

# CA-IDMS<sup>®</sup>

---

Database Administration

15.0



Computer Associates™

This documentation and related computer software program (hereinafter referred to as the "Documentation") is for the end user's informational purposes only and is subject to change or withdrawal by Computer Associates International, Inc. ("CA") at any time.

This documentation may not be copied, transferred, reproduced, disclosed or duplicated, in whole or in part, without the prior written consent of CA. This documentation is proprietary information of CA and protected by the copyright laws of the United States and international treaties.

Notwithstanding the foregoing, licensed users may print a reasonable number of copies of this documentation for their own internal use, provided that all CA copyright notices and legends are affixed to each reproduced copy. Only authorized employees, consultants, or agents of the user who are bound by the confidentiality provisions of the license for the software are permitted to have access to such copies.

This right to print copies is limited to the period during which the license for the product remains in full force and effect. Should the license terminate for any reason, it shall be the user's responsibility to return to CA the reproduced copies or to certify to CA that same have been destroyed.

To the extent permitted by applicable law, CA provides this documentation "as is" without warranty of any kind, including without limitation, any implied warranties of merchantability, fitness for a particular purpose or noninfringement. In no event will CA be liable to the end user or any third party for any loss or damage, direct or indirect, from the use of this documentation, including without limitation, lost profits, business interruption, goodwill, or lost data, even if CA is expressly advised of such loss or damage.

The use of any product referenced in this documentation and this documentation is governed by the end user's applicable license agreement.

The manufacturer of this documentation is Computer Associates International, Inc.

Provided with "Restricted Rights" as set forth in 48 C.F.R. Section 12.212, 48 C.F.R. Sections 52.227-19(c)(1) and (2) or DFARS Section 252.227-7013(c)(1)(ii) or applicable successor provisions.

**Second edition, October 2001**

© 2001 Computer Associates International, Inc.  
All rights reserved.

All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

# Contents

---

<b>How to use this manual</b> . . . . .	xvii
---	------

---

## Volume 1. Database Definition

<b>Chapter 1. The CA-IDMS Environment</b> . . . . .	1-1
1.1 The environment . . . . .	1-3
1.1.1 Multiuser environment . . . . .	1-3
1.1.2 Single-user environment . . . . .	1-4
1.1.3 Data sharing environment . . . . .	1-5
1.2 CA-IDMS/DC and CA-IDMS/UCF . . . . .	1-6
1.3 CA-IDMS/DB components . . . . .	1-7
1.3.1 The database management system . . . . .	1-7
1.3.2 Dictionaries . . . . .	1-7
1.3.3 Physical database definition . . . . .	1-8
1.3.4 Logical database definition . . . . .	1-8
1.4 Security . . . . .	1-9
1.5 Getting started . . . . .	1-10
1.5.1 Towards a production environment . . . . .	1-10
1.6 Tools for database definition and maintenance . . . . .	1-13
<b>Chapter 2. Defining Physical Databases</b> . . . . .	2-1
2.1 About physical databases . . . . .	2-3
2.1.1 Segments . . . . .	2-4
2.1.2 DMCLs . . . . .	2-4
2.1.3 Database name tables . . . . .	2-5
2.2 Separating logical and physical database definitions . . . . .	2-7
2.3 Before you begin . . . . .	2-8
<b>Chapter 3. Defining Segments, Files, and Areas</b> . . . . .	3-1
3.1 About segments, files, and areas . . . . .	3-3
3.1.1 Files . . . . .	3-3
3.1.2 Areas . . . . .	3-4
3.2 Planning . . . . .	3-5
3.2.1 Segment boundaries . . . . .	3-5
3.2.2 Mapping areas to files . . . . .	3-6
3.2.3 Page ranges . . . . .	3-6
3.2.4 Page groups . . . . .	3-6
3.2.5 Records per page . . . . .	3-7
3.2.6 Page reserve . . . . .	3-8
3.2.7 Resolving symbolic parameters . . . . .	3-8
3.2.8 Synchronization stamps . . . . .	3-9
3.2.9 Specifying data set name information . . . . .	3-10
3.3 Procedure for defining segments . . . . .	3-12
3.4 Related information . . . . .	3-15
<b>Chapter 4. Defining, Generating, and Punching a DMCL</b> . . . . .	4-1

4.1	About DMCLs	4-3
4.2	Data sharing attributes	4-6
4.3	Database buffers	4-8
4.4	Journal buffers and journal files	4-9
4.4.1	Sizing the journal buffer	4-10
4.4.2	Sizing journal files	4-11
4.5	Adding segments to the DMCL	4-13
4.5.1	Required segments	4-13
4.5.2	File limitations	4-14
4.5.3	Area status	4-14
4.5.4	Sharing update access to data	4-15
4.5.5	Area overrides	4-16
4.5.6	File overrides	4-17
4.6	Procedure for defining a DMCL	4-19
4.7	Making the DMCL accessible to the runtime environment	4-22
4.8	Related information	4-23
<b>Chapter 5. Defining a Database Name Table</b>		5-1
5.1	About database name tables	5-3
5.2	Planning	5-5
5.2.1	SQL considerations	5-5
5.2.2	Non-SQL considerations	5-6
5.2.3	Restricting subschema names	5-8
5.2.4	Application dictionaries	5-8
5.2.5	Defining the default dictionary	5-9
5.2.6	Conflicting names	5-10
5.2.7	Mixed page groups and maximum records per page	5-10
5.2.8	Sharing database name tables	5-12
5.3	Defining and generating the database name table	5-13
5.4	Related information	5-15
<b>Chapter 6. Physical Database DDL Statements</b>		6-1
6.1	Statement summary	6-3
6.2	Components of a physical DDL statement	6-6
6.3	Naming conventions	6-7
6.3.1	Using lowercase letters in identifiers	6-8
6.3.2	Keywords as identifiers	6-8
6.3.3	Entity currency	6-8
6.4	Generic DISPLAY/PUNCH statement	6-10
6.4.1	Usage	6-11
6.4.2	Examples	6-11
6.5	DISPLAY/PUNCH ALL statement	6-12
6.5.1	Usage	6-15
6.5.2	Date selection criteria	6-20
6.5.3	Example	6-21
6.6	ARCHIVE JOURNAL statements	6-22
6.6.1	Usage	6-24
6.6.2	Examples	6-25
6.6.3	For more information	6-25
6.7	AREA statements	6-26
6.7.1	Usage	6-36

6.7.2 Examples	6-41
6.7.3 For more information	6-42
6.8 BUFFER statements	6-43
6.8.1 Usage	6-47
6.8.2 Examples	6-48
6.8.3 For more information	6-48
6.9 DBGROUP statements	6-49
6.9.1 Usage	6-51
6.9.2 Examples	6-51
6.9.3 For more information	6-51
6.10 DBNAME statements	6-52
6.10.1 Usage	6-55
6.10.2 Examples	6-56
6.10.3 For more information	6-57
6.11 DBTABLE statements	6-58
6.11.1 Usage	6-60
6.11.2 Examples	6-62
6.11.3 For more information	6-62
6.12 DISK JOURNAL statements	6-63
6.12.1 Usage	6-65
6.12.2 Examples	6-66
6.12.3 For more information	6-66
6.13 DMCL statements	6-67
6.13.1 Usage	6-80
6.13.2 Examples	6-81
6.13.3 For more information	6-81
6.14 FILE statements	6-82
6.14.1 Usage	6-87
6.14.2 Examples	6-88
6.14.3 For more information	6-88
6.15 JOURNAL BUFFER statements	6-89
6.15.1 Usage	6-91
6.15.2 Examples	6-91
6.15.3 For more information	6-92
6.16 SEGMENT statements	6-93
6.16.1 Usage	6-96
6.16.2 Examples	6-98
6.16.3 For more information	6-98
6.17 TAPE JOURNAL statements	6-99
6.17.1 Usage	6-101
6.17.2 Examples	6-101
6.17.3 For more information	6-102
6.18 Summary of physical database limits	6-103
<b>Chapter 7. Defining a Database Using SQL</b>	7-1
7.1 Executing SQL data description statements	7-4
7.2 Creating a schema	7-5
7.3 Creating a table	7-6
7.4 Defining a CALC key	7-8
7.5 Defining an index	7-9

7.6	Defining a referential constraint . . . . .	7-10
7.7	Dropping a default index . . . . .	7-12
7.8	Creating a view . . . . .	7-13
7.9	For further information . . . . .	7-15
<b>Chapter 8.</b>	<b>Defining a Database Using Non-SQL . . . . .</b>	<b>8-1</b>
8.1	About schemas and subschemas . . . . .	8-4
8.2	About the schema and subschema compilers . . . . .	8-6
8.3	Defining a schema . . . . .	8-7
8.3.1	SCHEMA statement . . . . .	8-7
8.3.2	AREA statements . . . . .	8-8
8.3.3	RECORD statements . . . . .	8-9
8.3.4	SET statements . . . . .	8-15
8.3.5	VALIDATE . . . . .	8-16
8.4	Defining a subschema . . . . .	8-18
8.4.1	Subschema statement . . . . .	8-18
8.4.2	AREA statements . . . . .	8-19
8.4.3	RECORD statements . . . . .	8-19
8.4.4	SET statements . . . . .	8-20
8.4.5	LOGICAL RECORD statements . . . . .	8-21
8.4.6	PATH-GROUP statements . . . . .	8-22
8.4.7	Subschema validation and generation . . . . .	8-23
8.5	Security checking . . . . .	8-25
8.5.1	Checking compiler security . . . . .	8-25
8.5.2	Checking registration override security . . . . .	8-26
8.5.3	Checking verb security . . . . .	8-27
8.5.4	Checking component security . . . . .	8-28
8.6	Establishing schema and subschema currency . . . . .	8-30
8.7	Reporting on schema and subschema definitions . . . . .	8-32
8.8	Related information . . . . .	8-33
<b>Chapter 9.</b>	<b>Using the Schema and Subschema Compilers . . . . .</b>	<b>9-1</b>
9.1	Online compiling . . . . .	9-4
9.2	Batch compiling . . . . .	9-6
9.3	Coding DDL schema and subschema statements . . . . .	9-7
9.3.1	Statement components . . . . .	9-7
9.3.2	Delimiting statements . . . . .	9-8
9.3.3	Compiler comments . . . . .	9-8
9.3.4	Input format . . . . .	9-9
9.3.5	Error handling . . . . .	9-10
9.4	Coding keywords, variables, and comment text . . . . .	9-12
9.4.1	Coding keywords . . . . .	9-12
9.4.2	Coding entity-occurrence names . . . . .	9-12
9.4.3	Coding user-supplied values . . . . .	9-13
9.4.4	Coding comment text . . . . .	9-14
9.5	About compiler-directive statements . . . . .	9-16
9.6	Output from the compilers . . . . .	9-17
9.6.1	Source code and load modules . . . . .	9-17
9.6.2	Schema and subschema listings . . . . .	9-18
<b>Chapter 10.</b>	<b>Compiler-Directive Statements . . . . .</b>	<b>10-1</b>

10.1 Overview	10-3
10.2 DISPLAY/PUNCH ALL statement	10-4
10.2.1 Usage	10-7
10.2.2 Example	10-10
10.3 DISPLAY/PUNCH IDD statement	10-11
10.3.1 Example	10-12
10.3.2 For more information	10-13
10.4 INCLUDE statement	10-14
10.4.1 Usage	10-14
10.4.2 Example	10-15
10.4.3 For more information	10-15
10.5 SET OPTIONS statement	10-16
10.5.1 Usage	10-27
10.5.2 Examples	10-32
10.5.3 For more information	10-32
10.6 SIGNOFF statement	10-33
10.6.1 Usage	10-33
10.7 SIGNON statement	10-34
10.7.1 Usage	10-35
10.7.2 For more information	10-37
<b>Chapter 11. Operations on Entities</b>	11-1
11.1 ADD operations	11-4
11.2 MODIFY operations	11-5
11.3 DELETE operations	11-6
11.4 VALIDATE operations	11-7
11.5 DISPLAY/PUNCH operations	11-8
11.5.1 Usage	11-10
11.5.2 Examples	11-11
11.5.3 For more information	11-11
<b>Chapter 12. Parameter Expansions</b>	12-1
12.1 Expansion of boolean-expression	12-4
12.1.1 Usage	12-6
12.2 Expansion of db-record-field	12-8
12.2.1 Usage	12-8
12.3 Expansion of lr-field	12-9
12.3.1 Usage	12-9
12.4 Expansion of module-specification	12-10
12.4.1 Usage	12-11
12.4.2 For more information	12-11
12.5 Expansion of user-specification	12-12
12.5.1 Usage	12-12
12.6 Expansion of user-options-specification	12-13
12.6.1 For more information	12-14
12.7 Expansion of version-specification	12-15
12.7.1 Examples	12-15
<b>Chapter 13. Schema Statements</b>	13-1
13.1 SCHEMA statement	13-4

13.1.1 Usage	13-12
13.1.2 Examples	13-13
13.1.3 Related information	13-14
13.2 AREA statement	13-15
13.2.1 Usage	13-19
13.2.2 Examples	13-20
13.2.3 Related information	13-20
13.3 RECORD statement	13-21
13.3.1 Usage	13-35
13.3.2 Examples	13-41
13.3.3 Related information	13-43
13.4 Element substatement	13-44
13.4.1 Usage	13-54
13.4.2 Examples	13-64
13.4.3 Related information	13-68
13.5 COPY ELEMENTS substatement	13-69
13.5.1 Usage	13-70
13.5.2 Examples	13-70
13.6 SET statement	13-72
13.6.1 Usage	13-85
13.6.2 Examples	13-88
13.6.3 Related information	13-91
13.7 VALIDATE statement	13-92
13.7.1 Usage	13-92
13.8 REGENERATE statement	13-93
13.8.1 Usage	13-93
<b>Chapter 14. Subschema Statements</b>	14-1
14.1 SUBSCHEMA statement	14-4
14.1.1 Usage	14-12
14.1.2 Examples	14-15
14.1.3 Related information	14-16
14.2 AREA statement	14-17
14.2.1 Usage	14-19
14.2.2 Example	14-20
14.2.3 Related information	14-20
14.3 RECORD statement	14-21
14.3.1 Usage	14-24
14.3.2 Example	14-27
14.4 SET statement	14-28
14.4.1 Usage	14-30
14.4.2 Example	14-30
14.5 LOGICAL RECORD statement	14-32
14.5.1 Usage	14-35
14.5.2 Examples	14-36
14.5.3 Related information	14-37
14.6 PATH-GROUP statement	14-38
14.6.1 Usage	14-57
14.6.2 Example	14-59
14.6.3 Related information	14-60
14.7 VALIDATE statement	14-61

14.7.1 Usage . . . . .	14-61
14.8 GENERATE statement . . . . .	14-62
14.9 LOAD MODULE statement . . . . .	14-63
14.9.1 Usage . . . . .	14-65
14.9.2 Examples . . . . .	14-66
14.9.3 Related information . . . . .	14-66
14.10 DISPLAY/PUNCH SCHEMA statement . . . . .	14-67
14.10.1 Example . . . . .	14-68
<b>Chapter 15. Writing Database Procedures . . . . .</b>	<b>15-1</b>
15.1 About database procedures . . . . .	15-3
15.2 Specifying a procedure . . . . .	15-4
15.3 Common uses of database procedures . . . . .	15-5
15.4 Coding database procedures . . . . .	15-7
15.4.1 Area procedures . . . . .	15-8
15.4.2 Record procedures . . . . .	15-8
15.4.3 Database procedure blocks . . . . .	15-8
15.4.4 Establishing communication between programs and procedures . . . . .	15-15
15.4.5 Invoking database procedures . . . . .	15-16
15.4.6 Link editing database procedures . . . . .	15-16
15.4.7 Calling non-reentrant or non-assembler database procedures . . . . .	15-17
15.4.8 Executing database procedures . . . . .	15-20
15.4.9 Resetting the error-status indicator . . . . .	15-20
15.5 Database procedure example . . . . .	15-22

---

## Volume 2. Database Maintenance

<b>Chapter 16. Allocating and Formatting Files . . . . .</b>	<b>16-1</b>
16.1 Making files accessible to CA-IDMS/DB . . . . .	16-3
16.2 Types of files . . . . .	16-4
16.3 File access methods . . . . .	16-5
16.4 Creating disk files . . . . .	16-7
16.4.1 File characteristics . . . . .	16-8
16.5 Formatting files . . . . .	16-10
16.6 Considerations for native VSAM files . . . . .	16-11
16.7 Related information . . . . .	16-12
<b>Chapter 17. Buffer Management . . . . .</b>	<b>17-1</b>
17.1 Planning database buffers . . . . .	17-3
17.1.1 How many buffers do you need? . . . . .	17-3
17.1.2 How many pages should a buffer contain? . . . . .	17-3
17.1.3 How large should a buffer page be? . . . . .	17-5
17.1.4 Choosing a method for storage acquisition . . . . .	17-5
17.2 Managing buffers dynamically . . . . .	17-7
17.3 Tuning buffers for performance . . . . .	17-8
17.4 Using chained reads . . . . .	17-9
17.5 Using read and write drivers . . . . .	17-11
17.6 Related information . . . . .	17-12

<b>Chapter 18. Journaling Procedures</b>	18-1
18.1 About journaling	18-3
18.1.1 Journaling under the central version	18-3
18.1.2 Journaling in local mode	18-4
18.2 About journal files	18-5
18.2.1 Journal record entries	18-5
18.2.2 Checkpoints	18-6
18.3 Offloading disk journal files	18-9
18.3.1 When CA-IDMS/DB switches journal files	18-9
18.3.2 How to offload the disk journal	18-10
18.3.3 After system shutdown	18-11
18.4 User exits and reports for journal management	18-12
18.5 Influencing journaling performance	18-13
18.5.1 Reducing journal file I/O	18-13
18.5.2 Improving warmstart performance	18-14
18.6 Related information	18-16
<b>Chapter 19. Backup and Recovery</b>	19-1
19.1 About database backup and recovery	19-3
19.2 Backup procedures	19-4
19.2.1 Back up after a normal system shutdown	19-5
19.2.2 Backup while the DC/UCF system is active	19-5
19.2.3 Back up before and after local mode jobs	19-10
19.2.4 Automating the backup process	19-11
19.3 Automatic recovery	19-14
19.3.1 Warmstart	19-14
19.3.2 Automatic rollback	19-16
19.4 Manual recovery	19-18
19.4.1 Recovery from a quiesced backup	19-19
19.4.2 Recovery from a hot backup	19-21
19.4.3 Reducing recovery time	19-28
19.4.4 Recovering a large number of files	19-30
19.5 Recovery procedures after a warmstart failure	19-31
19.6 Recovery procedures from database file I/O errors	19-33
19.7 Recovery procedures from journal file I/O errors	19-37
19.8 Recovery procedures for local mode operations	19-40
19.8.1 No journaling	19-40
19.8.2 Journaling to a tape device	19-40
19.8.3 Journaling to a disk device	19-40
19.8.4 Using an incomplete journal file	19-40
19.9 Recovery procedures for mixed-mode operations	19-42
19.10 Data sharing recovery considerations	19-44
19.11 Considerations for recovery of native VSAM files	19-47
<b>Chapter 20. Loading a Non-SQL Defined Database</b>	20-1
20.1 About database loading	20-3
20.2 Loading database records using FASTLOAD	20-4
20.2.1 General considerations	20-4
20.3 FASTLOAD procedure	20-6
20.4 Loading database records using a user-written program	20-7
20.4.1 Organizing input data for a user-written program	20-7

20.4.2 Loading the database . . . . .	20-9
20.5 Related information . . . . .	20-11
<b>Chapter 21. Loading an SQL-Defined Database . . . . .</b>	<b>21-1</b>
21.1 About database loading . . . . .	21-3
21.2 Loading considerations . . . . .	21-7
21.3 Contents of the input file . . . . .	21-10
21.4 Loading procedures . . . . .	21-12
21.4.1 Steps that apply to all load procedures . . . . .	21-12
21.4.2 Full load procedure . . . . .	21-13
21.4.3 Phased load procedure . . . . .	21-13
21.4.4 Segmented load procedure . . . . .	21-15
21.4.5 Stepped load procedure . . . . .	21-16
21.5 Related information . . . . .	21-20
<b>Chapter 22. Monitoring and Tuning Database Performance . . . . .</b>	<b>22-1</b>
22.1 Monitoring guidelines . . . . .	22-3
22.2 Monitoring facilities . . . . .	22-4
22.3 Items to monitor and tune . . . . .	22-5
22.3.1 Journal use . . . . .	22-5
22.3.2 Buffer utilization . . . . .	22-6
22.3.3 Space management and database design . . . . .	22-7
22.3.4 Indexing efficiency . . . . .	22-8
22.3.5 Database locks . . . . .	22-9
22.3.6 Longterm locks . . . . .	22-13
22.3.7 SQL processing . . . . .	22-14
22.4 Reducing I/O . . . . .	22-15
22.4.1 Through database reorganization . . . . .	22-15
22.4.2 Through application design . . . . .	22-16
22.4.3 Through database design . . . . .	22-16
22.4.4 By using UPDATE STATISTICS (SQL-accessed databases) . . . . .	22-16
<b>Chapter 23. Dictionaries and Runtime Environments . . . . .</b>	<b>23-1</b>
23.1 About dictionaries . . . . .	23-3
23.1.1 Physical components of a dictionary . . . . .	23-3
23.1.2 Logical components of a dictionary . . . . .	23-4
23.1.3 Assigning dictionary areas to segments . . . . .	23-5
23.1.4 Sharing dictionary areas . . . . .	23-6
23.2 CA-supplied dictionary definitions . . . . .	23-8
23.2.1 Logical database definitions . . . . .	23-9
23.2.2 Protocols, nondatabase structures, and modules . . . . .	23-11
23.3 Defining new dictionaries . . . . .	23-13
23.3.1 Defining new catalog components . . . . .	23-13
23.3.2 Defining new application dictionaries . . . . .	23-14
23.3.3 Defining new system dictionaries . . . . .	23-16
23.4 Establishing a default dictionary . . . . .	23-19
23.5 About runtime environments . . . . .	23-20
23.5.1 SYSIDMS parameter file . . . . .	23-22
23.5.2 Establishing session options . . . . .	23-23
23.6 Related information . . . . .	23-25

<b>Chapter 24. Migrating from Test to Production</b>	24-1
24.1 About migration	24-3
24.2 Establishing migration procedures	24-4
24.3 Implementing migration procedures	24-5
24.3.1 Step 1: Determine the types of components to migrate	24-5
24.3.2 Step 2: Determine the sequence of migration	24-9
24.3.3 Step 3: Identify the individual components	24-11
24.3.4 Step 4: Migrate the components	24-11
24.4 Identification aids	24-12
24.5 Migration tools	24-15
24.6 General methods	24-17
24.6.1 Using the DISPLAY statement	24-17
24.6.2 Using the PUNCH statement	24-18
24.6.3 Using the mapping compiler and mapping utility	24-22
24.6.4 For SQL-defined entities	24-23
24.7 Additional considerations	24-25
24.7.1 Additional tasks	24-25
<b>Chapter 25. Modifying Physical Database Definitions</b>	25-1
25.1 Modifications you can make	25-3
25.2 Making the changes available under the central version	25-7
25.3 Dynamic DMCL management	25-8
25.4 Changing a file's access method	25-10
25.4.1 Step 1: Expand the page size	25-10
25.4.2 Step 4: Copy the data to the new file	25-10
25.5 Increasing the size of an area	25-12
25.5.1 Increasing an area's page size	25-12
25.5.2 Extending an area's page range	25-13
25.6 Adding or dropping files associated with an area	25-14
25.7 Changing the size of a disk journal	25-15
25.8 Changing the access method of a disk journal	25-16
25.9 Related information	25-17
<b>Chapter 26. Modifying Database Name Tables</b>	26-1
26.1 Changes you can make	26-3
26.2 Procedure for modifying database name tables	26-4
26.3 Related information	26-5
<b>Chapter 27. About Modifying SQL-Defined Databases</b>	27-1
27.1 What you can modify	27-3
27.2 Methods for modifying	27-4
<b>Chapter 28. Modifying Schema, View, and Table Definitions</b>	28-1
28.1 Maintaining schemas	28-4
28.1.1 Dropping an existing schema	28-4
28.1.2 Modifying a schema	28-4
28.2 Maintaining views	28-5
28.2.1 Dropping a view	28-5
28.2.2 Modifying a view	28-5
28.3 Maintaining tables	28-7
28.3.1 Creating a table	28-7

28.3.2	Dropping a table	28-7
28.3.3	Adding a column to a table	28-8
28.3.4	Dropping a column from a table	28-9
28.3.5	Changing the characteristics of a column	28-10
28.3.6	Adding or removing data compression	28-10
28.3.7	Adding a new check constraint	28-10
28.3.8	Dropping a check constraint	28-11
28.3.9	Modifying a check constraint	28-11
28.3.10	Revising the estimated row count for a table	28-11
28.3.11	Changing a table's area	28-12
28.3.12	Dropping the default index associated with a table	28-12
28.4	Dropping and recreating a table	28-14
28.4.1	Method 1 — Using DDL and DML statements	28-14
28.4.2	Method 2 — Using DDL and utility statements	28-16
<b>Chapter 29. Modifying Indexes, CALC Keys, and Referential Constraints</b>		29-1
29.1	Maintaining indexes	29-4
29.1.1	Creating an index	29-4
29.1.2	Dropping an index	29-4
29.1.3	Changing index characteristics/ moving an index	29-5
29.2	Maintaining CALC keys	29-6
29.2.1	Creating a CALC key	29-6
29.2.2	Dropping a CALC key	29-6
29.3	Maintaining referential constraints	29-7
29.3.1	Creating a referential constraint	29-7
29.3.2	Dropping a referential constraint	29-7
29.3.3	Modifying referential constraint tuning characteristics	29-8
<b>Chapter 30. About Modifying Non-SQL Defined Databases</b>		30-1
30.1	Types of modifications	30-3
30.2	Overview	30-4
30.2.1	Methods for modifying	30-4
30.2.2	Procedure for modifying the non-SQL definitions	30-5
30.2.3	RESTRUCTURE SEGMENT utility statement	30-7
30.2.4	UNLOAD/RELOAD utility statements	30-7
30.2.5	MAINTAIN INDEX utility statement	30-8
<b>Chapter 31. Modifying Schema Entities</b>		31-1
31.1	Modifications to an unloaded database	31-4
31.2	Schema modifications	31-5
31.2.1	Deleting a schema	31-5
31.2.2	Changing schema characteristics	31-5
31.3	Area modifications	31-6
31.3.1	Adding or deleting an area	31-6
31.3.2	Changing area characteristics	31-7
31.4	Record modifications	31-8
31.4.1	Adding schema records	31-8
31.4.2	Deleting schema records	31-8
31.4.3	Changing a record's CALC key	31-9
31.4.4	Changing the DUPLICATES option on a CALC or SORT key	31-11

31.4.5	Changing the location mode of a record	31-12
31.4.6	Changing a record's area	31-13
31.4.7	Modifying record elements	31-14
31.4.8	Changing other record characteristics	31-15
31.4.9	Adding and dropping database procedures	31-16
31.5	Set modifications	31-17
31.5.1	Adding or deleting a set	31-17
31.5.2	Changing set mode	31-18
31.5.3	Adding and dropping set pointers	31-19
31.5.4	Changing set order	31-20
31.5.5	Changing set membership options	31-21
31.6	Index modifications	31-23
31.6.1	Adding or deleting system-owned indexes	31-23
31.6.2	Changing the location of an index	31-24
31.6.3	Changing index characteristics	31-24
31.6.4	Adding or deleting index pointers	31-25
<b>Chapter 32.</b>	<b>Modifying Subschema Entities</b>	32-1
32.1	Modifying or deleting a subschema	32-4
32.1.1	Modifying a subschema	32-4
32.1.2	Deleting a subschema	32-4
32.2	Adding, modifying, or deleting a record	32-6
32.3	Adding, modifying, or deleting a set	32-7
32.4	Adding, modifying, or deleting an area	32-8
32.5	Adding, modifying, or deleting a logical record or path group	32-9
<b>Chapter 33.</b>	<b>Space Management</b>	33-1
33.1	About space management	33-3
33.2	Database pages	33-4
33.3	Database keys	33-7
33.4	Area space management	33-10
33.4.1	SR1 records	33-11
33.4.2	Space management pages	33-12
<b>Chapter 34.</b>	<b>Record Storage and Deletion</b>	34-1
34.1	Record storage	34-3
34.1.1	Storing CALC records	34-4
34.1.2	Clustering records	34-7
34.1.2.1	Clustering records around a chained set	34-7
34.1.2.2	Storing records via an indexed set	34-9
34.1.3	Storing variable-length records	34-11
34.1.4	Relocated records	34-14
34.2	Record deletion	34-16
34.2.1	Physical deletion	34-16
34.2.2	Logical deletion	34-18
<b>Chapter 35.</b>	<b>Chained Set Management</b>	35-1
35.1	About chained sets	35-3
35.2	Chained sets	35-4
35.2.1	Connecting records to chained sets	35-5
35.2.2	Disconnecting records	35-6

