 record-element-name PICTURE.....
                                     USAGE.....
                                     OCCURS..... }.....

```

A repeating group is a group item with repetition factor. The repetition factor indicates the number of duplicates of this group item to be grouped into the repeating group.

A repeating group must be copied into a subschema if one of its items is to be copied.

When copied into a subschema, the repeating group can be reduced as follows:

- by specifying a lower repetition factor. It is possible to reduce a group item down to a single occurrence.
- by excluding record elements that are part of the repeating group from the subschema.

group-item-name specifies the name assigned to the repeating group by the user in the schema.

integer specifies the repetition factor. It must be at least 1 and may not exceed that specified in the schema.

The default value for *integer* is 1.

record-element-name denotes a record element which is defined as part of the repeating group in the schema and which is to be copied into the subschema. It is copied as described on [page 186](#) and [page 188](#) if it is an item, as described on [page 189](#) if it is a vector or as described in this section if it is a repeating group.

level-number-2 defines the record element as part of a repeating group in the subschema.

level-number-2 must be greater than *level-number-1*.

The following applies for all record elements to be copied from a repeating group in the schema into a subschema: Record elements must have the same level number if they have the next higher group item in common.

You use the GROUP-USAGE clause to declare a national repeating group, i.e. a repeating group which is treated in its entirety like a national data item. The GROUP-USAGE clause may only be specified if all lower-ranking record elements are of the type NATIONAL and *level-number-1* is not equal to 01.

Grouping record elements into a group item

```

level-number-1 group-item-name [ GROUP-USAGE IS NATIONAL ]
                               [ OCCURS integer TIMES ]_
{ level-number-2 record-element-name PICTURE.....
                               USAGE.....
                               OCCURS..... }....._

```

A group item is a named group of record elements within a record type. A record element can be an item, a vector or even a group item.

group-item-name specifies the name of a group item.

record-element-name specifies the record element to be declared part of the group item. This can be:

- a record element to be copied from the schema into the subschema. It is copied as described on pages [186](#) and [188](#) if it is an item, as described on [page 189](#) if it is a vector or as described on [page 190](#) if it is a repeating group.
- a group item, which is defined as described in this section.

level-number-2 must be greater than *level-number-1*.

The following applies for all record elements which are to be grouped into a group item: Record elements must have the same level number if they have the next higher group item in common.

You use the GROUP-USAGE clause to declare a national data group, i.e. a data group which is treated in its entirety like a national data item. The GROUP-USAGE clause may only be specified if all lower-ranking record elements are of the type NATIONAL and *level-number-1* is not equal to 01.

Defining a condition

Detailed information is provided in the COBOL2000 “[Language Reference Manual](#)”.

88 *condition-name*

$$\left. \begin{array}{l} \{ \text{VALUE IS} \} \\ \{ \text{VALUES ARE} \} \end{array} \right\} \{ \textit{literal-1} [\text{THROUGH} \textit{literal-2}] \}, \dots$$

The database programmer can make the execution of program statements dependent on conditions. A condition can be that certain items have certain item contents. Such items are then referred to as condition variables.

condition-name specifies the name assigned to the condition by the database programmer for reference purposes. The name must be assigned immediately following the description of the item which is to be the condition variable.

literal-1, *literal-2*, etc. denote value ranges applicable for the condition. The condition is fulfilled if the item assumes a value within the specified value range. The value range can be described by several individual values as well as smaller value ranges within the larger ranges. It must be within the value range defined for the item.

Several condition names and associated value ranges may be specified for one condition variable.

Example

```

01  ORDERS
    .
    .
    .
02  ORDER-STATUS PICTURE IS S9.
88  FINISHED VALUE IS 1.
88  OPEN      VALUE IS 0.

```

The condition FINISHED is met if the ORDER-STATUS item contains value 1. If it contains value 0, the OPEN condition is fulfilled.

6.1.6 Copying sets from the schema into the subschema

Format 1:

```
COPY ALL SETS.
```

Format 2:

```
COPY set-name-1, ...
```

Format 1 is used if all the sets in the schema are to be copied into the subschema.

Format 2 is used if only some of the sets in the schema are to be copied into the subschema.

set-name specifies the sets to be copied. All owner and member records of such sets must be present in the subschema.

If key items referencing the sets to be changed, the corresponding sets must have been defined in the subschema.

When SQL is used to access a record type that is a member record type in several sets, all the sets for the record type must be present in the subschema.

6.1.7 Copying realms from the schema into the subschema

Format 1:

```
COPY ALL AREAS.
```

Format 2:

```
COPY realm-name, ... .
```

Format 1 is used if all the realms in the schema are to be copied into the subschema.

Format 2 is used if only some of the realms in the schema are to be copied into the subschema.

realm-name specifies the realms to be copied. All realms that contain data relating to the record types and sets of the subschema and that the subschema user requires, must be copied. The [table 13](#) indicates which realms this may be. When SQL is used, the temporary realm must always be copied.

Type of data	Realm(s) from
Records	WITHIN clauses (DDL)
DBTTs	DBTT clauses
Hash areas for primary keys	Record POPULATION clauses (SSL)
Hash areas for record secondary keys	Record INDEX clauses (SSL)
Record SEARCH key tables	
Pointer arrays	MODE clauses (SSL)
Lists	
Sort key tables	Set INDEX clauses (SSL)
Set SEARCH key tables	
Hash areas for set secondary keys	

Table 13: Realms to be copied

6.1.8 Comprehensive example of subschema DDL