35.2.3	Retrieving records	35-7
<b>Chapter 36. Index Management</b> . . . . . 36-1		
36.1	About indexed sets	36-3
36.1.1	Structure of indexes	36-5
36.1.2	Connecting records to indexed sets	36-11
36.1.2.1	Connecting members to unsorted indexed sets	36-11
36.1.2.2	Connecting members to sorted indexed sets	36-14
36.1.3	Disconnecting records from indexed sets	36-15
36.1.4	Retrieving indexed records	36-16
<b>Chapter 37. Lock Management</b> . . . . . 37-1		
37.1	Controlling access to CA-IDMS databases	37-3
37.2	Readying areas	37-4
37.2.1	Area ready modes	37-4
37.2.2	Central version area status	37-7
37.2.3	Default ready mode using navigational DML	37-8
37.2.4	Ready modes and SQL access	37-8
37.3	Physical area locks	37-11
37.3.1	About physical area locks	37-11
37.3.2	Controlling update access	37-11
37.4	Locking within central version	37-13
37.4.1	Logical locks	37-13
37.4.2	Types of locks	37-14
37.4.3	Logical area locks	37-15
37.4.4	Area locking for SQL transactions	37-16
37.4.5	Record locks	37-18
37.4.6	System generation options affecting record locking	37-19
37.5	Locking within a data sharing group	37-21
37.5.1	Inter-CV-interest	37-21
37.5.2	Global transaction locks	37-21
37.5.3	Proxy locks	37-22
37.5.4	Page locks	37-23
37.6	Controlling access to native VSAM files	37-24
37.7	Deadlocks	37-25
37.7.1	How the system detects a deadlock	37-25
37.7.2	Global deadlock detection	37-26

---

## Appendixes

<b>Appendix A. Sample Physical Database Definition</b>	A-1
<b>Appendix B. Sample SQL Database Definition</b>	B-1
<b>Appendix C. Sample Non-SQL Database Definition</b>	C-1
<b>Appendix D. Native VSAM Considerations</b>	D-1
D.1 Native VSAM data set structures	D-4
D.2 CA-IDMS/DB native VSAM definitions	D-5

D.2.1 Schema definition . . . . .	D-5
D.2.2 DMCL definition . . . . .	D-6
D.3 DML functions with native VSAM . . . . .	D-8
<b>Appendix E. Batch Compiler Execution JCL . . . . .</b>	<b>E-1</b>
E.1 Overview of batch compilation . . . . .	E-4
E.2 OS/390 JCL . . . . .	E-7
E.2.1 Schema compiler . . . . .	E-7
E.2.2 Subschema compiler . . . . .	E-8
E.3 VSE/ESA JCL . . . . .	E-10
E.3.1 =COPY facility . . . . .	E-10
E.3.2 Schema compiler . . . . .	E-10
E.3.3 Subschema compiler . . . . .	E-12
E.3.4 IDMSLBLS procedure . . . . .	E-14
E.4 CMS commands . . . . .	E-20
E.4.1 Schema compiler . . . . .	E-20
E.4.2 Subschema compiler . . . . .	E-21
E.5 BS2000/OSD JCL . . . . .	E-23
E.5.1 =COPY facility . . . . .	E-23
E.5.2 Schema compiler . . . . .	E-23
E.5.3 Subschema compiler . . . . .	E-25
<b>Appendix F. System Record Types . . . . .</b>	<b>F-1</b>
<b>Appendix G. User-Exit Program for Schema and/or Subschema Compiler . . . . .</b>	<b>G-1</b>
G.1 When a user exit is called . . . . .	G-4
G.2 Rules for writing the user-exit program . . . . .	G-5
G.3 Control blocks and sample user-exit programs . . . . .	G-7
G.3.1 User-exit control block . . . . .	G-7
G.3.2 SIGNON Element Block . . . . .	G-7
G.3.3 SIGNON Block . . . . .	G-8
G.3.4 Entity control block . . . . .	G-8
G.3.5 Card-image control block . . . . .	G-9
G.4 Sample user-exit program for Schema and/or Subschema Compilers . . . . .	G-10
<b>Appendix H. SYSIDMS Parameter File . . . . .</b>	<b>H-1</b>
H.1 Parameter Summary . . . . .	H-3
H.2 Parameter Descriptions . . . . .	H-6
<b>Index . . . . .</b>	<b>X-1</b>

# How to use this manual

---

## **What this manual is about**

This manual contains all information necessary to define, load, and administer a CA-IDMS/DB database.

## Who should use this manual

This manual is intended for anyone who is responsible for administering one or more CA-IDMS/DB databases as well as for those whose responsibility lies in administering a portion of the database, such as database definition.

# How this manual is organized

This manual is divided into two volumes as follows:

- **Volume 1 — CA-IDMS/DB Database Definition**

- **Chapter 1** — describes the CA-IDMS environment
- **Chapter 2** — describes defining physical databases
- **Chapter 3** — describes defining segments, files, and areas
- **Chapter 4** — describes defining, generating, and punching a DMCL
- **Chapter 5** — discusses defining a database name table
- **Chapter 6** — discusses physical database DDL statements
- **Chapter 7** — describes defining a database using SQL
- **Chapter 8** — describes defining a database using non-SQL
- **Chapter 9** — describes using the schema and subschema compilers
- **Chapter 10** — discusses compiler-directive statements
- **Chapter 11** — discusses operations on entities
- **Chapter 12** — discusses parameter expansions
- **Chapter 13** — discusses schema statements
- **Chapter 14** — discusses subschema statements
- **Chapter 15** — discusses writing database procedures

- **Volume 2 — Database Maintenance**

- **Chapter 16** — discusses allocating and formatting files
- **Chapter 17** — discusses buffer management
- **Chapter 18** — discusses journaling procedures
- **Chapter 19** — discusses backup and recovery
- **Chapter 20** — describes loading a non-SQL defined database
- **Chapter 21** — describes loading an SQL-defined database
- **Chapter 22** — discusses monitoring and tuning database performance
- **Chapter 23** — describes dictionaries and runtime environments
- **Chapter 24** — discusses migrating from test to production
- **Chapter 25** — discusses modifying physical database definitions
- **Chapter 26** — discusses modifying database name tables
- **Chapter 27** — discusses modifying SQL-defined databases
- **Chapter 28** — describes modifying schema, view, and table definitions
- **Chapter 29** — discusses modifying indexes, CALC keys, and referential constraints

- **Chapter 30** — discusses modifying non-SQL defined databases
- **Chapter 31** — describes modifying schema entities
- **Chapter 32** — describes modifying subschema entities
- **Chapter 33** — describes space management
- **Chapter 34** — describes record storage and deletion
- **Chapter 35** — discusses chained set management
- **Chapter 36** — discusses index management
- **Chapter 37** — describes lock management
- **Appendix A** — presents a sample physical database definition
- **Appendix B** — presents a sample SQL database definition
- **Appendix C** — presents a sample non-SQL database definition
- **Appendix D** — discusses native VSAM considerations
- **Appendix E** — discusses batch compiler execution JCL
- **Appendix F** — discusses system record types
- **Appendix G** — discusses procedures for coding a user-exit program
- **Appendix H** — presents SYSIDMS parameters

## Related documentation

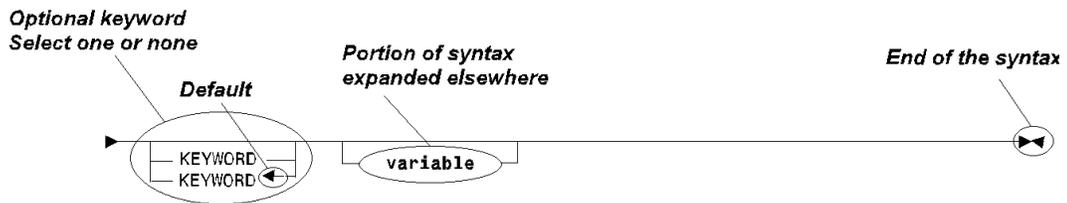
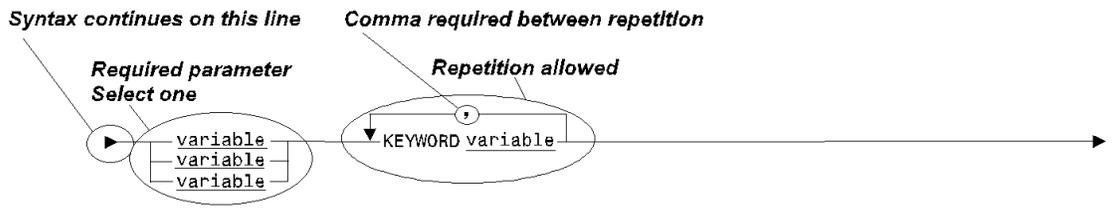
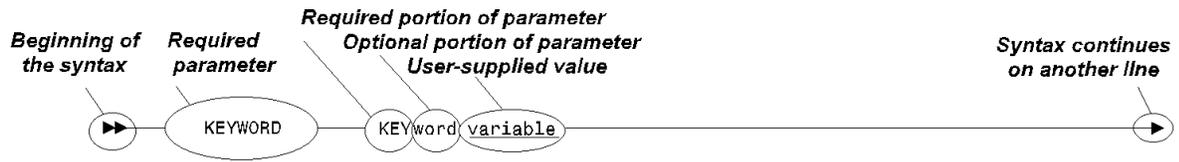
- *CA-IDMS Utilities*
- *CA-IDMS SQL Reference*
- *CA-IDMS Database Design*
- *CA-IDMS Database Administration Quick Reference*
- *CA-IDMS SQL Programming*
- *CA-IDMS Navigational DML Programming*

# Understanding Syntax Diagrams

Look at the list of notation conventions below to see how syntax is presented in this manual. The example following the list shows how the conventions are used.

UPPERCASE OR SPECIAL CHARACTERS	Represents a required keyword, partial keyword, character, or symbol that must be entered completely as shown.
lowercase	Represents an optional keyword or partial keyword that, if used, must be entered completely as shown.
<u>underlined lowercase</u>	Represents a value that you supply.
←	Points to the default in a list of choices.
<b>lowercase bold</b>	Represents a portion of the syntax shown in greater detail at the end of the syntax or elsewhere in the document.
▶—————▶	Shows the beginning of a complete piece of syntax.
—————▶◀	Shows the end of a complete piece of syntax.
—————▶	Shows that the syntax continues on the next line.
▶—————▶	Shows that the syntax continues on this line.
—————▶	Shows that the parameter continues on the next line.
▶—————▶	Shows that a parameter continues on this line.
▶— parameter —▶	Shows a required parameter.
▶— parameter —▶ └── parameter ─┘	Shows a choice of required parameters. You must select one.
▶— parameter —▶ └── parameter ─┘	Shows an optional parameter.
▶— parameter —▶ └── parameter ─┘	Shows a choice of optional parameters. Select one or none.
▶— parameter —▶ └── parameter ─┘	Shows that you can repeat the parameter or specify more than one parameter.
▶— parameter —▶ └── parameter ─┘	Shows that you must enter a comma between repetitions of the parameter.

# Sample Syntax Diagram



## Volume 2. Database Maintenance

---



# Chapter 16. Allocating and Formatting Files

---

- 16.1 Making files accessible to CA-IDMS/DB . . . . . 16-3
- 16.2 Types of files . . . . . 16-4
- 16.3 File access methods . . . . . 16-5
- 16.4 Creating disk files . . . . . 16-7
  - 16.4.1 File characteristics . . . . . 16-8
- 16.5 Formatting files . . . . . 16-10
- 16.6 Considerations for native VSAM files . . . . . 16-11
- 16.7 Related information . . . . . 16-12



## 16.1 Making files accessible to CA-IDMS/DB

**Steps:** To make a file accessible to CA-IDMS/DB, follow these steps:

1. Use physical DDL statements to: define the file within a new or existing segment and associate it with one or more new or existing areas; include the segment definition, with any file and/or area overrides, in a DMCL
2. Make available the DMCL in which the file's segment is included
3. Create the file using facilities provided by your operating system
4. Format the file

This chapter describes steps 3 and 4.

## 16.2 Types of files

**Available options:** CA-IDMS/DB can access data stored in the following types of files:

File type	Access method	File structure
Direct access	EXCP (OS/390,VSE/ESA)	A file block corresponds to a database page
Physical sequential	EXCP (OS/390) SAM (VSE/ESA)	A file block corresponds to a database page
CMS format minidisk	DASD block I/O (VM/ESA)	A file block corresponds to a database page
PAM	UPAM (BS2000/OSD)	One or more file blocks contain a single database page
CA-IDMS/DB VSAM	VSAM (OS/390, VSE/ESA)	An ESDS VSAM file in which each Control Interval contains a single database page plus 8 bytes of control information used by VSAM
Native VSAM	VSAM (OS/390, VSE/ESA)	An ESDS, KSDS, or RRDS VSAM file or PATH in which each VSAM record corresponds to an IDMS record

**Specifying the file's type in the FILE statement:** When you define a file using a physical DDL FILE statement, you specify the file's type using these parameters:

FILE statement parameter	Corresponding file type
NONVSAM or BDAM	Direct access (OS/390, VSE/ESA) Physical sequential (OS/390) CMS format minidisk (VM/ESA) PAM (BS2000/OSD)
VSAM	VSAM (OS/390, VSE/ESA)
ESDS KSDS RRDS PATH	Native VSAM (OS/390, VSE/ESA)

---

## 16.3 File access methods

**Determines how CA-IDMS/DB gains access to files:** When an application program issues a call to CA-IDMS/DB for retrieval or storage of a record or row of data, CA-IDMS/DB maps the database page that contains the record or row to the corresponding block or blocks in the file. The means by which this mapping occurs varies according to the access method in use:

- EXCP (OS/390,VSE/ESA)
- SAM (VSE/ESA)
- DASD Block I/O (VM/ESA)
- PAM (BS2000/OSD)
- VSAM (OS/390, VSE/ESA)

**EXCP access method:** The EXCP access method is used in OS/390 and VSE/ESA in order to take advantage of extended addressing. Using EXCP as an access method, CA-IDMS/DB maps the database page number to a relative track and record number. The database page size must equal the block size of the file.

**SAM access method:** Using SAM as an access method, CA-IDMS/DB maps the first database page number to a relative block number (RBN) within the sequential access file. It then reads forward sequentially from that RBN. The database page size must equal the block size of the file.

**DASD block I/O:** In VM/ESA, all CA-IDMS/DB files are allocated as separate minidisks and are accessed using DASD Block I/O.

►► For more information, refer to *CA-IDMS Installation and Maintenance Guide — VM/ESA*.

**UPAM access method:** Using UPAM as an access method, CA-IDMS/DB maps the database page number to a relative block number and requests one or more blocks using the UPAM access method.

Using the UPAM access method, CA-IDMS/DB can take advantage of extended addressing. All PAM file access macros are compiled with the PARMOD=31 parameter.

**VSAM access method:** CA-IDMS/DB can take advantage of extended addressing when accessing data by means of the VSAM access method. All VSAM macros use the AMODE=31 and RMODE=31 parameters. Therefore, all VSAM control blocks are allocated above the 16-megabyte line.

**Accessing VSAM database files:** Using VSAM as an access method to VSAM database files, CA-IDMS/DB maps the database page number to a VSAM control interval and issues a request to VSAM for that control interval.

**Accessing native VSAM files:** Existing VSAM files to be accessed by CA-IDMS/DB are referred to as **native VSAM files** because they are not formatted into pages as is the case with all other file types. CA-IDMS/DB accesses native VSAM files using VSAM record-level services. A native VSAM file can have one of the following structures:

- Key-sequenced (KSDS)
- Entry-sequenced (ESDS)
- Relative record (RRDS)

Regardless of the type of file being accessed, each is represented by a single record type described to CA-IDMS/DB in a non-SQL schema definition.

►► For more information, see 16.6, “Considerations for native VSAM files” on page 16-11 later in this chapter.

**Choosing between VSAM and non-VSAM file types:** In OS/390 and VSE/ESA, you may define database files as either VSAM or non-VSAM.

**VSE/ESA:** To define non-VSAM files on FBA disk devices (type 3310 or type 3370), use a sequential label (that is, an SD attribute on the DLBL statement).

## 16.4 Creating disk files

**Use operating system facilities:** Use facilities provided by your operating system to create and catalog the files.

**File placement on disk:** You can reduce I/O response time by planning where you place files on a disk. In general, spread high activity files across disk devices and channels. Particularly, consider the placement of disk journal files used by systems engaged in high-volume update activity.

**Valid disk devices for archive and tape journal files:** The table below summarizes the disk device types CA-IDMS/DB supports for archive and tape journal files:

System	Device types
OS/390	Any supported by QSAM
VSE/ESA	Any supported by SAM
BS2000/OSD	Any supported by DMS
VM/ESA	Any supported by QSAM

**Valid device types for disk journal files and database files:** The table below summarizes the device types CA-IDMS/DB supports for disk journal files and database files:

System	Device types
OS/390	Any supported by BDAM or VSAM
VSE/ESA	Any supported by SAM
BS2000/OSD	Any supported by UPAM
VM/ESA	Any supported by DASD Block I/O

**Maximum area page sizes:** When allocating non-VSAM files in OS/390 and VSE/ESA operating systems, the page size of an area is restricted by the track size of the disk device being used. The table below identifies the maximum page size for non-VSAM files in OS/390 and VSE/ESA operating systems:

Disk device	Maximum page size	Bytes per track
2311	3624	3625
2314	7292	7294
2321	2000	2092
3330/3330B	13028	13030
3340	8368	8535
3350	19068	19254
3375	32764	36000
3380	32764	47476
3390	32764	56664

### 16.4.1 File characteristics

**Non-VSAM files in OS/390:** To create a non-VSAM file in OS/390, use a JCL statement or a facility such as TSO. The DCB characteristics of the file must be:

Parameter	Value
DSORG	PS or DA
BLKSIZE	Page size of the area(s) mapped to the file
RECFM	F

**PAM files:** To create a PAM file, you use the BS2000/OSD /FILE command. All PAM files have a block size of 2048 and a database page is mapped to one or more blocks. In choosing a page size for an area that will be mapped to a PAM file, the page size should be a multiple of 2048 to optimize the use of disk space.

**VSAM files:** To create a VSAM database or journal file, you use the IDCAMS utility from IBM. The following IDCAMS statements are used:

- **DEFINE SPACE** — Allocates disk space for one or more VSAM files; alternatively, the database file can be defined in its own data space
- **DEFINE CLUSTER** — Creates the database file as an ESDS VSAM cluster specifying the following attributes:

---

RECORDS	Assign: <ul style="list-style-type: none"> <li>▪ PRIMARY SPACE as the number of pages mapped to the file</li> <li>▪ SECONDARY SPACE as the value 2</li> </ul>
RECORDSIZE	Assign: <ul style="list-style-type: none"> <li>▪ AVERAGE as the page size of the area mapped to the file</li> <li>▪ MAXIMUM as the page size of the area mapped to the file</li> </ul>
CONTROL INTERVALSIZE	<ul style="list-style-type: none"> <li>▪ For database files, assign a value at least 8 bytes larger than the page size of the area mapped to the file, but less than twice the page size minus 8 ((2 * page size)- 8)</li> <li>▪ For disk journal files, assign a value that is the same as the page size of the journal buffer</li> </ul>
SHAREOPTIONS	Assign (3 3)
REUSE SUBALLOCATE or UNIQUE	
NONSPANNED	
NONINDEXED	

---

**For more information**

- About creating CMS-format minidisks to be used by CA-IDMS/DB, refer to *CA-IDMS Installation and Maintenance — VM/ESA*
- About defining and accessing native VSAM files, see 16.6, “Considerations for native VSAM files” on page 16-11 later in this chapter.

## 16.5 Formatting files

**What formatting means:** **Formatting** means initializing database or disk journal files into database pages or blocks according to information provided by the DMCL.

**CAUTION:**

**NEVER format native VSAM files**

**Formatting database files:** When you issue a FORMAT command against a database file, CA-IDMS/DB:

- Establishes space management pages (SMPs) for the area(s) that map to the file
- Initializes the space management entry for each database page
- Establishes a header and footer on each database page
- Sets all data portions of database pages to binary zeros

**Formatting journal files:** When you issue a FORMAT command against a disk journal file, CA-IDMS/DB formats the file into blocks according to the journal file specification in the DMCL module. The disk journal file contains:

- Journal header records at the beginning
- Binary zeros in the remainder

**Before you begin:** Before you format a file, the DMCL that contains the file definition must be available. The DMCL provides the information CA-IDMS/DB needs to format the file into database pages or journal file blocks.

**Formatting options:** You can specify four options on the FORMAT utility statement. The table below identifies when to use these options:

Action	FORMAT option
Format <i>newly</i> -created database file(s)	FILE or SEGMENT
Re-format non-empty database file(s) 1	AREA or SEGMENT*
Format a disk journal file	JOURNAL
1 IDMS VSAM files must use the AREA option	

**Note:** \*IDMS VSAM files can only use the AREA option.

**Example:** The following example instructs CA-IDMS/DB to format all the database files contained in segment EMPSEG:

```
format segment empseg;
```

## 16.6 Considerations for native VSAM files

**About native VSAM files:** A native VSAM file is a file that is already defined to VSAM and contains VSAM records. Even though a native VSAM file is not structured as a CA-IDMS/DB database file, users can gain access to it using CA-IDMS/DB DML. To access data in native VSAM data sets, CA-IDMS/DB converts DML statements issued by an application program into record-level (not control-interval) VSAM requests and passes control to VSAM. A CA-IDMS/DB local run unit or the central version appears to VSAM as a single application that:

1. Opens VSAM data clusters
2. Activates VSAM paths using local-shared resources (LSR) or non-shared resources (NSR)
3. Accesses data records
4. Closes the clusters and paths

**Native VSAM files contain data:** CA-IDMS/DB can access native VSAM files only if they contain at least one record; that is, the files cannot be empty. This also implies that empty native VSAM files cannot be loaded using CA-IDMS/DB services.

**Defining native VSAM to IDMS:** Before an existing VSAM file can be accessed using CA-IDMS/DB DML statements, both a logical and physical description must be provided using non-SQL schema and physical DDL statements.

►► For more information about defining native VSAM files, see Appendix D, “Native VSAM Considerations” on page D-1.

## 16.7 Related information

- About creating and formatting VM/ESA files, *CA-IDMS Installation and Maintenance — VM/ESA*
- About database file definition and modification, see Chapter 3, “Defining Segments, Files, and Areas” on page 3-1 and Chapter 25, “Modifying Physical Database Definitions” on page 25-1
- About disk journal file definition and modification, see Chapter 4, “Defining, Generating, and Punching a DMCL” on page 4-1 and Chapter 25, “Modifying Physical Database Definitions” on page 25-1
- About syntax for the FILE and DISK JOURNAL statements, see Chapter 6, “Physical Database DDL Statements” on page 6-1
- About loading files, see Chapter 20, “Loading a Non-SQL Defined Database” on page 20-1 and Chapter 21, “Loading an SQL-Defined Database” on page 21-1
- About IDCAMS, see the appropriate IBM publication.
- About using native VSAM files, *CA-IDMS Database Design Guide*

# Chapter 17. Buffer Management

---

- 17.1 Planning database buffers . . . . . 17-3
  - 17.1.1 How many buffers do you need? . . . . . 17-3
  - 17.1.2 How many pages should a buffer contain? . . . . . 17-3
  - 17.1.3 How large should a buffer page be? . . . . . 17-5
  - 17.1.4 Choosing a method for storage acquisition . . . . . 17-5
- 17.2 Managing buffers dynamically . . . . . 17-7
- 17.3 Tuning buffers for performance . . . . . 17-8
- 17.4 Using chained reads . . . . . 17-9
- 17.5 Using read and write drivers . . . . . 17-11
- 17.6 Related information . . . . . 17-12



## 17.1 Planning database buffers

**Tradeoffs to consider:** Buffers use space in main memory, but reduce the amount of I/O performed on behalf of your applications. You want to choose the optimal buffer attributes to achieve a balance between storage resources and I/O.

**What follows:** Considerations for assigning values to these attributes appear below, beginning with a discussion on how many database buffers to define.

### 17.1.1 How many buffers do you need?

**Multiple buffers allowed:** As a general rule, one large buffer is often adequate for most processing situations. However, you may need to define more buffers to:

- Enhance database performance
- Optimize storage use

**Separate buffers to enhance performance:** To enhance run-time performance, you can associate individual files with separate buffers. This reduces contention for buffer pages.

For example, you can assign a frequently-used index to a separate file and then assign the file to a separate buffer. Applications can access this index in its own buffer, while CA-IDMS/DB uses other buffers to hold database pages.

**Separate buffers to optimize storage use:** The size of a buffer page must be as large as the largest database page that uses the buffer. Therefore, you can optimize storage use by assigning files that contain the same or similar block sizes to the same buffer.

### 17.1.2 How many pages should a buffer contain?

**Minimum number of pages:** The minimum number of pages in a buffer is three. However, a value of at least five is recommended to avoid excessive database I/O operations and to reduce contention among transactions for space in the buffer.

**Maximum number of pages:** The maximum number of pages is constrained only by available memory resources. However, if you allocate *too many* pages, you may degrade performance by increasing the amount of virtual paging performed by the operating system.

**Choosing an optimum:** Choosing an optimum number of pages comes with experience gained from tuning your database. However, if most files in the DMCL use a common buffer, a rule of thumb indicates that the number of buffer pages should be at least three times the maximum number of anticipated concurrent database transactions.

**Manage the size of the buffer dynamically in response to need:** Once a database is in operation under the central version, you can dynamically change the number of pages in the central version buffer with a DCMT VARY BUFFER statement. By changing the size dynamically, you can determine the optimum size for the buffer by monitoring the **buffer utilization ratio**, which is described in 17.3, “Tuning buffers for performance” on page 17-8 below.

**Local mode vs. central version specifications:** You can size a buffer differently for local mode and central version use. This feature allows you to optimize use of memory resources. For example, you could specify that a particular buffer will hold 100 pages when used in local mode and 500 pages when used under the central version. Under local mode, the buffer is smaller because it supports only a single application; under the central version, the buffer is larger because it supports multiple, concurrent applications.

**Initial and maximum allocations under the central version:** Buffers defined to run under the central version can be assigned an initial number of pages and a maximum number of pages. Depending on the amount of system activity, you can use the DCMT VARY BUFFER command to change the number of pages in the buffer; for example, use the DCMT VARY BUFFER command to increase the number of buffer pages during peak system usage or to reduce the number of buffer pages at other times.

**You can use JCL to increment size of local mode buffer:** At OS/390 sites, you may want to increase the size of the buffer for a specific application, such as loading a database. You can do this without modifying the buffer definition by specifying additional buffer pages in the BUFNO parameter of the JCL statement identifying a file associated with the buffer. At runtime, CA-IDMS/DB acquires storage for the buffer equal to the number of pages specified in the DMCL's buffer definition plus the value assigned to BUFNO for each file associated with the buffer.

**Associating buffers with files cached externally:** In certain operating systems, you can cache database files in an external cache.

- In OS/390 you can cache files in a dataspace or in a shared cache residing in a coupling facility
- In VSE/ESA you can cache files in a dataspace

If a file is cached externally, CA-IDMS reads database pages from the cache into the database buffer. If it modifies the database page, CA-IDMS writes the modified page back to disk and to the cache. One advantage of a cache is a reduction in the number of I/Os to the file. Another advantage is that you may be able to reduce the number of pages in your buffer pool, relying on the cache to hold pages while not in use.

Dataspaces provide larger caching capabilities than database buffers (even those allocated above the 16-megabyte line). However, you must have sufficient expanded storage on your machine to support the use of dataspace. Without adequate storage, the paging overhead associated with the system can increase significantly.

If using a coupling facility cache, you must have enough coupling facility space to hold the most frequently accessed pages, in order to make its use worthwhile. An additional advantage of a coupling facility cache is that it can be shared by more than one central version.

External caching in a dataspace or coupling facility is not available for native VSAM files.

►► For more information about using a shared cache, refer to *CA-IDMS System Operations*.

**Using Batch LSR for VSAM files:** At OS/390 sites, VSAM database files can make use of IBM's Batch Shared Resources Subsystem (Batch LSR) by specifying the SUBSYS JCL parameter. At runtime, CA-IDMS/DB opens the VSAM database file and the VSAM Batch LSR subsystem converts the buffer management technique to LSR processing and allows the buffer pool to be created in hiperspace. Batch LSR is also supported for native VSAM files.

**Batch LSR improves performance for actively used files:** By using Batch LSR, you can reduce the number of pages in the buffer associated with the file in your DMCL because VSAM and the Batch LSR subsystem can create a large buffer pool in hiperspace which will minimize the number of I/Os. This feature offers performance improvements for files that are actively used.