```
IDENTIFICATION DIVISION.  
    SUB-SCHEMA NAME IS ADMIN OF SCHEMA MAIL-ORDERS  
    PRIVACY KEY FOR COPY IS "SHIP-KEY".  
*  
*  
*  
DATA DIVISION.  
*  
AREA SECTION.  
    COPY ALL AREAS.  
*  
RECORD SECTION.  
    COPY ALL RECORDS.  
*  
SET SECTION.  
    COPY ALL SETS.  
*  
*  
*
```

6.2 Relational schema

The schema DDL can be used to create a schema complying with relational rules or with CODASYL rules. A schema defined according to relational rules contains no set relationships, and the primary and foreign keys are defined by the user.

If a schema was defined according to CODASYL rules, the utility routine BPSQLSIA can be used to provide SQL programmers with a relational view of the CODASYL subschema.

The BPSQLSIA utility routine analyzes all the objects in a subschema and represents them as objects in a relational database.

The list resulting from the conversion by BPSQLSIA contains all the information that programmers need to work with SQL.

If a CODASYL subschema is to be processed on a completely relational basis with SQL, it must satisfy certain requirements. Restrictions and notes on DDL, SSL and subschema DDL can be found in the relevant chapters.

A description of the BPSQLSIA utility routine, including an overview of the relevant restrictions and an example, is provided in another UDS/SQL manual (see the "[Recovery, Information and Reorganization](#)" manual, BPSQLSIA).

7 Structure of pages

When creating a database using DDL and SSL, the user does not usually require information on the structure of pages, records and tables.

The following two chapters are intended as a reference section for users interested in special details.

The entire storage space provided for the data in a database is divided into realms. Every realm consists of a specific number of pages that is specified when creating the database. This number may be subsequently modified (see the "[Creation and Restructuring](#)" manual).

The length of a page may be 2048, 4000 or 8096 bytes and must be uniform within a database. This page length is defined when creating the database with the BCREATE utility routine and can be subsequently extended with BPGSIZE (see the "[Creation and Restructuring](#)" manual for details).

Pages with a length of 4000 or 8096 bytes are each embedded in a page container, which consists of a 64-byte header that precedes the page itself, and a 32-byte trailer at the end.

The following types of pages can be distinguished in the structure of a realm. These pages differ in their data contents and structure:

Act-key-0 page

contains realm-specific data; is always the first page in a realm

Act-Key-N page

contains realm-specific data; is always the last page in a realm

FPA page

serves free place administration (FPA) on realm level. At least one FPA page is always present.

DBTT anchor page

manages the DBTT areas of a record type (DBTT: Database Key Translation Table).

DBTT page

page of a DBTT area. Manages the records of a record type.

CALC page

contains a hash area

Data page

contains records which are not stored in a hash area, and tables (except DBTT)

The following sections describe the page container and the various types of pages in detail. These descriptions of the individual page types are restricted to the pages themselves, i.e. the header and trailer for pages with a length of 4000 or 8096 bytes are not shown. Displacement values for individual page areas are always with reference to the start of the page. Unless explicitly specified otherwise, the given description applies to pages with lengths of 2048, 4000 and 8096 bytes. Specific differences, if any, are explained separately.

7.1 Page container

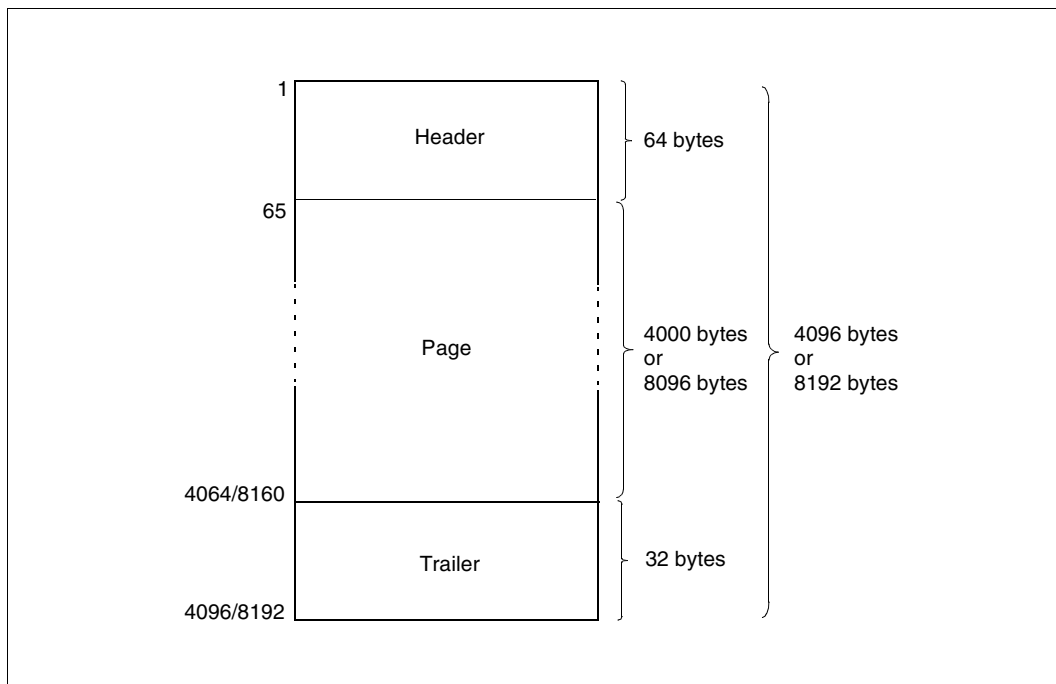


Figure 46: Structure of the page container for pages with a length of 4000 or 8096 bytes

Meaning of the bytes

Byte	Meaning
1-64	Header; bytes 17-24 contain a version number.
65-4064 or 65-8160	Page with a length of 4000 or 8096 bytes
4065-4096 or 8161-8192	Trailer; bytes 4089-4096 (for 4000-byte pages) and bytes 8185-8192 (for 8096-byte pages) contain a version number.

Table 14: Meanings of the bytes in a page container

Headers and trailers are used by UDS/SQL for administration purposes. Consistent pages have a matching version number in their header and trailer.

7.2 Act-key-0 and act-key-N page

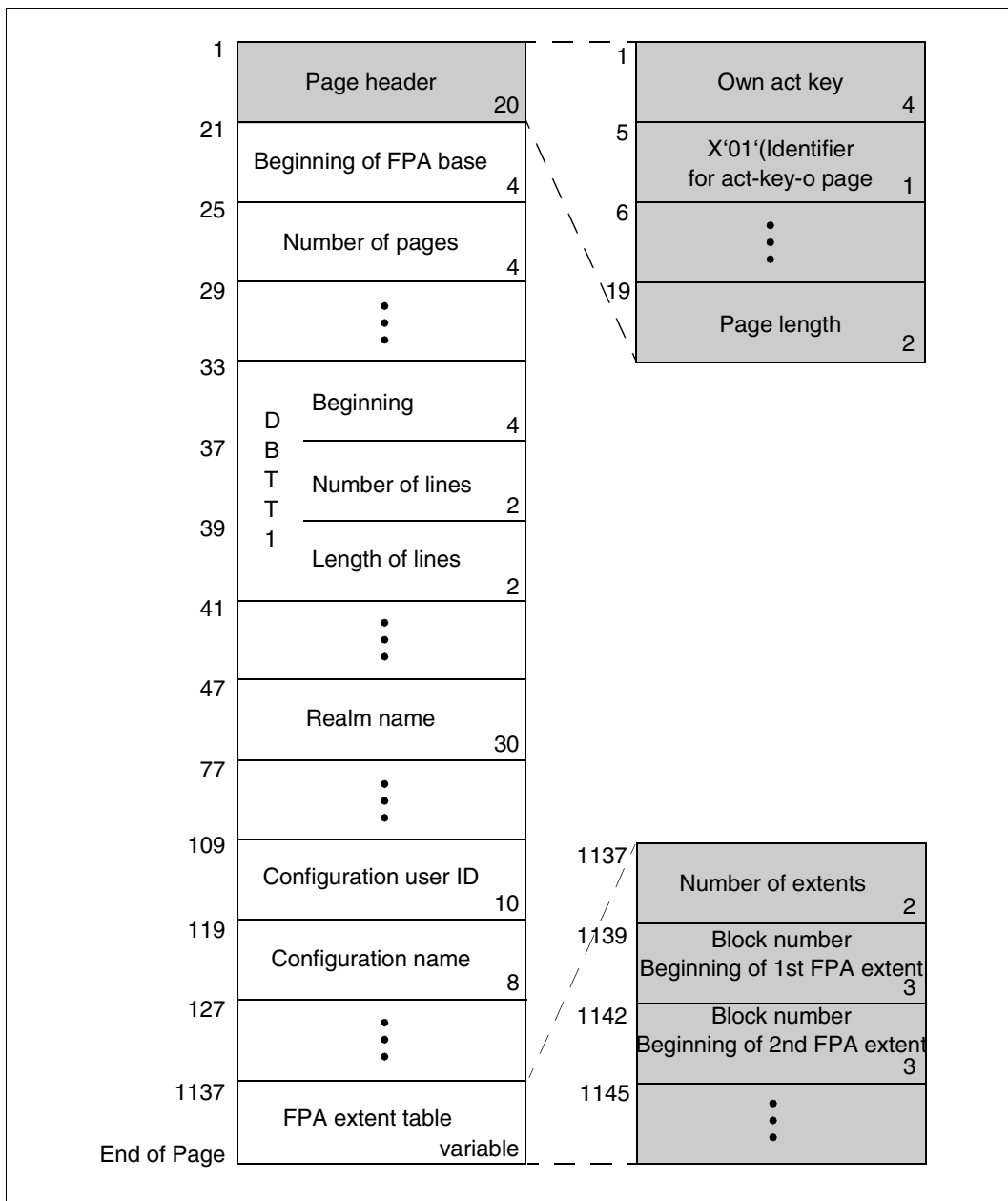


Figure 47: Structure of the act-key-0 and act-key-N page

Meaning of the bytes

Byte	Meaning
1-4	The act-key specifies the address of the page. Since the act-key-0 page is always the first page in a realm, the act-key of this page always contains page number 0.
19-20	The page length can be 2048, 4000 or 8096 bytes.
21-24	Specifies the act-key of the first FPA base page. For a description of the layout of an FPA base page, see section "FPA page" on page 202 .
25-28	Specifies the number of pages in the realm.
33-40	These fields are only filled in the DBDIR ¹ realm.
47-76	contains the name of the realm.
109-126	These fields are only filled in the DBDIR ¹ realm.
from 1137	Contains the FPA extent table. For a description of the layout of an FPA extent and an FPA extent page, see section "FPA page" on page 202 .

Table 15: Meanings of the bytes of the act-key-0 and act-key-N page

¹ DBDIR is the database directory.

7.3 FPA page

FPA pages constitute one level of the three-level UDS/SQL Free Place Administration and are used to administer free place on the realm level. There is also a free place administration facility on the page and table levels.

Note that if a database page contains tables, the value specified in the FPA page does not always indicate storage space actually occupied.

Every FPA page contains a separate act-key, which specifies the address of the page. The pair associated with a given act-key (i.e. the FPA page and entry within the page) can be specifically defined with the help of the information about the FPA base and the FPA extent table in the act-key-0 page.

Depending on which page length was defined for the database, the length of an FPA page may be 2048 bytes, 4000 bytes or 8096 bytes.

Additional free place administration tables (FPA extents) may be present in a realm in addition to the base free place administration table (FPA base).

The FPA base consists of the FPA pages created when the realm is created or when the realm is converted by BPGSIZE. FPA extents may be created when the realm is extended. It does not matter in this case if the realm is extended online or offline mode using the BREORG utility routine.

Every FPA extent is 128 KB in size. In 2 KB databases it consists of 64 pages, in 4 KB databases of 32 pages and in 8 KB databases of 16 pages. FPA extents are managed through the act-key-0 page. The BPRECORDER utility routine can be used to output information on the FPA base and the FPA extents (see the [“Recovery, Information and Reorganization”](#) manual).

Structure of an BASE FPA page with a length of 2048 bytes

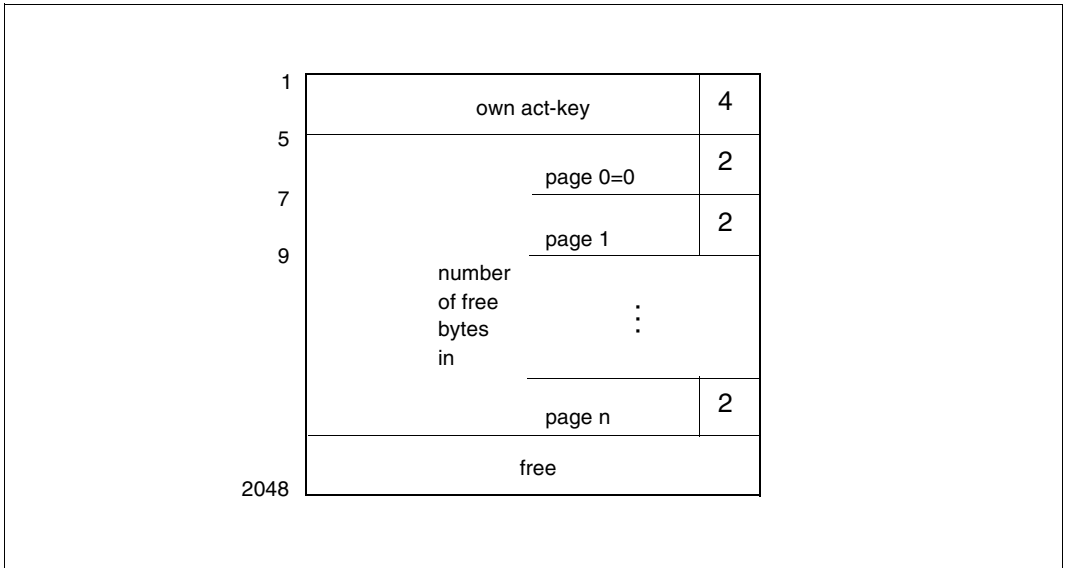


Figure 48: Structure of a FPA base page with a length of 2048 bytes

Meaning of the bytes

Byte	Meaning
1-4	The address of the page is contained in the act-key
5	Starting from this byte there is an entry of 2 bytes for every page of the realm, indicating the number of free bytes contained in the page. This number is 0 for: <ul style="list-style-type: none"> – the act-key-0 page, the act-key-N page – FPA pages – DBTT anchor pages – DBTT pages – CALC pages as these pages are exclusively reserved for a particular purpose and no further data can be allocated to them by the Free Place Administration. The entry for empty pages is 2028 bytes, which can actually be used, 20 bytes being reserved for the page header.

Table 16: Meanings of the bytes of a FPA base page with a length of 2048 bytes

Structure of an FPA extent page with a length of 2048 bytes or of an FPA page with a length of 4000 or 8096 bytes

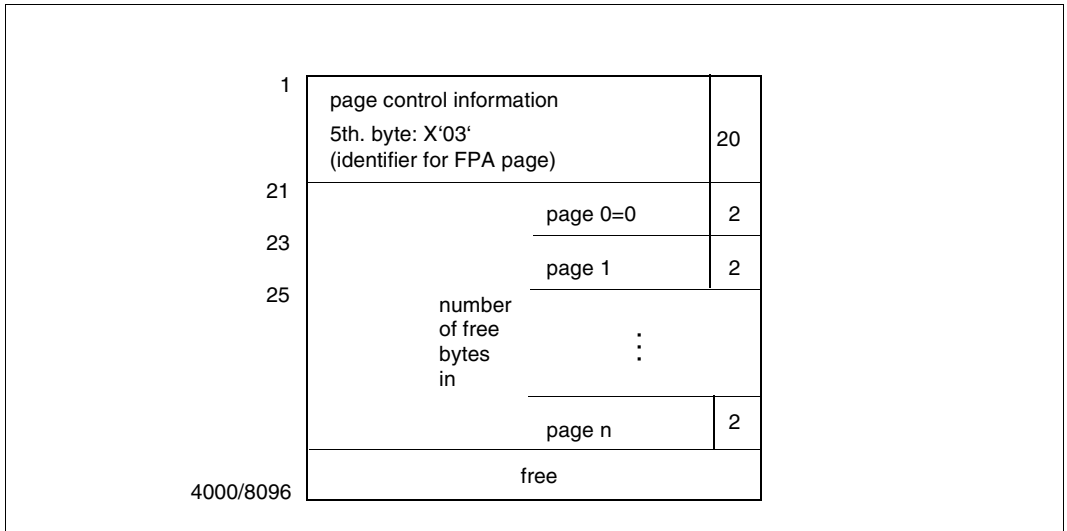


Figure 49: Structure of an FPA extent page with a length of 2048 bytes or of an FPA page with a length of 4000 or 8096 bytes

Meaning of the bytes

Byte	Meaning
1-20	Page control information; type=FPA_PAGE; contains, among other things, the act-key that indicates the page address.
25	Starting with this byte, there is a 2-byte entry for each page of the realm indicating the number of free bytes in the respective page. This is equal to 0 for <ul style="list-style-type: none"> – the act-key-0 page, the act-key-N page, – FPA pages, – DBTT anchor pages – DBTT pages, – CALC pages, since these pages are exclusively reserved for specific purposes and cannot therefore be assigned additional data via the Free Place Administration. The effectively usable storage space for empty pages is indicated as 2048 bytes, 3980 bytes or 8076 bytes, respectively, due to the 20 bytes taken up by the the page header.

Table 17: Meanings of the bytes of an FPA extent page with a length of 2048 bytes or of an FPA page with a length of 4000 or 8096 bytes

7.4 DBTT pages

For each record type, UDS/SQL requires a table - known as the DBTT (Database Key Translation Table) - in which all the currently stored records are administered via a database key. Basically, this is an index in the DBTT which points to the physical address of the record in the database as recorded in the table. The structure of the DBTT anchor pages and the actual DBTT pages are described below.

7.4.1 DBTT anchor page

Only for the SSIA-RECORD there is no DBTT anchor page. For all other record types, the DBTT is administered via DBTT anchor pages. The first DBTT- anchor page is recorded in the SIA and shown in the output of utility routine BPSIA. In order to manage the largest possible DBTT for a record type - in a 4-Kbyte database this means more than 2 billion DB keys - multiple chained DBTT anchor pages may be required. DBTT anchor pages are located in the realm containing the DBTT. They are only set up in the length that is required at the time of creation. In the great majority of cases, no more than one DBTT anchor page is required for each record type. The DBTT itself consists of the DBTT base with a variable number of DBTT pages and possibly DBTT extents each of which consists of 128 consecutive PAM pages.

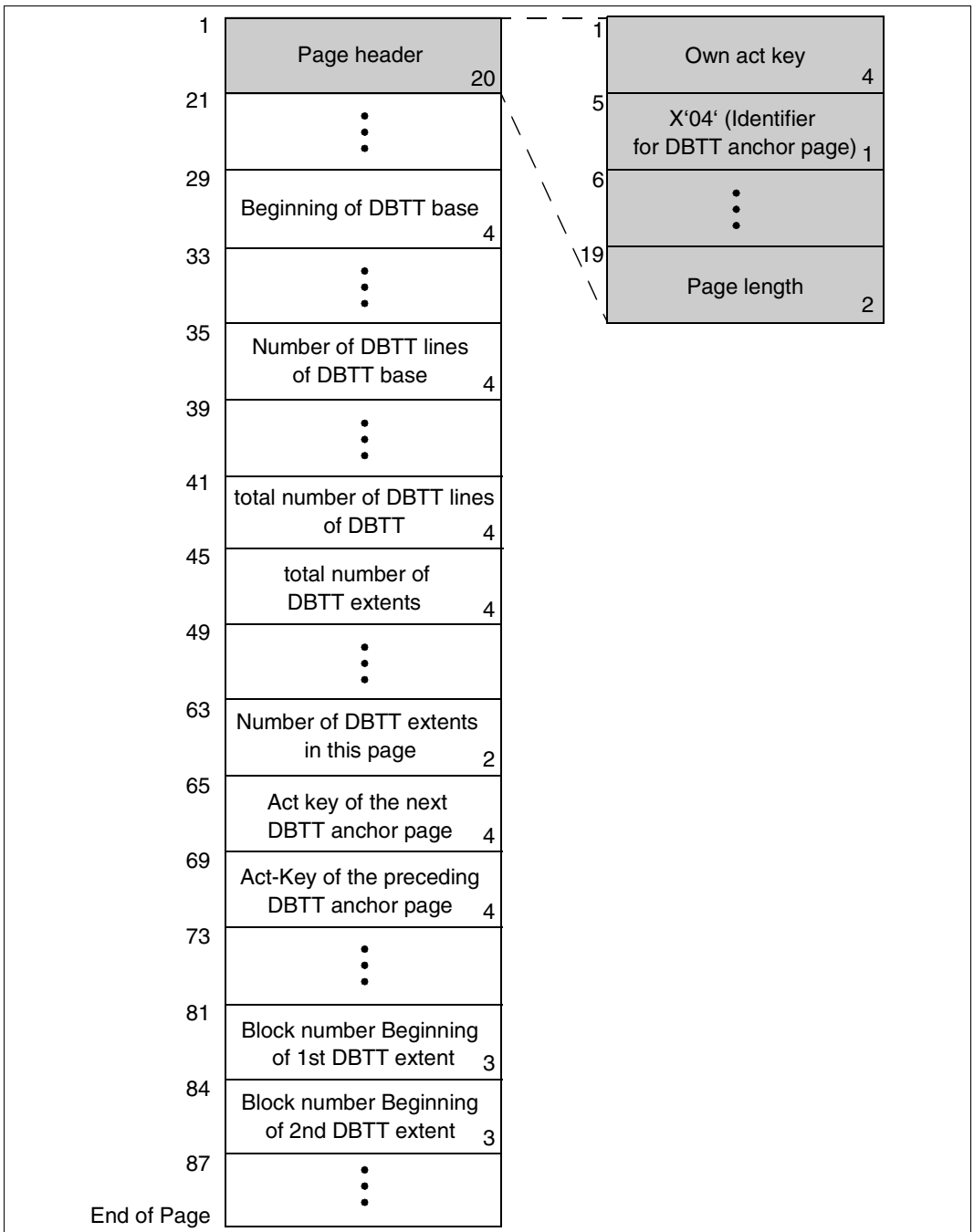


Figure 50: Layout of a DBTT anchor page

7.4.2 DBTT page

This section describes the layout of a DBTT page and the structure of the DBTT entries in terms of lines and columns. There is precisely one DBTT line for each record of a record type.