**SUBSYS subparameters:** Use of the Batch LSR subsystem and the number and location of the buffers is controlled by use of the SUBSYS JCL parameter and its subparameters. Use the MSG=I subparameter to display the batch LSR subsystem messages on the job log. Do *not* use DEFERW=YES because it could affect the integrity of your database in the event of a system failure.

### 17.1.3 How large should a buffer page be?

**Pages as large as largest database page:** The page size for a buffer must be able to hold the largest database page that will be read into that buffer. Therefore, to conserve system resources, try to assign files to the buffer with roughly equivalent block sizes (a block equals a database page). At BS2000/OSD sites using PAM files, the size of a buffer page should be a multiple of 2048 bytes.

### 17.1.4 Choosing a method for storage acquisition

**Choosing IDMS or OPSYS:** The IDMS and OPSYS options on the BUFFER statements determine *how* CA-IDMS/DB acquires storage for the buffer and the source of this storage:

- If you specify OPSYS storage, CA-IDMS/DB issues one or more requests to the operating system for a contiguous block of storage. If the operating system supports extended addressing, the storage will be acquired above the 16-megabyte line.

- If you specify IDMS storage, CA-IDMS/DB issues separate storage requests for each page in the buffer. The storage is acquired from IDMS-managed storage and will reside above the 16-megabyte line under the following conditions:
  - In local mode, if the operating system supports extended addressing
  - Under the central version, if an XA storage pool exists which supports system-type storage.

**Advantages of using OPSYS storage:** The OPSYS storage option offers an advantage to sites that define large buffers because of the way storage is acquired. For example, a buffer defined with an initial number of pages of 1000 will result in a single storage request for the entire 1000 pages if OPSYS is specified or 1000 storage requests if IDMS is specified. Another advantage is that the OPSYS storage is acquired outside the IDMS storage pool while IDMS storage is acquired from the IDMS storage pool. Therefore, the storage pool must be large enough to hold the buffer.

**Insufficient storage under the central version:** When initially allocating a buffer or when increasing the size of a buffer in response to a DCMT command, CA-IDMS/DB may be unable to acquire all the necessary storage. If this occurs and the storage acquisition mode is OPSYS, CA-IDMS/DB will attempt to acquire the storage from the IDMS storage pool. Whenever acquiring storage from the IDMS storage pool, if the necessary storage cannot be acquired or if the DC/UCF system is placed in a short-on-storage condition, the number of pages in the buffer is reduced by half until the necessary storage can be acquired without a short-on-storage condition.

---

## 17.2 Managing buffers dynamically

**Changing buffer characteristics:** Once a database is in operation, you can vary the characteristics of buffers dynamically by issuing the DCMT VARY BUFFER statement.

By making a temporary change to a buffer setting online, you can evaluate the potential impact this change might have on overall system performance. This allows you to identify the optimal settings for your buffers. When you have identified the optimal settings, you can make permanent changes to the buffer definitions by using the ALTER BUFFER statement.

**Types of changes:** The following buffer characteristics can be changed using DCMT commands:

- The number of pages in the buffer pool
- The number of pages to be acquired in each storage request (this value defaults to the initial number of pages in the buffer pool)
- The maximum number of pages in the buffer pool
- The storage acquisition mode (OPSYS or IDMS)
- Whether or not the chained read facility is activated and the number of pages that must be in the buffer to invoke chained reads as described under 17.4, “Using chained reads” on page 17-9 later in this chapter
- Whether or not a file is associated with a shared cache using a DCMT VARY FILE/AREA/SEGMENT command

If the number of pages in the buffer pool is changed to any value between the initial and maximum number of pages, the change is effective immediately. Changing the number of pages in the buffer pool beyond this range or changing other buffer characteristics takes effect only after the buffer is closed and re-opened. The buffer can be closed using a DCMT VARY BUFFER command and it will be re-opened automatically when the next read occurs for a file associated with the buffer.

**Varying a DMCL:** The following buffer changes can be made dynamically by varying a new copy of the DMCL:

- The page size of a buffer can be changed
- New buffers can be added to the system
- Existing buffers can be removed from the system
- Files can be associated with a different buffer

Other characteristics, such as the number of pages in the buffer or the storage acquisition mode, are not affected by varying a new copy of the DMCL. To dynamically make such changes, use the DCMT VARY BUFFER command.

## 17.3 Tuning buffers for performance

**When to add more database buffers:** If your monitoring operations reveal contention among applications for use of your buffers, you may need to add more buffers. For example, you may create a new buffer and assign it to a file that is accessed frequently; files that are accessed infrequently can share buffers without incurring contention among applications.

To determine which files within a buffer are accessed most frequently, issue the DCMT DISPLAY STATISTICS BUFFER command with the FILE option. This will show the number of pages requested as well as the number of reads and writes issued for each file associated with a specific buffer.

**When to change the database buffer page size:** You may have to change the buffer's page size if you associate different files with the buffer. The buffer's page size must be as large as the largest database page in any file associated with the buffer. Therefore, if new files assigned to the buffer contain larger database pages, the buffer page must be increased accordingly; likewise, if the files are removed from the buffer, you may be able to decrease the buffer page size to conserve memory resources.

**When to change the number of database buffer pages:** You can use the **buffer utilization ratio** to determine if a buffer has the optimal number of pages. This ratio is the number of database pages requested to the number of database pages CA-IDMS/DB reads from disk. A high ratio (above 2) indicates an effective buffer size. A lower ratio indicates that the buffer has too few pages.

You can use the DCMT DISPLAY STATISTICS BUFFER command to determine these values. You can also obtain them from the Performance Monitor, JREPORTs, and SREPORTs.

---

## 17.4 Using chained reads

**What chained reads do:** Chained reads allows IDMS/DB to read multiple blocks from disk with a single I/O request. It can significantly reduce both elapsed and CPU times for applications that process multiple contiguous pages within an area.

CA-IDMS/DB automatically uses chained reads under OS/390 and VSE/ESA both in local and central version processing under these conditions:

- The file being accessed is non-VSAM
- The file is not associated with a dataspace or a shared cache
- The buffer pool for the file contains a page count of at least 255 pages
- And, one or more of the following applies:
  - An area sweep is being performed
  - An SQL request is processed in such a way that multiple contiguous pages will likely be accessed (walking a clustered set or index, performing an index scan) — then need a buffer pool with at least 500 pages (or the prefetch\_buf SYSIDMS value)
  - One of the following utility functions is executing:
    - ARCHIVE LOG
    - BACKUP
    - BUILD INDEX
    - CLEANUP
    - MAINTAIN INDEX
    - PRINT LOG
    - PRINT SPACE
    - RESTRUCTURE SEGMENT
    - RESTRUCTURE CONNECT
    - UNLOAD
    - UPDATE STATISTICS
    - VALIDATE

**Note:** Several other utilities such as ARCHIVE JOURNAL use QSAM processing for their sequential processing.

**How chained reads work:** When chained reads is active, a single start I/O reads up to an entire track at one time. If some of the pages are already in core, those pages are skipped (that is, they are not read).

When IDMS/DB processes an entire area, it issues multiple start I/Os. Under the central version, without read drivers, two start I/Os will be issued; in local mode, as many as ten start I/Os will be issued (subject to buffer pool size). IDMS/DB overlaps multiple start I/Os to reduce elapsed time.

**Controlling the use of chained reads:** Under the central version, use the PREFETCH option of the DCMT VARY DMCL, AREA, FILE, or BUFFER commands to control when to use chained reads. ON is the default. OFF takes precedence over ON at a lower level. For example, varying PREFETCH OFF for an area will disable it for all files associated with that area. The default prefetch limit of 500 pages under the central version can be overridden by using the following command:

```
DCMT VARY BUFFER <buffer-name> PREFETCH <limit>
```

For example, if the limit for a buffer pool is set to 100, then (provided that there are at least 100 buffer pages) chained reads will be used for all files associated with the buffer.

**Monitoring effectiveness:** To determine the effectiveness of chained reads in your system, use the OPER WATCH DB IO command, which displays the number of start I/Os and number or page I/Os using chained I/O for a given task. It also reports, for a given area, the ratio of pages read to start I/Os.

It is possible that certain applications or processing loads may either experience no improvement or incur increased overhead because chained reads may cause pages to be prematurely flushed from the buffer. If such a situation occurs, you can disable chained reads for local mode or central version by specifying PREFETCH=OFF as a SYSIDMS parameter.

## 17.5 Using read and write drivers

**Read drivers:** A read driver performs "look-ahead" reads when IDMS/DB is instructed to sweep an area. When it is activated, it uses chained reads to read a track of pages beginning with the third or fourth tracks from the start of the area sweep and attempts to "stay ahead" of processing the pages. Use the DCMT VARY DB READ ON/OFF command to activate or de-activate the read driver for an area.

**Write drivers:** A write driver facilitates writing pages from the buffer to disk. IDMS/DB invokes a write driver under these conditions:

- When a transaction is committed and the buffer contains at least five updated pages. The driver writes all the pages in the buffer updated by the transaction.
- When more than 75% of the pages in the buffer are updated pages.

Use the DCMT VARY DB WRITE DRIVER ON/OFF command to activate or de-activate the write driver.

## 17.6 Related information

- About defining database buffers, see Chapter 4, “Defining, Generating, and Punching a DMCL” on page 4-1
- About DCMT commands, refer to *CA-IDMS System Tasks and Operator Commands*
- About shared cache, refer to *CA-IDMS System Operations*

# Chapter 18. Journaling Procedures

---

18.1 About journaling . . . . .	18-3
18.1.1 Journaling under the central version . . . . .	18-3
18.1.2 Journaling in local mode . . . . .	18-4
18.2 About journal files . . . . .	18-5
18.2.1 Journal record entries . . . . .	18-5
18.2.2 Checkpoints . . . . .	18-6
18.3 Offloading disk journal files . . . . .	18-9
18.3.1 When CA-IDMS/DB switches journal files . . . . .	18-9
18.3.2 How to offload the disk journal . . . . .	18-10
18.3.3 After system shutdown . . . . .	18-11
18.4 User exits and reports for journal management . . . . .	18-12
18.5 Influencing journaling performance . . . . .	18-13
18.5.1 Reducing journal file I/O . . . . .	18-13
18.5.2 Improving warmstart performance . . . . .	18-14
18.6 Related information . . . . .	18-16



## 18.1 About journaling

**Journals log database activity:** Journals log database activity. Specifically, journals log:

- The before and after images of modified records and rows
- The status of transactions accessing the database

**Note:** Throughout the remainder of this chapter, the term record is used to mean both record and row.

**What follows:** A brief description of journaling under the central version and in local mode follows. The remainder of the chapter describes:

- The contents of a journal file
- Offloading disk journal files
- User exits and reports CA supplies to assist journal management
- Managing journal files dynamically

### 18.1.1 Journaling under the central version

**Update and retrieval transactions:** Under the central version, several transactions can update the database concurrently. CA-IDMS/DB writes information about all update transactions to the journal files. CA-IDMS/DB also writes status information about retrieval-only non-SQL transactions if JOURNAL RETRIEVAL is specified in the system generation SYSTEM statement. No information is recorded on the journal file for retrieval-only SQL transactions.

**Use disk journals under the central version:** You must use disk journals for automatic recovery under the central version. Automatic recovery occurs during *warmstart*, following the abnormal termination of a transaction, and under the command facility due to a statement error.

►► For more information about automatic recovery, see Chapter 19, “Backup and Recovery” on page 19-1.

**Need at least two disk journals:** Under central version, you need at least two disk journals. As one file becomes full, CA-IDMS/DB automatically switches to an alternate file. While CA-IDMS/DB writes to the alternate file, the full disk journal file must be offloaded using the ARCHIVE JOURNAL utility statement. This procedure is described in more detail later in this chapter.

## 18.1.2 Journaling in local mode

**Journaling may not be necessary:** When you execute an application in local mode, that application is the only one that has access to any areas it updates. Therefore, journaling may not be necessary in local mode, provided you backup the database files before and after executing an application that updates the database. Typically, you journal in local mode when your database is too large to backup in a reasonable amount of time.

**Must use tape journals in local mode:** To journal in local mode, you must use a DMCL that defines a tape journal file. You can assign the tape journal file to either a disk or tape device. However, if you journal to a disk device, you must copy the file to a tape device before performing a manual recovery operation.

## 18.2 About journal files

**Journal record types:** Database activity is recorded on a journal file (tape or disk). CA-IDMS/DB writes the following information to the journal:

- Journal record entries that contain the image of database records
- Checkpoints that describe the status of transactions accessing the database

**Writing journal blocks:** CA-IDMS/DB accumulates journal records in the journal buffer. It writes the journal buffer to a journal file when one of the following conditions occurs:

- The buffer is full.
- A page containing an updated record occurrence whose before image is in the journal buffer, is to be written back to the database.
- A recovery unit (that is, that part of a transaction that falls between two checkpoints) terminates. A recovery unit terminates when the application issues a COMMIT (WORK), ROLLBACK (WORK), or FINISH command, or when the application aborts.

**Note:** All journal file blocks are the same length, whether or not the buffer is full when the buffer is written to a journal file.

### 18.2.1 Journal record entries

**Log changes in records:** CA-IDMS/DB uses journal record entries to log changes to the records in a database. A journal record entry is an image of a database record. As a database record is added, deleted, or modified, CA-IDMS/DB writes a **before image** that contains the image of the record before update and an **after image** that contains the image of the record after update.

**Journal images for modified records:** On a change to an existing record, the contents of before and after images are dependent on how the processing of the DML statement affects the database record:

Affect on database record	Contents of journal record entry
Data in the record changes	<ul style="list-style-type: none"> <li>▪ Database key of the record occurrence</li> <li>▪ Prefix portion of the record occurrence</li> <li>▪ Data portion of the record occurrence</li> </ul>
Record's relationships in a set changes	<ul style="list-style-type: none"> <li>▪ Database key of the record occurrence</li> <li>▪ Prefix portion of the record occurrence</li> </ul>

**Journal images for new or deleted records:** If a DML statement adds a new record occurrence into the database, the *before* image of the record is null. Similarly, if a DML statement removes a record occurrence, the *after* image of the record is null.

## 18.2.2 Checkpoints

**Describe transaction status:** Checkpoints describe the status of transactions accessing the database. CA-IDMS/DB writes these checkpoints to the journal buffer:

---

<b>Checkpoint</b>	<b>Description</b>
BGIN	Written automatically to the journal file when an application initiates a non-SQL database transaction if JOURNAL RETRIEVAL is specified, or when the first update occurs, if NOJOURNAL RETRIEVAL is specified.
ENDJ	Written automatically to the journal file when an application executes a FINISH or COMMIT WORK statement, marking the normal termination of a transaction.
COMT	Written to the journal file when an application executes a COMMIT or COMMIT WORK CONTINUE statement, marking the end of a recovery unit within the transaction.
ABRT	Written to the journal file when an application executes a ROLLBACK or ROLLBACK WORK statement or, if running under the central version, when the CV automatically recovers a failing transaction. An ABRT checkpoint marks the abnormal completion of a transaction.
AREA	Written for each area readied by an explicit DML READY command or readied automatically by the DBMS.
RTSV	Written automatically to the journal file each time CA-IDMS/DB encounters an error while executing an SQL or physical DDL statement that updated the database. During recovery, CA-IDMS/DB rolls back to the journal record designated by the RTSV checkpoint record.
TIME	Written to a journal each time the journal's buffer is initialized. However, the time and date fields contain binary zeros until the journal buffer is written to the journal file.
BFOR	Written to a journal each time a record is updated and carries the image of that record before the change was made
AFTR	Written to a journal each time a record is updated and carries the image of that record after the change was made
CKPT	Written to a journal each time one or more transactions are committed. This record is used to coordinate the commit of several transactions at the same time.
USER	Written to a journal via the WRITE JOURNAL command issued by a user program
JSEG	Written to a journal at the beginning of each disk journal segment. This record identifies the transactions that were active when that journal segment was started.
DSEG	Written periodically to the journal to identify the transactions that are active at a given point in time.

---

**Note:** ENDJ, COMT, and ABRT checkpoints are written to the journal file only by transactions for which a BGIN checkpoint is also written.

---

## 18.3 Offloading disk journal files

**What happens when you offload a disk journal file:** The ARCHIVE JOURNAL utility statement offloads the contents of a disk journal file to an archive journal file. It also rebuilds the disk journal file, condensing all before images for each active transaction into new journal blocks at the beginning of the file. This process creates a journal file that contains only those before images that are needed if an active transaction aborts or requests rollback.

**Creating multiple archive files:** CA-IDMS/DB will offload the disk journal files to multiple archive files if more than one is defined in the DMCL used when executing the ARCHIVE JOURNAL utility statement. By creating multiple archive files, you increase the likelihood that a readable archive file is available in the event it is needed for manual recovery. If an I/O error is encountered while writing to one of the archive files, a warning message is issued and offloading continues without further writes to the damaged file. If all archive files incur write errors, execution is aborted.

**When to offload:** You normally offload disk journal files only when:

- CA-IDMS/DB switches to another disk journal file
- The DC/UCF system is shut down and the database is backed up

The procedure for each scenario is provided below followed by a description of how to restart an offload operation.

### 18.3.1 When CA-IDMS/DB switches journal files

**When switch occurs:** CA-IDMS/DB switches to another disk journal file when:

- The active disk journal becomes full
- You issue a DCMT VARY JOURNAL command under the central version
- An I/O error is detected on the active disk journal file

**What happens when the switch occurs:** When CA-IDMS/DB switches to another disk journal file, it writes a message to the operator, indicating that a swap has occurred and that the previously active journal file needs offloading. The operator should respond to this message by offloading the full file.

**Eliminating operator intervention:** You can eliminate the need for operator intervention by using a write-to-operator exit routine that intercepts and reviews the message to the operator and responds by automatically submitting a job to offload the full journal file.

►► For information about the WTOEXIT user exit and sample routines for each operating system, refer to *CA-IDMS System Operations*.

## 18.3.2 How to offload the disk journal

**ARCHIVE JOURNAL utility statement:** To offload the journal, you execute the ARCHIVE JOURNAL utility statement using the batch command facility. You should use the default option of AUTO so that the oldest non-archived journal file is selected for processing.

**System failure during offload:** If the operating system fails while an ARCHIVE JOURNAL statement is executing, resubmit the ARCHIVE JOURNAL job using the RESTART parameter and identifying the journal file that was being processed at the time of failure.

**Potential problems while offloading:** You may encounter two types of problems when you offload journal files in an active system:

1. The offloaded journal file is still full following the offload because it contains before images for uncommitted transactions active at the time of the offload. The ARCHIVE JOURNAL utility statement issues messages indicating how full the disk journal file is after being offloaded. If it is full, it is usually because a long-running batch job is updating the database without issuing intermediate COMMIT statements. In this case, consider cancelling the offending job. If you allow the job to continue, it may cause all disk journal files to fill (even after being offloaded), at which point the DC/UCF system must be cancelled allowing warmstart to recover the database.
2. The remaining disk journal files fill before ARCHIVE JOURNAL completes offloading a single file. When this occurs, CA-IDMS/DB temporarily halts further database activity until the offload job is complete.

**Prevention for problem 1:** To prevent a full disk journal following an offload, take one or more of the following steps:

- Ensure that batch update programs issue frequent COMMITs to reduce the number of before images that must be retained on the journal file
- Allocate larger disk journal files
- Execute long-running update programs in local mode

**Prevention for problem 2:** To prevent future disk journal file overloading, take one or more of the following steps:

- Allocate larger disk journal files
- Increase the number of disk journal files
- Execute long-running update programs in local mode.

### 18.3.3 After system shutdown

**Offload all files:** After a normal system shutdown, you may offload *all* non-empty journal files, by executing an ARCHIVE JOURNAL utility statement with the ALL option:

```
archive journal all;
```

**Usually done in conjunction with backup:** Offloading all journal files following a system shutdown is usually performed in conjunction with backing up the database.

►► For more information about backup, see Chapter 19, “Backup and Recovery” on page 19-1.

## 18.4 User exits and reports for journal management

**User exits:** The table below describes user exits that you can use in managing your journals:

IDMSAJNX	Can be used to collect statistics on database activities; CA-IDMS/DB invokes this exit as it offloads a journal record page to the archive file
IDMSDPLX	Can be used to maintain duplicate journal files; CA-IDMS/DB invokes this exit each time it writes to the disk journal or a database file;
IDMSJNL2	Can be used for duplicating journal information and statistics collection; CA-IDMS/DB invokes this exit each time it writes a journal buffer to the journal file
WTOEXIT	Can be used to automatically initiate a journal offload following a switch to a new journal file. CA-IDMS/DB invokes the exit each time a message is written to the operator.

►► For more information about these user exits and how to invoke them, refer to *CA-IDMS System Operations*.

**Reports:** The table below summarizes reports you can use to manage your journals:

JREPORTs	<p>Report on the content of the journal file as follows:</p> <ul style="list-style-type: none"> <li>■ Transaction summary</li> <li>■ Program termination statistics</li> <li>■ Program I/O statistics</li> <li>■ Program summary</li> <li>■ Transactions within an area</li> <li>■ Programs within an area</li> <li>■ Area summary</li> </ul> <p>You can also request a formatted dump of the journal file</p>
PRINT JOURNAL utility	Reports on checkpoint information for transactions recorded on the archive file; this information is useful for rollback and rollforward operations

## 18.5 Influencing journaling performance

CA-IDMS/DB provides facilities to:

- Reduce the amount of I/O activity for journal files under the central version
- Reduce the time needed to warmstart a central version following abnormal termination

### 18.5.1 Reducing journal file I/O

**Increasing journal buffer size::** If your system encounters frequent or sizable rollback operations, it may be possible to reduce the I/O to the journal file by increasing the number of pages in the journal buffer. Minimally, the journal buffer should hold at least 5 pages. Increasing the number of pages may significantly improve performance.

**Deferring journal writes:** You can reduce the amount of journal I/O by instructing CA-IDMS/DB to defer the writing of journal buffers. Normally CA-IDMS/DB forces the writing of a journal buffer to the journal file whenever a COMT, ENDJ, or ABRT record is written to the journal buffer. You can request that CA-IDMS/DB defer the write by specifying a non-zero JOURNAL TRANSACTION LEVEL either in the system generation SYSTEM statement or in a DCMT VARY JOURNAL command.

**How transaction levels work:** When the number of active transactions in the central version is greater than the journal transaction level, CA-IDMS/DB defers the writing of a journal buffer when a recovery unit terminates. If the journal write is deferred, the task associated with a terminating recovery unit is placed in a wait state until the journal block is written. The journal block is written when:

- The number of active transactions falls below the journal transaction level
- The journal buffer is full
- The journal buffer contains the before image of an updated record occurrence (or row) that exists on a page to be written to the database

**Note:** An 'active transaction' is one for which journal records are being created.

By deferring the journal write, CA-IDMS/DB is able to place more information on a journal block, thus reducing the need to write as many blocks.

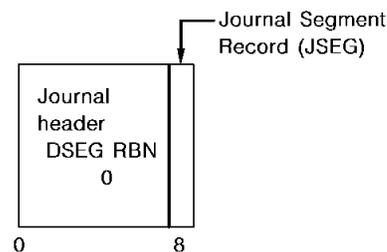
**Considerations:** The establishment of a journal transaction level is most effective in an active system; that is, one in which many update transactions are active at one time. If used, you should set the journal transaction level to be at least 4. The lower the number, the more likely journal writes will be deferred.

## 18.5.2 Improving warmstart performance

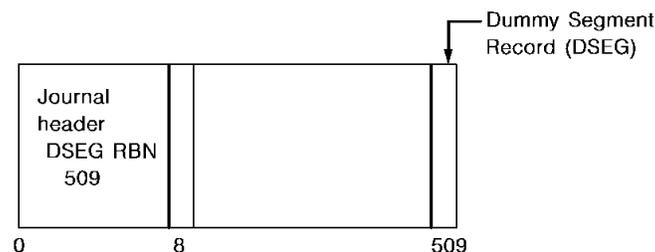
**Reducing warmstart time:** You can reduce the time it takes to warmstart a central version following an abnormal termination by specifying a non-zero value for a JOURNAL FRAGMENT INTERVAL in the system generation SYSTEM statement or in a DCMT VARY JOURNAL command.

**How the journal fragment works:** The journal fragment interval designates an interval for writing dummy segment (DSEG) records to the journal file. DC/UCF uses the DSEG records in the event of a system crash to determine the appropriate starting place for warmstart processing, as shown in the steps below:

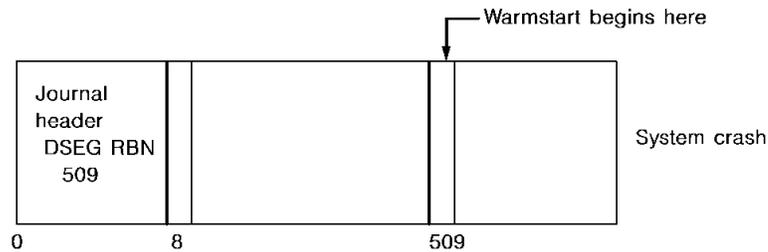
1. The new journal file is activated. It begins with header records. These records contain:
  - Information on currently open transactions
  - The relative block number (RBN) of the DSEG record. The RBN signifies which DSEG record is used to start forward processing in the event of a warmstart.



2. If the journal fragment interval is 500, the DC/UCF system will do the following before it writes the 509th journal block:
  - Creates and writes the DSEG record
  - Updates the DSEG RBN in the journal header



3. In the event of a system crash, the warmstart forward processing starts at the DSEG record at RBN 509 instead of at the JSEG record. This saves the time it would have taken for processing to read the first 500 journal blocks.



**Considerations:** If your journal files are large (in terms of the number of pages), a journal fragment interval can significantly reduce the amount of time it takes to warmstart a DC/UCF system. The warmstart logic goes to the most recently accessed journal fragment and starts its recovery processing from that point. However, because there is overhead required to write dummy segment headers, your journal fragment interval should be at least 100. Choose an interval that is between 100 and half the number of blocks in your journal file.

## 18.6 Related information

- About defining and modifying journal files, see Chapter 4, “Defining, Generating, and Punching a DMCL” on page 4-1 and Chapter 25, “Modifying Physical Database Definitions” on page 25-1
- On database backup and recovery, see Chapter 19, “Backup and Recovery” on page 19-1
- About allocating and formatting disk journal files, see Chapter 16, “Allocating and Formatting Files” on page 16-1
- About user exits, refer to *CA-IDMS System Operations*
- For the complete syntax and syntax rules for the ARCHIVE JOURNAL utility statement, refer to *CA-IDMS Utilities*
- About DCMT VARY JOURNAL and DCMT VARY FILE commands, refer to *CA-IDMS System Tasks and Operator Commands*
- About journal system generation parameters, refer to the SYSTEM statement in *CA-IDMS System Generation*

# Chapter 19. Backup and Recovery

---

19.1	About database backup and recovery	19-3
19.2	Backup procedures	19-4
19.2.1	Back up after a normal system shutdown	19-5
19.2.2	Backup while the DC/UCF system is active	19-5
19.2.3	Back up before and after local mode jobs	19-10
19.2.4	Automating the backup process	19-11
19.3	Automatic recovery	19-14
19.3.1	Warmstart	19-14
19.3.2	Automatic rollback	19-16
19.4	Manual recovery	19-18
19.4.1	Recovery from a quiesced backup	19-19
19.4.2	Recovery from a hot backup	19-21
19.4.3	Reducing recovery time	19-28
19.4.4	Recovering a large number of files	19-30
19.5	Recovery procedures after a warmstart failure	19-31
19.6	Recovery procedures from database file I/O errors	19-33
19.7	Recovery procedures from journal file I/O errors	19-37
19.8	Recovery procedures for local mode operations	19-40
19.8.1	No journaling	19-40
19.8.2	Journaling to a tape device	19-40
19.8.3	Journaling to a disk device	19-40
19.8.4	Using an incomplete journal file	19-40
19.9	Recovery procedures for mixed-mode operations	19-42
19.10	Data sharing recovery considerations	19-44
19.11	Considerations for recovery of native VSAM files	19-47



## 19.1 About database backup and recovery

**Protects your data:** Database backup and recovery are maintenance tasks that protect the changes made to your database:

- **Backup** is a routine database maintenance task that produces a copy of the database. If necessary, this backup copy can be used to restore lost data.
- **Recovery** restores the contents of the database when an error occurs that corrupts the database or disk journal file. Recovery procedures restore altered areas to their original state.

**Types of recovery:** Under the central version, recovery occurs *automatically* with no intervention from the DBA. If automatic recovery fails you must recover the database *manually*. You must also recover the database manually for local mode update jobs that terminate abnormally.

**What follows:** The remainder of this chapter describes:

- Procedures to back up database files
- How CA-IDMS/DB recovers data automatically
- Procedures for recovering data manually under different circumstances

## 19.2 Backup procedures

**Perform backups often:** Backup procedures are an essential part of database administration. To help protect the integrity of your database, you should perform backups as often as possible. As a general rule, always back up the database:

- At regular, scheduled intervals, such as daily or weekly
- Before and after structural changes to the database
- Whenever you initialize journal files
- Since automatic recovery is not available in local mode, before and after executing an application run in local mode.