In the case of record types which are not owner record types, the DBTT consists of only column 0, which contains the act-keys of the records.

Otherwise, one column each is added for every reference to

- a pointer array (MODE IS POINTER-ARRAY),
- a list (MODE IS LIST),
- a sort key table (MODE IS CHAIN, ORDER IS SORTED INDEXED), and
- an indexed set SEARCH key table.

UDS/SQL uses the DBTT line length and the database key value specifying the line in the DBTT, together with the block number of the associated DBTT component, DBTT base or DBTT extent to calculate the location of the associated DBTT line.

Depending on which page length was defined for the database, the length of a DBTT page may be 2048 bytes, 4000 bytes or 8096 bytes.

Structure of a DBTT page with a length of 2048 bytes

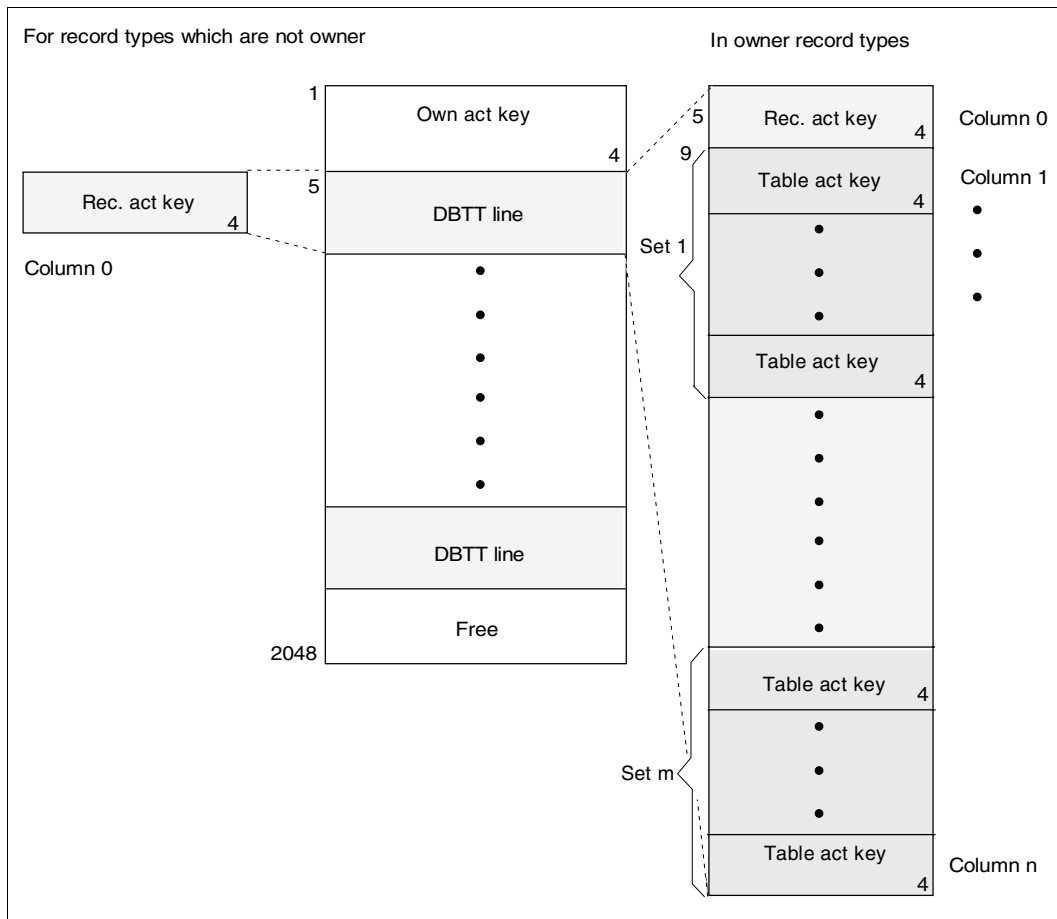


Figure 51: Structure of a DBTT page with a length of 2048 bytes

Structure of a DBTT page with a length of 4000 or 8096 bytes

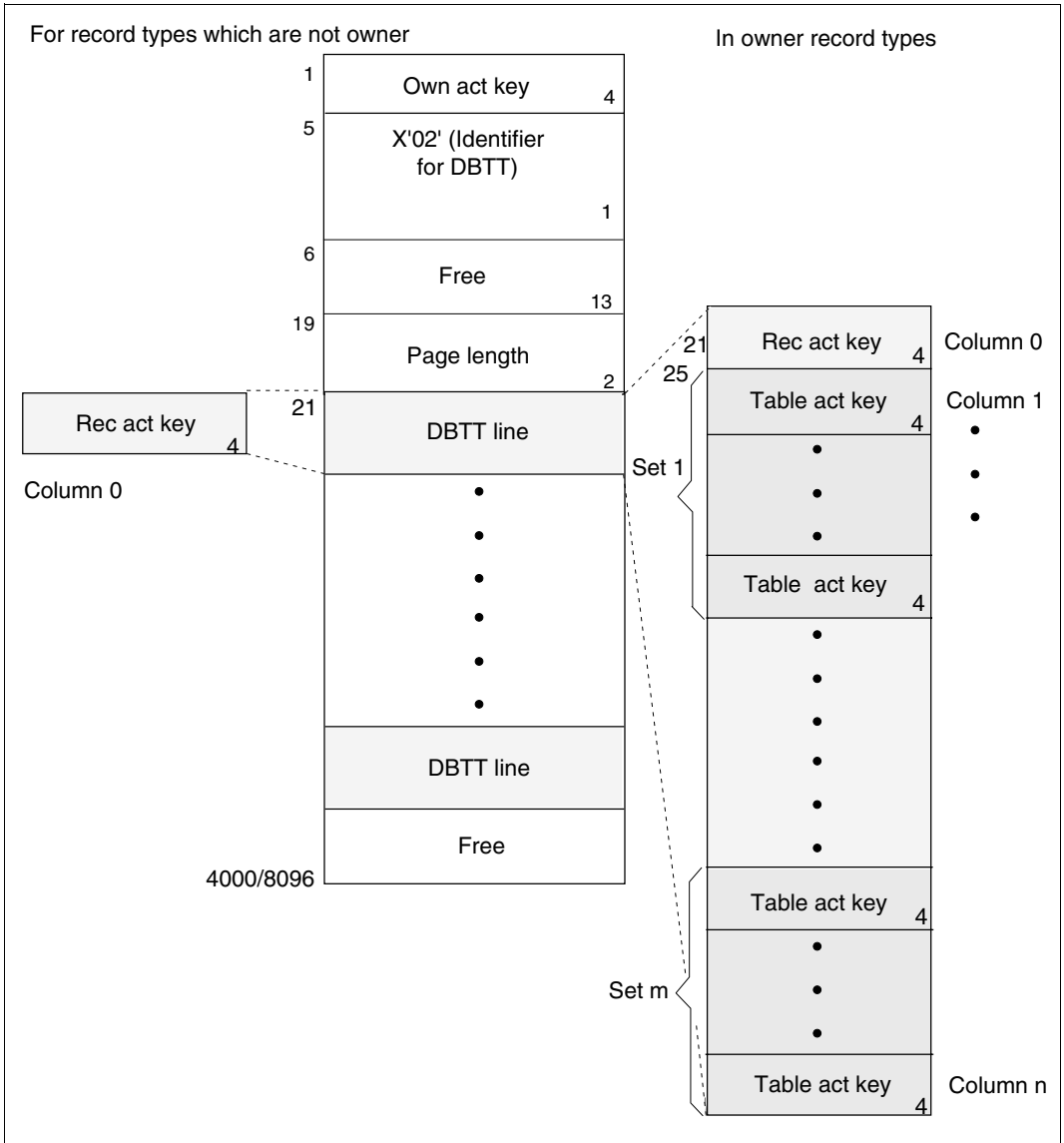


Figure 52: Structure of a DBTT page with a length of 4000 or 8096 bytes

7.5 Direct CALC page

Depending on which page length was defined for the database, the length of a direct CALC page may be 2048 bytes, 4000 bytes or 8096 bytes.

Direct CALC pages with a length of 2048 bytes differ from those consisting of 4000 or 8096 bytes with respect to the lengths of entries for the database key value, record sequence number (RSQ) and page index:

- In the case of direct CALC pages with a length of 2048 bytes, the entry for the database key value occupies 4 bytes; the length of the RSQ entry is 3 bytes, and that of the page index entry is 8 bytes.
- For direct CALC pages with a length of 4000 or 8096 bytes, the entry for the database key value occupies 8 bytes; the length of the RSQ entry is 6 bytes, and that of the page index entry is 12 bytes.

Note that the values given in the form “number1/number2” (e.g. 25/29) in [figure 53](#) must be interpreted as follows:

- “number1” indicates the applicable value for direct CALC pages with a length of 2048 bytes.
- “number2” indicates the applicable value for direct CALC pages with a length of 4000 or 8096 bytes.

All other length and displacement entries are equally applicable to direct CALC pages with a length of 2048 bytes and direct CALC pages with a length of 4000 or 8096 bytes.

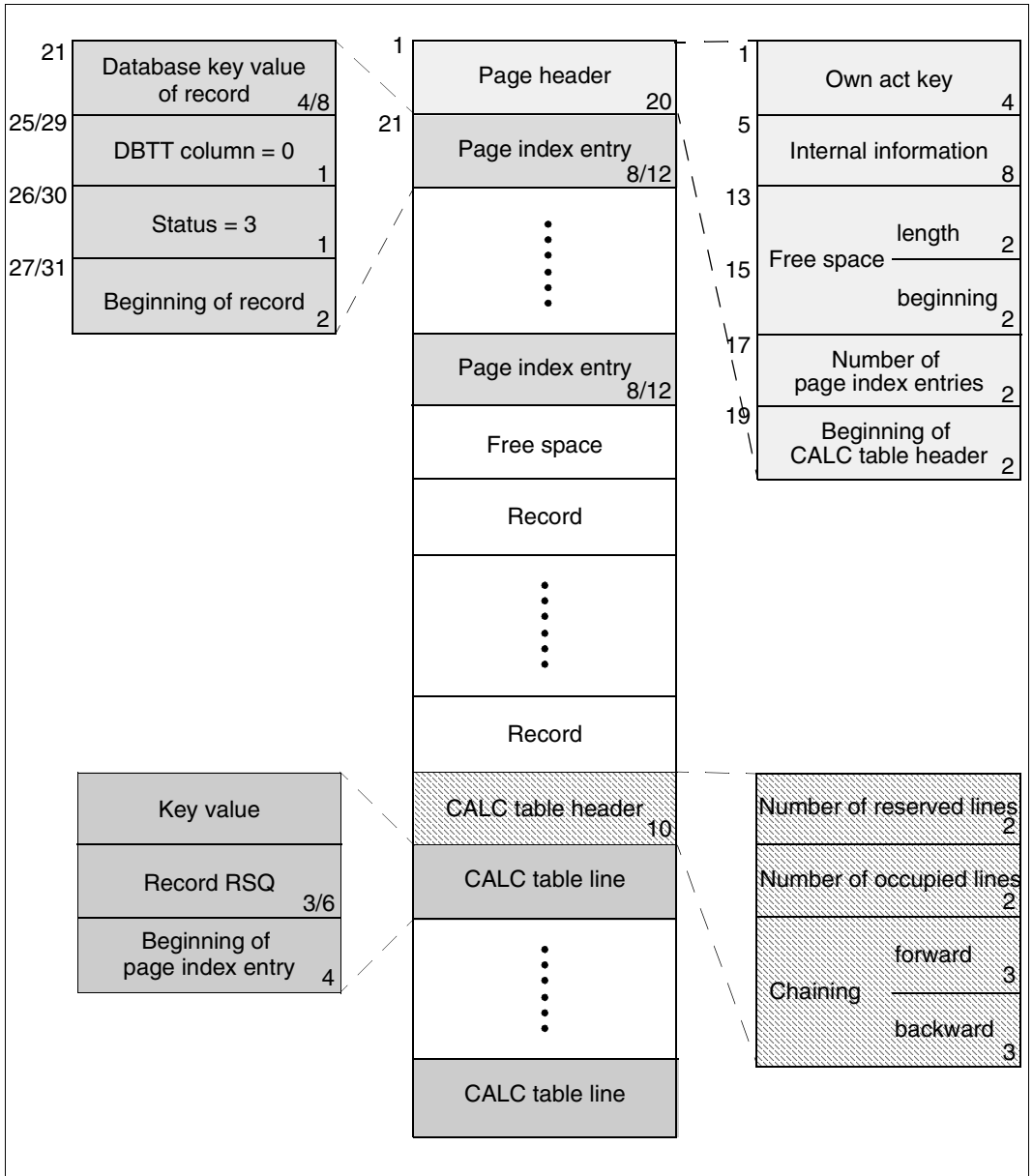


Figure 53: Structure of a direct CALC page

Meanings of bytes 1-28 or 1-32

Byte	Meaning
1-4	specifies the address of the page
13-16	contains the length and the beginning of the free space. Since the free place is filled with records starting from the CALC table header, the first record stored in the page borders on the CALC table header. The free place begins after the last page index entry and ends before the first record. This information on free place represents the second level of the three-level Free Place Administration facility.
17-18	contains the number of page index entries and thus the number of records contained in the page.
19-20	indicates the beginning of the CALC table header.
21-24 or 21-28	contains the database key value of the first record stored in the page.
25 or 29	Since the page index entries in CALC pages always refer to records, the corresponding DBTT column is column 0.
26 or 30	Status=3 identifies the record associated with the page index entry as a CALC record.
27-28 or 31-32	points to the first stored CALC record.

Table 18: Meanings of bytes 1-28 or 1-32 of the CALC page

The “chaining” items are used to link the overflowing pages to their overflow pages.

Based on the record length and the length of the key item, UDS/SQL calculates the maximum number of records that can be stored in a page. Then UDS/SQL sets up a CALC table at the end of the page with the calculated number of table lines, where each line refers to exactly one record. The table entries are sorted in ascending order by key values, and by RSQs for duplicate key values.

There is also a page index entry for each record.

If UDS/SQL knows the key value of a record, it can find the record in the page via the CALC table and the associated page index entry. If UDS/SQL knows the database key value, it locates the record via the page index entry alone.

7.6 Indirect CALC page

Indirect CALC pages are created for a record type stored with LOCATION MODE IS CALC if the record type was simultaneously defined with the SSL as one of the following:

- as a member with MODE IS LIST or
- with PLACEMENT OPTIMIZATION or
- with COMPRESSION FOR ALL ITEMS.

Indirect CALC pages are always set up when SEARCH KEY USING CALC has been specified.

Depending on which page length was defined for the database, the length of an indirect CALC page may be 2048 bytes, 4000 bytes or 8096 bytes.

Indirect CALC pages with a length of 2048 bytes differ from those consisting of 4000 or 8096 bytes with respect to the lengths of entries for the record sequence number (RSQ):

- In the case of indirect CALC pages with a length of 2048 bytes, the RSQ entry is 3 bytes long.
- For indirect CALC pages with a length of 4000 or 8096 bytes, the RSQ entry is 6 bytes long.

The structure of indirect CALC pages differs from that of direct CALC pages in that they contain no records or page index entries. The CALC table line therefore contains the PPP to the associated record. In every other respect, the description of the direct CALC page also applies to the indirect CALC page.

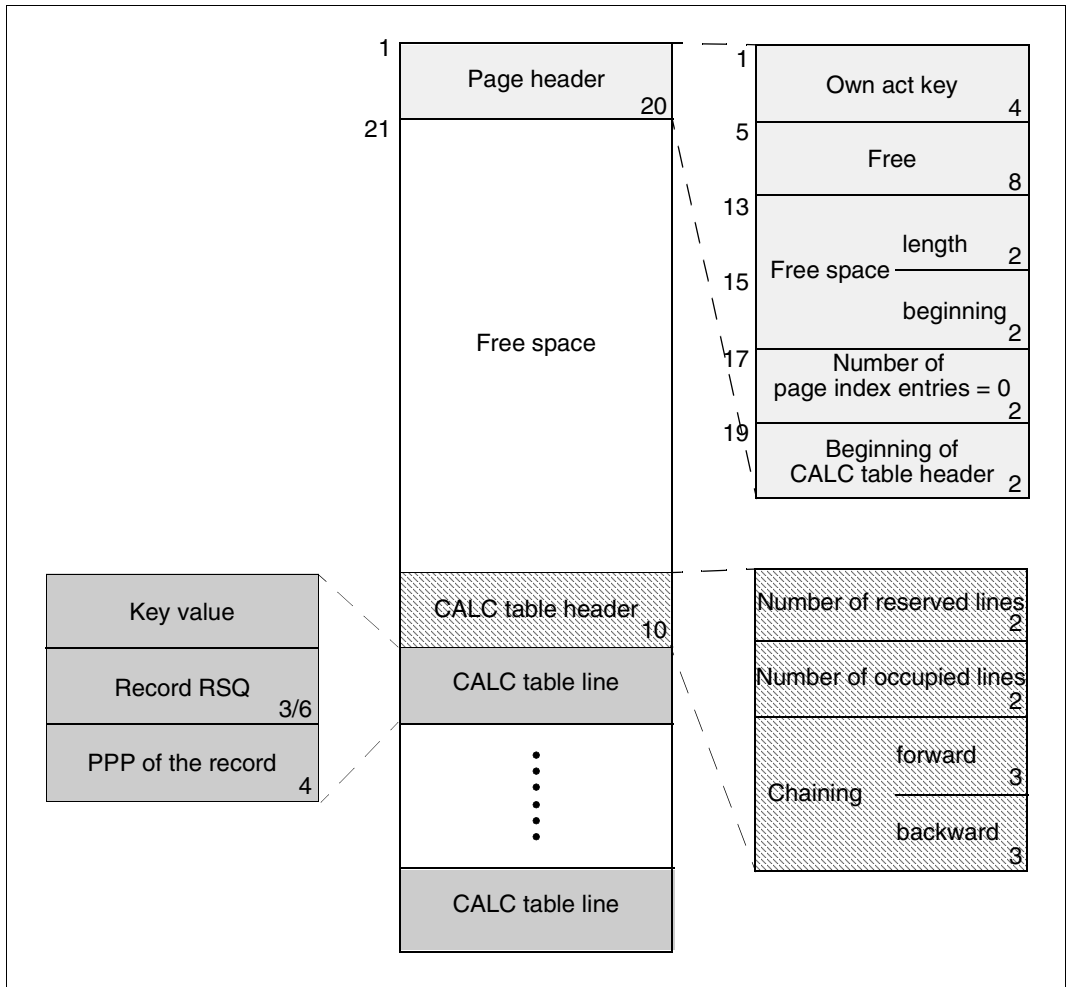


Figure 54: Structure of an indirect CALC page

7.7 Data page

Depending on which page length was defined for the database, the length of a data page may be 2048 bytes, 4000 bytes or 8096 bytes.

Data pages with a length of 2048 bytes differ from those consisting of 4000 or 8096 bytes with respect to the lengths of entries for the database key value and the page index:

- In the case of data pages with a length of 2048 bytes, the entry for the database key value is 4 bytes in length and that of the page index entry is 8 bytes.
- For data pages with a length of 4000 or 8096 bytes, the entry for the database key value is 8 bytes in length and that of the page index entry is 12 bytes.

The following data may be placed in a data page:

- pointer arrays
- lists
- sort key tables
- SEARCH key tables
- records that are not stored in a hash area

The values given in the form “number1/number2” (e.g. 25/29) in [figure 53](#) must be interpreted as follows:

- “number1” indicates the applicable value for data pages with a length of 2048 bytes.
- “number2” indicates the applicable value for data pages with a length of 4000 or 8096 bytes.

All other length and displacement entries are equally applicable to data pages with a length of 2048 bytes and those with a length of 4000 or 8096 bytes.

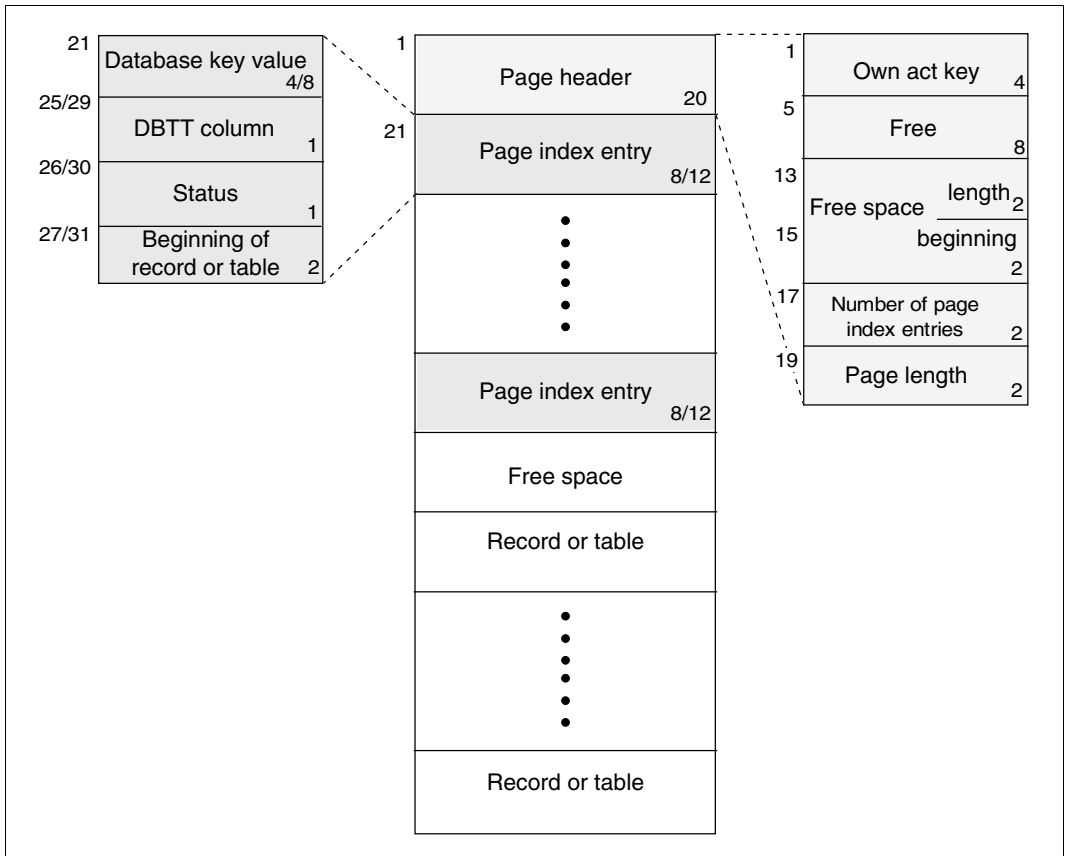


Figure 55: Structure of a data page

Meanings of bytes 1-28 or 1-32

Byte	Meaning	
1-4	contains the address of the page.	
13-16	specifies the length and the beginning of the free space. Since the free space is filled with records and tables starting at the end of the page, the beginning of the free space borders on the last stored record or table. This information on free place represents the second level of the UDS/SQL three-level Free Place Administration facility.	
17-18	contains the number of page index entries.	
19-20	The page length may be 2048 bytes, 4000 bytes or 8096 bytes.	
	Record	Table
21-24 or 21-28	Database key value of the record	Database key value of the associated owner record
25/29	DBTT column=0	DBTT column=1-n
26/30	Status=0: Record is not part of a list (anchor record) Status=2: Record is part of a list	Status=1
27-28 or 31-32	indicates the beginning of the record or table.	

Table 19: Meanings of bytes 1-28 or 1-32 of a data page

There is a page index entry for each record and each table in the page, indicating the position of the record or table. In the case of lists, there is a page index entry for the list itself and for every record contained in it. The logical sequence of records in a list is not based on their physical sequence, but on the sequence of the page index entries.

A data page must not contain two anchor records at the same time.

8 Structure of records and tables

In general, the user requires no information on the structure of records and tables when defining a database by means of DDL and SSL.

This chapter is a reference chapter for users interested in particular details.

8.1 Structure of records

User-defined record

As a rule, the physical structure of a record defined in the schema DDL comprises more than the items specified for this record type. In most cases, set connection data (SCD) is automatically added when a record is stored. The SCD can consist of the following components:

- the pointers UDS/SQL requires for storing a set occurrence as a chain,
- the pointer from the owner to its associated pointer array or list,
- the pointer from the member to the associated owner, and
- an indication if the record is part of a SYSTEM set.

If a variable item or compression has not been defined for a record type, its records are stored in the format below:

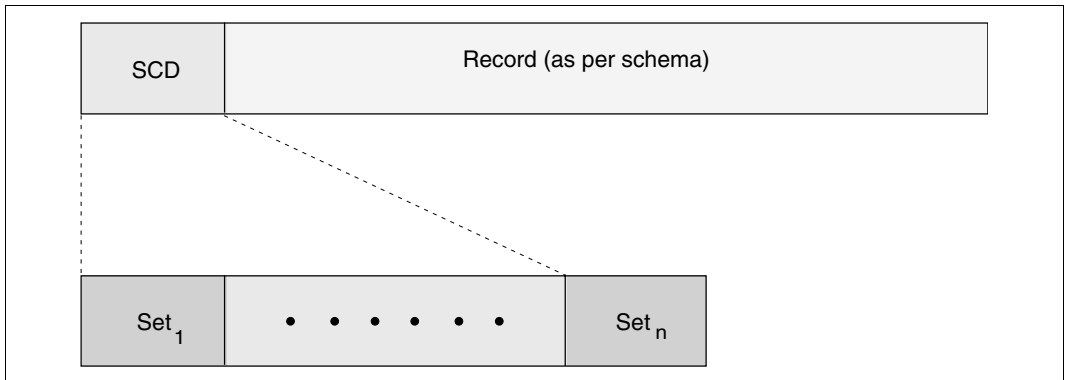


Figure 56: Standard format of a user-defined record

The following format is used to store the records of a record type if a variable item or compression has been specified for it:

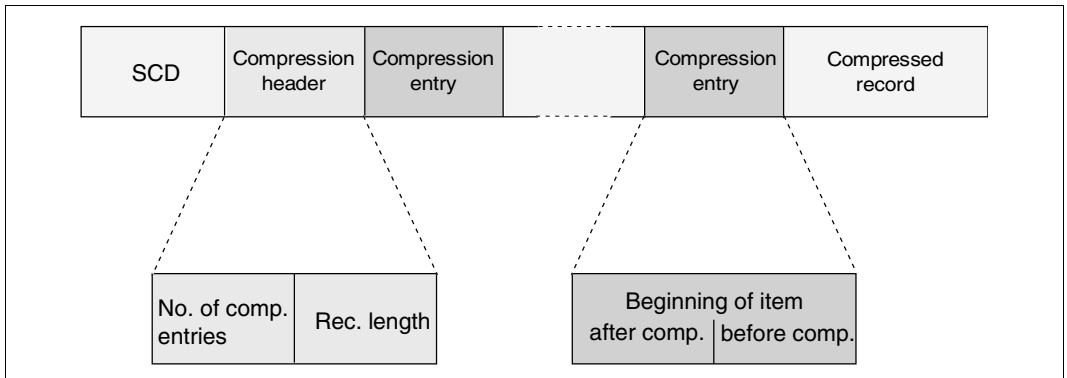


Figure 57: Compressed format of a user-defined record

Anchor record

In addition to the records defined in the schema DDL, UDS/SQL automatically generates one anchor record for each SYSTEM set. In SYSTEM sets, the anchor record assumes the function that the DBTT of an owner record type performs in a normal set. The anchor record contains its own act-key as well as the act-keys of the tables belonging to the SYSTEM set. It is extended by SCD if the records of the SYSTEM set are stored as a chain.

Set connection data

When a record is stored, UDS/SQL automatically adds set connection data (SCD) if the record has to be connected with other records or tables. [table 20](#) shows what the SCD consists of in each case.

The length of the SCD may vary, depending on which page length was defined for the database. The effect of the database page length on that of the SCD is indicated in [table 20](#) in the column “SCD length in bytes” by means of 2 values (e.g. 8/12):

- The first value shows the SCD length for databases with a page length of 2048 bytes.
- The second value shows the SCD length for databases with a page length of 4000 or 8096 bytes.

MODE IS	RECORD TYPE	OWNER IS SYSTEM	ORDER IS LAST	LINKED TO PRIOR	MEMBER LINKED	PHYSICAL LINK	SCD length in bytes	SCD content
CHAIN	Owner	Y/N	N	N	-	-	8/12	1
			Y	N			16/24	1,2
	Y/N	Y	Y	16/24	1,2			
Member	Y		N	-	-	8/12	3	
	N		Y	-	-	16/24	3,4	
POINTER-ARRAY or LIST	Owner	Y		N	N	-	11/18	3,5
		N		N	Y	-	15/22	3,5,6
	Y		Y	N	-	19/30	3,4,5	
	N		Y	Y	-	23/34	3,4,5,6	
Member	Y		-	-	-	0	-	
	N		-	-	N	0	7	
		Y		-	-	4	8	
		N		-	N	1	8	
				-	Y	3/6	5	
						7/10	5,6	

Table 20: Overview of the possible SCD combinations per set

- 1 Database key value + PPP of the first member record ... 8/12 bytes
- 2 Database key value + PPP of the last member record ... 8/12 bytes
- 3 Database key value + PPP of the following record ... 8/12 bytes
- 4 Database key value + PPP of the preceding record ... 8/12 bytes
- 5 RSQ of the owner ... 3/6 bytes
- 6 PPP of the owner ... 4 bytes
- 7 Act-key of the beginning of the table ... 4 bytes
- 8 Indicator for SYSTEM set ... 1 byte
 This byte contains X'00' if the member record belongs to the set and X'FF' if it is not part of the set.
 In the case of "MODE IS CHAIN", the byte is redefined as shown in point 3 (see above).

8.2 Structure of tables

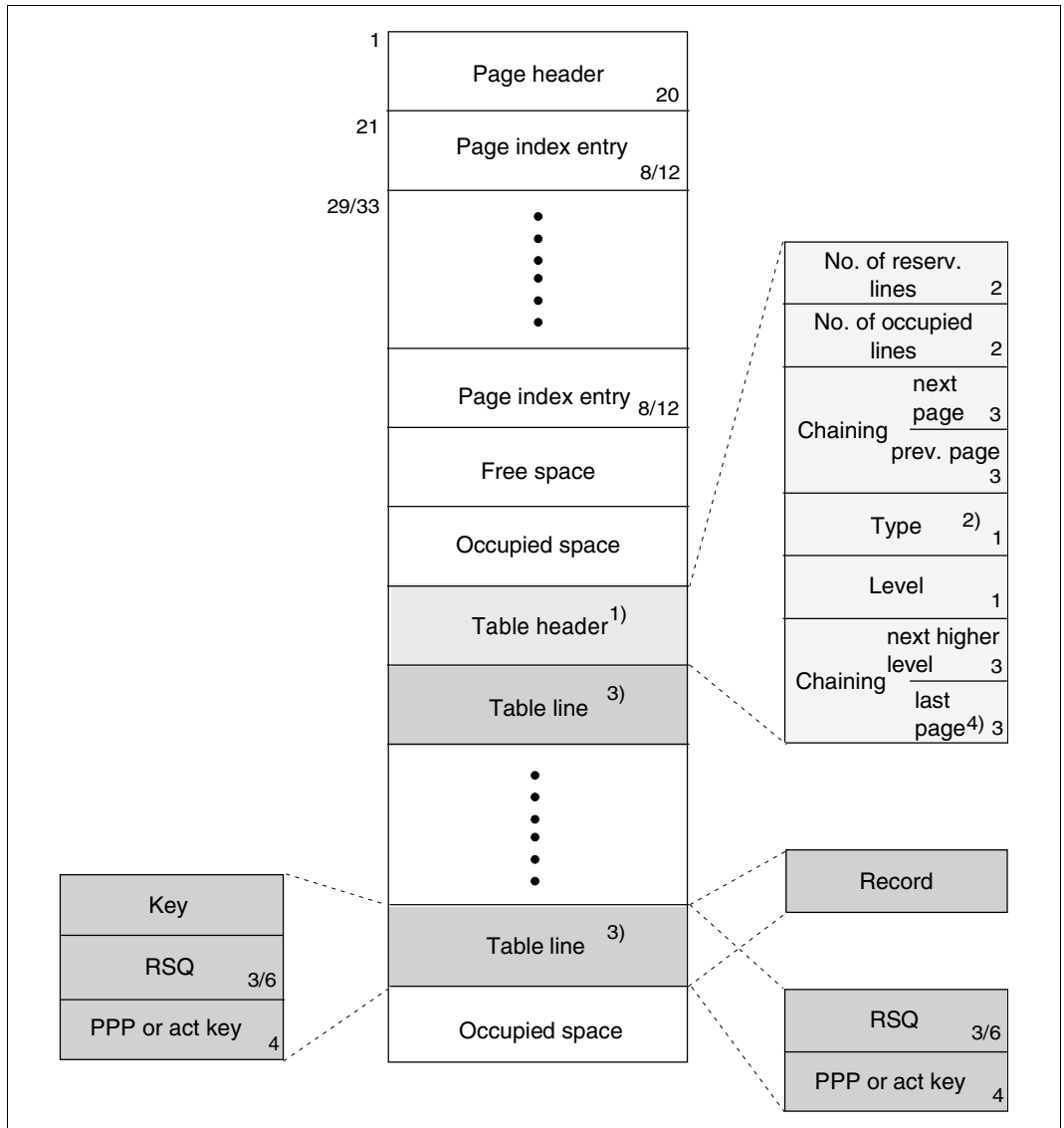


Figure 58: Structure of tables

Explanation of [figure 58](#):

- 1) follows the last table line in the case of lists.
- 2) bit $2^7=1$: list
 $2^6=1$: multi-level table
 $2^5=1$: table ATTACHED TO OWNER
 $2^4=1$: duplicates table
- 3) does not apply to duplicates tables (see [figure 60](#) and [figure 61](#)).
- 4) applies to the highest table page of a multi-level table or to the first table page of a single-level table only.

The length of the entries for the record sequence number (RSQ) and page index depend on the page length that was defined for the database:

- In tables of a database with a 2048-byte page length, the RSQ entry is 3 bytes long, and the page index entry is 8 bytes.
- In tables of a database with a 4000-byte or 8096-byte page length, the RSQ entry is 6 bytes long, and the page index entry is 12 bytes.

Pointer arrays, lists, sort key tables and standard SEARCH key tables are structured according to the principle shown in [figure 58](#). (The representation of the table line does not apply to duplicates tables.) They are always stored in data pages. If a table does not fill a page completely, the remaining space can be used for records or further tables. The table headers contain the space reservations specified in the POPULATION clauses. This represents the third level of the three-level UDS/SQL Free Place Administration facility. Furthermore, the table header contains pointers to other table parts if the table exceeds one page and a pointer to the next higher table level if the page belongs to a lower level. Two further indicators show the table level of the page itself and the type of the table.

The following overview shows which of the three available table lines corresponds to which table type:

DDL/SSL clauses	Table type	Content of a table line:		Name of the table
		lowest level	higher level	
MODE IS POINTER-ARRAY ORDER IS LAST/ FIRST/NEXT/PRIOR	single-level	RSQ, PPP	-	single-level pointer array
MODE IS POINTER-ARRAY ORDER IS SORTED INDEXED BY DATABASE-KEY	multi-level	RSQ, PPP	RSQ, act-key	multi-level pointer array
MODE IS POINTER-ARRAY ORDER IS SORTED INDEXED BY DEFINED KEYS ASCENDING/ DESCENDING KEY IS...	multi-level	key value, RSQ, PPP	key value, RSQ, act-key	
MODE IS LIST ORDER IS LAST/ FIRST/NEXT/PRIOR	single-level	member record	-	single-level list
MODE IS LIST ORDER IS SORTED INDEXED BY DATABASE-KEY	multi-level	member record	RSQ, act-key	multi-level pointer array
MODE IS LIST ORDER IS SORTED INDEXED BY DEFINED KEYS ASCENDING/ DESCENDING KEY IS...	multi-level	member record	key value, RSQ, act-key	

Table 21: Overview of the table types

(part 1 of 2)

DDL/SSL clauses	Table type	Content of a table line:		Name of the table
		lowest level	higher level	
MODE IS CHAIN ORDER IS SORTED INDEXED BY DATABASE-KEY	multi-level	RSQ, PPP	RSQ, act-key	sort key table
MODE IS CHAIN ORDER IS SORTED INDEXED BY DEFINED KEYS ASCENDING/ DESCENDING KEY IS...	multi-level	key value, RSQ, PPP	key value, RSQ, act-key	
SEARCH KEY IS... USING INDEX TYPE IS REPEATED-KEY (on set or record type level)	multi-level	key value, RSQ, PPP	key value, RSQ, act-key	SEARCH key table
SEARCH KEY IS... USING INDEX TYPE IS DATABASE- KEY-LIST	multi-level	see figure 60 and figure 61	key value, ACT_KEY	duplicates table

Table 21: Overview of the table types

(part 2 of 2)

The table lines of a higher table level contain act-keys pointing to the associated pages of the next lower table level.

The table lines of the lowest table level contain PPPs which indicate the pages where the associated records are stored. If a PPP is not current, UDS/SQL finds the record by means of the RSQ via the DBTT.

The hierarchic structure of a multi-level table is shown in [Figure 59](#) below. The illustrated example applies to pointer arrays, sort key tables and SEARCH key tables. In the case of a list, level 0 contains the records themselves.

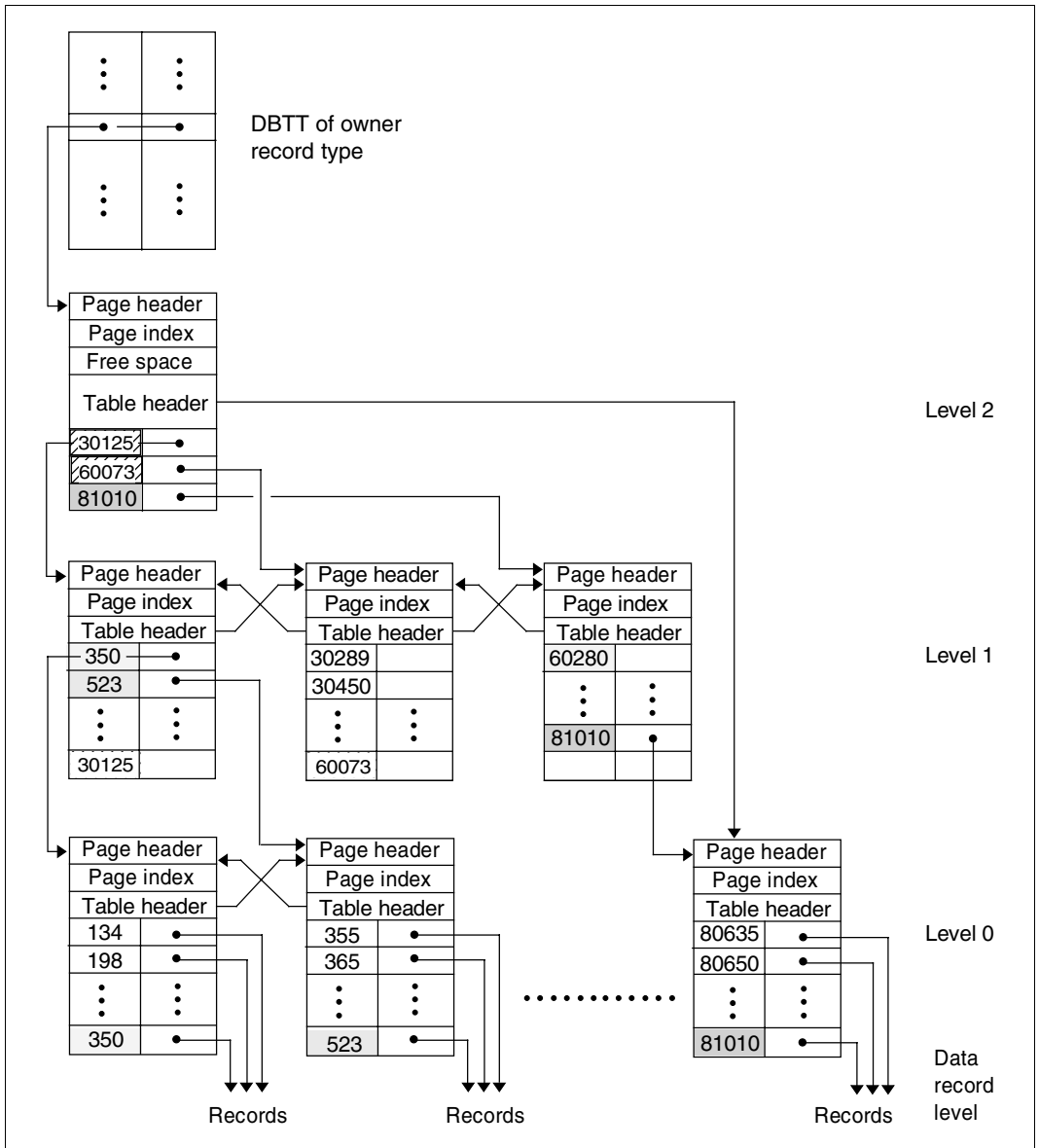


Figure 59: Hierarchic structure of a multi-level table

Duplicates table

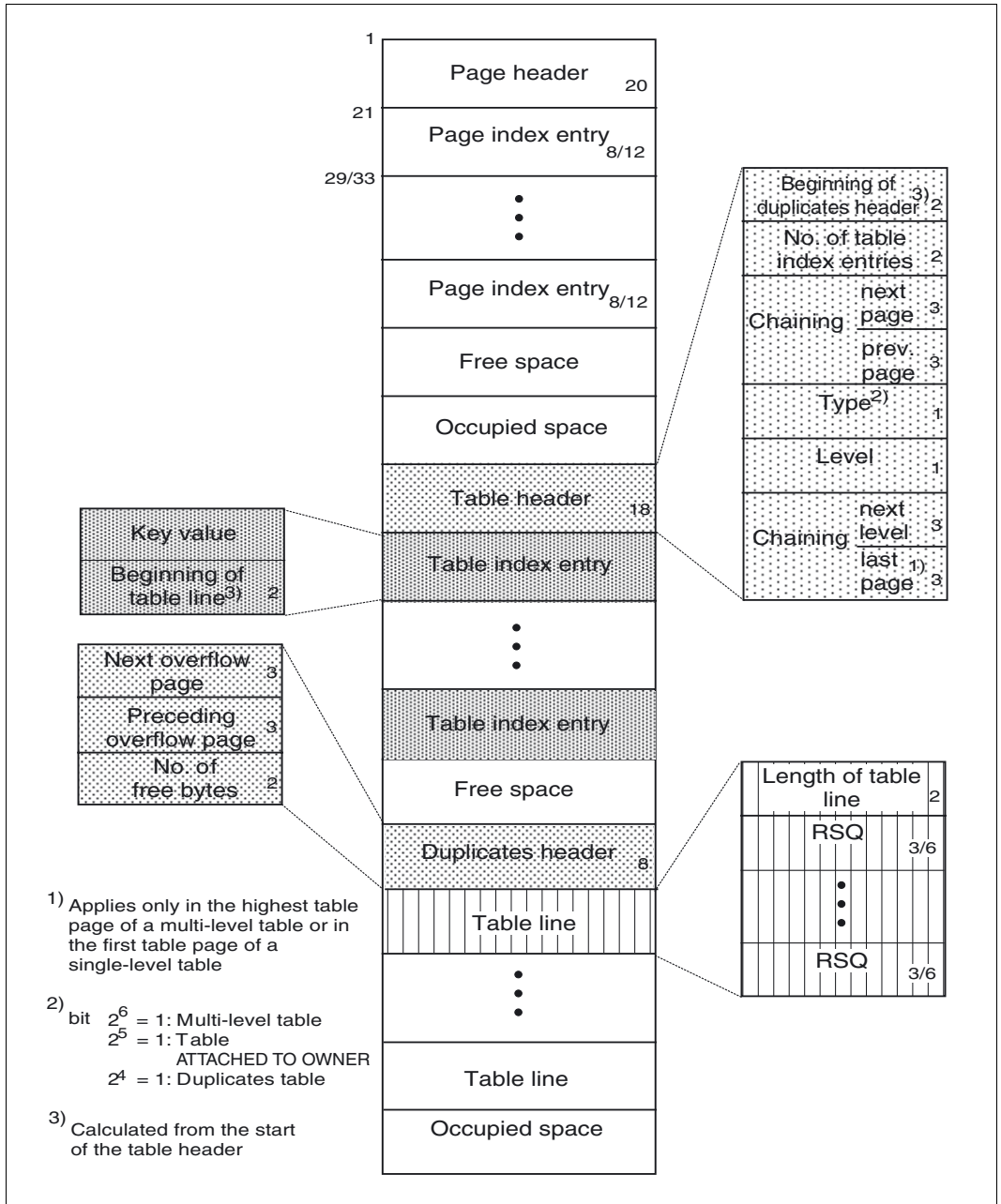


Figure 60: Structure of a duplicates table

The lengths of the entries for the database key value, the record sequence number (RSQ), and for the page index entries depend on the page length that was defined for the database:

- In tables of a database with a 2048-byte page length, the entry for the database key value is 4 bytes long; the RSQ entry is 3 bytes, and the page index entry is 8 bytes.
- In tables of a database with a 4000-byte or 8096-byte page length, the entry for the database key value is 8 bytes long; the RSQ entry is 6bytes, and the page index entry is 12 bytes.

Overflow page of a duplicates table

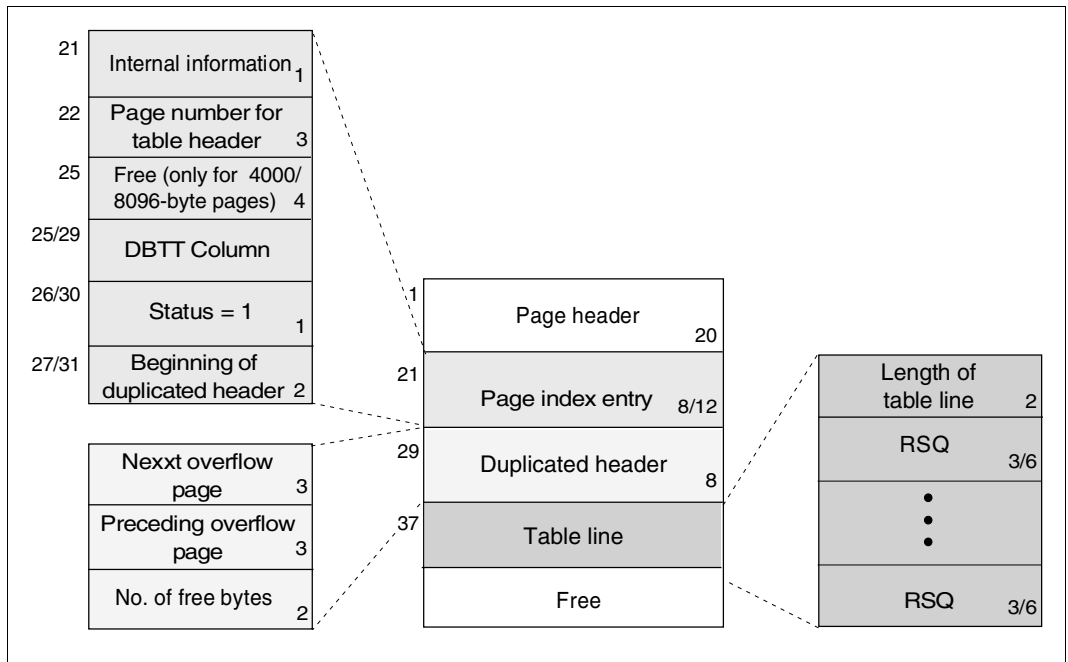


Figure 61: Overflow page for a duplicates table

Depending on which page length was defined for a database, the length of an overflow page may be 2048 bytes, 4000 bytes or 8096 bytes.

The following applies with respect to the lengths of entries for the record sequence number (RSQ) and page index:

- In a 2048-byte overflow page, the RSQ entry is 3 bytes long, and the page index entry is 8 bytes.
- In an overflow page of 4000 or 8096 bytes, the RSQ entry comprises 6 bytes, and the page index entry is 12 bytes.

A duplicates table is a special type of SEARCH key table in which key values that occur several times are represented only once. When a part of a duplicates table that contains only one key value is to be extended to cover more than one page, UDS/SQL creates an overflow page, in which the table line related to the key is continued. The connection to the overflow page is not established via the table header but via the duplicates header.

9 Reference section

The previous chapters cover the functions and applications of the schema DDL, schema SSL and subschema DDL clauses.

This chapter deals with the syntax rules you must observe in order to use the respective language correctly.

The notational conventions are described on [page 22](#).

General syntax rules

variable

must be replaced by a current value when applying the format. Four categories of variables can be distinguished:

Variable	Current value
<i>schema-name</i> <i>subschema-name</i> <i>realm-name</i> <i>set-name</i> <i>record-name</i> <i>record-element-name</i> <i>group-item-name</i> <i>vector-name</i> <i>item-name</i> <i>identifier</i> <i>hashroutine</i>	<p>must begin with a letter and may consist of up to 30 characters. The characters used may include letters, digits and hyphens. One hyphen must not follow another it must not be the last character. The current value must not be identical to optional word or keywords.</p> <p><i>record-element-name</i>, <i>group-item-name</i>, <i>vector-name</i> and <i>item-name</i> must be unique within the record type only.</p> <p>By specifying</p> <p>{ <u>IN</u> } <i>record-name</i> { <u>OF</u> }</p> <p>the user can also refer to names which have already been assigned.</p> <p>All other names must be unique within the entire database.</p> <p>The first 6 characters of <i>subschema-name</i> must be unique within a database configuration.</p>
<i>literal</i>	<p>must be enclosed in quotation marks (if QUOTE IS SINGLE, in apostrophes; see the "Creation and Restructuring" manual, Schema DDL compilation). They are not part of the value of the literal.</p>
<i>integer</i>	<p>consists of up to 15 digits. The minus sign is also permitted in the TYPE clause (see page 236, TYPE clause).</p>
<i>mask-string</i>	<p>see page "Defining an item" on page 55</p>

Comment

is indicated by * in column 7. The text in columns 8 through 72 is then recognized as a comment by UDS/SQL.

Semicolon

may optionally be used as separator between clauses.

Page feed

is indicated by / in column 7.

Continuation line

Entries exceeding column 72 can be continued in a new lines line that must start with a hyphen in column 7.

Uppercase

The COBOL compiler accepts uppercase letters only.

Column conventions

Each of the three languages is made up of clauses.

Clauses are generally written starting at column 12.

Entries starting at column 8 are:

- the first clause of an entry,
- the MEMBER clause;

and in the subschema DDL

- the first line of a division,
- the first line of a section,
- level number 01.

The syntax description of each language starts with an overview of the entries and their clauses.

Reserved words

For the list of reserved words, see the reserved word list of the COBOL version that you are using in the manual "[COBOL2000 \(BS2000/OSD\)](#) - Reference manual".

9.1 Schema DDL syntax

Schema entry	{	SCHEMA NAME clause
	{	[PRIVACY LOCK clause]_
Realm entry	{	AREA NAME clause
	{	[TEMPORARY clause]_
Record entry	{	RECORD NAME clause
	{	[LOCATION MODE clause]
	{	WITHIN clause
	{	[SEARCH KEY clause]_
	{	record element name clause
	{	[PICTURE clause]
	{	[TYPE clause]
	{	[OCCURS clause]_
Set entry	{	SET NAME clause
	{	[DYNAMIC clause]
	{	ORDER clause
	{	OWNER clause_
	{	[MEMBER clause]
	{	[ASCENDING/DESCENDING-KEY clause]
	{	[SEARCH KEY clause]
	{	[SET OCCURRENCE SELECTION clause]]_

Figure 62: Structure of schema DDL

The description of the logical data structure should always be started with the schema entry and at least one realm entry.

The following applies for the subsequent realm, record and set entries:

- The two associated record types must be defined before a set can be defined.
- All the realms mentioned in the WITHIN clause for the record must be defined before a record type can be defined.

9.1.1 Schema entry