**Design a backup plan:** To ensure that your backup procedures meet the data processing needs of your company, you need to decide how often to take backups and how long to retain them. Develop a schedule and procedures for performing backups and stick to it.

**General guidelines:** The following list identifies some guidelines to follow in designing a backup plan:

- Define the backup and recovery requirements for an application while the application is being designed. Test all backup and recovery procedures before the application is put into production.
- Make sure you backup the database after making changes to its physical definition (such as changing the page size, page range, and so on).
- Identify all archive files created since the last backup.
- If you need to concatenate archive tapes for historical records, make sure that the tapes included in the concatenation are not required for recovering the database. For example, you might concatenate the archive tapes from the previous week at the end of the current week.
- Bear in mind that restoring a database from a date several weeks in the past can be a very time-consuming process because of the volume of journal data that needs to be processed.

**BACKUP utility statement:** The examples outlined in this chapter use the BACKUP utility statement provided with CA-IDMS/DB to backup the database. You can use other utilities (such as IEBGENER in OS/390) to perform the backup and recovery operation provided they restore disk files to the state they were in when copied.

If you use the CA-provided BACKUP utility statement for regularly scheduled backups, specify the FILE option rather than the AREA option. FILE lets you recover an individual file in the event it is damaged rather than having to recover the entire area. Use the AREA option only if multiple areas are stored in a single file.

**What follows:** The following topics tell you how to back up the database under the following conditions:

- After a normal system shutdown
- While the system remains active
- Before and after running a local mode update job

### 19.2.1 Back up after a normal system shutdown

**Steps:** While the system is inactive, back up the database using the following procedure:

Action	Statement
Offload the active journal (that is, the journal in use at the time you shut down the system)	ARCHIVE JOURNAL utility statement with the AUTO or AUTOALL option
Copy all files associated with the database	BACKUP utility statement or any comparable backup utility

### 19.2.2 Backup while the DC/UCF system is active

*Types of backup while system is active:* There are two types of backup that can be done while DC/UCF remains active:

- A quiesced backup during which no updates are made to the areas being copied
- A hot backup during which the areas that are copied are updated by transactions executing within the central version

While it is preferable to back up a database when it is quiesced, a site with high-availability requirements may not be able to disable updates long enough to complete the backup.

*Considerations:* If you decide to use a hot backup strategy, consider the following:

- The time to recover using a hot backup may be longer than with a backup produced while the area was quiesced due to additional steps in the recovery process.
- In order to recover using a file produced during a hot backup, all archive journal files created while the backup was taking place must also be available; without these files, the backup file cannot be used. Although the EXTRACT JOURNAL utility statement can be used to preprocess the journal images generated during this time period, the original archive files must also be available in order to perform a successful recovery.
- To ensure the availability of the archive journal files you should treat them in the same way as the backup file; for example, if a copy of the backup file is sent offsite, a copy of all corresponding archive files should also be sent offsite.

►► For more information on the impact of a hot backup on recovering a database, see 19.4, “Manual recovery” on page 19-18 later in this chapter.

*Quiesced backup procedure:* The procedure outlined below describes how to perform a quiesced backup.

Action	Steps
Quiesce update activity in the target areas. (See considerations below)	Issue one or more of the following commands: <ul style="list-style-type: none"> <li>■ DCMT VARY AREA ... RETRIEVAL</li> <li>■ DCMT VARY AREA ... OFFLINE</li> <li>■ DCMT QUIESCE AREA ...</li> <li>■ DCMT VARY SEGMENT ... RETRIEVAL</li> <li>■ DCMT VARY SEGMENT ... OFFLINE</li> <li>■ DCMT QUIESCE SEGMENT ...</li> <li>■ DCMT QUIESCE DBNAME ...</li> <li>■ DCMT VARY RUN UNIT ... OFFLINE</li> </ul>
Note the quiesce point	Record the date and time that the areas were quiesced.  Optionally force a new archive journal file to be created: <ul style="list-style-type: none"> <li>■ Issue a DCMT VARY JOURNAL command</li> <li>■ Execute the ARCHIVE JOURNAL utility statement</li> </ul>
Copy all files containing the target areas.	Execute the BACKUP utility statement using the FILE option or any comparable backup utility.
Restart update activity in the target areas.	Issue one or more of the following commands: <ul style="list-style-type: none"> <li>■ DCMT VARY AREA ... ONLINE</li> <li>■ DCMT VARY SEGMENT ... ONLINE</li> <li>■ DCMT VARY ID ... TERMINATE</li> <li>■ DCMT VARY RUN UNIT ... ONLINE</li> </ul>

*Hot backup procedure:* The procedure for a hot backup is similar to that for a quiesced backup, except that updates are re-enabled before the backup is complete. The procedure described below includes establishing a second quiesce point. This is not necessary if the appropriate recovery procedure is followed.

►► For more information on the impact of a hot backup and a second quiesce point on recovery, see 19.4, “Manual recovery” on page 19-18 later in this chapter.

Action	Steps
Quiesce update activity in the target areas. (See considerations below)	Issue one or more of the following commands: <ul style="list-style-type: none"> <li>■ DCMT VARY AREA ... RETRIEVAL</li> <li>■ DCMT VARY AREA ... OFFLINE</li> <li>■ DCMT QUIESCE AREA ...</li> <li>■ DCMT VARY SEGMENT ... RETRIEVAL</li> <li>■ DCMT VARY SEGMENT ... OFFLINE</li> <li>■ DCMT QUIESCE SEGMENT ...</li> <li>■ DCMT QUIESCE DBNAME ...</li> <li>■ DCMT VARY RUN UNIT ... OFFLINE</li> </ul>
Note the quiesce point	Record the date and time that the areas were quiesced.  Optionally force a new archive journal file to be created: <ul style="list-style-type: none"> <li>■ Issue a DCMT VARY JOURNAL command</li> <li>■ Execute the ARCHIVE JOURNAL utility statement</li> </ul>
Restart update activity in the target areas.	Issue one or more of the following commands: <ul style="list-style-type: none"> <li>■ DCMT VARY AREA ... ONLINE</li> <li>■ DCMT VARY SEGMENT ... ONLINE</li> <li>■ DCMT VARY ID ... TERMINATE</li> <li>■ DCMT VARY RUN UNIT ... ONLINE</li> </ul>
Copy all files containing the target areas.	Execute the BACKUP utility statement using the FILE option or any comparable backup utility.

Action	Steps
Optionally, establish a second quiesce point for the target areas.	<p data-bbox="938 310 1321 369">Issue one or more of the following commands:</p> <ul style="list-style-type: none"> <li data-bbox="954 394 1414 420">■ DCMT VARY AREA ... RETRIEVAL</li> <li data-bbox="954 445 1377 470">■ DCMT VARY AREA ... OFFLINE</li> <li data-bbox="954 495 1295 520">■ DCMT QUIESCE AREA ...</li> <li data-bbox="954 546 1312 604">■ DCMT VARY SEGMENT ... RETRIEVAL</li> <li data-bbox="954 630 1430 655">■ DCMT VARY SEGMENT ... OFFLINE</li> <li data-bbox="954 680 1349 705">■ DCMT QUIESCE SEGMENT ...</li> <li data-bbox="954 730 1338 756">■ DCMT QUIESCE DBNAME ...</li> <li data-bbox="954 781 1317 831">■ DCMT VARY RUN UNIT ... OFFLINE</li> </ul>
Mark the end of the backup process.	<p data-bbox="938 856 1354 915">Force a new archive journal file to be created:</p> <ul style="list-style-type: none"> <li data-bbox="954 940 1365 999">■ Issue a DCMT VARY JOURNAL command</li> <li data-bbox="954 1024 1370 1083">■ Execute the ARCHIVE JOURNAL utility statement</li> </ul> <p data-bbox="938 1108 1398 1167">If a second quiesce point was established, record its date and time.</p>
If a second quiesce point was established, restart update activity in the target areas.	<p data-bbox="938 1192 1321 1251">Issue one or more of the following commands:</p> <ul style="list-style-type: none"> <li data-bbox="954 1276 1365 1302">■ DCMT VARY AREA ... ONLINE</li> <li data-bbox="954 1327 1422 1352">■ DCMT VARY SEGMENT ... ONLINE</li> <li data-bbox="954 1377 1377 1402">■ DCMT VARY ID ... TERMINATE</li> <li data-bbox="954 1428 1422 1453">■ DCMT VARY RUN UNIT ... ONLINE</li> </ul>

*Quiescing update activity:* Both DCMT VARY AREA (and SEGMENT) and DCMT QUIESCE can be used to quiesce update activity in one or more areas of the database. Consider the following when choosing which of these to use:

- If DCMT VARY is used, tasks which subsequently attempt to access a target area in an update mode (or any mode if the area is varied offline) will receive an 0966 error status. Unless the application program handles this condition, the associated task will fail. If DCMT QUIESCE is used, such tasks will wait until update activity is restarted, unless their quiesce wait time is exceeded.
- DCMT QUIESCE provides more control over the quiesce operation. For example, it is possible to specify how long the quiesce operation should wait for conflicting

---

tasks to finish and what action should be taken in the event that the quiesce point has not been reached in the specified time interval.

- In a data sharing environment, DCMT QUIESCE will quiesce update activity across all members of the data sharing group. DCMT VARY will quiesce update activity only within the DC/UCF system in which it is executed.
- DCMT QUIESCE can be used to automate much of the backup process.

►► For more information on backup automation, see 19.2.4, “Automating the backup process” on page 19-11 later in this chapter.

►► For more information on the DCMT system task, refer to *CA-IDMS System Tasks and Operator Commands*.

*Quiescing update activity for system areas:* When backing up a system area, such as a load area, it may be necessary to terminate predefined system run units by issuing a DCMT VARY RUN UNIT ... OFFLINE command. This will be necessary if predefined run units for the target area have been defined in the system definition and such run units access the area in update mode. You can determine this by issuing a DCMT DISPLAY RUN UNIT command.

Varying a system run unit offline does not prevent overflow run units from being started to service requests for the area. It simply terminates predefined run units of the specified type. Since varying an area offline will impact the system's ability to service requests for the area, it is advisable to quiesce update activity to system areas either by varying their status to retrieval or by using the DCMT QUIESCE command.

Depending on the options specified when issuing a DCMT VARY AREA, DCMT VARY SEGMENT, or DCMT QUIESCE command, the system may automatically terminate conflicting predefined system run units.

►► For more information on when predefined system run units are automatically terminated, refer to the individual commands in *CA-IDMS System Tasks and Operator Commands*.

*Data sharing considerations:* In a data sharing environment, whenever update activity is quiesced, it must be quiesced in all DC/UCF systems that are members of the data sharing group. If a DCMT QUIESCE command is used, then update activity will automatically be quiesced on all members within the group. If a DCMT VARY AREA or DCMT VARY SEGMENT command is used, it must be executed on each system that is a member of the group. This can be accomplished by broadcasting the DCMT command.

►► For more information on broadcasting DCMT commands, refer to *CA-IDMS System Tasks and Operator Commands*.

### 19.2.3 Back up before and after local mode jobs

**Two options:** To protect data to be accessed by an update job running in local mode, you can either:

- Use local mode journaling. This option is best for large databases that would require a long time to backup and restore.
- Back up the database before and after you run the job. This option is best for small databases that can be backed up within a reasonable time frame.

►► For information about local mode journaling, see Chapter 18, “Journaling Procedures” on page 18-1.

**Steps to back up the database:** Follow the steps below to back up a database before and after running an update application in local mode:

Action	Steps
Make the areas to be accessed by the application unavailable under the central version	DCMT VARY AREA or SEGMENT with the OFFLINE, RETRIEVAL, or TRANSIENT RETRIEVAL option
Before running an application, back up each file of the database	BACKUP or any comparable backup utility
Dummy the journal file DD statements in the execution JCL of the application if the DMCL being used has a tape journal file defined	
After running the application, back up each file of the database	BACKUP or any comparable backup utility
Swap to another disk journal file in order to coordinate journal files with the backup	DCMT VARY JOURNAL
Re-activate the areas for use under the central version	<ul style="list-style-type: none"> <li>■ If the areas are OFFLINE or in RETRIEVAL mode, issue DCMT VARY AREA or SEGMENT ONLINE</li> <li>■ If the areas are in TRANSIENT RETRIEVAL mode, first vary them OFFLINE and then ONLINE</li> </ul>

**CA-ADS:** When you vary an area in preparation for a local mode update, CA-ADS users should vary the area to either OFFLINE or TRANSIENT RETRIEVAL mode; do *not* use RETRIEVAL mode.

## 19.2.4 Automating the backup process

*Exploiting DCMT QUIESCE:* Backing up a database while the DC/UCF system is active can be automated through the use of the DCMT QUIESCE command. To assist in this effort, the following can be specified as options:

- A unique identifier for use in subsequent DCMT DISPLAY ID and DCMT VARY ID commands to query or terminate an outstanding quiesce operation.
- The action that should be taken in the event that a quiesce point cannot be reached within a specified time interval. The available choices are to abandon the quiesce operation or force the quiesce by canceling conflicting tasks.
- An indication of whether a new archive journal file should be created when the quiesce point is reached.
- An indication of whether update activity in the target areas should be restarted automatically once the areas are quiesced.

*Quiesce user exit:* When a quiesce point is achieved, numbered exit, Exit 38 is invoked. This exit can be used to initiate the next step in the backup process. For example, it can submit a job to the internal reader, thus enabling the QUIESCE task to automatically initiate a copy operation. Once the files are copied, a subsequent UCF batch job step can invoke further system tasks to complete the backup process.

Rather than submitting a batch job, exit 38 might instead use an API to directly interface to a "zero-time copy" facility if the database resides on a storage device that provides such a capability.

►► For details on how to code an Exit 38 routine, refer to *CA-IDMS System Operations*.

►► For more information on the DCMT QUIESCE command, refer to *CA-IDMS System Tasks and Operator Commands*.

*Automating a quiesced backup:* The following illustrates how the DCMT QUIESCE command can be used to automate a quiesced backup operation.

Activity	Description
dcmt quiesce dbname CUST hold swap CUSTBKP	This command initiates a quiesce operation identified as CUSTBKP. All areas in all segments included in the database name CUST will be quiesced. When the quiesce point is reached, a new archive journal file will be created and exit 38 will be invoked. The quiesce point will be held until the quiesce operation is explicitly terminated.

---

Activity	Description
Exit 38 is invoked	Exit 38 submits a batch job through the internal reader (or an equivalent mechanism) to initiate the copy operation.
Batch job is executed	The batch job first copies all files containing areas of the CUST database and then invokes a UCF batch job step that terminates the quiesce operation by issuing a DCMT VARY ID command.
dcmt vary id CUSTBKP terminate	This command terminates the quiesce operation and makes the CUST areas available for update.

---

*Automating a hot backup:* The following illustrates how the DCMT QUIESCE command can be used to automate a hot backup operation.

---

Activity	Description
dcmt quiesce dbname CUST nohold swap CUSTBKP1	This command initiates a quiesce operation identified as CUSTBKP1. All areas in all segments included in the database name CUST will be quiesced. When the quiesce point is reached, a new archive journal file will be created and exit 38 will be invoked. The quiesce operation will then terminate and make the areas available for update.
Exit 38 is invoked	Exit 38 submits a batch job through the internal reader (or an equivalent facility depending on the operating system) to initiate the copy operation.
Batch job is executed	<p>The batch job first copies all files containing areas of the CUST database and then invokes a UCF batch job step.</p> <p>The UCF batch job step either initiates a second quiesce operation by issuing a DCMT QUIESCE command or forces a new archive journal file to be created by issuing a DCMT VARY JOURNAL command.</p>

---

---

<b>Activity</b>	<b>Description</b>
dcmt quiesce dbname CUST nohold swap CUSTBKP2	<p>This command initiates a quiesce operation identified as CUSTBKP2. All areas in all segments included in the database name CUST will be quiesced. When the quiesce point is reached, a new archive journal file will be created and exit 38 will be invoked. The quiesce operation will then terminate and make the areas available for update.</p> <p>Exit 38 examines the quiesce identifier and determines that no further action is needed.</p>
dcmt vary journal	<p>This command forces the use of another disk journal file which in turn causes a batch execution of the ARCHIVE JOURNAL utility statement.</p> <p><b>Note:</b> Automatic submission of the ARCHIVE JOURNAL job is dependent on the implementation of a site-specific means (such as WTOEXIT) to examine console messages and use operating system facilities to submit a batch job.</p>

---

## 19.3 Automatic recovery

**Available only under the central version:** Automatic recovery is available only under the central version. Automatic recovery occurs when CA-IDMS/DB:

- **Warmstarts**, following a system failure
- **Automatically rolls back** a failing transaction

Each is described below.

### 19.3.1 Warmstart

**Due to system failure:** **Warmstart** occurs when CA-IDMS/DB starts up and by examining the journal files it detects that the previous execution of the DC/UCF system was not shutdown normally. CA-IDMS uses the journal files to rollback all transactions that were active when the system failed.

**How you respond to a system failure:** In response to a DC/UCF system failure, you should immediately restart the system. In a data sharing environment, it is particularly important to restart failing systems as soon as possible, since data that was being updated at the time of failure remains inaccessible to other group members until the failing system has completed its warmstart.

**Note:** Do *not* offload any journal files between the time of system failure and your first attempt to warmstart the system. If you must offload, use the READ option of the ARCHIVE JOURNAL utility statement.

**Data sharing considerations:** In general, you respond to a DC/UCF system failure in the same way regardless of whether or not the system is a member of a data sharing group. However, certain types of failures, such as a loss in connectivity to a coupling facility, require special action. Additionally, if a member is unable to warmstart and manual recovery becomes necessary, then data sharing introduces additional considerations.

►► For more information on recovery considerations in a data sharing environment, refer to *CA-IDMS System Operations*.

►► For more information on the impact of data sharing to manual recovery, see 19.4, “Manual recovery” on page 19-18 later in this chapter.

**Incomplete warmstart:** Certain errors, such as I/O errors or open failures, may prevent warmstart from rolling out the changes in one or more database files. If this occurs, warmstart will continue, the system will start up and the transactions affected by the error will be restarted. Once restarted, automatic rollback will be invoked to again attempt to remove the effect of the unrecovered transactions. If automatic rollback is successful, no further action is necessary although the reason for the original failure should be investigated and corrective action taken if necessary. If automatic rollback is not successful, the unrecovered transactions will be suspended just as if they had encountered an I/O error. To correct the situation, You respond as if

a database file I/O error occurred. First take whatever action is necessary to make the file available, such as restoring a damaged file or using DCMT commands to correct a data set name. Then restart the suspended transactions by issuing a DCMT VARY FILE ACTIVE command.

►► For more information on responding to I/O errors, see 19.6, “Recovery procedures from database file I/O errors” on page 19-33 later in this chapter.

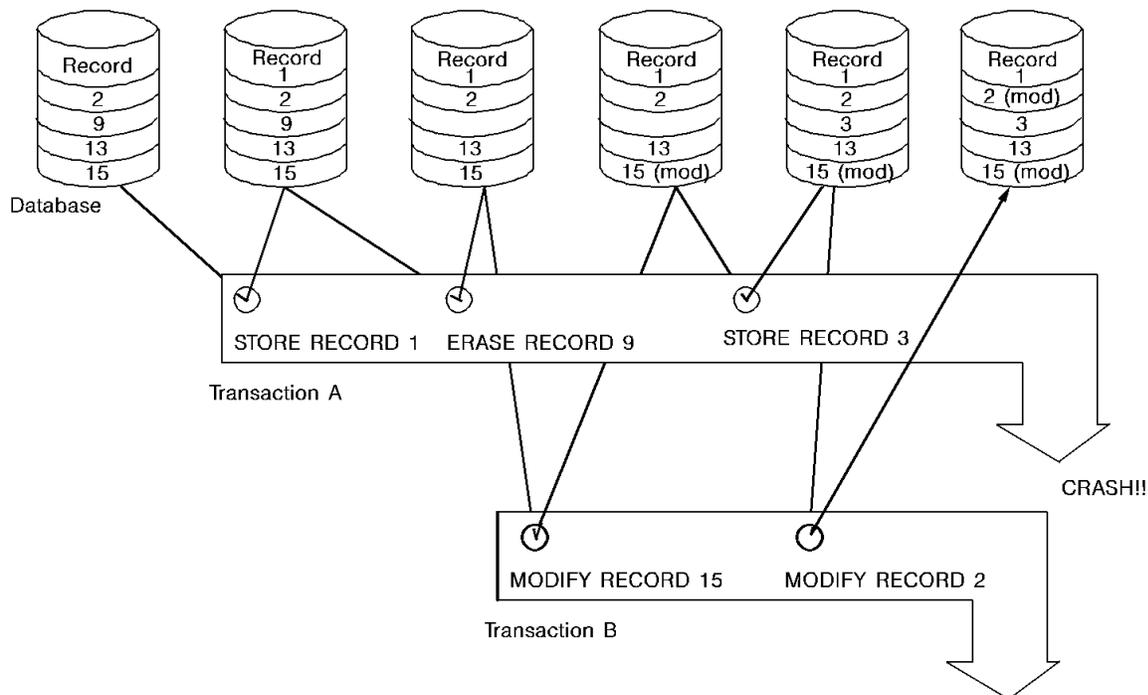
**How warmstart works:** To restore all transactions active at the time of a system failure, CA-IDMS/DB does the following:

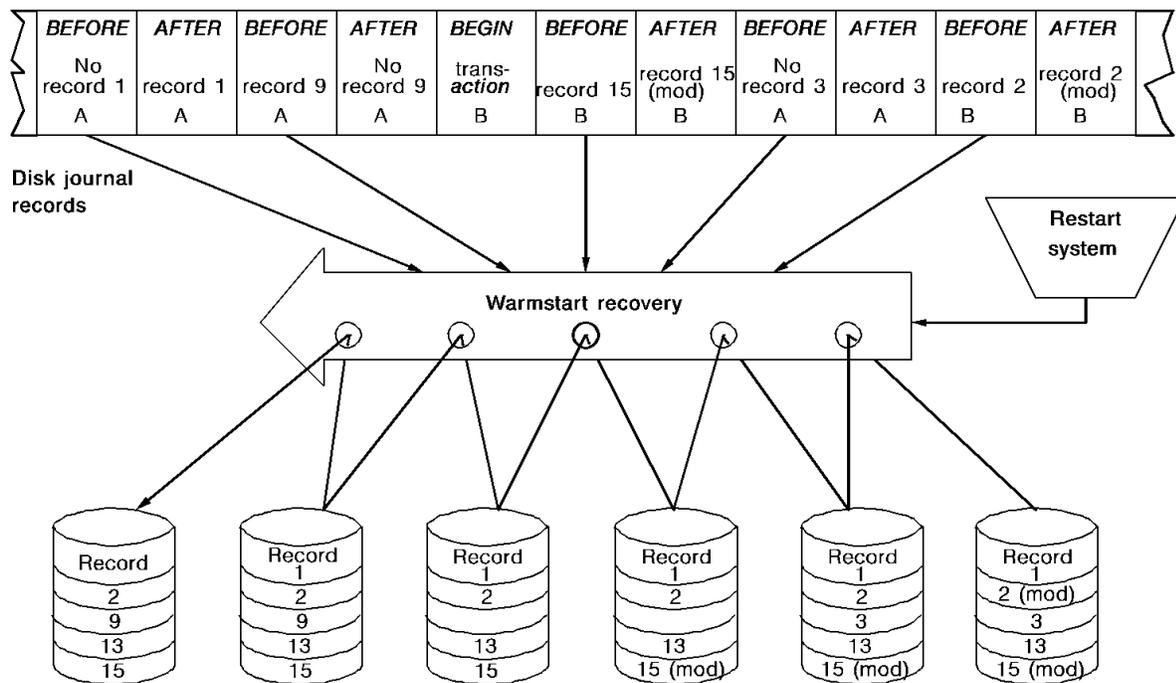
1. Establishes which disk journal file was active at the time of the failure
2. Locates the last journal record written before the system failed
3. Rolls back and writes ABRT checkpoints for all incomplete transactions.

All transactions can then be restarted with no further interruption in processing.

►► For information about journal checkpoints, see Chapter 18, “Journaling Procedures” on page 18-1.

**Example:** The example below shows how a warmstart operation is performed. In this example, two transactions are active at the time of the system crash. Both are recovered automatically when the system is restarted.



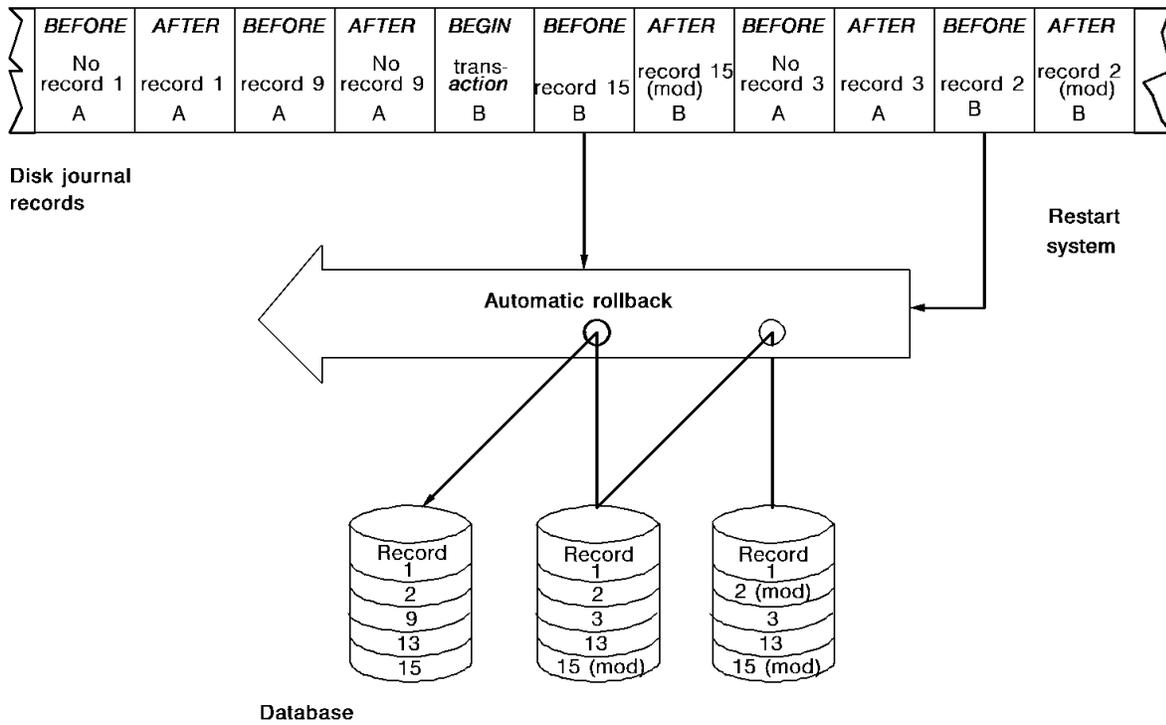
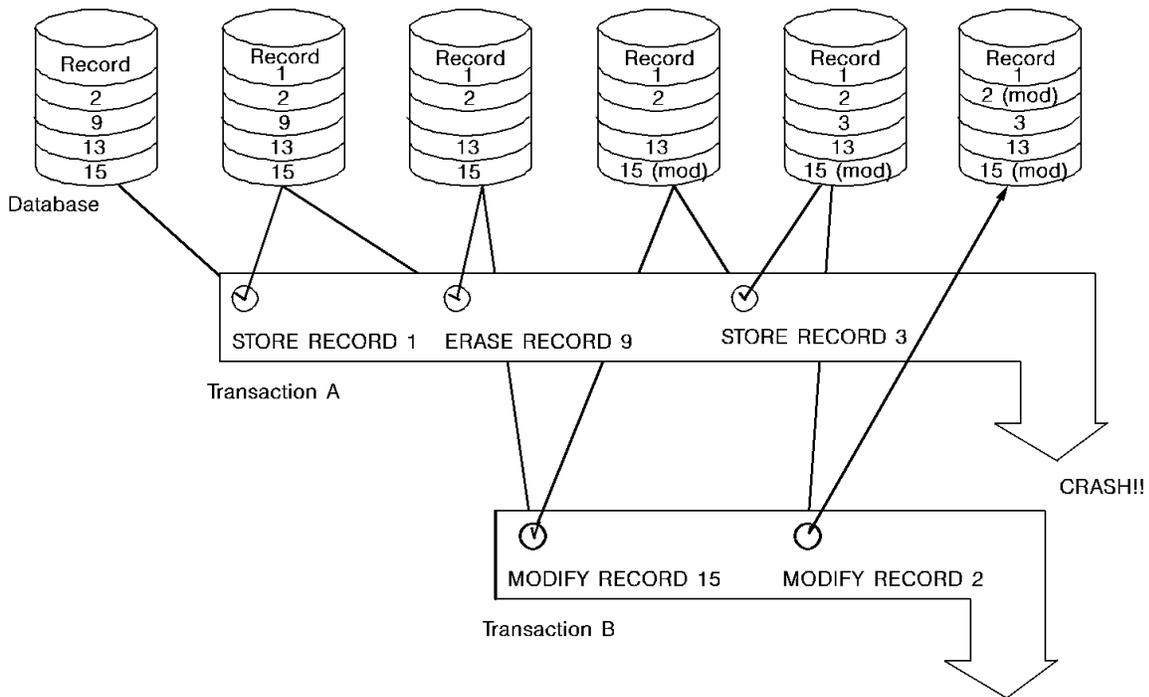


### 19.3.2 Automatic rollback

**Due to transaction failure:** **Automatic rollback** occurs when a transaction fails or an application requests recovery by means of the ROLLBACK command.