```
SCHEMA NAME IS schema-name  
[PRIVACY LOCK FOR COPY IS literal-1[ OR literal-2]]_
```

literal-1,-2

may consist of up to 10 characters.

The schema entry is used to assign a name to the schema. Passwords can be specified to prevent unauthorized creation of subschemas from the schema.

9.1.2 Realm entry

```
AREA NAME IS realm-name  
[AREA IS TEMPORARY]]_
```

The realm entry is used to assign a name to a realm and, if necessary, to define it as a temporary realm.



A maximum of 123 realms may be defined in a database with a page length of 2048 bytes, and a maximum of 245 realms may be defined in databases with a page length of 4000 or 8096 bytes.

Only *one* temporary realm may be defined.

9.1.3 Record entry

RECORD NAME IS *record-name*

[LOCATION MODE IS { DIRECT } { *item-name-1* { IN } *record-name* }
 { DIRECT-LONG } { *identifier-1* }]
 [CALC[*hash-routine*] USING *item-name-2*,...
DUPLICATES ARE[NOT] ALLOWED]

WITHIN *realm-name-1*[,*realm-name-2*,... AREA-ID IS *identifier-2*]

[SEARCH KEY IS *item-name-3*,...USING { CALC[*hash-routine*] } [NAME IS *name*]
 { INDEX }]

DUPLICATES ARE[NOT] ALLOWED]....

{ [*level-number*] *record-element-name*

{ PICTURE IS { *mask-string* }
 { LX(*integer-1*) DEPENDING ON *item-name-4* } }
 { TYPE IS { FIXED REAL { BINARY[{ 15 }] }
 { DECIMAL[*integer-2*[,*integer-3*]] } }
CHARACTER[*integer-4*[DEPENDING ON *item-name-5*]]
DATABASE-KEY
DATABASE-KEY-LONG }]

[OCCURS *integer-5* TIMES]...}

The record entry is used to assign a name to a record type. At the same time, it can be used to define:

- the allocation of records to realms,
- the sequence of records for sequential processing,
- additional access paths for direct access via primary and secondary keys,
- all record elements to be included in the record type.



A maximum of 253 record types may be defined in a database with a page length of 2048 bytes, and a maximum of 32 766 record types may be defined in databases with a page length of 4000 or 8096 bytes.

The individual clauses of the record entry are explained below.

RECORD NAME IS *record-name*

A name is assigned to a record type.

$$\left[\text{LOCATION MODE IS } \left\{ \begin{array}{l} \left\{ \begin{array}{l} \text{DIRECT} \\ \text{DIRECT-LONG} \end{array} \right\} \left\{ \begin{array}{l} \text{item-name-1} \left\{ \begin{array}{l} \text{IN} \\ \text{OF} \end{array} \right\} \text{record-name} \\ \text{identifier} \end{array} \right\} \end{array} \right\} \right. \\ \left. \begin{array}{l} \text{CALC[has h-routine] USING item-name-2, ...} \\ \text{DUPLICATES ARE[NOT] ALLOWED} \end{array} \right\} \right]$$

item-name-1

must designate a database key item.

If you specify LOCATION MODE IS DIRECT, you must define *item-name-1* as a DATABASE-KEY item.

If you specify LOCATION MODE IS DIRECT-LONG, you must define *item-name-1* as a DATABASE-KEY-LONG item.

item-name-2

must specify an item of fixed length belonging to the record type.

You use LOCATION MODE IS DIRECT/DIRECT-LONG to enable you to assign the database key of a record which is to be saved and to specify the sequence for sequential processing.

You use LOCATION MODE IS CALC to specify a primary key to permit direct access to a particular record or to a set of records with the same key values.

WITHIN *realm-name-1*[,*realm-name-2*,... AREA-ID IS *identifier*]

realm-name-1,-2,...

must not be temporary realms.

The records of the record type are allocated to certain realms.

[SEARCH KEY IS *item-name*,... USING {CALC[*hash-routine*]} [NAME IS *name*]
INDEX]
DUPLICATES ARE [NOT] ALLOWED]...]

item-name

specifies an item of fixed length belonging to the record type.

name specifies tables for SEARCH keys; referred to in the SSL statements.

Additional direct access paths via secondary key are specified and a name is assigned to the SEARCH KEY table or the hash area, which can be referenced in the SSL.

[*level-number*] *record-element-name*

level-number

must be an integer between 1 and 99.

Default value: 1

A name is assigned to a record element and optionally a level number can be defined.



The total length of all record elements of one record type must not exceed the maximum record length.

The maximum record length is equal to:

- 2020 bytes in a database with a page length of 2048 bytes (2-Kbyte format)
- 3968 bytes in a database with a page length of 4000 bytes (4-Kbyte format)
- 8064 bytes in a database with a page length of 8096 bytes (8-Kbyte format)

Depending on the length of the SCD, the maximum record length may also be somewhat lower.