CA-IDMS/DB writes an ABRT checkpoint for the transaction and automatically rolls out the changes made to the database by the transaction. The recovery occurs while the system continues to process requests by other concurrently active transactions.

**Example:** The example below shows how an automatic rollback occurs. In this example, transaction B aborts. CA-IDMS/DB then performs an automatic rollback for transaction B while other transactions continue to process.



## 19.4 Manual recovery

**Before you begin:** Before you attempt to manually recover the areas or files of the database, gather the available facts, such as:

1. The time of the system or transaction failure
2. Whether the failure occurred under the central version or in local mode
3. What applications were running at the time the system failed
4. Which areas of the database were in use and whether these were in update mode
5. The time of the preceding quiesce point

You can use the `PRINT JOURNAL` or `MERGE ARCHIVE` utility statements to determine the information in items 3, 4, and 5.

**Locate backup and archive files:** After you've determined the nature of the failure, locate the most recent backup of the database and all archive journal files created since the backup.

**Note:** To successfully recover the database, all of the archive files must be readable. To increase the likelihood of this, you can define multiple archive files in the DMCL used to execute the `ARCHIVE JOURNAL` utility statement. This directs CA-IDMS/DB to create multiple archive files during offload.

**Minimize scope of recovery:** You can limit the recovery process by recovering only the areas or files that were impacted by the failure. Areas that were available for retrieval do not have to be recovered. Depending on the nature of the failure, recovery may be restricted to an individual file. If the recovery is due to an application error, all areas updated by the application may need to be recovered to insure the logical integrity of the database. This may in turn necessitate the recovery of other areas, if another application has updated both the original and additional areas.

**After you're done:** After you recover an area or file, check the validity of the recovery by:

- Following procedures you designed to check the validity of the data; for example, by executing a report you run regularly and comparing the output to output produced before the recovery
- Verifying the structure of the database by executing the `IDMSDBAN` utility

►► For more information on `IDMSDBAN`, refer to *CA-IDMS Utilities*.

**What follows:** The remainder of this section describes the general recovery procedure to be followed when using a quiesced backup or a hot backup procedure.

The remainder of this chapter describes manual recovery procedures under the following circumstances:

- After a warmstart fails

- I/O errors in a database file
- I/O errors in a journal file
- When journaling in local mode
- When using the database in both local mode and under the central version (mixed-mode recovery)

It also provides special considerations for data sharing environments and native VSAM files.

### 19.4.1 Recovery from a quiesced backup

*Quiesced backup:* A quiesced backup is a backup that is performed while no updates are being made to the data that is being copied. The following types of backup are quiesced backups:

- A backup performed after the DC/UCF system is shutdown
- A backup performed while the DC/UCF system is active, provided that the affected areas are quiesced at the time of the backup
- A backup performed before and after a local mode job

►► For information on how to backup a database, see 19.2, “Backup procedures” on page 19-4 earlier in this chapter.

*Recovery procedure:* The procedure outlined below describes the general approach to recovery from a quiesced backup. See the later sections in this chapter for additional considerations specific to certain types of failures.

Action	Steps
Copy the files that need to be recovered from the backup	Execute the RESTORE utility statement using the FILE option or another comparable utility.
<b>When required:</b> Always.	

Action	Steps
<p>Consolidate, in the sequence in which they were created, the archive journal files created since the quiesce point established at the start of the backup procedure.</p> <p><b>When required:</b> This step is necessary only under the following conditions:</p> <ul style="list-style-type: none"> <li>■ In OS/390 and MSP/EX environments, if the subsequent ROLLFORWARD utility statement will be executed with the SEQUENTIAL option and more than one archive journal file must be processed.</li> <li>■ In a data sharing environment, if more than one member has updated the affected areas and the subsequent ROLLFORWARD utility statement will be executed with either the SEQUENTIAL or the ALL and STOP TIME options.</li> </ul>	<p>Execute one of the following and use as input the properly concatenated set of archive files:</p> <ul style="list-style-type: none"> <li>■ FIX ARCHIVE utility statement</li> <li>■ MERGE ARCHIVE utility statement</li> <li>■ EXTRACT JOURNAL utility statement</li> <li>■ another comparable utility</li> </ul> <p><b>Note:</b> If consolidating archive files from multiple members and the subsequent rollforward will be executed with either the SEQUENTIAL or the ALL and STOP TIME options, use the MERGE ARCHIVE utility statement.</p> <p>►► For more information, see 19.10, “Data sharing recovery considerations” on page 19-44 later in this chapter.</p> <p><b>Note:</b> If recovery involves local mode journal files, the MERGE ARCHIVE utility statement can be used to consolidate both local mode journal files and archive files.</p> <p>►► For more information, see 19.9, “Recovery procedures for mixed-mode operations” on page 19-42 later in this chapter.</p>

Action	Steps
<p>Reapply to the restored files all updates made since the backup was taken</p> <p><b>When required:</b> Always.</p>	<p>Execute the ROLLFORWARD utility statement using either the consolidated journal file or individual archive files concatenated in the sequence in which they were created.</p> <p>If the journal files were consolidated using the EXTRACT JOURNAL utility, specify the FROM EXTRACT option.</p> <p>If FROM EXTRACT is not specified, then the following considerations apply:</p> <ul style="list-style-type: none"> <li>■ Specify the SORTED option unless there is insufficient disk space available. SORTED must be specified if: <ul style="list-style-type: none"> <li>– A consolidated journal file is not used as input in OS/390 and MSP/EX environments and more than one archive file must be processed.</li> <li>– The input journal file is on a device, such as a disk or a 3490 that does not support reading backwards.</li> <li>– Running ROLLFORWARD in a VM/ESA environment.</li> </ul> </li> <li>■ If the SEQUENTIAL option is used and the quiesce point for the affected areas does not coincide with the start of the first input file, use the START TIME parameter to identify the quiesce point.</li> </ul>

## 19.4.2 Recovery from a hot backup

*Hot backup:* A hot backup is a backup that is performed while the database is being updated. The steps that must be taken to create a usable hot backup are described under 19.2, “Backup procedures” on page 19-4 earlier in this chapter.

*Recovery procedures:* Following are two approaches to recovery from a hot backup. The first involves the use of both the ROLLBACK and ROLLFORWARD utility statements; the second involves two executions of the ROLLFORWARD utility statement. Either approach can be used to successfully recover from a hot backup; however certain conditions must be satisfied in order to use the second approach.

For additional considerations associated with specific types of failure, refer to later sections in this chapter.

*Restore procedure 1:* This approach can always be used to recover from a hot backup provided that the correct procedures were followed when the backup was taken and the necessary journal and backup files are available.

Action	Steps
Copy the files that need to be recovered from the backup  <b>When required:</b> Always.	Execute the RESTORE utility statement using the FILE option or another comparable utility.
Identify: <ul style="list-style-type: none"> <li>■ The quiesce point that was taken at the beginning of the backup procedure.</li> <li>■ The archive journal files created since this quiesce point up to and including the one created at the end of the backup procedure.</li> <li>■ All archive journal files created since the quiesce point up to the point of failure.</li> </ul> <b>When required:</b> Always.	Use the PRINT JOURNAL utility statement, or if the quiesce point was established using the DCMT QUIESCE command, examine the operating system log for the DC/UCF system on which the DCMT command was issued.
Consolidate, in the sequence in which they were created, the archive journal files created between the quiesce point and the end of the backup procedure.  <b>When required:</b> This step is necessary only under the following conditions: <ul style="list-style-type: none"> <li>■ In OS/390 and MSP/EX environments if more than one input journal file must be processed.</li> <li>■ In a data sharing environment, if the SEQUENTIAL option will be specified on the subsequent ROLLBACK utility statement and more than one member's journal images must be processed.</li> </ul>	Execute one of the following and use as input the properly concatenated set of archive files: <ul style="list-style-type: none"> <li>■ FIX ARCHIVE utility statement</li> <li>■ MERGE ARCHIVE utility statement</li> <li>■ Another comparable utility</li> </ul> <b>Note:</b> If consolidating archive files from multiple members and the subsequent rollback will be executed with the SEQUENTIAL option, use the MERGE ARCHIVE utility statement.  ►► For more information, see 19.10, "Data sharing recovery considerations" on page 19-44 later in this chapter.  <b>Note:</b> This and the subsequent step can be combined by using a sort utility to do the consolidation unless the use of MERGE ARCHIVE is required.

Action	Steps
<p>If backward read is not supported, presort the journal blocks created between the quiesce point and the end of the backup procedure in reverse sequence.</p> <p>Multiple archive files may be consolidated into a single sorted output file.</p> <p><b>When required:</b> This step is necessary in a VM/ESA environment or if the journal files reside on devices such as disk or 3490s that do not support backward read.</p>	<p>Execute the sort utility and use as input either a set of archive journal files or the consolidated journal file produced in the preceding step.</p> <p>►► For the sort parameters to use, refer to the ROLLBACK utility statement in <i>CA-IDMS Utilities</i>.</p>
<p>Remove from the restored files the effects of all updates made between the quiesce point and the end of the backup process.</p> <p><b>When required:</b> Always</p>	<p>Execute the ROLLBACK utility statement specifying the HOTBACKUP option and using either the consolidated journal file or individual archive files concatenated in the sequence in which they were created.</p> <p>If the quiesce point for the affected areas does not coincide with the start of the input (or the end of the input if it was sorted in reverse sequence), use the STOP TIME parameter to identify the quiesce point.</p> <p>If STOP TIME is specified, also specify ACTIVE; otherwise specify ALL.</p> <p>If backward read is not supported for the device on which the input journal file resides, specify ROLLBACK3490 in the SYSIDMS parameter file associated with the ROLLBACK job step. This parameter is not necessary in a VM/ESA environment.</p> <p>If a consolidated journal file is not used as input in OS/390 or MSP/EX environments, specify the SORTED option.</p>

---

Action	Steps
<p>Consolidate, in the sequence in which they were created, the archive journal files created since the quiesce point established at the start of the backup procedure.</p> <p><b>When required:</b> This step is necessary only under the following conditions:</p> <ul style="list-style-type: none"><li>■ In OS/390 and MSP/EX environments, if the subsequent ROLLFORWARD utility statement will be executed with the SEQUENTIAL option and more than one archive journal file must be processed.</li><li>■ In a data sharing environment, if more than one member has updated the affected areas and the subsequent ROLLFORWARD utility statement will be executed with either the SEQUENTIAL or the ALL and STOP TIME options.</li></ul>	<p>Execute one of the following and use as input the properly concatenated set of archive files:</p> <ul style="list-style-type: none"><li>■ FIX ARCHIVE utility statement</li><li>■ MERGE ARCHIVE utility statement</li><li>■ EXTRACT JOURNAL utility statement</li><li>■ Another comparable utility</li></ul> <p><b>Note:</b> If consolidating archive files from multiple members and the subsequent rollforward will be executed with either the SEQUENTIAL or the ALL and STOP TIME parameters, use the MERGE ARCHIVE utility statement.</p> <p>►► For more information, see 19.10, “Data sharing recovery considerations” on page 19-44 later in this chapter.</p> <p><b>Note:</b> If recovery also involves local mode journal files, the MERGE ARCHIVE utility statement can be used to consolidate local mode journal files and archive files.</p> <p>►► For more information, see 19.9, “Recovery procedures for mixed-mode operations” on page 19-42 later in this chapter.</p>

---

Action	Steps
<p>Reapply to the restored files all updates made since the quiesce point established at the beginning of the backup procedure.</p> <p><b>When required:</b> Always.</p>	<p>Execute the ROLLFORWARD utility statement using either the consolidated journal file or individual archive files concatenated in the sequence in which they were created.</p> <p>If the journal files were consolidated using the EXTRACT JOURNAL utility, specify the FROM EXTRACT option.</p> <p>If FROM EXTRACT is not specified, then the following considerations apply:</p> <ul style="list-style-type: none"> <li>■ Specify the SORTED option unless there is insufficient disk space available. SORTED must be specified if: <ul style="list-style-type: none"> <li>– A consolidated journal file is not used as input in OS/390 and MSP/EX environments and more than one archive file must be processed.</li> <li>– The input journal file is on a device, such as a disk or a 3490 that does not support reading backwards.</li> <li>– Running ROLLFORWARD in a VM/ESA environment.</li> </ul> </li> <li>■ If the SEQUENTIAL option is used and the quiesce point for the affected areas does not coincide with the start of the first input file, use the START TIME parameter to identify the quiesce point.</li> </ul>

*Restore procedure 2:* The use of this approach requires that:

- Two quiesce points were established during the hot backup procedure
- Backward read is supported for the input journal files. Backward read is not available in VM/ESA environments nor when the journal files reside on disk or a device such as a 3490

If either of these conditions are not satisfied, the first recovery approach must be followed.

Action	Steps
<p>Copy the files that need to be recovered from the backup.</p> <p><b>When required:</b> Always.</p>	<p>Execute the RESTORE utility statement using the FILE option or another comparable utility.</p>
<p>Identify the two quiesce points that were taken during the backup process. Also identify the archive journal files that were created between those quiesce points and after the second quiesce point.</p> <p><b>When required:</b> Always.</p>	<p>Use the PRINT JOURNAL utility statement, or if the quiesce point was established using the DCMT QUIESCE command, examine the operating system log for the DC/UCF system on which the DCMT command was issued.</p>
<p>Consolidate, in the sequence in which they were created, the archive journal files created between the two quiesce points established during the backup procedure.</p> <p><b>When required:</b> This step is necessary only in OS/390, MSP/EX, and data sharing environments if more than one archive journal file must be processed.</p>	<p>Execute one of the following and use as input the properly concatenated set of archive files:</p> <ul style="list-style-type: none"> <li>■ FIX ARCHIVE utility statement</li> <li>■ MERGE ARCHIVE utility statement</li> <li>■ Another comparable utility</li> </ul> <p><b>Note:</b> If consolidating archive files from multiple data sharing members, use the MERGE ARCHIVE utility statement. For more information, see 19.10, “Data sharing recovery considerations” on page 19-44 later in this chapter.</p>
<p>Reapply to the restored files all updates made between the two quiesce points.</p> <p><b>When required:</b></p>	<p>Execute the ROLLFORWARD utility statement specifying the SEQUENTIAL option and using either the consolidated journal file or individual archive files concatenated in the sequence in which they were created.</p> <p>If the first quiesce point for the affected areas does not coincide with the start of the first input file, use the START TIME parameter to identify the quiesce point.</p> <p>If the second quiesce point does not coincide with the end of the last input file, use the STOP TIME parameter to identify the second quiesce point.</p> <p><b>Note:</b> Output from the EXTRACT utility statement cannot be used to apply the images during this step.</p>

Action	Steps
<p>Consolidate, in the sequence in which they were created, the archive journal files created after the second quiesce point established during the backup procedure.</p> <p><b>When required:</b> This step is necessary only under the following conditions:</p> <ul style="list-style-type: none"> <li>■ In OS/390 and MSP/EX environments, if the subsequent ROLLFORWARD utility statement will be executed with the SEQUENTIAL option and more than one archive journal file must be processed.</li> <li>■ In a data sharing environment, if more than one member has updated the affected areas and the subsequent ROLLFORWARD utility statement will be executed with either the SEQUENTIAL or the ALL and STOP TIME options.</li> </ul>	<p>Execute one of the following and use as input the properly concatenated set of archive files:</p> <ul style="list-style-type: none"> <li>■ FIX ARCHIVE utility statement</li> <li>■ MERGE ARCHIVE utility statement</li> <li>■ EXTRACT JOURNAL utility statement</li> <li>■ Another comparable utility</li> </ul> <p><b>Note:</b> If consolidating archive files from multiple members and the subsequent rollforward will be executed with either the SEQUENTIAL or the ALL and STOP TIME parameters, use the MERGE ARCHIVE utility statement.</p> <p>►► For more information, see 19.10, “Data sharing recovery considerations” on page 19-44 later in this chapter.</p> <p><b>Note:</b> If recovery also involves local mode journal files, the MERGE ARCHIVE utility statement can be used to consolidate local mode journal files and archive files.</p> <p>►► For more information, see 19.9, “Recovery procedures for mixed-mode operations” on page 19-42 later in this chapter.</p>

Action	Steps
<p>Reapply to the restored files, all updates made since the second quiesce point</p> <p><b>Note:</b> Updates made prior to the second quiesce point may also be reapplied during this step; however there is no need to do so.</p> <p><b>When required:</b> Always.</p>	<p>Execute the ROLLFORWARD utility statement using either the consolidated journal file or individual archive files concatenated in the sequence in which they were created.</p> <p>If the journal files were consolidated using the EXTRACT JOURNAL utility, specify the FROM EXTRACT option.</p> <p>If FROM EXTRACT is not specified, then the following considerations apply:</p> <ul style="list-style-type: none"> <li>■ Specify the SORTED option unless there is insufficient disk space available to perform the sort. SORTED must be specified if a consolidated journal file is not used as input in OS/390 and MSP/EX environments and more than one archive file must be processed.</li> <li>■ If the SEQUENTIAL option is used and the quiesce point for the affected areas does not coincide with the start of the first input file, use the START TIME parameter to identify the quiesce point.</li> </ul>

### 19.4.3 Reducing recovery time

*Ways to reduce recovery time:* It is often critical to recover a database as quickly as possible in order to meet availability demands. The length of time it takes to recover can be reduced by:

- Limiting the scope of the recovery
- Reducing the time between backups
- Sorting journal images
- Pre-processing archive files

*Limiting scope of recovery:* One of the most significant factors affecting recovery time is the number of files being recovered. If recovering due to an I/O error, only a single file may need to be recovered. If recovering due to a journal I/O error, it may be necessary to recover all files in the database. To reduce time, recover only those files or areas impacted by the failure.

*Reducing time between backups:* Another factor that affects recovery time is the number of journal images that must be applied to a restored file. One way to reduce the volume of journal images is to backup more frequently. Backups should be taken frequently enough that recovery times meet your operational requirements.

*Sorting journal images:* Another way to reduce the number of journal images applied to a restored file is to use the SORTED option of the ROLLFORWARD or ROLLBACK utility statement. By specifying this option, only the last AFTR image (in the case of ROLLFORWARD) or the first BFOR image (in the case of ROLLBACK) is applied to the database. While time and resources are required to sort the journal images, the number of I/Os to the database (and therefore the length of time needed to recover) may be significantly reduced using this option.

**Note:** There are restrictions on the use of the SORTED option when recovering from a hot backup. For more information, see 19.4.2, “Recovery from a hot backup” on page 19-21 earlier in this section.

*Preprocessing archive files:* Another way to reduce the time needed to recover is to preprocess journal images using the EXTRACT JOURNAL utility statement. This utility eliminates redundant journal images by retaining only the last AFTR image for a dbkey. It creates an output file (called an extract file) that subsequently can be used as input to the ROLLFORWARD utility statement.

A backup plan may include the regular use of EXTRACT JOURNAL to pre-process archive journal files. If a recovery then becomes necessary, the extract files already exist and can be used in place of the original archive files to reduce the volume of journal images that must be applied to the database, thereby reducing the length of time it takes to recover.

To illustrate how this may be done, the EXTRACT JOURNAL utility might be executed each night. Its input would consist of all archive files produced since the previous night's extract or since the previous backup, whichever occurred most recently. If a recovery becomes necessary, the EXTRACT JOURNAL utility must be executed one more time to process the remaining archive files. After the database files are restored from the backup, the ROLLFORWARD utility is used to reapply updates. Its input is the concatenated set of extract files produced since the backup.

**Note:** There are restrictions on the use of extract files when recovering from a hot backup.

►► For more information, see 19.4.2, “Recovery from a hot backup” on page 19-21 earlier in this section.

►► For considerations in the use of the EXTRACT JOURNAL utility statement in a data sharing environment, see 19.10, “Data sharing recovery considerations” on page 19-44 later in this chapter.

## 19.4.4 Recovering a large number of files

*Operating system file limitations:* Some operating systems impose a limit on the number of files that can be accessed within a single job step. Except when exploiting extended file support in OS/390, the limit for a central version is the same as that for a batch job and so there are no special considerations involved in recovery.

*Extended file support:* CA-IDMS has extended the number of files that can be accessed by a central version in an OS/390 operating system to exceed that which can be accessed by a batch job step. While useful, this capability may impact manual recovery.

*Extended file support and manual recovery:* Under rare circumstances, it may be necessary to recover more files than can be accessed by a single batch job step. If this occurs, it will be necessary to split the recovery operation into multiple job steps each of which recovers a subset of the areas, files or segments within the DMCL. Each job step can access up to 3273 files.

## 19.5 Recovery procedures after a warmstart failure

**Before you begin:** Before you begin the recovery process, determine why the warmstart failed. Start by checking any shutdown or warmstart messages. The failure could be due to:

- Changes made to the DMCL or startup JCL
- Hardware problems
- Software maintenance

**Corrective action:** If the failure is due to:

Change	Action
Changes in the DMCL and a timestamp mismatch is detected	Warmstart the system using the prior version of the DMCL load module
Changes in the startup JCL	Correct the JCL and restart the system
Software maintenance	Backup the maintenance and restart the system

**Steps:** In the unlikely event that hardware or software problems prevent the warmstart process from recovering the database, follow these steps:

Action	Statement
Offload all journal files	ARCHIVE JOURNAL with the FULL option to offload all full journal files. This should be followed by an ARCHIVE JOURNAL with the READ option to offload the journal that was active when the abnormal system failure occurred.
Recover the transactions that were active at the time of the system failure (that is, abended transactions)	ROLLBACK with the ACTIVE option
Unlock the areas that were not accessed during the rollback process. The ROLLBACK statement identifies what areas it unlocked.	UNLOCK
Reinitialize the journal files	FORMAT with the JOURNAL option

**Data sharing considerations:** If a member of a data sharing group is unable to warmstart and manual recovery must be undertaken, any shared area that was being updated by the failing member must be quiesced in all other members of the data sharing group before the ROLLBACK utility is executed. To quiesce the area, change

its status to OFFLINE or TRANSIENT RETRIEVAL. Do not use the DCMT QUIESCE command to quiesce the area.

►► For additional data sharing considerations, see 19.10, “Data sharing recovery considerations” on page 19-44 later in this chapter.

## 19.6 Recovery procedures from database file I/O errors

**What an I/O error means:** An I/O error occurring on a database file indicates that the file either cannot be read or cannot be written to. This may be caused by hardware malfunctions such as a channel problem, which if corrected, means that no recovery operation is needed. An I/O error can also be caused by a physically damaged file or disk device; this type of error requires recovery of the file.

**Identifying a database file I/O error:** When CA-IDMS/DB encounters an I/O error in a database file, the following events occur:

1. CA-IDMS/DB issues one of the following messages:
  - **DC205007**, which indicates a read error
  - **DC205008**, which indicates a write error
2. The transaction abends with a code of 3010 or 3011.
3. CA-IDMS/DB performs automatic recovery processing.

**If recovery is successful:** If the recovery process is successful, CA-IDMS/DB continues processing. To fix the I/O error, you must follow these steps:

Action	Statement
Take the area(s) associated with the bad database file offline	DCMT VARY AREA with the OFFLINE option
Identify the problem and fix it. If the problem is not associated with the database file itself (for example, the problem is due to a bad channel), perform step 3 after the problem is corrected; if the problem is due to a damaged file, perform the steps outlined for an unsuccessful recovery.	
Bring the area(s) associated with the database file online	DCMT VARY AREA with the ONLINE option

**If the recovery is unsuccessful:** If the recovery process is unsuccessful, CA-IDMS/DB suspends the transaction and issues the following message:

DC205009 TRANSACTION SUSPENDED. TRANSACTION ID: transaction-id

When CA-IDMS/DB issues this message, quiesce the area in which the problem occurred as quickly as possible to prevent additional transactions from readying the area. The table below identifies all the steps:

Action	Statement
Quiesce the affected area (see considerations below)	DCMT VARY AREA with the TRANSIENT RETRIEVAL or OFFLINE options
Switch to a new journal file	DCMT VARY JOURNAL
De-allocate the file	DCMT VARY FILE with the DEALLOCATE option; use the FORCE option if the file cannot be closed (for example, because of a channel problem)
Restore a copy of the damaged file using the last backup tape as input. If the FORCE option was used in step 3, recreate the file with a new name	RESTORE with the FILE option
Rollforward the restored copy of the file using the archive journal files in the order they were created	ROLLFORWARD FILE with the ALL option
If the file was restored to a new location: <ul style="list-style-type: none"> <li>■ Recatalog it in OS/390 and BS2000/OSD</li> <li>■ Update the standard labels in VSE/ESA</li> </ul>	Operating system facilities
If the file was renamed in OS/390, BS2000/OSD, or VM/ESA, change its dataset name	DCMT VARY FILE with the DSNAME option
Make the new file available to the central version	DCMT VARY FILE with the ALLOCATE option
Re-activate the suspended transactions so they complete automatic recovery	DCMT VARY FILE with the ACTIVE option
Re-activate the area for update processing	<ul style="list-style-type: none"> <li>■ If the area was varied OFFLINE, issue DCMT VARY AREA with the ONLINE option</li> <li>■ If the area was varied to TRANSIENT RETRIEVAL mode, first vary it OFFLINE and then ONLINE</li> </ul>

---

## Considerations

*Quiescing the area:* Quiesce the area by varying it offline or retrieval. The differences are:

- If the area is varied offline, no new transactions will be able to access the area until the recovery is complete and the area is varied online; existing transactions will complete if possible.
- If the area is varied to transient retrieval, transactions can continue to read data from the area but cannot update until the recovery is complete and the area is varied offline and back online. This may be useful if the area is mapped to many files (only one of which is damaged) or if only a small portion of the file is damaged. It can also be beneficial if most of the file blocks are in a buffer or a dataspace.

If the area to be recovered is a system area, it may be necessary to terminate predefined system run units by issuing a `DCMT VARY RUN UNIT ... OFFLINE` command in order to quiesce activity to the area. It is advisable to vary the status of a system area to transient retrieval rather than offline.

In a data sharing environment, it is important to quiesce a shared area in all members of the data sharing group. The broadcast capability of DCMT commands can be used to do this easily.

*Renaming the file:* If you restored the file under a new name, you must do one of the following:

- Rename (and recatalog) the restored file to its original name before restarting the DC/UCF system.
- Alter the system startup JCL to reference the new dataset name.
- After recovery is complete, modify the dataset name in the definition of the file, regenerate all DMCLs which include the file's segment and make the new DMCL available to the DC/UCF system.

►► For more information on making a DMCL available to a runtime system, see Chapter 4, “Defining, Generating, and Punching a DMCL” on page 4-1.

If you fail to do one of the above, CA-IDMS/DB will attempt to access the wrong file the next time the system is started. This may have serious consequences if the original file still exists.

*Use of deallocate force:* If the damaged file was de-allocated using the FORCE option, the DC/UCF system marks the file as closed and de-allocated but does not actually issue the corresponding operating system requests. For this reason, you must restore the file under a different dataset name. When the DC/UCF system is eventually shutdown, it will not shutdown successfully because the operating system will attempt to close the original file. This will either cause an abend or the DC/UCF system will hang. In either case, examine the messages produced on the log. If the

following message appears, the database system has completed processing and no additional action is required:

```
DC200010 CA-IDMS/DB Inactive
```

If this message does not appear, you should restart the system (after taking appropriate steps such as renaming the file) and then shut it down.

*Correcting the lock option of an area and file:* If the area associated with a damaged database file is in retrieval mode or offline and the file was restored with the area lock on, then the area status is incompatible with the file status. If you try to vary the area online, IDMS responds with an error. To correct this situation, issue a DCMT VARY AREA command with the UPDATE LOCKED option. This command allows IDMS to vary the area to an update mode even though the file is locked.

## 19.7 Recovery procedures from journal file I/O errors

**What happens:** If an I/O error is encountered when accessing a journal file, the system responds differently depending on whether a read or write error is encountered:

- If a write error occurs, CA-IDMS/DB swaps to a new journal file, re-issues the journal write and disables the use of the file on which the error occurred.
- If a read error occurs, CA-IDMS/DB writes message DC205007 to the system log, indicating a read error, and it disables the journal file from further use.