```

PICTURE IS { mask-string
                LX( integer-1) DEPENDING ON item-name }

```

mask-string

may consist of the following symbols:

Symbol	Syntax rule
S	can be specified once only at the beginning of the mask string.
X	at least one of these symbols must be entered. Each can be specified more than once or followed by a repetition symbol. 9 may not be placed to the left of A or X. N may not be combined with other characters.
A	
9	
N	
V	cannot be specified in combination with X, N or A.
P	cannot be specified in combination with X, N or A. P can be specified more than once or followed by a repetition symbol. P may be specified either to the left or right of 9, but not both at the same time.
(<i>integer</i>)	may be added after X, N, A, 9 or P.

Table 22: Mask string

The mask string may define an item length of up to 255 bytes. This generally corresponds to the number of characters. However, you may only repeat the symbol N up to 127 times because in this case a character occupies 2 bytes. 9 may be repeated up to 18 times.

integer-1

must be greater than 0. The maximum value depends on the record structure.

item-name

must refer to an item that has just been defined with TYPE IS BINARY 15.

The PICTURE IS clause is used to define unpacked numeric items or alphanumeric items of fixed or variable length or national items.

```

TYPE IS {
  FIXED REAL {
    BINARY [ { 15 } ]
    DECIMAL [ integer-1 [ , integer-2 ] ]
  }
  CHARACTER [ integer-3 [ DEPENDING ON item-name ] ]
  DATABASE-KEY
  DATABASE-KEY-LONG
}

```

BINARY

If no number is specified, the default value used by UDS/SQL is 15.

integer-1

must be an integer between 1 and 18.

Default value: 18

integer-2

must not be greater than 18 and not less than {integer-1} - 18.

Default value: 0.

integer-3

Unless DEPENDING ON is used, *integer-3* must be between 1 and 255. Otherwise, the maximum value depends on the record structure.

item-name

specifies the item which has just been defined with TYPE IS BINARY 15.

This clause is used to define

- packed numeric items,
- binary items,
- alphanumeric items of fixed or variable length, or
- database key items

[OCCURS *integer* TIMES]

integer

must be greater than 0. The maximum value depends on the record structure.

A repetition factor is defined for a vector or a repeating group.

9.1.4 Set entry

```

SET NAME IS set-name
  [SET IS DYNAMIC]
  ORDER IS {
    LAST
    FIRST
    NEXT
    PRIOR
    IMMATERIAL
    SORTED[ INDEXED[ NAME IS name]]
    BY { DATABASE-KEY
        { DEFINED KEYS DUPLICATES ARE[ NOT] ALLOWED } }
  }
  OWNER IS { record-name
             SYSTEM }
  [MEMBER IS record-name { MANDATORY } { AUTOMATIC }
                          { OPTIONAL } { MANUAL } ]
  [ { ASCENDING }
    { DESCENDING } KEY IS item-name-1,...]
  [SEARCH KEY IS item-name-2,... USING { CALC[ hash-routine]
                                             { INDEX } ]
    [NAME IS name]
    DUPLICATES ARE[ NOT] ALLOWED]...
  [SET OCCURRENCE SELECTION IS
    THRU { CURRENT OF SET
          { LOCATION MODE OF OWNER[ ALIAS FOR { item-name-3
                                                  { identifier-1 } } ] } ]
          IS identifier-2]...
  
```

This clause is used to assign a name to a set and to

- declare the set a dynamic set if required,
- define the sequence of the member records within the set occurrences for sequential processing,
- define additional access paths via primary and secondary keys,
- declare a record type to be the owner record type of the set,
- declare a record type to be a member record type of the set if required, and define the type of membership of member records in a set, and
- specify the selection option for the set occurrences.



A maximum of 32 766 sets can be defined per database.

For each record type which is owner of a set you can generate a maximum of 255 tables in these sets. A table is created when the set mode pointer array or list or chain is of the type sorted indexed, also for each secondary key in these sets.

Irrespective of this you may define up to 255 secondary keys per record type on record type level and per singular set on set level; hash routines are not counted here.

The individual clauses of the set entry are explained below.

SET NAME IS *set-name*

A name is assigned to the set.

[SET IS DYNAMIC]

The set is declared a dynamic set.

```

ORDER IS {
  LAST
  FIRST
  NEXT
  PRIOR
  IMMATERIAL
  SORTED[ INDEXED[ NAME IS name]]
  BY { DATABASE-KEY
      { DEFINED KEYS DUPLICATES ARE[ NOT] ALLOWED }
    }
}

```

This clause is used to define

- the sequence of the records within the set occurrences for sequential processing.
 - an additional direct access path via the primary key.
-

```

OWNER IS { record-name
          { SYSTEM }
}

```

A record type defined by the user or a symbolic record type SYSTEM is declared owner record type of the set.

```

MEMBER IS record-name { MANDATORY } { AUTOMATIC }
                     { OPTIONAL }   { MANUAL }

```

A description of the member record type is not required for dynamic sets. In all other cases, the above clause is used to declare a record type member record type and to specify the type of membership of the member records in the set.

```
[ { ASCENDING }
  { DESCENDING } KEY IS item-name,... ]
```

item-name,...

denotes an item of fixed length that belongs to the record type.

This clause is used to define an item or a combination of items of the member record type as sort key. The member records within the set occurrence are sorted in ascending or descending order, according to the values of this key.

```
[ SEARCH KEY IS item-name,... USING { CALC [ hash-routine ]
                                       INDEX } [ NAME IS name ]
  [ DUPLICATES ARE [ NOT ] ALLOWED ] ... ]
```

item-name,...

must specify an item of fixed length that belongs to this record type.

name specifies the name of the table; referred to in the SSL statements.

This clause is used to define additional direct access paths via secondary keys and to assign a name to the SEARCH key table or the hash area, which can be referenced in the SSL.

SEARCH KEY ... USING CALC is permitted only with a SYSTEM set.

[SET OCCURRENCE SELECTION IS

THRU { CURRENT OF SET
LOCATION MODE OF OWNER [ALIAS FOR { *item-name*
identifier-1 }]
IS *identifier-2*]... }

item-name

must denote an item specified in the LOCATION MODE clause for the owner record type.

identifier-1

must be an identifier specified in the LOCATION MODE clause for the owner record type.

identifier-2

identifier-2 assigns a name for the additional item to be generated. UDS/SQL automatically creates this item with the same item type and length as *item-name* or *identifier-1*.

This clause must be specified if the set is not a SYSTEM set.
It is used to define the selection option for the set occurrences.

9.2 SSL syntax

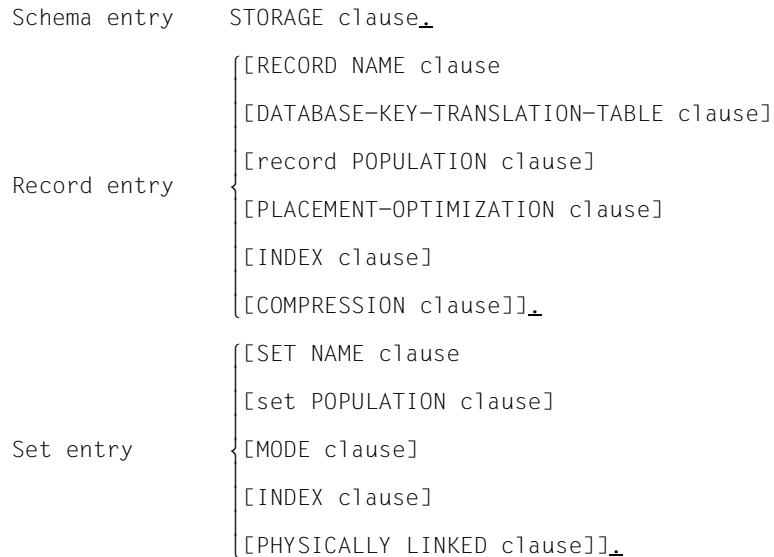


Figure 63: Structure of SSL

The description of the physical storage structure is optional. If it is omitted, UDS/SQL uses the default values indicated in the explanations for the individual syntax elements.

Otherwise, the description always starts with the STORAGE clause. The sequence of the record and set descriptions is arbitrary. All names referred to in the storage structure description must have been previously defined in the schema DDL.

9.2.1 Schema entry

STORAGE STRUCTURE OF SCHEMA *schema-name_*

The schema entry is used to specify the name of the schema to which the storage structure description applies.

9.2.2 Record entry

RECORD NAME IS *record-name*

[DATABASE-KEY-TRANSLATION-TABLE IS *integer-1*][WITHIN *realm-name-1*]

[POPULATION IS {*integer-2* WITHIN *realm-name-2*},...]

[PLACEMENT OPTIMIZATION FOR SET *set-name*]

[INDEX NAME IS *name*

[PLACING IS WITHIN *realm-name-3*]

[TYPE IS { DATABASE-KEY-LIST
REPEATED-KEY
 [DYNAMIC REORGANIZATION SPANS *integer-3* PAGES] }]] ...

[COMPRESSION FOR ALL ITEMS]_

The record entry is used to specify the name of the record type to which the storage structure description applies and

- to describe the size and physical position of the DBTT and the size of the hash areas for record SEARCH keys,
- to specify the number of records of the record type or the size of the hash area for the primary key within certain realms,
- to describe the physical position of the records within a realm if the record type is a member of a set,
- to describe the physical position, the type and the extent of reorganization employed for record SEARCH key tables or the physical position of hash areas for record SEARCH keys, and
- to indicate compression.

The individual clauses of the record entry are explained below.

RECORD NAME IS *record-name*

This clause is used to specify the name of the record type to which the record entry applies.

`[DATABASE-KEY-TRANSLATION-TABLE IS integer][WITHIN realm-name]]`

integer

must be greater than 0. If this entry is omitted, UDS/SQL reserves one page each for the DBTT and the hash area of a record SEARCH key.

realm-name

must not denote a temporary realm. If this entry is omitted, the DBTT is placed in the realm first mentioned in the DDL WITHIN clause for this record type.

This clause is used to describe the size and position of the DBTT and at the same time the size of the hash areas for the record SEARCH keys.

`[POPULATION IS {integer WITHIN realm-name},...]`

integer

must be greater than 0.

realm-name

must be a realm name specified in the DDL WITHIN clause for this record type. All realm names specified in the DDL WITHIN clause for this record type must be listed.

This clause is used to describe size and position of the hash areas for the primary key (LOCATION MODE IS CALC). In addition, UDS/SQL bases its assessment of the realm sizes on this entry.

If this clause is omitted, UDS/SQL reserves one page for the hash area in each of the realms mentioned in the WITHIN clause.

[PLACEMENT OPTIMIZATION FOR SET *set-name*]

set-name

must not denote a SYSTEM set.

This clause is used to store the member records of the set *set-name* in the vicinity of their owner record.



The record type must be an AUTOMATIC member of the set *set-name*.

Each realm referenced in the DDL WITHIN clause for this record type must also be specified in the WITHIN clause for the owner record type of the set *set-name*.

integer-1 in the set POPULATION clause of the set *set-name* must be greater than 0.

An indirect hash area is created if LOCATION MODE IS CALC has been defined for this record type.

This clause has no effect

- if the record type is a member of a set for which MODE IS LIST has been defined,
- if the owner record type of the set *set-name* is a member of a set for which MODE IS LIST has been defined,
- if LOCATION MODE IS CALC or PLACEMENT OPTIMIZATION has been defined for the owner record type of the set *set-name*.

```
[INDEX NAME IS name
  [PLACING IS WITHIN realm-name]
  [TYPE IS { DATABASE-KEY-LIST
             REPEATED-KEY
             [DYNAMIC REORGANIZATION SPANS integer PAGES]
           }]]...
```

name must have been defined in the schema DDL for a record SEARCH key table or a hash area of this record type.

realm-name

must not be a temporary realm. If this entry is omitted, UDS/SQL places the record SEARCH key table or the hash area in the first realm referenced in the DDL WITHIN clause for this record type.

integer

must be an integer between 1 and 20.
Default value: 2

This clause is used to specify the name of the record SEARCH key table or the hash area to which the description applies and to define

- for a record SEARCH key table; the physical placement, the type and the extent of reorganization,
- for a hash area; the physical placement.



If the description applies to a hash area, only PLACING can be specified.

[COMPRESSION FOR ALL ITEMS]

This clause causes UDS/SQL to store the records in compressed form, provided they are made available in this form at the DML interface.



If the record type has been defined with `LOCATION MODE IS CALC`, indirect `CALC` pages are created due to the compression.

Compression is not allowed if the record type contains an item of variable length or is a member of a set which has been defined with `MODE IS LIST`.

9.2.3 Set entry

```

SET NAME IS set-name
  [POPULATION IS integer-1 [ INCREASE IS integer-2]]
  [MODE IS {
    { CHAIN [ LINKED TO PRIOR]
      { POINTER-ARRAY { ATTACHED TO OWNER
                      { DETACHED [ WITHIN realm-name-1]
                        [ WITH PHYSICAL LINK]
                      }
            }
      }
    LIST
  } ] ]
  [DYNAMIC REORGANIZATION SPANS integer-3 PAGES]
  [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-4 PAGES]
             }
            } ] ] ...
  [MEMBER IS PHYSICALLY LINKED TO OWNER]_

```

The set entry is used to specify the set to which the storage structure description applies as well as

- to indicate the average size of the set occurrences
- to give information on the connection of the records within the set occurrences
- to define placement and extent of reorganization for pointer arrays, lists, sort key tables and set SEARCH key tables
- to indicate the type of set SEARCH key tables
- to add a pointer from member to owner.

The individual clauses of the set entry are explained below.

SET NAME IS *set-name*

This clause is used to specify the name of the set to which the set entry applies.

[POPULATION IS *integer-1* [INCREASE IS *integer-2*]]

integer-1

must be greater or equal to 0. Default value: 0

integer-2

must be greater than 0. Default value: 1

This clause is used to indicate the average number of member records which are expected to be in the set occurrences when the database is initially loaded or when the set occurrences are extended at a later date.

[MODE IS { CHAIN [LINKED TO PRIOR]
 { POINTER-ARRAY } { ATTACHED TO OWNER
 { LIST } { DETACHED [WITHIN *realm-name*]}
 [WITH PHYSICAL LINK]}]

[DYNAMIC REORGANIZATION SPANS *integer* PAGES]

CHAIN [LINKED TO PRIOR]

is the only valid specification if the set was defined with **ORDER IS SORTED** (without **INDEXED**) in the schema DDL.

POINTER-ARRAY DETACHED WITHIN *realm-name*

is the only valid specification if the set is a dynamic set.

LIST may be specified only if the following conditions are satisfied:

- The membership of the member record type in the set was defined as MANDATORY AUTOMATIC.
- Member records (including pointers, see [page 219](#), SCD) are no longer than
 - 993 bytes for databases with a page length of 2048 bytes,
 - 1963 bytes for databases with a page length of 4000 bytes, and
 - 4011 bytes for databases with a page length of 8096 bytes.

LIST ATTACHED or LIST DETACHED (without WITHIN)

may be specified only if the same realms were specified in the DDL WITHIN clauses for the owner and the member record types of this set, provided the set is not a SYSTEM set. The placement of a list is determined without a DETACHED WITHIN clause by the placement of the owner.

ATTACHED

may be specified only if the set is not a SYSTEM set.

realm-name

must denote a temporary realm if the set is dynamic.

In the case of LIST, it must be identical to that specified in the DDL WITHIN clause for the member record type.

integer

specifies an integer between 1 and 20.

Default value: 2

This clause is used to specify the linking of records within the set occurrences.

When setting up a pointer array, list or sort key table, the user can also define the extent of reorganization for such tables.

If this clause is omitted, the following default values apply for:

- dynamic sets: POINTER-ARRAY DETACHED WITHIN *realm-name*
- ORDER IS SORTED INDEXED: POINTER-ARRAY DETACHED
- ORDER IS LAST/FIRST/NEXT/PRIOR/SORTED: CHAIN
- non-dynamic sets if ORDER IS IMMATERIAL: CHAIN

```

[INDEX NAME IS name

  [PLACING IS { ATTACHED TO OWNER
                DETACHED[ WITHIN realm-name]}]

  [TYPE IS { DATABASE-KEY-LIST
             REPEATED-KEY
             [DYNAMIC REORGANIZATION SPANS integer PAGES]}]]...

```

name must be the name of a table or of a hash area specified in the Schema DDL.

ATTACHED

may be specified only if the set is not a SYSTEM set.

realm-name

must not denote a temporary realm. If this entry is omitted, UDS/SQL selects the default value as indicated on [page 162](#), [figure 40](#).

integer must be an integer between 1 and 20. Default value: 2

If a table of this set has been specified in the Schema DDL, this table can be referenced here in order to define its placement, type and extent of reorganization.

If a SEARCH key has been defined for storage in a hash area, the realm which is to contain this hash area can be specified here.

Default values

- for PLACING: DETACHED
- for TYPE: REPEATED-KEY

MEMBER IS PHYSICALLY LINKED TO OWNER

This clause is used to include an additional pointer to its owner in each member record of the set. The SCD of a member record also contains the PPP of the associated owner record.

9.3 Subschema DDL syntax

```

IDENTIFICATION DIVISION_
SUB-SCHEMA NAME clause
[PRIVACY LOCK clause]
[PRIVACY KEY clause]_
DATA DIVISION_
AREA SECTION_
COPY clause_
RECORD SECTION_
[COPY clause_]

Record entry {
[record name clause_]
record element name clause
[GROUP-USAGE clause]
[PICTURE clause]
[USAGE clause]
[OCCURS clause]_
[condition name clause]
[VALUE clause_]

[SET SECTION_
COPY clause_]

```

Figure 64: Structure of subschema DDL

The sequence of clauses shown in [figure 64](#) must be observed, with the following exceptions:

- The sequence of PICTURE and USAGE clauses is arbitrary.
- In the RECORD SECTION, a COPY clause may also follow a record entry.

Names used in the definition of the subschema must have been defined in the schema DDL. This does not apply to group item and condition names, which are newly defined in the subschema.

9.3.1 IDENTIFICATION DIVISION

IDENTIFICATION DIVISION.

SUB-SCHEMA NAME IS *subschema-name* OF SCHEMA NAME *schema-name*
 [PRIVACY LOCK FOR COMPILE IS *literal-1*[OR *literal-2*]]
 [PRIVACY KEY FOR COPY IS *literal-3*].

literal-1,-2

may consist of up to 10 characters.

literal-3

must be a password defined in the schema entry of the schema DDL.

This entry is used to assign a name to the subschema, and

- to indicate from which schema the subschema is to be copied,
- to define passwords to prevent unauthorized compilation of a DML program with this subschema, and
- to enter, where appropriate, one of the passwords preventing unauthorized copying of a subschema from the schema.

9.3.2 AREA SECTION

DATA DIVISION.

AREA SECTION.

```
{COPY ALL AREAS.
 {COPY realm-name, ...} ... }
```

This entry is used to copy all realms or a selection of realms from the schema into the subschema.

9.3.3 RECORD SECTION

RECORD SECTION.

[{ COPY ALL RECORDS. }]
 [{ COPY *record-name-1*, ... }] ...]

[01 *record-name-2*_
 { *level-number* *record-element-name* [PICTURE IS *mask-string*]

[GROUP-USAGE IS NATIONAL]

[USAGE IS { DISPLAY
COMPUTATIONAL-3
COMPUTATIONAL
NATIONAL
DATABASE-KEY
DATABASE-KEY-LONG }]

[OCCURS *integer* TIMES] ...]

[88 *condition-name*

{ VALUE IS } { *literal-1* [THROUGH *literal-2*], ... } ...] ...]
 { VALUES ARE }

record-name-2

must not be identical with *record-name-1* and must not be used in combination with COPY ALL RECORDS.

level-number

must be an integer between 02 and 49.

mask-string

See [page 241](#), [table 22](#).

GROUP-USAGE

If the GROUP-USAGE clause is specified, all lower-ranking record elements must be of the type NATIONAL.

USAGE

If this entry is omitted, DISPLAY is assumed by default.

Exception: if the PICTURE clause contains the symbol N, NATIONAL is assumed if the USAGE clause is missing.

literal-1

must be less than *literal-2*.

The user has the choice of either completely copying all record types contained in the schema into the subschema or only a selection of records or items.

In the latter case, the user specifies the record types that are to be copied completely or in part. For records to be copied in part, the user must specify all record elements that are to be copied.

It is also possible to define group items and conditions.

9.3.4 SET SECTION

SET SECTION.