The DC/UCF system will continue to operate without the use of the damaged journal file, although processing may be slower due to the availability of fewer journal files.

**Automatic recovery failure:** If a transaction abends (or issues a rollback) and, in order to recover, CA-IDMS/DB must access a disabled journal file, it places the failing transaction in a suspended state and issues the following message to the log:

```
DC205009 Transaction suspended. Transaction id xxxxxx
```

**Recovery procedure steps:** To recover from an I/O error on a journal file, follow these steps:

<b>Action</b>	<b>Statement</b>
1. Quiesce all activity within the system	For every area in update mode within the system (except load, scratch, queue and log areas), issue DCMT VARY AREA or SEGMENT with the TRANSIENT RETRIEVAL option
2. Monitor the transactions within the system	DCMT DISPLAY TRANSACTIONS
3. If all the areas quiesce and the only transactions that remain are internal rununits:	
3.1. Backup the areas that were in update mode at the time of the error	BACKUP using the FILE option
3.2. Initialize the damaged journal file	FORMAT JOURNAL
3.3. Re-activate the areas within the system	DCMT VARY AREA or SEGMENT
4 If the areas will not quiesce and the only transactions left are suspended, cancel the system	Operating system facilities
5. Restore all areas that were open at the time of the I/O error (including load and queue areas)	RESTORE
6. Roll forward all restored areas using the archive journal files created since each backup was taken	ROLLFORWARD with the COMPLETE and AREA options
7. Initialize all journal files	FORMAT JOURNAL with the ALL option
8. Backup all recovered database areas	BACKUP with the FILE option
9. Re-start the system	
10. Re-run all transactions that were not recovered	

### Considerations

*Quiescing system activity:* The approach outlined above quiesces activity within the system by varying the areas to a transient retrieval mode. This allows the system to remain active and capable of processing retrieval transactions until it is determined whether or not manual recovery is necessary. Varying the areas offline or attempting to shut the system down would also be valid alternatives.

In a data sharing environment, it is important to quiesce a shared area in all members of the data sharing group. The broadcast capability of DCMT commands can be used to do this easily.

*Do not cancel tasks:* Under no circumstances should you cancel a task or batch job executing under the central version if a journal I/O error is encountered. By cancelling the task, there is more chance that automatic recovery will fail because of the damaged journal file, thus necessitating manual recovery.

*Conservative approach:* The steps outlined above take a conservative approach to the recovery process in two ways:

- No attempt is made to try and offload the damaged journal file. If the file can be offloaded, then the areas can be recovered using ROLLBACK (with the ACTIVE option) rather than using RESTORE and ROLLFORWARD.
- All areas being updated by the system are recovered. If you identify the areas that were being updated by the suspended transactions, then recovery can be limited to those areas and **other areas which are logically-associated**. To identify the areas that were being updated, you can use the DCMT DISPLAY TRANSACTION command for the suspended transactions or the DCMT DISPLAY AREA command which will identify the areas that have not quiesced.

## 19.8 Recovery procedures for local mode operations

**What follows:** Recovery procedures for local mode operations differ depending on whether or not you are journaling and if so, whether you are journaling to a disk device or tape device. The topics below provide the recovery procedures for each situation.

### 19.8.1 No journaling

**Use the backup file:** If you are not maintaining journal files during execution of a local mode job and the job terminates abnormally, you must restore all areas updated by the local mode application.

### 19.8.2 Journaling to a tape device

**Steps:** To recover a local mode database when journaling to a tape device, follow these steps:

Action	Statement
Rollback the database or areas of the database using as input the tape journal file created by the local mode job	<ul style="list-style-type: none"> <li>■ If the job can be re-started from the last COMMIT point, use ROLLBACK with the ACTIVE option</li> <li>■ If the job has to be re-run from the beginning, use ROLLBACK with the ALL option</li> </ul>
Re-run the application	

### 19.8.3 Journaling to a disk device

**Steps:** If you are journaling a local mode job to a disk device, follow these steps:

Action	Statement
Copy the journal file to a tape device	Operating system utility
Follow the steps outlined above for journaling to a tape device above	

### 19.8.4 Using an incomplete journal file

**What is an incomplete journal file?:** An incomplete journal file is a journal file that does not contain a final ABRT checkpoint for the active transaction or even an end-of-file mark. This occurs when the journal file has been unexpectedly interrupted, for example, when the operating system crashes. An incomplete journal file is not

suitable for recovering your database. To make a suitable journal file for recovery, use the `FIX JOURNAL` utility statement, which:

- Reads the damaged file and creates a new one
- Writes an ABRT checkpoint at the end of the new file

**Steps:** To recover a database in local mode, using an incomplete journal file, follow these steps:

---

<b>Action</b>	<b>Statement</b>
Fix the journal file	<code>FIX JOURNAL</code>
Recover the database using the output from the <code>FIX JOURNAL</code> utility statement as described for journaling to a tape device above.	

---

## 19.9 Recovery procedures for mixed-mode operations

**What is a mixed-mode operation?:** When database areas have been updated both in local mode and under the central version (for example, when an area has been varied offline, subsequently updated by a local transaction that used journaling, and then varied back online), the database must be restored by using both the local and the central version journals.

**Mixed mode recovery:** The following scenario is an example of synchronizing the recovery operations by explicitly using both the central version and local journals to ensure proper recovery of all database areas:

6 a.m.	Nightly backups taken
8 a.m.	System startup: AREA1, AREA2, AREA3 are readied in update mode under the central version.
10:30 a.m.	AREA1 is varied offline. While offline, a local mode program (using a tape journal) updates AREA1 while the central version continues to update AREA2 and AREA3.
11:30 a.m.	A VARY JOURNAL command is issued for the central version journal. AREA1 is varied back online and the central version continues to update AREA1, AREA2, and AREA3.
12:00 p.m.	Database file I/O error occurs on AREA1.

When the database file I/O error occurs, the affected file associated with AREA1 must be restored by using both the local and central version journals.

**Steps to recover the database:** The following steps illustrate one approach to recovery, given the situation outlined above. Note that with this approach two separate rollforward operations are used. In order to process journal images from both central version and local mode operations in a single execution of the ROLLFORWARD utility, you must use the alternate recovery approach described below.

---

<b>Action</b>	<b>Statement</b>
Restore the damaged file using the backup tapes produced at 6 a.m.	RESTORE with the FILE option
Rollforward all archive files produced before 11:30	ROLLFORWARD FILE specifying ALL
Rollforward the local journal file, restoring the file up to 11:30 a.m.	ROLLFORWARD FILE specifying ALL
Rollforward the second archive journal file	ROLLFORWARD FILE with the COMPLETE option

---

Complete the recovery process by following the steps outlined in 19.6, “Recovery procedures from database file I/O errors” on page 19-33 earlier in this chapter.

**An alternate approach:** The following steps illustrate an alternate approach to recovery in a mixed-mode environment. With this approach, the local mode journal file is first merged with the archive files produced by the central version and the merged output file is used to recover the database in a single rollforward operation.

---

<b>Action</b>	<b>Statement</b>
Restore the damaged file using the backup tapes produced at 6 a.m.	RESTORE with the FILE option.
Merge the local mode journal file with the archive files produced since 8 a.m.	MERGE ARCHIVE specifying the COMPLETE option
Rollforward all updates made since the backup was taken	ROLLFORWARD FILE specifying the COMPLETE option.

---

Complete the recovery process by following the steps outlined in 19.6, “Recovery procedures from database file I/O errors” on page 19-33 earlier in this chapter.

## 19.10 Data sharing recovery considerations

*Quiescing update activity:* Whenever it becomes necessary to quiesce access to an area during a recovery operation, the quiesce must apply to all members of a data sharing group. For recovery purposes, the quiesce will usually be done by varying the area status to OFFLINE or TRANSIENT RETRIEVAL using a DCMT VARY command. This command must be executed in every member to establish a group-wide quiesce for a shared area. To do this easily, the command may be broadcast to other members of the group.

Occasionally, it will be sufficient to quiesce update access to an area through the use of a DCMT QUIESCE command. This command will automatically propagate the quiesce for a shared area to all group members, so there is no need to execute it on more than one member of the group.

►► For more information on DCMT commands and how to broadcast them, refer to *CA-IDMS System Tasks and Operator Commands*.

*Recovery from a warmstart failure:* If warmstart fails for a single member of a data sharing group, recovery can proceed just as if the DC/UCF system were not a data sharing member provided that all shared areas being updated by the member at the time of failure are quiesced in other members of the group. The quiesce should be done by varying the status of the affected areas to OFFLINE using a DCMT VARY command.

If warmstart fails for more than one member, each member can be recovered independently provided that:

- Only the ROLLBACK recovery utility is used
- The ACTIVE option is specified when executing ROLLBACK, and
- Executions of the ROLLBACK utility are serialized

Failure to comply with these conditions, may result in database corruption.

*Recovery from other types of failures:* Except when following the above procedure to recover from a warmstart failure, the archive files from all members that have updated a shared area since the backup was taken must be included in any manual recovery. Furthermore, the journal images from all members must be processed together in the same execution of the ROLLFORWARD or ROLLBACK utility. It is not valid to process the images from one member in one execution followed by the images from another member in another execution, since journal images must be processed in chronological sequence.

Recovery from a warmstart failure is an exception to this rule only because records updated by one member cannot be accessed by another member until the changes are committed or rolled out. If warmstart fails, the unrecovered records remain locked, so no other member can update them. This means that there will never be more than a

single member with a before image for an unrecovered record and so inter-member sequencing is not important.

*Using MERGE ARCHIVE:* The MERGE ARCHIVE utility is used to merge the journal images from multiple members so that they are in chronological sequence. As noted above, most recovery utilities require that journal images be processed chronologically. In a data sharing environment, the journal images produced by each member are in chronological sequence, but the images for areas concurrently updated by multiple members are contained in each member's archive files. The MERGE ARCHIVE utility interleaves the journal images from multiple members so that they occur in date/time sequence. The resulting output file may then be used as input to a ROLLFORWARD, ROLLBACK, or EXTRACT JOURNAL utility statement.

When executing the MERGE ARCHIVE utility statement, the input consists of a concatenated set of archive files and optionally a merge archive file produced from a previous execution of the MERGE ARCHIVE utility. Archive files produced by a single member must be processed in the order in which they were created. Archive files from different members may be processed in any order relative to those of other members.

*When to use MERGE ARCHIVE:* The output of the MERGE ARCHIVE utility can always be used as input to the ROLLFORWARD, ROLLBACK, and EXTRACT JOURNAL utility statements in place of the original archive files. It can also be used to combine local mode journal files and archive files when mixed-mode updates must be recovered.

However, while optional in most cases, MERGE ARCHIVE **must** be used to merge the journal images of multiple data sharing group members before those images are processed by:

- ROLLFORWARD or ROLLBACK utility statements that specify the SEQUENTIAL option.
- ROLLFORWARD, ROLLBACK, or EXTRACT JOURNAL utility statements that specify both the ALL and STOP TIME options.

*Using EXTRACT JOURNAL:* The EXTRACT JOURNAL utility is used to preprocess journal images in order to reduce recovery time. This utility can also be used in a data sharing environment. Any of the following are valid approaches to its use:

- Separately preprocess the archive files of each member
- Preprocess the archive files of multiple members together
- Merge the archive files of multiple members using the MERGE ARCHIVE utility and then preprocess the merge file

You must use the third approach if the ALL and STOP TIME parameters are specified on the EXTRACT JOURNAL utility statement; otherwise, any of the above approaches can be used to preprocess journal files in a data sharing environment.

If using either of the first two approaches, the EXTRACT JOURNAL utility statement can be executed on a periodic basis to preprocess the archive files created since its previous execution, or since a backup was taken. If recovery becomes necessary, all extract files produced since the backup must be concatenated as input to a single execution of the ROLLFORWARD utility. The order in which the extract files are concatenated must be such that the journal images for each member are in chronological sequence. It makes no difference in which order the images of one member occur in relation to those of another member.

If using the third approach, the entire set of archive files produced by group members that have updated the affected areas must be merged prior to executing the EXTRACT JOURNAL utility. The MERGE ARCHIVE utility can be executed on a periodic basis to merge the archive files created since its previous execution with the previously created merge file. The EXTRACT JOURNAL utility can then be used to preprocess the final merge file.

►► For more information on executing both the MERGE ARCHIVE and the EXTRACT JOURNAL utility statements, refer to *CA-IDMS Utilities*.

*Coupling facility failures:* Certain types of failures are unique to a data sharing environment, such as the loss of a coupling facility or a structure within the coupling facility. In some cases, all members of a group will fail and recovery must be coordinated across the group, a process called "group restart."

►► For more information on recovering from coupling facility failures and group restart, refer to *CA-IDMS System Operations*.

## 19.11 Considerations for recovery of native VSAM files

**About recovery for native VSAM files:** CA-IDMS/DB performs journaling for native VSAM files just like it does for other types of files it supports. The recovery procedures described in this chapter apply to native VSAM files also. The processing difference is that the BACKUP and RESTORE utility statements cannot be used with native VSAM files. Instead, use IDCAMS or some other utility for backing up and restoring the file.

**Potential problems:** Since VSAM controls the actual updating of the data sets, recovery problems may occur. If a total system failure occurs after CA-IDMS/DB passes control to VSAM, automatic recovery is not guaranteed. Therefore, you should back up native VSAM data sets frequently, as described in the appropriate VSAM documentation. Recovery can then be accomplished by restoring your file Using IDCAMS (or some other utility) and ROLLFORWARD utility statements.

**File verification after failure:** If a DC/UCF system fails or a local mode application terminates abnormally, you must issue the IDCAMS VERIFY command for native VSAM files that were open for update at the time of the failure.

**Limitations for ESDS areas:** You cannot use the ROLLBACK utility statement for an ESDS area to which a record has been added, because VSAM does not allow the necessary erase.

**Limitations for KSDS areas:** Due to limitations within the VSAM access method, ROLLFORWARD and ROLLBACK cannot be run with the SORTED option to recover native VSAM KSDS areas. If you need to use the SORTED option, because of the volume of data, and a database that contains a mixture of KSDS, ESDS, and/or RRDS native VSAM files, follow these steps:

Action	Statement
Restore the native VSAM files	Operating system facility
Rollforward or rollback the area that maps to the KSDS file; the utility statement recovers the KSDS file and any associated alternate indexes.	ROLLFORWARD or ROLLBACK with the SEQUENTIAL option
Rollforward or rollback all the remaining areas or files.	ROLLFORWARD or ROLLBACK with the SORTED option



# Chapter 20. Loading a Non-SQL Defined Database

---

- 20.1 About database loading . . . . . 20-3
- 20.2 Loading database records using FASTLOAD . . . . . 20-4
  - 20.2.1 General considerations . . . . . 20-4
- 20.3 FASTLOAD procedure . . . . . 20-6
- 20.4 Loading database records using a user-written program . . . . . 20-7
  - 20.4.1 Organizing input data for a user-written program . . . . . 20-7
  - 20.4.2 Loading the database . . . . . 20-9
- 20.5 Related information . . . . . 20-11



## 20.1 About database loading

**Loading options:** To load a database defined with non-SQL DDL statements, you can use either:

- The FASTLOAD utility statement
- A user-written load program

**FASTLOAD utility statement:** To use FASTLOAD, you must write and compile a format program that specifies how to load the data. After executing the format program, you invoke the FASTLOAD utility statement, which loads record occurrences into the database and makes set connections using the output from the format program. It also builds indexes during the load process.

**User-written program:** You can also load a database by using DML commands in a user-written application. The application can be written in any of the languages CA-IDMS/DB supports.

If you use a user-written program to load the database, you should organize the record occurrences in the input file so that they mimic the structure of the database. For example, you should sort the records so that a CALC record is followed by its VIA member record occurrences. Steps for organizing the input file appear in more detail later in this chapter.

**Advantages of FASTLOAD:** FASTLOAD is often more efficient than a user-written program for loading a database with complicated structures (for example, multiple-member sets or multi-level record relationships). In addition, FASTLOAD does not require pre-sorted data. As part of its internal processing, FASTLOAD sorts the data at certain points during the load process.

## 20.2 Loading database records using FASTLOAD

**Requires a user-written format program:** To use FASTLOAD, you must write a format program that uses subroutines provided by CA to prepare data for input to the FASTLOAD utility statement.

►► For a description of the format program, refer to the FASTLOAD statement in *CA-IDMS Utilities*.

### 20.2.1 General considerations

**Always load in local mode:** You must load a database in local mode. Journaling is not required, and is not recommended when loading a database for these reasons:

- The load utility does not maintain checkpoints
- It's easier to re-run a failed job step than to recover the database
- Journaling can impact performance

#### Cross-area sets

- If the owner and member records of an automatic set exist in different database areas, load the areas together.
- If the owner and member records of a manual set exist in different database areas, either:
  - Load the areas together
  - Run a user-written program to connect the records after loading the entire database

**CALC records:** The target page for CALC records to be loaded into a database can be determined in one of two ways:

- By the standard CA-IDMS/DB CALC routine (IDMSCALC).
- By a user-written CALC exit routine (IDMSCLCX) that was compiled and link-edited with IDMSUTIL.

**Important:** If you determine the target page using IDMSCALC, you must use it whenever the database is accessed; likewise, if you use the IDMSCLCX user exit, you must link-edit it with IDMSDBMS.

**Compressed data:** If the schema definition specifies compression for a record type, CA-IDMS/DB compresses the record before it stores it during a load operation. Therefore, before you begin the load procedure, be sure the schema definition includes the information it requires to compress the record occurrences.

**Reserving space on the page:** To reserve space for the storage of additional records on a page or for increases in the length of records stored on a page, add an area override to the DMCL that specifies a page reserve. When the load is complete, you can remove the area override.

**Buffers:** The DMCL that you use to load the database should contain a local mode buffer that contains at least 10 pages. One large buffer should be sufficient. However, you may obtain performance improvements by assigning the files associated with each area to a separate buffer. If you don't have enough resources, then try to assign the files associated with the following areas to separate buffers:

- Index area
- Areas for which the owner record exists in one area and the member record exists in another area

**Considerations for large databases:** A large database should be loaded in portions. The FASTLOAD statement assumes that all record occurrences that are connected by automatic sets will be loaded at the same time. For a large database, this assumption can be limiting. To load a large database:

1. Group the record types so that there are not automatic sets between the groups
2. Load each group of record types
3. If manual set connections exist between records in different groups, connect the records by executing a user-written program

**Subschema requirements:** The subschema that you use in the load process must:

- Include all records being loaded and all set relationships in which the records participate
- Allow all affected areas to be readied in exclusive update mode

## 20.3 FASTLOAD procedure

**Steps:** To load a database for the first time, follow these steps:

1. Write and compile a format program that specifies how to load the data. Refer to *CA-IDMS Utilities* for information about the format program.
2. Link-edit the format program with IDMSDBLU
3. Define the segments, areas, and files that represent the physical database
4. Add the segment definition to the DMCL and make the DMCL available to the runtime environment
5. Format the database files to be loaded using the FORMAT utility statement with the FILE option
6. Execute the format program
7. Load the database using the output from the format program as the input to FASTLOAD
8. Back up the database areas using the BACKUP utility statement or any comparable backup utility
9. Verify the validity of the loaded database using:
  - IDMSDBAN, to verify the physical integrity of the database
  - CA-OLQ, CA-CULPRIT, or some other retrieval job to verify the data in the database

## 20.4 Loading database records using a user-written program

**What follows:** Before you load a database using a user-written program, you must first organize the data in the input file. What follows is a discussion of how to organize the record occurrences followed by the procedure to load the database.

### 20.4.1 Organizing input data for a user-written program

**Organize record occurrences to match schema:** To make the database load as efficient as possible, you need to organize the record occurrences to match the structure of the database. For example, you want a CALC owner record to be followed by its VIA member records. The discussion below identifies how to organize the data.

**Step 1: Identify the record types:** The first step in organizing input data is to identify the type of each record. To identify the type of record, add the record's ID to the beginning of each record occurrence. For example, the ID of the DEPARTMENT record is 410; the ID of the EMPLOYEE record is 415.

**Step 2: Identify CALC clusters:** A CALC cluster is an occurrence of a CALC record, all of its VIA member records, and all VIA member records of a VIA member record occurrence. For efficient database processing, all the records within a CALC cluster should fit on one page (and thereby, can be processed with one I/O). If the records do not fit on one page, then store the most frequently accessed record types immediately following the CALC record occurrence so that they have a better chance of being stored on the same page as the owner.

**Step 3: Form CALC cluster hierarchies:** A hierarchy is a collection of CALC clusters. For example, if a CALC record occurrence in one cluster is owned by a record in another cluster, you have a hierarchy of CALC clusters. In the Commonweather database, both the OFFICE and DEPARTMENT records own occurrences of the EMPLOYEE record, which in turn owns VIA member record occurrences. In deciding what records to include in the CALC cluster hierarchy, consider the number of CALC record occurrences. For example, if the DEPARTMENT record has many more occurrences than the OFFICE record, then store the EMPLOYEE records immediately after the owning DEPARTMENT record. This potentially saves an I/O because you won't need to reestablish currency on the DEPARTMENT record occurrence later on.

Hierarchies are loaded from top-to-bottom, left-to-right order. When you store the owner of a CALC cluster, you establish currency to store the member of a CALC cluster.

**Step 4: Sort the records in a hierarchy:** To sort records within a hierarchy, add a prefix to the beginning of the record occurrence. The prefix contains the record id and sequence number for each level of the hierarchy. For example, the

DEPARTMENT, EMPLOYEE, EMPOSITION record hierarchy might have a prefix that looks like this:

ID and sequence number			Record EMPLOYEE	Record EMPOSITION
410/1	0/0	0/0	410	Department record 1
410/1	415/1	0/0	415	Employee record 1
410/1	415/1	420/1	420	Emposition record 1
410/1	415/1	420/2	420	Emposition record 2

**Step 5: Order the occurrences of each hierarchy:** A database page will typically hold more than one database cluster. Therefore, you can load multiple clusters with one I/O if you load all the hierarchies that target to the same database page. To sort the hierarchy occurrences, add the CALC target page number of the top cluster in the hierarchy to the beginning of the input record.

**Tip:** To determine the CALC target page, use IDMSCALC in the program that creates the input file; for more information on IDMSCALC, refer to *CA-IDMS Utilities*.

**Step 6: Include records excluded from the hierarchies:** Some records do not fall within a hierarchy. For example, suppose you did not include the OFFICE record, which owns EMPLOYEE record occurrences in a CALC cluster hierarchy. To load owner records that fall outside of a hierarchy:

1. Position the non-VIA owner records at the beginning of the input file, before any records that form part of a hierarchy, by adding an identifier to the beginning of each input record. For example, the identifier of the OFFICE record type might be 4 and the identifier of the DEPARTMENT, EMPLOYEE, EMPOSITION hierarchy might be 5.
2. Add the key of the non-VIA owner record to the end of the hierarchy record occurrence; at load time, use the key to find the owner before storing the member. For example, add the OFFICE-CODE-0450 field to the end of each EMPLOYEE record occurrence.

**Step 7: Order sorted and indexed sets:** Sorted sets should always be loaded in the same order as the sort sequence. To sort the input data:

- For a set within a hierarchy, replace the sequence number field at the record's level in the hierarchy with the sort key of the set; for example, if the EMP-EMPOSITION set is a sorted set, replace the sequence number for occurrences of the EMPOSITION record with the record's sort key in the prefix portion of the input record.

- For a set outside of a hierarchy, follow these steps:
  1. Re-define the set as manual
  2. Create a file containing records with these fields: the owner's page, the set name, the owner's CALC key, the set's sort key, the dbkey of the member record
  3. Sort the file in:
    - Descending order by page
    - Ascending order by set name and owner key
    - Either ascending or descending order by sort key
  4. After loading the database, connect the set members using a user-written program

**Step 8: Sort the input records:** Sort the input records in:

- Ascending order by identifier
- Descending order by target page number
- Ascending order by the concatenation of all ID and sequence fields that represent a hierarchy

**Note:** If records are to be stored VIA a system-level index, they should be sorted in the reverse order of their VIA index so records at the end of the index will be processed first by the user-written format program. This will insure that the physical sequence of the records on the database will match the sequence of the index.

## 20.4.2 Loading the database

To load a database for the first time using a user-written program, follow these steps:

Action	Statement
Write and compile a load program that specifies how to load the data.	
Optionally, tailor the DMCL to be used for the load operation	ALTER DMCL
If altered, make the DMCL available to the local mode runtime environment	See Chapter 4, “Defining, Generating, and Punching a DMCL” on page 4-1
Format the database files to be loaded	FORMAT with the FILE option
Load the database using as input the sorted input file	Execute the user-written program
If necessary, connect members to sets treated as manual during the load	Execute the user-written program
Back up the database areas	BACKUP or any comparable backup utility
Verify the validity of the loaded database	CA-OLQ, CA-CULPRIT, or some other retrieval job to verify the data in the database

**Example:** The following example shows code to load the DEPARTMENT hierarchy:

```

read an input record
repeat until end-of-file
  if record-id = 410
    move DEPARTMENT record
    store DEPARTMENT
  else if record-id = 415
    move OFFICE key
    find calc OFFICE
    move EMPLOYEE record
    store EMPLOYEE
  else if record-id = 420
    move JOB key
    find calc job
    move EMPOSITION record
    store EMPOSITION
  else if record-id = 425
    move SKILL key
    find calc SKILL
    move EXPERTISE record
    store EXPERTISE
  end-if
read next input record
end-repeat

```

## 20.5 Related information

- About utility statements mentioned in this chapter, refer to *CA-IDMS Utilities*
- About loading an SQL-defined database, see Chapter 21, “Loading an SQL-Defined Database” on page 21-1
- About the IDMSCLCX user exit, refer to *CA-IDMS System Operations*



# Chapter 21. Loading an SQL-Defined Database

---

- 21.1 About database loading . . . . . 21-3
- 21.2 Loading considerations . . . . . 21-7
- 21.3 Contents of the input file . . . . . 21-10
- 21.4 Loading procedures . . . . . 21-12
  - 21.4.1 Steps that apply to all load procedures . . . . . 21-12
  - 21.4.2 Full load procedure . . . . . 21-13
  - 21.4.3 Phased load procedure . . . . . 21-13
  - 21.4.4 Segmented load procedure . . . . . 21-15
  - 21.4.5 Stepped load procedure . . . . . 21-16
- 21.5 Related information . . . . . 21-20



## 21.1 About database loading

**Loading phases:** The table below summarizes the phases involved with loading an SQL-defined database. The load process was designed to accommodate both small databases and very large databases as well as allow flexibility in tailoring the load process to the characteristics of the data being loaded:

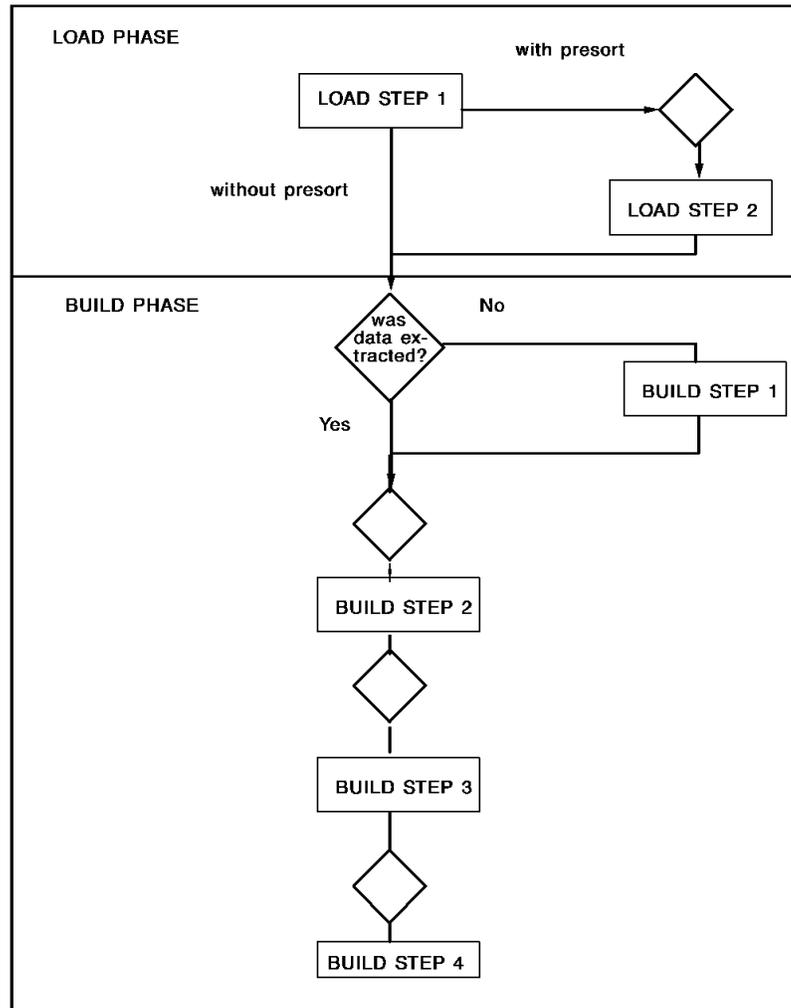
Phase	What it does
Load	Loads the specified tables
Build	Builds indexes and linked index constraints for the specified tables; this phase can be bypassed if neither linked index constraints or non-clustering indexes are defined on the specified tables
Validate	Validates referential constraints in which the specified tables participate

**Steps within phases:** Each of these phases, in turn, is composed of sub-phases called **steps**. The table below summarizes the function of each step:

---

<b>Phase</b>	<b>Step</b>	<b>What it does</b>
Load	Step 1	Processes data in preparation for sorting; this step can be bypassed if data is already sorted
	Step 2	<ul style="list-style-type: none"><li>■ Loads the table rows</li><li>■ Connects linked, clustered constraints</li><li>■ Builds clustering indexes</li></ul>
Build	Step 1	Performs an area sweep in the absence of an intermediate extract file
	Step 2	Finds the db-keys of rows that participate in the referenced table of a linked index referential constraint
	Step 3	Builds non-clustering indexes and linked indexes
	Step 4	Updates the prefixes of rows that participate as the referencing table of a linked index referential constraint
Validate	Step 1	Validates only those constraints that can be processed efficiently in a single pass and extracts information about other referential constraints
	Step 2	Validates any referential constraints bypassed in Step 1

**Load flow diagram:** The diagram below illustrates the load and build phases of the process described above:



**Loading options:** CA-IDMS/DB offers you the following loading options:

<b>Option</b>	<b>Description</b>	<b>When to use it</b>
Full load	Loads, builds and validates the specified tables	Always, unless special considerations apply
Phased load	Executes each phase (load, build, and validate) separately	When loading a number of tables one at a time or in groups; defer build and/or validate phases until all the tables have been loaded
Segmented load	Loads portions of input in separate operations	When loading extremely large tables; defer the build and validate steps until all the input records have been processed
Stepped load	Executes each step of a phase (load, build, and validate) separately	When loading extremely large tables for which external sort packages may be more efficient or when space for intermediate work files or tape drives is at a premium

**CA-IDMS/DB enforces all constraints during the load:** CA-IDMS/DB enforces all constraints during the load process. That is, it enforces:

- Referential constraints
- Data type constraints
- Check constraints
- Unique constraints

For example, if a table allows only specified values to be stored in a column, CA-IDMS/DB stores only valid values. CA-IDMS/DB also assigns default values for columns for which no input values are supplied, provided the column was defined to allow null or default values.

## 21.2 Loading considerations

**Using pre-sorted data:** Before CA-IDMS/DB loads data, it sorts the data using a sort sequence best suited to the table's characteristics. If you have already sorted the input data, you can tell CA-IDMS/DB to skip the sort phase.

**Providing sorted data:** To sort the data yourself, follow these recommendations to achieve the most efficient load for your tables:

Table characteristic	Recommended sort sequence
Table has a clustered index	Sort on index key
Table has a clustered referential constraint	Sort on foreign key of the referencing table
Table has a CALC key	Sort on CALC-key target page; to do this, use the IDMSCALC utility program to determine the target page and append the target page to the input record

**Database buffers used during load:** You must load a database in local mode. The DMCL that you use for the load should specify buffers for the areas being loaded that contain at least 10 pages. The larger the buffer, the more efficient the load.

**Reserving space on the page:** If you want to leave free space on the database pages following the load, add an area override in the DMCL that specifies a page reserve. After the load is complete, remove the area override so that new rows and index entries can use the free space. This technique is especially useful for areas that contain only indexes or that contain tables clustered on an index.

**Error handling:** CA-IDMS/DB may encounter errors during each phase of the load process. You can instruct CA-IDMS/DB what to do in response to these errors, for example, to continue processing or to quit following a specified number of errors. The table below summarizes the types of errors that can occur within each phase:

Phase	Type of error	Corrective action
All phases	<ul style="list-style-type: none"> <li>■ Table not defined in the catalog</li> </ul>	Define the table in the catalog
Load	<ul style="list-style-type: none"> <li>■ Check constraint violation</li> <li>■ Invalid data values</li> <li>■ Unique constraint violation on a CALC key, clustering index, or linked clustered constraint</li> <li>■ Referential constraint violation on a linked clustered constraint</li> </ul>	No corrective action needed; however, row is not inserted and subsequent build and validate phases may fail
Build	<ul style="list-style-type: none"> <li>■ Unique constraint violation on non-clustering index or linked index constraint</li> <li>■ Referential constraint violation on a linked index constraint</li> </ul>	FIX PAGE utility statement or reload data
Validate	Invalid referential constraint	<ul style="list-style-type: none"> <li>■ INSERT to store missing owner</li> <li>■ UPDATE to change invalid foreign key</li> <li>■ DELETE to remove invalid referencing rows</li> </ul>

**Input data used in the build phase:** You can enter the BUILD phase of the load process using data stored in intermediate work files created by the LOAD phase or by instructing CA-IDMS/DB to extract the necessary information as the first step in the build process. Intermediate work files are generally used when you intend to enter the BUILD phase immediately following the LOAD phase; typically, you instruct CA-IDMS/DB to extract the information if some time elapses between the two phases.

The table below summarizes how to specify these options:

---

<b>BUILD phase input</b>	<b>Load and build option</b>	<b>LOAD statement</b>	<b>BUILD statement</b>
Intermediate work file	Phased load and build	LOAD NO VALIDATE	None
	Stepped load	LOAD STEP1 EXTRACT	Start with BUILD STEP2
Extracted work file	Phased load	LOAD NO BUILD	BUILD
	Stepped load	LOAD STEP1 NOEXTRACT	Start with BUILD STEP1

---

**Enhancing load performance:** The list below identifies some ways to enhance the performance of your load operations:

- If possible, load several tables at the same time
- Validate several tables at once
- Always load using sorted data; either letting CA-IDMS/DB sort the data or by supplying pre-sorted data
- Increase the number of pages in the buffer(s)

## 21.3 Contents of the input file

**Mixed input records:** The input file to the load process can contain different types of input records. For example, the input file might contain an EMPLOYEE record, followed by a DEPARTMENT record, followed by an OFFICE record and so on; to distinguish the different types of records, you must include a record identifier (in this example, at the end of the record):

```
0574SMITH   JOHN   254 WILLOW ST   NEEDHAM MA   4035 415
4001PERSONNEL      MASON   PAULA   5538   0020 410
0020CHICAGO  3 CORPORATE PLACE                450
```

**Tip:** By including record identifiers at the end of the input records, you may be able to avoid listing individual column definitions in the LOAD statement.

**Loading multiple tables:** You can load more than one table in the same load operation by using one of the following techniques:

- By specifying selection criteria applied against records in the input file. For example, to load the EMPLOYEE table, using the example above, you could select all input records with value '415' as a record identifier.
- By selecting specific fields from one input record that contains data pertinent to multiple tables. For example, the input record may contain values to be stored in table EMPLOYEE and values to be stored in table DEPARTMENT.

**Identifying columns implicitly:** If, in the LOAD statement, you do not explicitly list the columns in the table to be loaded, CA-IDMS/DB assumes that values are supplied for all columns in the table. It starts with position 1 of the input record and extracts input values for each column of the table. To be successful, the input data must match the order, data type, length, and null criteria specified in the table definition. Columns that allow null values must be represented by a data field and an indicator field, which is described below under "Null values".

**Identifying columns explicitly:** If you supply values for only some of the columns within the table or if the order or data types of the values in the input file do not match that of the columns in the table, you must tell CA-IDMS/DB:

- Where to find the column values in the input record by specifying their start position relative to the beginning of the input record
- The data type of the input record value
- The null value criteria for input values, if applicable

If you omit a column name, the column must either:

- Allow null values
- Allow a default value

**Data types:** If you explicitly list the columns to be loaded, the data type of the value to be stored can be different from the data type defined for the column provided the data types are compatible. For example, a column defined as CHARACTER is compatible with data types VARCHAR, DATE, TIME, and TIMESTAMP.

►► For more information about compatible data types, refer to the *CA-IDMS SQL Reference*.

**Null values:** Null values in an input file can be represented as either:

- A specific data value, designated by you in the NULL IF clause of the LOAD statement.
- An indicator field, immediately following the data field. This field is either a 1, 2, or 4 byte binary field and must contain a value of:
  - 0, to indicate a non-null data value
  - -1, to indicate a null value

If you do not explicitly list the columns to be loaded, then all columns that permit null values must be followed by a **4-byte** indicator field.

**Sequence for loading tables:** If you are loading multiple tables, you may have to use separate load operations. If so, use these rules to load the tables in the correct sequence:

- Tables clustered through a linked or unlinked constraint cannot be loaded until the referenced table is loaded and, if necessary, the index on the primary key of the referenced table is built.
- Linked index constraints cannot be *built* until the referenced table is loaded and, if necessary, its primary key index is built.

## 21.4 Loading procedures

**What follows:** The remainder of this chapter provides procedures and examples for:

- Steps that apply to all load procedures
- A full load procedure
- A phased load procedure
- A segmented load procedure
- A stepped load procedure

**Note:** Only one LOAD, BUILD, or VALIDATE statement may be performed during one execution of the batch command facility; for example, you cannot submit two LOAD statements at one time.

### 21.4.1 Steps that apply to all load procedures

**Steps before the load:** Regardless of what load procedure you use, perform the following actions *before* loading the data:

Action	Statement
Define the tables to be loaded	CREATE TABLE
Create the input file or files of data to be loaded using CA-CULPRIT, CA-OLQ (batch), or a user-written program	
Vary the areas in which the tables reside offline to DC/UCF	DCMT VARY AREA with the OFFLINE option
Back up the areas, if they aren't empty	BACKUP or a comparable backup utility

**Steps after the load:** *After* loading the data, perform these steps:

Action	Statement
Optionally, verify the result by retrieving data from the loaded tables	SELECT statements
Back up the areas in which the tables reside	BACKUP or a comparable backup utility
Vary the areas online	DCMT VARY AREA with the ONLINE option

## 21.4.2 Full load procedure

**Steps:** Follow these steps to perform a full load of an SQL-defined database:

Action	Statement
In local mode, load, build, and validate one or more database tables	LOAD

**Example:** This example loads, builds, and validates tables ASSIGNMENT, CONSULTANT, EXPERTISE, SKILL, and PROJECT. Each of these tables is independent of those in other areas of the EMPLOYEE database. By default, CA-IDMS/DB sorts the input data before it loads the tables. Also by default, if it finds any errors during any phase of the load procedure, it stops.

To load each table, CA-IDMS/DB applies selection criteria against each input record it reads. For example, the ASSIGNMENT table receives all input records where the value in position 210 of the input record equals '512'. Similarly, the EXPERTISE table receives all input records where the value in position 210 equals '320'.

```
load
  into demoproj.assignment
    where position 210 = '512'
    (emp_id position 1 smallint,
     proj_id position 3 char(4),
     start_date position 13 date,
     end_date position 23 char(8) null if '01-01-01')

  into demoproj.consultant
    where position 210 = '222'

  into demoproj.expertise
    where position 210 = '320'
    (emp_id position 1 smallint,
     skill_id position 3 smallint,
     skill_level position 5 char(2) null if '99',
     exp_date position 7 date)

  into demoproj.project
    where position 210 = '416'

  into demoproj.skill
    where position 210 = '445';
```

## 21.4.3 Phased load procedure

**Steps:** Follow the steps below to perform a phased load:

**Tip:** Optionally, back up the database areas between the load and build steps if you want to recover the data in the event of a failed job step.

Action	Statement
Identify the following tables: <ul style="list-style-type: none"> <li>■ All tables clustered through referential constraints; if multiple levels of clustering exist, the tables in each level must be loaded in a separate operation before those at a lower level</li> <li>■ All referencing tables in linked index constraints where the primary key is an index; if multiple levels of such a structure exist, the tables in the higher levels must be loaded before those at a lower level</li> </ul>	
In local mode, load and build all tables not identified in Step 1 above.	LOAD with the NO VALIDATE option
<ul style="list-style-type: none"> <li>■ For each clustering level, load and build all tables clustered through referential constraints</li> <li>■ For each linked index level, load and build all tables that participate in linked index constraints</li> </ul>	LOAD with the NO VALIDATE option
Validate the referential constraints of all the loaded tables	VALIDATE SEGMENT

**Example:** In this example, the tables BENEFITS, COVERAGE, EMPLOYEE, and POSITION are loaded in a phased load procedure. The tables have the following characteristics:

Table	Characteristics
BENEFITS	References EMPLOYEE in a linked, clustered constraint
COVERAGE	References EMPLOYEE in a linked, clustered constraint
EMPLOYEE	References DEPARTMENT in an unlinked constraint
POSITION	References EMPLOYEE in a linked, clustered constraint

To load the tables, load and build the EMPLOYEE table first, followed by the remaining tables. After all 4 tables are loaded, validate the referential constraints that exist between them:

```

load no validate
  into demoempl.employee
  where position 150 = '415';

load no validate
  into demoempl.benefits
  where position 150 = '478'

  into demoempl.coverage
  where position 150 = '488'

  into demoempl.position
  where position 150 = '492';

validate segment demoempl;

```

### 21.4.4 Segmented load procedure

**Steps:** Follow the steps below to perform a segmented load:

Action	Statement
Load the input records in groups; for example, the first 1,000,000, the second 1,000,000 and so on	LOAD NO BUILD using the FROM and FOR clauses for each group of input records
Build the table indexes	BUILD INDEXES NO VALIDATE
Build the indexed constraints	BUILD CONSTRAINTS NO VALIDATE
Validate the referential constraints of the tables within the segment	VALIDATE

**Example:** This example uses a segmented load to load table EMPLOYEE, which contains more than 2,000,000 rows. By default, each input record is to be stored in the EMPLOYEE table, with each field in the input record corresponding in length and data type to each column defined for the EMPLOYEE table.

The first LOAD statement loads 1,000,000 rows in the table, starting with the first input record. CA-IDMS/DB will notify the user for each 100,000 input records processed. The second LOAD statement processes the next 999,999 input records beginning with input record 1,000,001. The third LOAD statement processes the remaining input records.

Because the table is so large, indexes and indexed constraints are built in separate steps using the BUILD statements. Finally, the referential constraints for the table are validated.

```

load no build
  into demoempl.employee
  for 1000000
  notify 100000;

load no build
  into demoempl.employee
  from 1000001
  for 999999
  notify 100000;

load no build
  into demoempl.employee
  from 2000000
  notify 100000;

build indexes
  no validate
  for demoempl.employee
  notify 100000;

build constraints
  no validate
  for demoempl.employee
  notify 100000;

validate table demoempl.employee
  notify 100000;

```

### 21.4.5 Stepped load procedure

**Steps:** Follow the steps below to perform a stepped load:

**Tip:** If you want to be able to recover the database in the event of a failed job step, back up the database areas between each job step.

Action	Statement
1. In local mode, load one or more database tables choosing one of the options below. If you intend to build indexes and/or relationships for the tables immediately following the load step, choose one of the options that creates an intermediate work file:	
1.1 Load, creating intermediate extract files for the build phase	LOAD STEP1 EXTRACT BOTH (the default)
1.2 Load, creating an intermediate extract file for building indexes	LOAD STEP1 EXTRACT INDEXES
1.3 Load, creating an intermediate extract file for building relationships	LOAD STEP1 EXTRACT RELATIONSHIPS

<b>Action</b>	<b>Statement</b>
1.4 Load, creating no intermediate extract file	LOAD STEP1 NO EXTRACT
2. If you specified WITHOUT PRESORT, skip this step. Otherwise, sort the data using an external sort program and the sort cards supplied by CA-IDMS/DB. Then continue the load phase of the stepped load procedure.	LOAD STEP2
3. If you specified LOAD STEP1 NO EXTRACT, perform this step to collect the data necessary to build the table indexes and indexed constraints	BUILD STEP1
4. Sort the data using an external sort program and the sort cards supplied by CA-IDMS/DB	
5. After all of the tables have been loaded or after completing the previous step, determine the db-keys of rows in any tables that participate as the referenced table in a linked index referential constraint	BUILD STEP2
6. Sort the data using an external sort program and the sort cards supplied by CA-IDMS/DB	
7. Build unclustered indexes and linked index referential constraints	BUILD STEP3
8. Sort the database using an external sort program and the sort cards supplied by CA-IDMS/DB	
9. Update the prefixes of any tables that participate as the referencing table in a linked index referential constraint	BUILD STEP4
10. Perform the first pass at validating the relationships between tables that participate in referential constraints	VALIDATE STEP1
11. Sort the database using an external sort program and the sort cards supplied by CA-IDMS/DB	

Action	Statement
12. Perform the second pass of validating referential constraints; generally, a second pass is required for unlinked relationships if either the referenced table or referencing table contains a CALC key.	VALIDATE STEP2

**Example:** In the example below, the DBA loads an SQL-defined database in the following steps:

- Loads tables CUSTOMER and INVENTORY using pre-sorted data
- Loads tables ORDERS and PARTS using pre-sorted data
- Using an area sweep, extracts information for building indexes and indexed constraints
- Builds the indexes and constraints for the table using separate steps and external sorts
- Validates referential constraints

```
load step1
  without presort
  no extract
  into custschm.customer
    where position 300 = '435'
  into custschm.inventory
    where position 300 = '457';
```

```
load step1
  without presort
  no extract
  into custschm.orders
    where position 200 = '335'
  into custschm.parts
    where position 200 = '345';
```

```
build step1
  for custschm.customer
  custschm.inventory
  custschm.orders
  custschm.parts;
```

**Sort the data:**

```
build step2;
```

**Sort the data:**

```
build step3;
```

**Sort the data:**

```
build step4;  
validate step1;  
  
Sort the data:  
validate step2;
```

## 21.5 Related information

- About the BACKUP, LOAD, BUILD, VALIDATE utility statements and the IDMSCALC utility program, see *CA-IDMS Utilities*
- About SQL data types, refer to the *CA-IDMS SQL Reference*
- About loading a non-SQL defined database, see Chapter 20, “Loading a Non-SQL Defined Database” on page 20-1
- About DCMT commands, refer to *CA-IDMS System Tasks and Operator Commands*
- About CA-CULPRIT and CA-OLQ, see the document set for each product

# Chapter 22. Monitoring and Tuning Database Performance

---

22.1	Monitoring guidelines	22-3
22.2	Monitoring facilities	22-4
22.3	Items to monitor and tune	22-5
22.3.1	Journal use	22-5
22.3.2	Buffer utilization	22-6
22.3.3	Space management and database design	22-7
22.3.4	Indexing efficiency	22-8
22.3.5	Database locks	22-9
22.3.6	Longterm locks	22-13
22.3.7	SQL processing	22-14
22.4	Reducing I/O	22-15
22.4.1	Through database reorganization	22-15
22.4.2	Through application design	22-16
22.4.3	Through database design	22-16
22.4.4	By using UPDATE STATISTICS (SQL-accessed databases)	22-16



## 22.1 Monitoring guidelines

**Why you need to monitor:** Eventually, a database may begin to outgrow its initial allocation of space, with resulting increased I/O and poor response time. If you don't monitor your databases on a regular basis, these conditions can become critical, forcing you to take emergency actions at an inconvenient time.

**Suggested monitoring schedule:** Consider using the schedule below as the basis for monitoring database performance:

Monitoring tool	Monitoring frequency	Information provided
JREPORT 004	Daily	Summary information on the database processing activities for each program recorded in the journal file
IDMSDBAN report 2	Weekly	Area detail statistics, such as number of logically full pages and number of relocated records
IDMSDBAN report 5	Monthly	Set detail statistics, such as the number of pages needed to store a chained set
PRINT SPACE	Daily	Area space utilization statistics
IDMSDBAN (all reports)	Monthly or as needed	Set statistics, including broken chains, record data, and area data

**Keep a history of meaningful statistics:** Keep a history of meaningful statistics so that you can identify abnormal conditions when they arise.

**SQL considerations:** Most of the information in this chapter applies to both SQL and non-SQL defined databases. Text that applies to only one or the other will be noted. In addition, much of the chapter applies to the physical structures that underlie the database definition. Therefore, one set of terms will be used for these physical entities. For example, chain sets are the physical structure used to implement both SQL linked constraints and non-SQL sets defined with the MODE IS CHAIN clause.

## 22.2 Monitoring facilities

**Online and batch components:** CA-IDMS/DB offers the following online and batch tools for you to use to monitor the performance of your databases:

Facility	Uses
CA-IDMS Performance Monitor	To monitor: <ul style="list-style-type: none"> <li>■ Real-time database and system statistics</li> <li>■ System-wide, wait-time statistics for a unit of time</li> <li>■ Statistics about resource usage by individual programs</li> </ul>
DCMT commands	To display definitions and run-time statistics for entities associated with a DC/UCF system
IDMSDBAN utility program	To check for broken chains and to display statistics and data for sets, records, and areas
OPER WATCH commands	To display dynamic system run-time statistics associated with DC/UCF systems
PRINT INDEX utility statement	To monitor the structure of user-owned and system-owned indexes
PRINT SPACE utility statement	To monitor space utilization in segments or areas
PRINT JOURNAL utility statement	To display checkpoint information about transactions recorded on an archive or tape journal file
PRINT utility statement	To display the contents of requested database pages
JREPORTs	To monitor journal and database usage statistics
SREPORTs	To monitor system and database usage statistics
Online print log (PLOG)	To display system messages, system trace information, and snap dumps from the DDLDCLOG area
UPDATE STATISTICS utility statement	To refresh statistical information about SQL defined databases, and non-SQL defined databases that are accessed by SQL commands.

►► For more information about CA-IDMS Performance Monitor, refer to *CA-IDMS Performance Monitor User Guide* For more information about utility programs and statements, refer to *CA-IDMS Utilities* For more information about DCMT and OPER commands, refer to *CA-IDMS System Tasks and Operator Commands* For more information about JREPORTs and SREPORTs, refer to *CA-IDMS Reports*

## 22.3 Items to monitor and tune

**Monitoring the database:** For your database, the major areas of degradation are:

- Pages over 70% full
- CALC and VIA (clustered) record overflow
- Fragmented records
- Inefficient index structures
- An increase in logically-deleted or relocated records

**Monitoring transactions:** For each transaction, the major performance indicators are:

- The number of I/Os and/or the number of calls to CA-IDMS/DB
- The number of waits and deadlocks

**What follows:** The remainder of this section identifies useful statistics to monitor in the following areas:

- Journal use
- Buffer utilization
- Space management and database design
- Indexing efficiency
- Database locks
- Longterm locks
- Access modules

### 22.3.1 Journal use

#### Useful statistics to monitor

Statistic	Meaning	Action
Journal read waits	Indicates CA-IDMS/DB must wait to read a page from a journal file into the journal buffer during a rollback operation.	Increase the number of pages in the journal buffer

Statistic	Meaning	Action
Journal page utilization	Indicates the fullness of journal pages written from the journal buffer.	Create fuller journal buffers by: <ul style="list-style-type: none"> <li>▪ Adjusting the journal buffer page size in the definition of the journal buffer</li> <li>▪ Increasing the journal TRANSACTION LEVEL option at system generation or using a DCMT VARY JOURNAL command</li> </ul>

#### Where the statistics are reported

- ARCHIVE JOURNAL utility statement report
- JREPORT 004
- CA-IDMS Performance Monitor
- DCMT DISPLAY JOURNAL

## 22.3.2 Buffer utilization

#### Useful statistics to monitor

Statistic	Meaning	Possible action
Buffer utilization ratio	Indicates the ratio of the number of pages requested to the number read; values less than 2 indicate a problem with the buffer size or with the design of the database	<ul style="list-style-type: none"> <li>▪ Increase the number of buffer pages</li> <li>▪ Reassign files to buffers</li> </ul>
Forced writes	Indicates the number of times CA-IDMS/DB had to write a buffer page to storage in order to read a database page	<ul style="list-style-type: none"> <li>▪ Increase the number of buffer pages</li> <li>▪ Reassign files to buffers</li> <li>▪ Issue COMMITs more frequently in update jobs</li> </ul>
Buffer waits	Indicates the number of times the buffer was requested but was not available	<ul style="list-style-type: none"> <li>▪ Increase the number of buffer pages</li> <li>▪ Reassign files to buffers</li> </ul>

#### Where the statistics are reported

- CA-IDMS Performance Monitor

- SREPORT 003
- DCMT DISPLAY/VARY BUFFER

### 22.3.3 Space management and database design

#### Useful statistics to monitor

Statistic	Meaning	Possible action
Clustering ratio	Indicates the ratio of the number of records requested to the number of pages requested; ratios less than 4 indicate poor database design or space availability problems	<ul style="list-style-type: none"> <li>▪ Redesign the database using clustering more effectively</li> <li>▪ Increase the area's page size or page range and unload and reload the database</li> <li>▪ Reassign files to buffers</li> </ul>
Page space availability	Indicates how full database pages are	<ul style="list-style-type: none"> <li>▪ Increase the database page size</li> <li>▪ Increase the number of pages</li> </ul>
Fragments stored	Indicates the number of fragments stored for a variable-length record.	<ul style="list-style-type: none"> <li>▪ Increase the page size and read each record in an update mode</li> <li>▪ Increase the page reserve size</li> <li>▪ Reassign fragmentation specifications</li> </ul>
Records relocated	Indicates the number of expanded records moved to a new page due to lack of space	<ul style="list-style-type: none"> <li>▪ Unload/reload the database</li> <li>▪ Increase the page size and read each record in an update mode</li> </ul>
CALC cluster ratio	Indicates the ratio of CALC records stored on the target page to the total number (that is, hits plus overflow) stored; values less than 1 indicate space availability problems	Increase the area's page size or number of pages and unload and reload the database

<b>Statistic</b>	<b>Meaning</b>	<b>Possible action</b>
VIA cluster ratio	Indicates the ratio of VIA (or clustered) records stored on the target page to the total number (that is, hits plus overflow) stored; values less than 1 indicate large clusters, space availability problems, or small page size	Increase the area's page size or number of pages and unload and reload the database
Effectiveness ratio	Indicates the ratio of number of records CA-IDMS/DB requests to the number that are current-of-run-unit. Values much higher than 1 indicate poor program logic or set options	Review application/database design. Consider use of PRIOR or OWNER pointers and possible elimination of some sorted sets. (Note that linked constraints in SQL-defined databases always include PRIOR and OWNER pointers.)
Logically deleted records	Indicates the number of logically deleted records	Physically delete the logically deleted records using the CLEANUP utility statement

#### Where statistics are reported

- JREPORT 004
- SREPORTs 003, 007, and 009
- CA-IDMS Performance Monitor
- IDMSDBAN utility report 5
- PRINT SPACE utility statement report
- PRINT JOURNAL utility statement
- UPDATE STATISTICS utility statement report for the SQL catalog

### 22.3.4 Indexing efficiency

#### Useful statistics to monitor

<b>Statistic</b>	<b>Meaning</b>	<b>Possible action</b>
Orphan count	Indicates the number of orphaned SR8 records.	Rebuild the index if more than 25% of the member records are orphaned.
Index levels	Indicates the number of levels in the index.	Rebuild the index if the number of levels exceeds the number originally calculated

---

<b>Statistic</b>	<b>Meaning</b>	<b>Possible action</b>
SR8 split	Indicates the number of SR8 splits.	If the number of SR8 splits is high, determine if applications frequently insert a large group of index entries in one spot; rebuild the index to balance it and cleanup orphan index records.

---

#### Where the statistics are reported

- CA-IDMS Performance Monitor (Realtime monitor) Run Unit Detail screen
- PRINT INDEX utility statement
- IDMSDBAN utility report 5

## 22.3.5 Database locks

#### Useful statistics to monitor

---

<b>Statistic</b>	<b>Meaning</b>	<b>Possible action</b>
Number of non-share locks held	Indicates the number of non-share locks (primarily update locks) held. The larger the number of update locks held, the greater the probability of contention between the tasks holding the locks and other tasks accessing the same database.	<ul style="list-style-type: none"> <li>▪ Issue COMMITs more frequently in update jobs</li> <li>▪ If overall throughput is constrained, identify the source; for example, CPU or DASD usage</li> <li>▪ If overall throughput is not constrained, identify potential deficiencies in database or application design or implementation; for example, look at the number of locks held by individual programs; determine if tasks contend for OOK and FOAK records in which case lowering the DEADLOCK DETECTION INTERVAL will temporarily solve the problem</li> </ul>

---

<b>Statistic</b>	<b>Meaning</b>	<b>Possible action</b>
Task wait status	Indicates whether a task is waiting for access to an area or record	Tasks that are waiting on locks have an ECB type of 'LMGR Lock'. If you notice a task waiting a long time on one or more locks, review ready modes and database design, especially for contention for OOK and FOAK records, by examining all tasks exhibiting this behavior for common programs, functions, and database references.
ECB type	Denotes the type of resource being waited on. In the case of area locks and dbkey locks, this statistic will contain the literal 'LMGR to ECB'. (Note: in the Performance Monitor this information is listed under the column headings 'First ECB', 'Second ECB', and 'Third ECB').	
Number of shared locks held	Indicates the number of share locks held. Share locks allow transactions other than the owning transaction to read the row, but not to update it. Thus, higher levels of share locks can impede concurrency (and throughput) if they are placed on rows in areas that are heavily accessed.	The number of locks held can be reduced by increasing the COMMIT frequency within the application.
ISO (SQL only)	Indicates the isolation level of the transaction. The isolation level of a transaction defines how long retrieval locks are held.	Ensure that the transaction is running in the appropriate isolation level for the level of data integrity required by the application.