```
{COPY ALL SETS.          }  
{COPY set-name, . . .} . . . }
```

This entry is used to copy all sets or a selection of sets from the schema.

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.

act-key-N page

Characteristic page of a *realm*, with the highest *page number*.

Copy of the *act-key-0 page*.

address, physical

See *act-key* or *PPP*.

administrator task

Task of the *independent DBH*; The *database administrator* can control execution of the *independent DBH* via this task.

AFIM

See *after-image*.

after-image

Modified portion of a *page* **after** its content has been updated.

The *DBH* writes after-images to the *RLOG file* as well as the *ALOG file*.

after-image, ALOG file

The after-images are written to the *ALOG file* when the *ALOG buffer* is full. The purpose of the after-images in the *ALOG file* is to secure the data contained in the database and thus they must be maintained for a long period of time. They are used to reconstruct an original database or update a *shadow database*.

after-image, RLOG file

After-images are logged in the *RLOG file* **before** the updates are applied to the *database*. The after-images held in the *RLOG file* are required for *warm start* only. They are thus periodically overwritten.

ALOG file

File for securing the data contained in the database in the long term; see *after-image*.

ALOG sequence number

See *sequence number*.

anchor record

Record automatically created by UDS/SQL as *owner record* for *SYSTEM sets*. It cannot contain any *items* defined with the *schema DDL* and cannot be accessed.

application

Realization of a job in one or several *user programs* working with UDS/SQL *databases*.

application program (AP)

E.g. *COBOL DML* program or *IQS*.

area

See *realm*.

ascending key (ASC key)

Primary key of a *set*. Defines the sequence of *member records* in the *set occurrences* by ascending key values.

authorization

Identification used for user groups.

authorized users

Specified user groups who are authorized to access the *database*.

automatic DBTT extension

Some utility routines automatically extend the number of records possible for a record type if too few are available; no separate administration is required to do this.

See also *online DBTT extension*.

automatic realm extension

Some utility routines automatically extend realms when insufficient free space is available; no separate administration is required to do this.

See also *online realm extension*.

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/OSD) 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 "[Introductory Guide to Systems Support](#)".

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*.

compound key

Key consisting of several *key items*.

compression

Only the filled *items* of a *record* are stored (see *SSL* clause *COMPRESSION*).

configuration

See *DB configuration*.

configuration user ID

User ID in which the *database administrator* starts the *DBH*.

configuration name

Freely selectable name of the *database configuration* for a particular *session*. The *DBH* uses it to form:

- the name of the *Session Log File*,
- the names of the *DB status file* and its backup copy,
- the names of the *RLOG files*,
- the names of the temporary *realms*,
- the names of session job variables,
- the *event names* of *PI eventing*,
- the *DCAM application* name for the administration,
- the names of the *common pools*
- the names of the dump files.

connection module

Module that must be linked into every UDS/SQL *application program* and which establishes the connection with the *DBH*.

consistency

State of the database without conflicts in the data stored in it.

consistency, logical

State of the database in which the stored data has no internal conflicts and reflects the real-world situation.

consistency, physical

State of the database in which the stored data is consistent with regard to correct physical storage, *access paths* and description information.

consistency, storage

See *physical consistency*.

consistency error

A violation of the *physical consistency* of the stored data.

consistency point

Point (in time) at which the *database* is consistent, i.e. all modifying transaction have been terminated and their modifications have been executed in the database.

consistency record

Administration record with consistency time and date stamps in the *DBDIR*. For an update in a *realm* the *DBH* enters the date and time in the consistency record and in the updated realm. When realms or *databases* are attached for a *session*, the *DBH* uses this time stamp to check the consistency of the realms within each database.

contending access

Different *transactions* attempting to access a *page* simultaneously.

conversation

SQL-specific administration data is retained across transaction boundaries in an *SQL* application. This kind of data administration unit is called a conversation. In openUTM such an administrative unit is also called a service.

copy

See *database copy*.

COSSD

See *COBOL Subschema Directory*.

CRA

(Current Record of Area) *Record* which is marked in the *currency table* as the current record of a particular *realm* (area).

CRR

(Current Record of Record) *Record* which is marked in the *currency table* as the current record of a particular *record type* (Record).

CRS

(Current Record of Set) *Record* which is marked in the *currency table* as the current record of a particular *set*.

CRU

(Current Record of Rununit) *Record* which is marked in the *currency table* as the current record of the *processing chain*.

CSV

see *Character Separated Values*

currency table

The currency table contains:

- CURRENT OF AREA table (table of CRAs),
- CURRENT OF RECORD table (table of CRRs) and
- CURRENT OF SET table (table of CRSs).

CURRENT OF AREA table

See *currency table*.

CURRENT OF RECORD table

See *currency table*.

CURRENT OF SET table

See *currency table*.

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 disconnected during the session with *DAL* commands.

DBC.COM

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.

exclusive buffer pool

Buffer which, in addition to the *system buffer pools*, is used exclusively for buffering *pages* of the specified *database*.

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*).

linked-in DBH

Module linked in to or dynamically loaded for the current DB *application program* and controlling access to a single *database (mono-DB operation)* or several databases simultaneously (*multi-DB operation*).

list

Table containing the *member records* of a *set occurrence*. Used for *sequential* and *direct access* to member records.

load parameters (DBH)

Parameters requested by the *DBH* at the beginning of the *session*. They define the basic characteristics of a session.

local application program

An *application program* is local with regard to a *configuration* if it was linked to the configuration using /SET-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=conf-name

local configuration

The *configuration* assigned to an *application program* before it is called using `/SET-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=conf-name`.

The application program communicates with the local configuration via the *communication pool*. The local configuration is in the same host as the application program.

local database

Database in a *local configuration*.

local distribution table

A *distribution table* is considered local to a *DBH* if it is held in the *DBH's distribution pool*.

local host

Host computer containing the *application program*.

local transaction

Transaction that only addresses the *local configuration*.

logging

Recording of all updates in the *database*.

logical connection

Assignment of two *communication partners* that enables them to exchange data. *DCAM applications* communicate via logical connections.

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 number of 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. For the purposes of administration of online extensibility of realms, you have the *DAL* commands *ACT INCR*, *DEACT INCR*, *DISPLAY INCR* and *REACT INCR*.

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 *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.'). 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.

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*.

primary key (DDL)

The *key* of a *record type* which is defined by means of "LOCATION MODE IS CALC" or the *key* of an order-determining *key* of a set occurrence which is defined by means of "ORDER IS SORTED [INDEXED]". Also used for *direct access* to a *record* or a set of records with the same key values or within a search interval.

primary key (SQL)

In the broader sense (SQL), a *record element* uniquely identifying a record.

In UDS-SQL, the database key of an owner record output as the "PRIMARY KEY" in the BPSQLSIA log (see also *foreign key*).

A *record element* which uniquely identifies a record is flagged as "UNIQUE" in the BPSQLSIA log unless it is the aforementioned "PRIMARY KEY".

primary subtransaction

Subtransaction that runs in the *local configuration*.

The primary subtransaction is opened by the first *READY* statement in a *transaction* on a *local database*.

If the first *READY* statement addresses a *remote database*, UDS-D generates a *dummy subtransaction* as the primary subtransaction.

PRIVACY-AND-IQF SCHEMA

UDS/SQL-internal *schema* for protection against unauthorized access.

PRIVACY-AND-IQF SUBSCHEMA

UDS/SQL-internal *subschema* for protection against unauthorized access.

probable position pointer

See *PPP*.

processing chain

Sequence of DML statements applied to a *database* within a *transaction*.

PPP (probable position pointer)

Probable address of a *page*, comprising *realm number* and *page number*.

UDS/SQL does not always update PPPs when the storage location of data is changed.

PTC state

See *prepared to commit*.

pubset declaration

See *UDS/SQL pubset declaration*

pubset declaration job variable

Job variable in which a *UDS/SQL pubset declaration* is specified.

P1 eventing

Manner in which tasks communicate with each other.

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 *DBCUM*, 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.

record hierarchy

Owner/member relationship between *record types*: the *owner record type* is the higher-ranking part of the relationship; the *member record type* is the lower-ranking part.

REC-REF

See *record reference number*.

record reference number

Record types are numbered consecutively in ascending order, starting at 1. The record reference number is part of the *database key*.

record SEARCH KEY table

SEARCH KEY table for selection of a *record* from a *record type*.

record sequence number (RSQ)

The record sequence number can be assigned by the database programmer; if not, UDS/SQL numbers the *records* of a *record type* contiguously in ascending order, in the sequence in which they are stored; numbering starts at 1. The record sequence number is part of the *database key*.

record type

Nameable grouping of *record elements*.

record type, linear

Record type that is neither the *owner* nor the *member* of a set (corresponds to record types of a conventional file).

referential integrity

Integrity of the relationships between tables (UDS/SQL *record types*).

remote application program

Application program that is not local with regard to a particular *configuration*.

remote configuration

DB-configurations that are not assigned to the *application program* via /SET-FILE-LINK LINK-NAME=DATABASE,FILE-NAME=conf-name but via the *distribution table* once the application program is running. The *connection module* of the application program communicates with the remote configurations via *DCAM applications*.

Remote configurations can be situated on *local* or *remote* hosts.

remote database

Database in a *remote configuration*.

remote host

Host computer that is not local.

repeating group

Group item with repetition factor. The repetition factor, which must be greater than 1, specifies the number of duplicates of the group item to be incorporated in the repeating group.

request

The functions of the *DAL* commands ADD DB, ADD RN, DROP DB, DROP RN, NEW RLOG and CHECKPOINT are held in the *DBH* as "requests" and are not executed until the *DAL* command PERFORM is entered.

restart of BMEND

Resumption of an aborted BMEND run.

restart of a session

See *session restart*.

restructuring

Modification of the *Schema DDL* or *SSL* for *databases* already containing data.

return code

Internal code which the called program sends to the calling program; Return code $\neq 0$ means an error has occurred.

RLOG file

Backup file used by the *DBH* during a session to store *before-images* (BFIMs) and *after-images* (AFIMs) of data which is updated. With the aid of the *RLOG file*, the *DBH* can cancel updates effected by incomplete *transactions*. There is one RLOG file per *configuration*. An RLOG file consists of two physical files.

rollback

Canceling of all updates effected within a *transaction*.

RSQ

See *record sequence number*.

RUNUNIT-ID

See *transaction identification*.

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*.

Schema Information Area (SIA)

The SIA contains the complete database definition. The *DBH* loads the SIA into main memory at the start of DB processing.

SEARCH KEY

Secondary key; *access paths* using secondary keys are created by UDS/SQL by means of *hash routines* and *multi-level tables*.

SEARCH KEY table

Multi-level table used by UDS/SQL as an *access path* via a *secondary key*.

secondary key

Any *key* which is not a *primary key*. Used for *direct access* to a *record* or a set of records with the same key values or within a search interval.

secondary subtransactions

Subtransactions that address *remote configurations*.

sequence number

Identifier in the name of the *ALOG files* (000000001 - 999999999). The first *ALOG* file of a *database* is always numbered 000000001.

sequential access

Accessing a *record* on the basis of its position within a predefined record sequence.

server task

Task of the *independent DBH* in which the *UDSSUB* module executes; processes the requests of the *DB application programs*.

session

Period between starting and normal termination of the *DBH* (*independent/ linked-in*) in which it is possible to work with the *databases* of the *configuration*. Normally, a session consists of a sequence of *session sections* and *session interrupts*.

session abort

Occurs when the *DBH* is terminated abnormally after a successful *session start*. A session abort can be caused by: power failure, computer failure, BS2000 problems, *DBH* problems, %TERM.

session end

Is the result of:

- *DAL* when using *independent DBH*,
- TERM in the *DML application program* when using *linked-in DBH*,
- *DBH* error handling.

During a *session interrupt*, the user can also effect session end by invalidating the *SLF* contents. Inconsistent *databases* can be made consistent again by a *warm start*, even without an *SLF*.

session interrupt

The period between a *session abort* and the related *session restart*.

session job variable

Job variable in which *UDS/SQL* stores information about a session.

Session Log File (SLF)

File which is permanently assigned to a *session* and which is required by the *DBH* in the event of a *session restart*. It contains information on the current *DB configuration*, the number of current file identifiers and the current values of the *DBH load parameters*.

session restart

Starting of the *DBH*, under the same *configuration name* and *configuration user ID*, after a *session abort*. With the aid of the *SLF*, the *DBH load parameters* and the current file identifiers which existed when the session aborted are re-established, and the *databases* of the previous *configuration* are reconnected, if necessary by means of a *warm start*.

session section

Period from the start of the *DBH*, either at the *session start* or a *restart*, to the normal *session end* or to a *session abort*.

session section number

Number which identifies a session section unambiguously.

session start

State of a session in which the *DBH* is started under a configuration name for which there is no *Session Log File (SLF)* with valid contents.

set

Nameable relationship between two *record types*.

set, dynamic

See *dynamic set*.

set, implicit

See *implicit set*.

set, singular

See *SYSTEM set*.

set, standard

See *standard set*.

Set Connection Data (SCD)

Linkage information for the *records* of a *set occurrence*.

set occurrence

Single instance of a *set*. Comprises exactly one *owner record* and any number of subordinate *member records*.

set reference number

Sets are numbered contiguously in ascending order, beginning at 1.

set SEARCH KEY table

SEARCH KEY table for selecting a *member record* from a *set occurrence*.

SF pubset

See *single feature pubset*

shadow database

Backup of all the files of a database, each saved under the name "*dbname.dbfile.copyname*".

A shadow database can be created at any time and processed parallel to the original database in RETRIEVAL mode.

In addition BMEND can be used to apply *ALOG files* that have already been closed to the database parallel to the UDS/SQL *session*.

Shared user buffer pool

Shared buffer of several databases which is used in addition to the *System Buffer Pool*, solely for buffering *pages* of the *databases* that have been assigned to it.

SIA

See *Schema Information Area*.

SIB

See *SQL Interface Block*.

single feature pubset

A single feature pubset (SF pubset) consists of one or more homogeneous disks which must have the same major properties (disk format, allocation unit).

SLF

See *session log file*.

SM pubset

See *system managed pubset*

snap pair, snap pubset, snap session, snap unit

A snap unit is the copy of an (original) unit (logical disk in BS2000/OSD) at a particular time ("Point-in-Time copy"). The TimeFinder/Snap component creates this copy as a "snapshot" in accordance with the "Copy-On-First-Write strategy": Only if data is modified is the original data concerned written beforehand into a central save pool of the Symmetrix system. The snap unit contains the references (track pointers) to the original data. In the case of unmodified data the references point to the unit, in the case of modified data to the save pool.

After they have been activated, the unit and snap unit are split; applications can access both.

The unit and snap unit together form a snap pair. TimeFinder/Snap manages this pair in what is known as a snap session.

If snap units exist for all units of a pubset, these snap units together form the snap pubset.

Details of this are provided in the manual "[Introductory Guide to Systems Support](#)".

sort key table

Table pointing to the *member records* of a set *occurrence*.

source program

Program written in a programming language and not yet translated into machine language.

spanned record

Record exceeding the length of a *page*. **Only UDS/SQL-internal records** can be spanned records;

User record types must not exceed

- 2020 bytes for a page length of 2048 bytes
- 3968 bytes for a page length of 4000 bytes
- 8064 bytes for a page length of 8096 bytes.

SQL

SQL is a relational database language which has been standardized by ISO (International Organization for Standardization).

SQL conversation

See *conversation*.

SQL DML

SQL Data Manipulation Language for querying and updating data.

SQL Interface Block (SIB)

Interface between UDS/SQL and SQL application program(s); contains the SQL statement, any existing parameters and the statement results.

SQL transaction

Related sequence of *SQL* 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.

SSIA

See *Subschema Information Area*.

SSIA-RECORD

UDS/SQL-internal *record type*, located in the *DBDIR*. *Records* belonging to this type are, for example, the Schema Information Area (*SIA*) and the Subschema Information Areas (*SSIAs*).

SSITAB module

Module generated by the BCALLSI utility routine; makes available the subschema information required by *CALL DML* programs.

SSL

See *Storage Structure Language*.

standard set

A *set* other than a *dynamic*, *implicit* or *SYSTEM set*.

statement code

Number stored in the first part of the *DATABASE-STATUS* item. Its function is to indicate which *DML* statement resulted in an exception condition.

status code

Number stored in the second part of the *DATABASE-STATUS* item. It indicates which exception condition has occurred.

Storage Structure Language (SSL)

Formalized language for describing the storage structure.

string

A series of consecutive alphanumeric characters.

subcontrol system

Component for the *independent DBH*. Responsible for control functions.

subschema

Section of a *schema* required for a particular *application*; it can be restructured, within limits, for the intended application; a subschema is defined by means of the *Subschema DDL*.

Subschema DDL

Formalized language for defining a *subschema*.

Subschema Information Area (SSIA)

The SSIA contains all subschema information required by the *DBH* to carry out, on behalf of the user, the *database* accesses permitted within the specified *subschema*. The *DBH* loads the SSIA into main memory when it is referenced in a *READY* command.

subschema module

Module resulting from *subschema* compilation when a *COBOL DML* program is compiled. It must be linked in to the *application program* and includes the *USER WORK AREA (UWA)* as well as the *RECORD AREA*, which is also part of the *base interface block (BIB)*. The name of the subschema module is the first 8 bytes of the subschema name.

subschema record

Record defined in the *Subschema DDL*.

SUB-SCHEMA SECTION

In COBOL programs with *DML* statements: section of the DATA DIVISION used for specifying the schema name and the subschema name.

subtransaction

In a distributed *transaction*, all the *processing chains* that address the databases in **one** *configuration* form a subtransaction.

system area

Realm required only by UDS/SQL. The system areas of a database include:

- the *Database Directory (DBDIR)*,
- the *Database Compiler Realm (DBCOM)*,
- the *COBOL Subschema Directory (COSSD)*

system break information

Indicates whether the *database* is consistent or inconsistent.

system buffer pools

Input/output buffer for database pages (see *page*). The buffer is part of the *common pool (independent DBH)* or the *DBH work area (linked-in DBH)*. Its size is determined by the *DBH load parameters* 2KB-BUFFER-SIZE, 4KB-BUFFER-SIZE or 8KB-BUFFER-SIZE.

system managed pubset

A system managed pubset consists of one or more volume sets which, as with an *SF pubset*, comprise a collection of multiple homogeneous disks; here, too, homogeneity relates to particular physical properties such as disk format and allocation unit.

SYSTEM record

See *anchor record*.

SYSTEM set

Set whose *owner record type* is the symbolic *record type* SYSTEM.

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/OSD 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/OSD: 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/OSD, 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.

transaction, committing a

Terminating a *transaction* with FINISH, i.e. all updates performed within the transaction are committed to the *database*.

transaction, rolling back a

Terminating a *transaction* with FINISH WITH CANCEL, i.e. all updates performed on the *database* within the transaction are rolled back.

Transaction Currency Area (TCUA)

Contains currency information.

transaction identification (TA-ID)

Assigned by the *DBH* to identify a particular *transaction*. Can be requested with the *DAL* command DISPLAY.

transfer pool

UDS-D-specific storage area in which the *UDSCT* receives the *BIBs* from *remote application programs*.

two-phase commit protocol

Procedure by which a *distributed transaction* that has made changes in at least one *remote configuration* is terminated in such a way as to safeguard *inter-configuration consistency* or UDS/SQL openUTM-D consistency. The two-phase commit is controlled

- by the distribution component in the *user task* if the *transaction* is distributed via UDS-D.
- by openUTM-D if the transaction is distributed via openUTM-D or via openUTM-D and UDS-D.

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.

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
DBC.COM	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

Abbreviations

FC	Function Code
FPA	Free Place Administration
GS	Global Store
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	Server Task
STT	Sekundäre Teiltransaktion (= secondary subtransaction)
TA	TransAction
TA-ID	TransAction IDentification
TANGRAM	TAsk aNd GRoup Affinity Management
TCUA	Transaction CUurrency Area
UDS/SQL	Universal Database System/Structured Query Language
UWA	User Work Area

Related publications

The manuals are available as online manuals, see <http://manuals.fujitsu-siemens.com>, or in printed form which must be paid for and ordered separately at <http://FSC-manualshop.com>.

UDS/SQL (BS2000/OSD)
Application Programming
User Guide

UDS/SQL (BS2000/OSD)
Creation and Restructuring
User Guide

UDS/SQL (BS2000/OSD)
Database Operation
User Guide

UDS/SQL (BS2000/OSD)
Design and Definition
User Guide

UDS/SQL (BS2000/OSD)
Messages
User Guide

UDS/SQL (BS2000/OSD)
Recovery, Information and Reorganization
User Guide

UDS/SQL (BS2000/OSD)
Ready Reference

UDS (BS2000)
Interactive Query System IQS
User's Guide

UDS-KDBS (BS2000/OSD)
Compatible Database Interface
User Guide

SQL for UDS/SQL
Language Reference Manual

BS2000/OSD-BC
Commands, Volumes 1 - 5
User Guide

BS2000/OSD-BC
Commands, Volume 6, Output in S Variables and SDF-P-BASYS
User Guide

BS2000/OSD-BC
System Messages, Volumes 1 - 3
User Guide

BS2000/OSD-BC
Introductory Guide to Systems Support
User Guide

BS2000/OSD-BC
Executive Macros
User Guide

BS2000/OSD-BC
Introductory Guide to DMS
User Guide

SDF (BS2000/OSD)
Introductory Guide to the SDF Dialog Interface
User Guide

SORT (BS2000/OSD)
User Guide

SPACEOPT (BS2000/OSD)
Disk Optimization and Reorganization
User Guide

LMS (BS2000)

SDF Format

User Guide

DSSM/SSCM

Subsystem Management in BS2000/OSD

User Guide

ARCHIVE (BS2000/OSD)

User Guide

DRV (BS2000/OSD)

Dual Recording by Volume

User Guide

HSMS / HSMS-SV (BS2000/OSD)

Hierarchical Storage Management System

Volume 1: Functions, Management and Installation

User Guide

SECOS (BS2000/OSD)

Security Control System

User Guide

openNet Server (BS2000/OSD)

BCAM

Reference Manual

DCAM (BS2000/OSD, TRANSDATA)

Program Interfaces

Reference Manual

DCAM (BS2000/OSD, TRANSDATA)

Macros

User Guide

TransView-NMA/-NMAE, TransView-NTAC2, NTAC2E (TRANSDATA, BS2000)

Network Management in BS2000

User Guide

OMNIS/OMNIS-MENU (TRANSDATA, BS2000/OSD)

Functions and Commands

User Guide

OMNIS/OMNIS-MENU (TRANSDATA, BS2000)

Administration and Programming

User Guide

openUTM

Concepts and Functions

User Guide

openUTM (BS2000/OSD, UNIX, Windows)

Programming Applications with KDCS for COBOL, C and C++

User Guide

openUTM (BS2000/OSD, UNIX, Windows)

Generating Applications

User Guide

openUTM (BS2000/OSD, UNIX, Windows)

Administering Applications

User Guide

openUTM (BS2000/OSD)

Messages, Debugging and Diagnostics

User Guide

COBOL2000 (BS2000/OSD)

COBOL Compiler

Reference Manual

COBOL2000 (BS2000/OSD)

COBOL Compiler

User's Guide

COBOL85 (BS2000)

COBOL Compiler

Reference Manual

COBOL85 (BS2000)

COBOL Compiler

User's Guide

CRTE (BS2000/OSD)

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/OSD)

Disk Access Buffer

User Guide

Unicode in BS2000/OSD

Introduction

XHCS (BS2000/OSD)

8-Bit Code and Unicode Processing in BS2000/OSD

Benutzerhandbuch

BS2000/OSD

Softbooks English

CD-ROM

Index

4GL 40

A

access

- concurrent 54, 109
- contending 263
- direct 62, 84, 93, 100, 153, 238, 245, 246, 263
- sequential 84, 153, 263

access authorization 263

access path 84, 87, 100, 151, 165, 238, 244, 246, 263

access rights 42, 111, 263

access type 84, 263

act-key 131, 138, 264

act-key-0 page 197, 200, 264

act-key-N page 197, 200, 264

act-keys 207

address

physical 264

address, physical 85, 89, 93, 100, 132

administrator task 264

AFIM 264

after-image 108, 264

ALOG file 264

RLOG file 264

ALIAS entry 247

ALOG file 264

ALOG sequence number 265

analytical process 44

anchor record 105, 150, 221, 265

ANSI 34

application 265

application program 46

application program (AP) 265

area 54, 108, 131, 138, 161, 168, 183, 194, 259, 265

AREA NAME clause 109, 237

ascending key (ASC key) 265

ASCENDING-KEY clause 98, 100, 245

attribute 32

attribute value 32

authorization 265

authorized users 265

AUTOMATIC 79, 82, 104, 149, 245, 251

automatic DBTT extension 265

automatic realm extension 265

B

backup database 266

backward chaining 152, 155

Base Interface Block (BIB) 266

base table 32

before-image 266

BFIM 266

BIB (Base Interface Block) 266

BMEND 108

BPRIVACY 108

BPSQLSIA 36, 51

BREORG 137, 142

buffer pools

see exclusive buffer pool

see system buffer pools

C

CALC key 87, 103, 266

CALC page 197, 266

direct 210

indirect 149, 176, 213

- CALC SEARCH key 266
- CALC table 210, 267
- CALC table header 210
- calculation formulas, storage space requirement 177
- CALL DML 47, 267
- catalog identifier 267
- CC 39
- CHAIN 130, 136, 144, 156, 223
- chain 130, 144, 151, 155, 267
- Character Separated Values (CSV) 267
- check records 267
- checkpoint 267
- CHECK-TABLE 267
- clone 268
- COBOL DML 30, 47, 268
- COBOL runtime system 268
- COBOL Subschema Directory (COSSD) 268
- CODASYL 53
- CODASYL model 28, 35, 36
- Codd 31
- coexistence 36
- column 32
- column conventions 235
- comment 234
- common memory 268
- common pool 268
- communication partners 268
- communication pool 268
- compatible database interface 268, 280
- compiler database 269
- COMPILER-SCHEMA 269
- COMPILER-SUBSCHEMA 269
- compound key 87, 92, 269
- compression 176, 269
- COMPRESSION clause 176, 251
- computer 48, 49
- computer network 48
- conceptual schema 44
- condition 192, 261
- condition name clause 192, 260
- condition variable 192
- configuration 46, 47, 48, 49, 269
- configuration identification 269
- configuration name 269
- connection data 222
- connection module 269
- consistency 42, 46, 47, 269
 - logical 270
 - physical 270
 - storage 270
- consistency error 270
- consistency point 270
- consistency record 270
- container
 - see page container
- contending access 270
- continuation lines 235
- conversation 270
- copy 270
- COPY clause 185, 193, 259
- Copying a database key item 188
- copying a numeric item, an alphanumeric item of fixed length or a national item 186
- COSSD 270
- CRA 270
- CRR 271
- CRS 98, 103
- CSV 271
- currency table 271
- CURRENT
 - OF AREA table 271
 - OF RECORD table 271
 - OF SET table 271
- Current
 - Record of Rununit 271
 - Record of Set (CRS) 271
- cyclic data structure 82

- D**
- data analysis 44
- data backup 271
- data deadlock 271
- Data Definition Language 50
- Data Description Language 30, 50
- data independence 32, 41
- data manipulation 33
- Data Manipulation Language 30

- Data Manipulation Language (DML) 276
- data modeling 43
- data models 28
- data organization 38
- data page 197, 215
- data privacy 54
- data protection 42, 54, 183, 184
- data protection (privacy) 272
- data relationship 29, 30, 33, 50
- data retrieval 33
- data security 54, 108
- data structure
 - logical 130
 - physical 130
- database (DB) 272
- database administrator 272
- Database Administrator Language (DAL) 271
- database compiler realm (DBCOM) 272
- database configuration 46, 47, 48, 49
- database copy 272
- database copy update 272
- database description
 - relational 36
- database directory (DBDIR) 272
- database handler, see DBH
- database job variable 272
- database key 68, 86, 99, 105, 132, 151, 242, 272
 - item 273
- database key item 239
- database key translation table 85, 130
- database key translation table (DBTT) 249
 - placement 160, 168, 249
- database key value 64, 86, 207, 210, 215
- database page 273
- database system 273
- DATABASE-KEY item 64, 85, 86, 188, 239, 242, 273
- DATABASE-KEY-LONG item 64, 86, 188, 239, 242, 273
- DATABASE-KEY-TRANSLATION-TABLE
 - clause 136, 139, 250
- DATABASE-STATUS 273
- DB configuration 273
- DB key 274
- DB status file 273
- DBCOM 273
- DBDIR 274
- DBH 46, 47, 274
 - end 274
 - independent 274
 - linked-in 274
 - start 274
- DBH load parameters 274
- DBTT 86, 130, 131, 143, 274
 - column 133
 - line 133, 136
 - page 275
 - placement 160, 168
 - size 143
 - storage space requirement 136
- DBTT anchor page 197, 205, 274
- DBTT base 274
- DBTT clause 143
- DBTT extension
 - automatic 265
 - online 284
- DBTT extent 275
- DBTT page 197, 207
- DCAM 275
- DCAM application 275
- DDL 30, 50, 275
- deadlock 42, 47, 275
- decimal point symbol 56, 57
- descending key (DESC key) 275
- DESCENDING-KEY clause 98, 100, 245
- Description 50
- design 43
- digit symbol 56, 60
- direct access 62, 84, 100, 105, 153, 238, 245, 275
- direct CALC page 210
- direct hash area 275
- distributed database 275
- distributed transaction 276
- distribution 45
- distribution pool 276
- distribution table 276
- DML 30

- domain [32](#)
 - DRIVE [40](#)
 - DRV [42](#)
 - dummy subtransaction [276](#)
 - duplicate table [157](#)
 - DUPLICATES clause [87](#)
 - duplicates header [276](#)
 - duplicates table [229, 277](#)
 - main level [277](#)
 - DYNAMIC clause [106, 244](#)
 - dynamic set [106, 277](#)
- E**
- ESTIMATE-REPORT [277](#)
 - event name [277](#)
 - exclusive buffer pool [277](#)
- F**
- FASTPAM access method [42](#)
 - flexibility [35](#)
 - foreign key [32, 33, 36, 50, 277](#)
 - FPA [277](#)
 - FPA base [202, 278](#)
 - FPA extent [202, 278](#)
 - FPA page [197, 202, 278](#)
 - free place administration [197, 202, 212, 215, 225, 278](#)
 - function analysis [43](#)
 - function code [278](#)
- G**
- group item [53, 66, 191, 261, 278](#)
- H**
- hash area [87, 149, 168, 252, 278](#)
 - name [92](#)
 - naming [107, 240, 246](#)
 - placement [129, 160, 168, 172, 252](#)
 - position [107](#)
 - size [136, 142, 249](#)
 - hash routine [87, 89, 92, 100, 105, 137, 142, 278](#)
 - hashing [278](#)
 - HASHLIB [279](#)
 - host [47, 48](#)
 - host computer [49](#)
 - host network [47](#)
- I**
- IDENTIFICATION DIVISION [259](#)
 - identifier [279](#)
 - implicit set [279](#)
 - inconsistency [279](#)
 - independent DBH [279](#)
 - INDEX clause [160, 166, 251, 257](#)
 - index level [279](#)
 - index page [279](#)
 - INDEX search key [279](#)
 - indirect hash area [279](#)
 - information analysis [43](#)
 - integrity [280](#)
 - referential [30, 33, 35, 38, 50](#)
 - interconfiguration [280](#)
 - consistency [280](#)
 - deadlock [280](#)
 - interface [280](#)
 - internal version number [280](#)
 - IQL [106](#)
 - ISO [34](#)
 - item [53, 280](#)
 - alphanumeric [60, 186, 241, 242](#)
 - binary [59, 61, 187, 242](#)
 - fixed length [241](#)
 - fixed-length [241](#)
 - national [63, 187, 241](#)
 - numeric [56, 186, 241](#)
 - packed [58, 187, 242](#)
 - unpacked [56, 187, 241](#)
 - variable length [61, 149, 176, 241](#)
 - variable-length [220, 241](#)
 - item contents [55, 84](#)
 - item name [53](#)
 - item type [53, 187](#)
- K**
- KDBS [268, 280](#)
 - key [87, 92, 98, 145, 156, 281](#)
 - compound [281](#)
 - key item [85, 281](#)

key reference number 281
keyword 22

L

level 189
linkage of records 129, 144
linked-in control system 281
linked-in DBH 281
linking of records 256
LIST 136, 223
list 130, 136, 144, 148, 153, 154, 172, 225, 254, 256, 281
 placement 160, 172
 storage requirements 143
 storage space requirement 140, 177
literal 234
load parameters DBH 281
local application program 281
local configuration 282
local database 282
local distribution table 282
local host 282
local transaction 282
LOCATION MODE clause 85, 86, 87, 239
logging 282
logical connection 282
logical structure 44

M

main reference 282
mainref number 282
MANDATORY 79, 83, 149
manipulation 33
MANUAL 79
many-to-many relationship 83
mask 56
mask string 56, 60, 63
master task 282
member 29, 36, 283
 AUTOMATIC 283
 MANDATORY 283
 MANUAL 283
 OPTIONAL 283
MEMBER clause 79, 245

member record 79, 94, 110, 129, 140, 144, 159, 160, 163, 165, 257, 283
member record type 70, 165, 244, 245, 283
microcosm 43
mirrored disks 42
MODE clause 145, 160, 166, 255
MODE-Klausel 136
mono-DB configuration 283
mono-DB operation 283
multi-DB configuration 283
multi-DB operation 46, 47, 283
multi-DB program 283
multi-level table 283
multithreading 284

N

N, PICTURE symbol 63, 187, 241
NATIONAL 63, 187, 241
natural optimization 163
network 47, 48, 284
network model 28, 35, 36
network structures 40
normalized 44
notational conventions 22

O

occupancy level, table pages 172
OCCURS clause 65, 189, 241, 242
OLTP 40, 284
online backup 284
online DBTT extension 284
online realm extension 284
open transaction 284
openUTM 285
operator task (OT) 285
OPTIONAL 79
optional word 22
ORDER clause 94, 107, 145, 148, 153, 245
original database 285
overflow page
 of CALC page 138, 142
 of duplicate table 157
 of duplicates table 231
overflow pages 285

owner [29](#), [36](#), [285](#)
OWNER clause [105](#), [245](#)
owner record [70](#), [103](#), [110](#), [140](#), [147](#), [150](#), [159](#),
[161](#), [163](#), [165](#), [251](#), [285](#)
owner record type [70](#), [244](#), [285](#)

P

P1 eventing [288](#)
page [131](#), [197](#), [198](#), [285](#)
 overflow [138](#)
 structure [200](#)
page address [131](#), [285](#)
page container [197](#), [199](#), [286](#)
page feed [235](#)
page format [61](#), [68](#), [108](#), [149](#), [240](#)
page header
page header (page info) [286](#)
page index entry [210](#), [215](#), [286](#)
page length [61](#), [68](#), [108](#), [133](#), [149](#), [197](#), [202](#), [207](#),
[210](#), [213](#), [215](#), [225](#), [230](#), [239](#), [240](#)
page number [131](#), [286](#)
 relative [87](#), [92](#), [106](#)
parts list [114](#), [159](#)
parts list processing [40](#)
password [111](#), [184](#), [237](#), [259](#)
password for UDS/SQL files [286](#)
pattern [286](#)
pattern string [286](#)
performance [35](#)
PETA [287](#)
physical
 page address [131](#)
PHYSICALLY LINKED clause [159](#), [257](#)
PICTURE clause [56](#), [60](#), [63](#), [186](#), [189](#), [241](#)
placement
 data at realm level [161](#)
 data within realm [163](#)
 define [160](#)
PLACEMENT OPTIMIZATION [143](#), [149](#), [163](#),
[165](#)
PLACEMENT OPTIMIZATION clause [251](#)
pointer [92](#), [106](#), [131](#), [145](#), [150](#), [156](#), [219](#), [254](#),
[257](#)

pointer array [130](#), [136](#), [144](#), [153](#), [172](#), [225](#), [254](#),
[256](#), [287](#)
 placement [160](#), [172](#)
 storage requirements [143](#)
 storage space requirement [140](#), [177](#)
POINTER-ARRAY [130](#), [136](#), [144](#), [145](#), [153](#), [223](#)
population, declare [129](#), [136](#)
PPP [131](#), [159](#)
PPP (Probable Position Pointer) [145](#), [154](#), [155](#)
PPP (probable position pointer) [288](#)
prepared to commit (PTC) [287](#)
primary key [32](#), [33](#), [36](#), [50](#), [87](#), [94](#), [98](#), [136](#), [143](#),
[145](#), [156](#), [168](#), [172](#), [238](#), [244](#), [249](#), [287](#)
primary key (DDL) [239](#), [287](#)
primary key (SQL) [287](#)
primary subtransaction [288](#)
PRIVACY KEY clause [184](#)
PRIVACY LOCK clause [111](#), [184](#)
PRIVACY-AND-IQF SCHEMA [288](#)
PRIVACY-AND-IQF SUBSCHEMA [288](#)
probable position pointer (PPP) [288](#)
processing chain [288](#)
program interface [38](#)
PTC state [288](#)
pubset declaration [288](#)
pubset declaration job variable [288](#)

R

RC [38](#)
READY [289](#)
READYC [289](#)
realm [54](#), [108](#), [109](#), [131](#), [138](#), [161](#), [168](#), [183](#),
[194](#), [259](#), [289](#)
 size [250](#)
 structure [197](#)
 temporary [106](#), [110](#), [161](#)
realm configuration [289](#)
realm copy [108](#), [289](#)
realm entry
 DDL [237](#)
realm extension
 automatic [265](#)
 online [284](#)

- realm reference [131](#)
 - realm reference number [289](#)
 - REALM-REF (Realm Reference) [131](#)
 - reconfiguration [289](#)
 - record [32](#), [68](#), [110](#), [130](#), [136](#), [143](#), [149](#), [256](#), [289](#)
 - structure [219](#)
 - record address [92](#), [132](#), [138](#), [142](#), [289](#)
 - RECORD AREA [290](#)
 - record element [28](#), [32](#), [33](#), [53](#), [66](#), [185](#), [238](#), [240](#), [261](#), [290](#)
 - record entry
 - DDL [238](#)
 - SSL [249](#)
 - subschema DDL [258](#)
 - record hierarchy [290](#)
 - record length item [62](#)
 - RECORD NAME clause [110](#), [239](#), [249](#)
 - record name clause [185](#)
 - record POPULATION clause [136](#)
 - record reference number [85](#), [132](#), [290](#)
 - record SEARCH KEY table [290](#)
 - record SEARCH key table [93](#), [130](#), [136](#), [143](#), [224](#), [249](#), [252](#)
 - placement [160](#), [168](#), [172](#)
 - record sequence [84](#), [94](#), [105](#), [148](#), [151](#), [154](#), [238](#), [244](#)
 - record sequence number [85](#), [132](#), [133](#), [290](#)
 - record sequence number (RSQ) [210](#), [213](#)
 - record type [28](#), [53](#), [68](#), [130](#), [136](#), [183](#), [185](#), [238](#), [249](#), [261](#), [290](#)
 - linear [290](#)
 - record type reference [132](#)
 - record-element-name clause [240](#)
 - record-POPULATION clause [250](#)
 - REC-REF [290](#)
 - redundancy [107](#), [157](#)
 - redundancy-free data storage [41](#)
 - referential integrity [30](#), [33](#), [35](#), [38](#), [50](#), [290](#)
 - relation [32](#)
 - relational database description [36](#)
 - relational model [31](#), [35](#), [36](#)
 - relational schema [51](#)
 - relational view [36](#), [51](#)
 - relationship [29](#), [30](#), [33](#), [50](#)
 - remote application program [290](#)
 - remote configuration [291](#)
 - remote database [291](#)
 - remote host [291](#)
 - reorganization, dynamic [107](#), [129](#), [172](#), [249](#), [252](#), [255](#)
 - repeating group [53](#), [66](#), [183](#), [190](#), [242](#), [291](#)
 - repetition factor [53](#), [242](#)
 - repetition symbol [56](#), [57](#), [60](#), [63](#)
 - request [291](#)
 - restart
 - of a session [291](#)
 - of BMEND [291](#)
 - restructuring [291](#)
 - result table [32](#)
 - retrieval [33](#)
 - return code [291](#)
 - RLOG file [291](#)
 - rollback [292](#)
 - RR [39](#)
 - RSQ [133](#), [145](#), [158](#), [210](#), [213](#), [292](#)
 - RUNUNIT-ID [292](#)
- ## S
- SCD [219](#), [222](#)
 - schema [292](#)
 - conceptional [44](#)
 - naming [237](#)
 - relational [51](#)
 - schema DDL [30](#), [50](#), [53](#), [138](#), [145](#), [149](#), [292](#)
 - structure [236](#)
 - schema entry
 - DDL [236](#)
 - SSL [248](#)
 - Schema Information Area (SIA) [292](#)
 - SCHEMA NAME clause [111](#)
 - SEARCH KEY [292](#)
 - SEARCH key [92](#), [100](#), [102](#), [105](#), [143](#), [172](#), [240](#), [246](#), [249](#)
 - naming [107](#)
 - SEARCH KEY clause [92](#), [102](#), [105](#), [107](#), [240](#), [245](#)
 - SEARCH KEY table [292](#)

- SEARCH key table [93](#), [102](#), [130](#), [136](#), [140](#), [154](#),
[157](#), [172](#), [177](#), [225](#), [254](#), [257](#)
 - form [157](#)
 - naming [240](#), [246](#)
 - placement [160](#), [167](#), [172](#)
 - storage space requirement [177](#)
- secondary key [92](#), [100](#), [102](#), [107](#), [136](#), [139](#), [142](#),
[161](#), [168](#), [172](#), [238](#), [240](#), [244](#), [246](#), [292](#)
- secondary subtransaction [292](#)
- security [42](#)
- security concept [42](#)
- selection method for set occurrences [103](#)
- selection option for set occurrences [244](#), [247](#)
- semicolon [234](#)
- sequence number [292](#)
- sequence of records [71](#)
- sequential access [293](#)
- server task [293](#)
- session [293](#)
 - abort [293](#)
 - end [293](#)
 - interrupt [293](#)
 - start [294](#)
- session job variable [293](#)
- Session Log File (SLF) [293](#)
- session restart [294](#)
- session section [294](#)
- session section number [294](#)
- set [53](#), [70](#), [105](#), [183](#), [193](#), [244](#), [254](#), [261](#), [294](#)
 - dynamic [106](#), [110](#), [161](#), [244](#), [294](#)
 - implicit [294](#)
 - singular [294](#)
 - standard [294](#)
- set connection data [219](#), [222](#)
- Set Connection Data (SCD) [294](#)
- set entry
 - DDL [243](#)
 - SSL [254](#)
- set membership [79](#), [106](#), [149](#), [244](#), [245](#)
- SET NAME clause [70](#), [244](#), [255](#)
- set occurrence [70](#), [95](#), [130](#), [144](#), [148](#), [151](#), [160](#),
[163](#), [254](#), [294](#)
 - representation [71](#)
 - size [140](#), [154](#)
- SET OCCURRENCE SELECTION clause [103](#),
[247](#)
 - set operations [33](#)
 - set POPULATION clause [139](#), [140](#), [255](#)
 - set reference number [294](#)
 - set relationship [29](#), [30](#), [33](#)
 - set SEARCH KEY table [295](#)
 - set SEARCH key table [102](#), [130](#), [136](#), [143](#), [172](#),
[225](#), [254](#)
 - placement [160](#), [167](#), [172](#)
 - storage space requirement [140](#)
- SF pubset [295](#)
- shadow database [295](#)
- Shared User buffer pool [295](#)
- SIA [131](#), [295](#)
- SIB [295](#)
- sign symbol [56](#)
- single feature pubset [295](#)
- SLF [295](#)
- SM pubset [295](#)
- snap [296](#)
- sort key table [130](#), [136](#), [143](#), [151](#), [172](#), [225](#), [254](#),
[256](#), [296](#)
 - naming [107](#)
 - placement [160](#), [167](#), [172](#)
 - storage space requirement [140](#), [177](#)
- source program [296](#)
- spanned record [296](#)
- SQL [34](#), [36](#), [296](#)
- SQL access [57](#), [58](#), [62](#), [85](#), [97](#), [110](#), [176](#), [193](#),
[194](#)
- SQL conversation [296](#)
- SQL DML [296](#)
- SQL Interface Block (SIB) [297](#)
- SQL transaction [297](#)
- SSIA [297](#)
- SSIA-RECORD [297](#)
- SSITAB module [297](#)
- SSL [51](#), [98](#), [107](#), [110](#), [129](#), [248](#), [297](#)
 - structure [248](#)
- standard set [297](#)
- standardization [34](#)
- statement code [297](#)
- status code [297](#)

storage mode 99, 144
 storage space reservation 143
 storage structure 129
 Storage Structure Language (SSL) 51, 297
 string 297
 structure, logical 44
 subcontrol system 297
 subschema 51, 183, 258, 298
 naming 184, 259
 Subschema DDL 298
 subschema DDL 30, 51, 111, 183, 258
 structure 258
 Subschema Information Area (SSIA) 298
 subschema module 298
 SUB-SCHEMA NAME clause 184
 subschema record 298
 SUB-SCHEMA SECTION 298
 subtransaction 298
 suffixes data types 16
 syntax rules 234
 system area 298
 system break information 298
 system buffer pools 299
 system managed pubset 299
 SYSTEM record 299
 SYSTEM set 100, 105, 106, 140, 221, 299

T

table 32, 33, 160
 multi-level 299
 table (SQL) 299
 table extension 140, 172
 table header 224, 299
 table level 146, 157
 table line 141
 table pages 299
 table structure 224
 TANGRAM 300
 task attribute TP 300
 task communication 300
 task deadlock 300
 task priority 300
 TCUA 300

TEMPORARY clause 109, 237
 temporary realm 110
 time acknowledgment 300
 time conversion, uninterrupted 42
 transaction 300
 committing a 300
 roll back 301
 Transaction Currency Area 301
 transaction identification (TA-ID) 301
 transfer pool 301
 tuple 32
 two-phase commit protocol 301
 TYPE clause 58, 157, 241, 242

U

UDS 301
 UDS/SQL 302
 UDS/SQL / openUTM-D consistency 302
 UDS/SQL pubset declaration 302
 UDSADM 301
 UDS-D 47
 UDS-D task UDSCT 302
 UDSHASH 301
 UDSNET 301
 UDSSUB 301
 uninterrupted time conversion 42
 unique throughout the network 302
 unit of data 53
 uppercase letters 235
 USAGE clause 188
 user database 302
 user realm 302
 user task 302
 user view 38
 USER-WORK-AREA (UWA) 302
 UTF-16 63, 187, 241
 utility 109
 utility routine 111, 137, 142
 UWA 302

V

value 32
 VALUE clause 192

Index

value range [32](#)
 of condition [192](#)
 of item [55](#)
variable [22](#)
vector [53](#), [65](#), [183](#), [189](#), [242](#), [303](#)
version number
 internal [303](#)

view [32](#)
 relational [51](#)

W

warm start [303](#)
WITHIN clause [109](#), [161](#), [168](#), [240](#)

Free Manuals Download Website

<http://myh66.com>

<http://usermanuals.us>

<http://www.somanuals.com>

<http://www.4manuals.cc>

<http://www.manual-lib.com>

<http://www.404manual.com>

<http://www.luxmanual.com>

<http://aubethermostatmanual.com>

Golf course search by state

<http://golfingnear.com>

Email search by domain

<http://emailbydomain.com>

Auto manuals search

<http://auto.somanuals.com>

TV manuals search

<http://tv.somanuals.com>