<b>Statistic</b>	<b>Meaning</b>	<b>Possible action</b>
State (SQL only)	<p>Indicates the state of the transaction which defines how the transaction is affecting the data it is processing:</p> <ul style="list-style-type: none"> <li>■ Read only (RO) specifies that the transaction is reading data but not adding or updating.</li> <li>■ Read write (RW) specifies that the transaction intends to add and update data.</li> </ul>	<p>Ensure that the transaction state is appropriate for the type of processing being performed. Transactions that only read data should have a state of RO.</p>
Ratio of global resource lock requests to local lock requests	<p>Indicates the number of times that CA-IDMS had to acquire or alter a global lock on an area, page, or record in order to service the indicated number of local lock requests. The larger this ratio, the greater the inter-member contention for resources, since CA-IDMS acquires global record and page locks only if contention exists and global area locks, once acquired, will usually be retained in an active system.</p>	<ul style="list-style-type: none"> <li>■ Issue COMMITs more frequently in update transactions</li> <li>■ Disperse frequently updated data across more pages within the area</li> <li>■ Increase the size of the area, especially if frequently inserting or deleting data in an area that is more than 70 percent full</li> </ul>
Ratio of the number of global lock waits to the number of global lock requests.	<p>Indicates the number of times that CA-IDMS had to wait for a global lock request to complete. This ratio is a measure of one or more of the following types of contention:</p> <ul style="list-style-type: none"> <li>■ Inter-member contention for transaction resources</li> <li>■ False contention caused by synonyms when hashing to the global lock table</li> <li>■ Contention for operating system resources such as channels</li> </ul>	<ul style="list-style-type: none"> <li>■ Use operating system tools to determine the nature of the contention</li> <li>■ Take the actions outlined above to reduce inter-member contention for transaction resources</li> <li>■ Increase the number of lock table entries to reduce false contention</li> </ul>

<b>Statistic</b>	<b>Meaning</b>	<b>Possible action</b>
Number of times lock storage overflowed	Indicates the number of times that CA-IDMS had to acquire lock storage dynamically in order to satisfy a lock request. The larger this number the more CPU cycles that were expended to satisfy lock requests. Additionally the storage pool may become fragmented since dynamically acquired storage may not always be releasable.	<ul style="list-style-type: none"> <li>■ Examine the overflow details to determine the type of storage overflows that occurred</li> <li>■ Determine the applicable base factor for the type of overflowing storage: <ul style="list-style-type: none"> <li>– Session and class storage is based on the number of logical terminal elements (LTERMs) defined for the system.</li> <li>– Resource and proxy storage is based on the SYSLOCKS system definition parameter</li> <li>– XES Request storage is based on the maximum number of tasks specified in the system definition.</li> </ul> </li> <li>■ Increase the appropriate base factor (the number of LTERMs, SYSLOCKS, or maximum number of tasks) to increase the size of the initial storage allocation, and thus reduce the number of overflows.</li> </ul>

#### Where the statistics are reported

- For area contention:
  - SREPORTs
  - JREPORT 006
  - CA-IDMS Performance Monitor (Realtime monitor): Active User Task Detail, Active System Task Detail screen, Transaction Detail screen, and SQL Detail screen
  - DCMT DISPLAY ACTIVE TASKS
  - Area status codes from DCMT DISPLAY TRANSACTION transaction id

- Area status codes from OPER WATCH DB
- OPER WATCH TIME
- For record contention:
  - Status codes from OPER WATCH DB
  - Status codes from DCMT DISPLAY TRANSACTION transaction id
  - DCMT DISPLAY LOCK (shows longterm and notify locks held by logical terminals)
- For lock storage overflows:
  - DCMT DISPLAY LOCK STATS
- For inter-member contention in a data sharing environment:
  - DCMT DISPLAY LOCK STATS
  - DCMT DISPLAY DATA SHARING XES LOCKS

#### Reducing area contention

- Ready online program areas in shared ready modes
- Create a window for batch jobs

#### Reducing record contention

- Have the application issue more COMMITs
- Run applications that contend for a record serially, rather than concurrently
- Have some applications use a different access route that avoids the record under contention
- Change the database design so that access can be less serialized

### 22.3.6 Longterm locks

#### Useful statistics to monitor

Statistic	Meaning	Possible action
Tasks having areas locked	Shows which tasks have areas locked	Use this information to identify tasks that ready an area in protected or exclusive mode, which increase the potential for throughput degradation
Longterm/notify locks	Displays longterm or notify lock statistics by logical terminal	Use this information to identify tasks that hold a large number of longterm and/or notify locks

**Where the statistics are reported**

- DCMT DISPLAY AREA
- DCMT DISPLAY LOCK AREA/LTERM

**22.3.7 SQL processing****Useful statistics to monitor**

<b>Statistic</b>	<b>Meaning</b>	<b>Possible action</b>
Sorts performed	The number of sorts performed as the result of an SQL statement (the result of processing the ORDER BY clause)	Add additional indexes or sorted constraints to reduce the number of sorts
Maximum rows sorted	The largest number of rows sorted as the result of an ORDER BY clause	Add additional indexes to eliminate the sort
AM recompiles	The number of times access modules were automatically recompiled at runtime because of a recompilation of the corresponding program or dialog, or because of a change in the underlying database definition.	Examine the cause of compilations. If necessary, move frequently altered tables to areas with table level stamp synchronization.

**Where the statistics are reported**

- SREPORTs
- CA-IDMS Performance Monitor
- IDMSDBAN utility reports (database structure)

---

## 22.4 Reducing I/O

I/O can be reduced through:

- Database reorganization
- Application design
- Database design
- The UPDATE STATISTICS utility command (for SQL-accessed databases)

Each of these is discussed below.

### 22.4.1 Through database reorganization

Database reorganization includes:

- Reducing full pages by changing the size of a database page or increasing the number of pages
- Reducing overflow by changing the size of a database page or increasing the number of pages
- Decreasing fragmentation for non-SQL defined databases by:
  - Specifying page reserve
  - Changing page size
  - Reassigning records
  - Redefining fragmentation specifications
  - Increasing the number of pages
- Increasing the efficiency of an index's structure by decreasing the number of levels in the index and/or assigning SR8 records to a separate page range
- Reducing logically deleted and/or relocated records by physically deleting logically deleted occurrences using the CLEANUP utility statement and/or unloading and reloading the data
- Reducing the number of fragments and/or relocated records by increasing the page size and reading all records in an update mode

#### For more information

- About changing page size, see Chapter 25, “Modifying Physical Database Definitions” on page 25-1
- About modifying indexes, see Chapter 29, “Modifying Indexes, CALC Keys, and Referential Constraints” on page 29-1 and Chapter 31, “Modifying Schema Entities” on page 31-1
- About reassigning records and redefining fragmentation specifications, see Chapter 31, “Modifying Schema Entities” on page 31-1

- About utility statements, refer to *CA-IDMS Utilities*

## 22.4.2 Through application design

**Selecting the optimal path:** The first step to determine if the application is optimally designed is to determine if it is accessing the data it needs, using the access path that will create the fewest number of I/Os. To determine if this is true:

1. Walk through the application and identify the actual transaction path
2. Review the existing database design and determine if there is a more efficient way to:
  - Access the needed records
  - Process the necessary relationships

## 22.4.3 Through database design

Take into account the following database design considerations for reducing I/O:

- Adding sets, indexes, pointers, redundancy
- Changing set type, set (index) order for non-SQL defined databases
- Changing location (area) of record or index, index and/or set stored VIA (or clustered)
- Changing UNLINKED constraints to LINKED (SQL-defined databases) to repeating item, index and/or set stored VIA, location of record or index
- Splitting a record

## 22.4.4 By using UPDATE STATISTICS (SQL-accessed databases)

**When to use UPDATE STATISTICS:** Execute the UPDATE STATISTICS utility statement at the following times:

- Periodically (according to the needs of the application) to reflect shifts in the distribution of data in the database (for example, changes in owner/member ratios, area space utilization, index layout)
- After individual applications that alter the distribution of data; for example, monthly or year-end summary and offload processing

**Use UPDATE STATISTICS on SQL-defined tables or areas:** Run UPDATE STATISTICS on individual tables or whole areas. The resulting statistics are stored in the SQL catalog and are used by the Access Module Compiler to generate optimal access strategies for SQL processing. Access modules that reference the tables whose statistics have been updated can then be recompiled to take advantage of the updated information. Table/access module cross-reference information on the catalog can be used to determine which access modules to recompile.

**Use UPDATE STATISTICS on NON-SQL Schemas if they are accessed by**

**SQL:** Run UPDATE STATISTICS on some or all areas defined in a non-SQL Schema. The resulting statistics are kept in the non-SQL dictionary that defines the schema. If the database is accessed by SQL the statistics will be used by the Access Module Compiler to generate optimal access strategies for SQL processing.

**Restrictions on statistics and non-SQL schemas:** Non-SQL statistics are kept with the schema definition in the dictionary. This means statistics may be kept for only one physical database per schema. When processing an SQL command, only the current set of statistics will be used for that command regardless of the physical database being accessed by that command. The user must decide which physical database will provide the statistics that best meets their needs and run UPDATE STATISTICS against that database.



# Chapter 23. Dictionaries and Runtime Environments

---

23.1 About dictionaries . . . . .	23-3
23.1.1 Physical components of a dictionary . . . . .	23-3
23.1.2 Logical components of a dictionary . . . . .	23-4
23.1.3 Assigning dictionary areas to segments . . . . .	23-5
23.1.4 Sharing dictionary areas . . . . .	23-6
23.2 CA-supplied dictionary definitions . . . . .	23-8
23.2.1 Logical database definitions . . . . .	23-9
23.2.2 Protocols, nondatabase structures, and modules . . . . .	23-11
23.3 Defining new dictionaries . . . . .	23-13
23.3.1 Defining new catalog components . . . . .	23-13
23.3.2 Defining new application dictionaries . . . . .	23-14
23.3.3 Defining new system dictionaries . . . . .	23-16
23.4 Establishing a default dictionary . . . . .	23-19
23.5 About runtime environments . . . . .	23-20
23.5.1 SYSIDMS parameter file . . . . .	23-22
23.5.2 Establishing session options . . . . .	23-23
23.6 Related information . . . . .	23-25



## 23.1 About dictionaries

**What is a dictionary?:** A dictionary is a special CA-IDMS/DB database that contains definitions of other databases, DC/UCF systems, and applications. Information in the dictionary is organized into **entity types** that correspond to major data processing components (for example, tables, records, programs). The dictionary becomes populated with information about the data processing environment as various CA-IDMS/DB software components are executed.

**System and application dictionaries:** Each DC/UCF system must contain a system dictionary. Any number of application dictionaries can also exist in a CA-IDMS/DB runtime environment. The table below describes both types of dictionaries:

Dictionary	Description
System	<p>Includes all information required to establish, maintain, and control the processing environment:</p> <ul style="list-style-type: none"> <li>▪ The DC/UCF system definition</li> <li>▪ The physical database definitions</li> </ul> <p>Each runtime environment must have a system dictionary named SYSTEM.</p>
Application	<p>Optional dictionaries that contain information specific to a particular application, group of applications, or development groups:</p> <ul style="list-style-type: none"> <li>▪ The logical database definitions</li> <li>▪ Maps, dialogs, records, programs, elements</li> </ul> <p>A runtime environment may contain zero or more application dictionaries the names of which are user-defined.</p>

### 23.1.1 Physical components of a dictionary

**Dictionary areas:** Dictionaries are composed of the areas listed below:

Area name	Description
DDLDMML	Contains the following types of information: DC/UCF system definitions Non-SQL schema and subschema definitions Maps Dialogs Source modules Record and element descriptions IDD users Classes and attributes
DDLDCLOD	Contains load modules associated with entities contained in the DDLDMML area; for example: Map load modules Dialog load modules Subschema load modules
DDLCAT	Contains definitions of physical databases (segments, DMCLs, database name tables); at sites with the SQL option, contains definitions of SQL entities (tables, constraints, indexes, and so on)
DDLCATX	Contains indexes defined on entities stored in the DDLCAT area
DDLCATLOD	Contains: DMCL load modules Database name table load modules Access modules at sites with the SQL option
DDLDCMSG	Contains system and user-defined messages

### 23.1.2 Logical components of a dictionary

**Dictionary components:** You can group the six areas of the dictionary into logical components based on the inherent relationships that exist between the dictionary areas:

Logical component	Dictionary areas
Base definition component	DDLDMML DDLDCLOD
Message component	DDLDCMSG

---

Logical component	Dictionary areas
Catalog component	DDLCAT DDLCATX DDLCATL0D

---

**Components of a system dictionary:** A system dictionary always contains all 3 components:

- A base definition component
- A catalog component
- A message component

**Components of an application dictionary:** An application dictionary may contain all or a subset of the components. At sites without the SQL option, an application dictionary usually contains only a base definition component and a shared message component.

**Sharing the message area:** In most cases, an application dictionary will not have its own message area. Since the runtime system uses only the system message area (SYSMSG.DDLDCMSG) to display messages, most application dictionaries will share the system message area, rather than having a separate message area.

### 23.1.3 Assigning dictionary areas to segments

**Segment by component:** The six areas that make up a dictionary should be segmented by logical component. That is, a segment should be defined for each of the base definition, catalog, and message components of a dictionary.

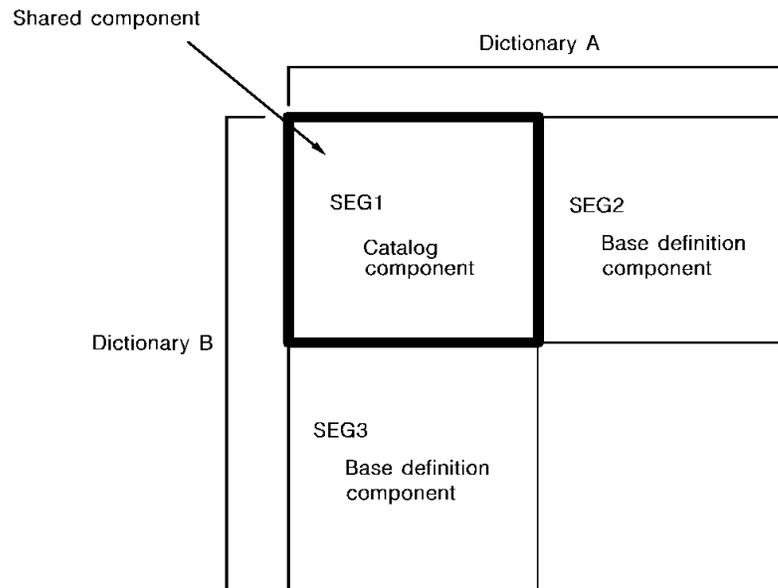
In most cases, a dictionary will not have its own message component, but will share the system message area SYSMSG.DDLDCMSG. Sites without the SQL option do not need to define a catalog segment for their application dictionary.

**Define a database name:** If a dictionary is made up of more than one segment, you must define a database name to represent the dictionary. The database name identifies all of the segments that together make up the dictionary.

The one exception to this is a dictionary comprised of only two segments, one of which is the SYSMSG segment. A database name is unnecessary because CA-IDMS/DB automatically uses the system message area (in the SYSMSG segment) if no message area is associated with the dictionary.

### 23.1.4 Sharing dictionary areas

**Sharing components:** By separating dictionary components into segments, you can share those components between dictionaries, as illustrated below:



To share SEG1 between dictionary A and dictionary B, define a database name for each that includes the SEG1 segment.

**System dictionary components:** You should not share the base definition component and the catalog component of the system dictionary with application dictionaries. Since the system dictionary contains critical information needed to control and execute your CA-IDMS/DB environment, it should be accessed only by authorized personnel and should be reserved for the following information:

- DC/UCF system definitions
- Physical database definitions

**Sharing individual areas:** It is possible to separate a component into multiple segments so that individual areas (such as a load area) can be shared across dictionaries. While this is supported, it is not recommended because of the potential for naming conflicts between the dictionaries. For example, a dialog in one dictionary could have the same name as a map in another dictionary, both of which have an associated load module.

**Important:** Under no circumstances should the DDLCAT and DDLCATX areas be placed in different segments.

**Page groups:** All segments associated with a dictionary must have the same page group (and maximum number of records per page). If you have different page groups, you will receive errors when you attempt to access the dictionary through IDD or other dictionary tools.

This rule also applies to the system message area (SYSMSG.DDLDCMSG). It can only be included in dictionaries whose other segments have the same page group as the SYSMSG segment. When processing a dictionary with a difference page group, IDD cannot be used to display or update messages. Maintenance of the system message area can only be done from a dictionary that has the same page group as the SYSMSG segment.

**Page groups and SQL:** When defining an application dictionary that contains a catalog component, the page groups of the base and catalog components may be different. The page group of the catalog component has no impact on the page group of data that may be accessed while connected to the dictionary.

## 23.2 CA-supplied dictionary definitions

**Provided on install tape:** As part of installation, you receive definitions for entities required to operate your CA-IDMS/DB environment. These definitions are described below:

Definitions	Description
Non-SQL descriptions of the dictionary	A schema and subschemas describing the base definition and message components and that part of the catalog component used for physical database definitions
At sites with the SQL option, an SQL description of the catalog component	Table definitions of the catalog component of the SYSTEM schema and views based on those tables in the SYSCA schema
Runtime messages	Messages used by CA-written software
Entity, class, and attribute definitions	Definitions of base entity, class, and attributes used by CA-IDMS tools
Protocols and standard error routines	Generalized source modules that the DML processors use to convert DML statements into calls for DBMS services
DC/UCF device types, task, and program definitions	Definitions used to generate DC/UCF systems
CA-CULPRIT report modules	CA-CULPRIT source modules used to produce standard reports; for example, JREPORTs, SREPORTs, and DREPORTs
Nondatabase structures	Records that are not associated with a CA-IDMS/DB database. CA-IDMS/DB stores the definitions of nondatabase structures as records in the dictionary; applications can copy the definitions of the records at compile time by means of COPY IDMS or INCLUDE IDMS compiler-directive commands.

**How the dictionary gets populated:** Dictionaries are populated with CA-supplied definitions in one of three ways:

IDMSDIRL	Loads the non-SQL schema and subschemas that define the base definition and message components of the dictionary
IDD, IDMSCHEM, IDMSUBSC	Populates the base definition and message components of the dictionary using source members provided at installation

---

Command facility	Populates the catalog component of the dictionary with system table and view definitions (SQL-option only)
------------------	--

---

**Where information should reside:** The information listed above can reside in either a system dictionary or an application dictionary, or both:

Information	Where it should reside
Non-SQL schema and subschema definitions	In <i>one</i> dictionary associated with each system; the definitions may be shared across systems
SQL definitions	In all dictionaries having a catalog component (SQL-option only)
Messages	In all system message areas
Entity, class, and attribute definitions	In all system and application dictionaries
Protocols and standard error routines	In all application dictionaries
DC/UCF device types, task, and program definitions	In all system dictionaries
CA-CULPRIT report modules	In the same dictionary that contains the non-SQL schema and subschema definitions of the dictionary

---

### 23.2.1 Logical database definitions

**CA-supplied schema:** The table below describes the non-SQL schema supplied by CA that describes a dictionary. Its definitions are stored in a dictionary by the IDMSDIRL utility.

Schema	Areas
IDMSNTWK	DDLML DDLCLD DDLCAT DDLCATX DDLDCMSG

---

**CA-supplied subschemas:** The following table describes subschemas supplied by CA and the CA-IDMS products or facilities that make use of them. Most of these subschemas are distributed as object modules only. The source definitions of IDMSNWKA and IDMSNWKG are also stored in a dictionary by IDMSDIRL for user-reporting purposes.

<b>Subschema</b>	<b>Areas</b>	<b>Used by</b>
IDMSCATL	DDLCLATL	<ul style="list-style-type: none"> <li>■ Loader processing</li> <li>■ CLOD DC/UCF system task</li> <li>■ PUNCH utility statement</li> <li>■ Database administrators when executing utilities such as UNLOAD/RELOAD against the DDLCLATL area</li> </ul>
IDMSCATZ	DDLCLATZ DDLCLATX DDLCLATL	<ul style="list-style-type: none"> <li>■ The command facility for SQL processing and physical database definition</li> <li>■ User applications issuing dynamic SQL requiring automatic recompilation of an access module or issuing SQL DDL statements</li> <li>■ Database administrators when executing utilities such as UNLOAD/RELOAD against SQL-defined DDLCLAT and DDLCLATX areas</li> </ul>
IDMSNWKA	DDLML DDLCLCLOD DDLCLMSG	<ul style="list-style-type: none"> <li>■ IDD DDDL compiler (IDMSDDL)</li> <li>■ DC/UCF system generation compiler (RHDCSGEN)</li> <li>■ DC/UCF startup</li> <li>■ Schema and subschema compilers (IDMSCHEM and IDMSUBSC)</li> <li>■ CA-IDMS-DC mapping compilers (MAPC and batch)</li> <li>■ CA-ADS compilers</li> <li>■ CA-OLQ</li> <li>■ CA-CULPRIT</li> <li>■ The Automatic System Facility (ASF)</li> </ul>
IDMSNWKL	DDLCLCLOD	Loader processing and the CLOD DC/UCF system task
IDMSNWKT	DDLML	SQL processing to access non-SQL defined database descriptions

Subschema	Areas	Used by
IDMSNWKU	DDLML DDLDCLOD DDLDCMSG DDLCAT DDLCATX	Database administrators when executing utilities such as UNLOAD/RELOAD against dictionary areas DDLML, DDLDCLOD, DDLDCMSG and DDLCAT and DDLCATX for non-SQL defined segments only
IDMSNWKG	DDLML DDLDCLOD DDLDCMSG DDLCAT DDLCATX	■ IDMSRPTS
IDMSNWK6	DDLDCMSG	System message processing
IDMSNWK7	DDLDCRUN	QUED and QUEM DC/UCF system tasks and queue processing
IDMSNWK8	DDLML	CLIST and send-message processing
IDMSNWK9	DDLDCLOG	Online print log (OLP) and PRINT and ARCHIVE LOG utility statements

**Note:** Additional non-SQL schemas and subschemas are supplied at installation time. For more information, refer to *CA-IDMS Security Administration*.

**SQL table definitions:** At sites with the SQL option, CA-IDMS/DB also provides the table and view definitions that describe the catalog component of the dictionary. These definitions are distributed under two schema names:

SYSTEM	Contains the catalog table definitions; no changes can be made to any of the entities in the SYSTEM schema
SYSCA	Contains the CA-supplied views of the SYSTEM tables and records in the IDMSNWK schema; these views restrict access to table definition information based on a user's SELECT authority on the table.

►► For a description of the table definitions, refer to the *CA-ADS Reference*.

## 23.2.2 Protocols, nondatabase structures, and modules

The following table summarizes the protocols, nondatabase structures, and modules placed in the dictionary at installation time:

<b>Language</b>	<b>Protocol</b>	<b>Non-database structure</b>	<b>Module</b>
COBOL	BATCH BATCH-AUTOSTATUS CICS CICS-AUTOSTATUS CICS-EXEC CICS-EXEC-AUTO CICS-STANDARD CICS-STD-AUTO DC-BATCH IDMS-DC UTM UTM-AUTOSTATUS IDMS-DC-NONAUTO IDMSDML-PROTOCOL-SQL	DB-STATISTICS SUBSCHEMA-CTRL for IDMS-DC IDMS-DC-NONAUTO DC-BATCH CICS CICS-AUTOSTATUS CICS-EXEC CICS-EXEC-AUTO CICS-STANDARD CICS-STD-AUTO UTM UTM-AUTOSTATUS SUBSCHEMA-LR-CTRL	IDMS-STATUS for BATCH-AUTOSTATUS IDMS-DC DC-BATCH all others
PL/I	BATCH CICS CICS_EXEC DC_BATCH IDMS_DC IDMSDML_PROTOCOL_SQL	DB-STATISTICS SUBSCHEMA_CTRL for CICS CICS_EXEC IDMS_DC DC_BATCH SUBSCHEMA_LR_CTRL	IDMS_STATUS IDMS_STATUS (IDMS_DC)
FORTRAN	BATCH FOR77	SSCTRL SSLRCTL	
Assembler	BATCH CICS CICS-AUTOSTATUS CICS-EXEC CICS-EXEC-AUTO IDMSDC	SSCTRL for CICS CICS-AUTOSTATUS CICS-EXEC CICS-EXEC-AUTO SSLRCTL	DBSTATS
RPG II	BATCH	SSCT SSLRCTL	

## 23.3 Defining new dictionaries

### 23.3.1 Defining new catalog components

**Physical characteristics:** The segment definition for all catalog components must have the following characteristics:

- The names of the areas must be DDLCAT, DDLCATX, and DDLCATLOD
- The page size of the areas should be at least 4856 plus page reserve

All other physical characteristics can be chosen based on processing requirements, hardware configuration, and standard database design techniques. For example, choose an access method and page size appropriate for your disk devices and consider using a page reserve on the DDLCATX area.

**Catalog components for non-SQL use:** Without the SQL option, only a system dictionary requires a catalog component. When defining the corresponding segment, specify FOR NONSQL (or take the default).

**Catalog components for SQL use:** If the SQL option has been installed at your site, one or more of your application dictionaries will have an associated catalog component in order to define tables and views. The corresponding segment must have the following attributes:

- FOR SQL specification on the segment
- STAMP BY AREA for the DDLCAT and DDLCATLOD areas
- STAMP BY TABLE for the DDLCATX area

The catalog associated with the system dictionary can also be defined with these attributes. If it is, SQL can be used to examine the physical database definitions stored in the system dictionary.

When a new SQL catalog component is defined, take the following steps after the new segment has been formatted:

1. Define the system tables and views in the new catalog using the TABLEDDL and VIEWDDL members in the installation source library
2. Issue the UPDATE STATISTICS utility statement against the new DDLCAT area
3. Grant appropriate authorities to permit authorized users to create schemas in the new dictionary

## 23.3.2 Defining new application dictionaries

**Steps:** To create a new application dictionary, follow these steps:

Action	Steps
Start a session in the command facility	CONNECT TO SYSTEM
Define segments for the base definition and the catalog components of the dictionary  Note that you need the catalog component only if the SQL option is installed at your site.	CREATE SEGMENT
Add the new segment(s) to the DMCL used at runtime	ALTER DMCL with the ADD SEGMENT clause
If you created two segments, define a new database name in the database name table	CREATE DBNAME
Generate, punch, and linkedit the new DMCL	See Chapter 25, "Modifying Physical Database Definitions" on page 25-1
If you created a new database name, generate, punch, and linkedit the new database name table	See Chapter 26, "Modifying Database Name Tables" on page 26-1
Create and format new dictionary files	See Chapter 16, "Allocating and Formatting Files" on page 16-1
Make the DMCL available to the runtime system	See Chapter 25, "Modifying Physical Database Definitions" on page 25-1
Populate the dictionary with CA-supplied definitions	Use IDD DDDL statements to add entity, class, and attribute definitions, protocols, and standard error routines
If you created a new catalog component: <ul style="list-style-type: none"> <li>■ Populate it with the system table and view definitions</li> <li>■ Execute UPDATE STATISTICS for the DDLCAT area of the new dictionary</li> <li>■ Grant appropriate authorities to define schemas in the new dictionary</li> </ul>	

**Example:** The following example illustrates how to define a new application dictionary. It consists of a new definition component in segment TESTDICT, a new catalog component in segment TESTCAT, and the system message component.

The database name for the dictionary is TESTDICT.

1. Define a new segment containing the necessary areas:

```
create segment testdict
  for nonsql
  page group 0
  maximum records per page 255;

create segment testcat
  for sql
  page group 0
  maximum records per page 255
  stamp by area;

add file testcat
  assign to testcat
  dsname 'test.testcat';

add file testcatx
  assign to testcats
  dsname 'test.testcatx';

add file testcatl
  assign to testcatl
  dsname 'test.testcatl';

add file testdml
  assign to testdml
  dsname 'test.ddldml';

add file testlod
  assign to testlod
  dsname 'test.ddldclod';

add area ddldml
  primary space 10000 pages
  from page 5000001
  maximum space 20000 pages
  page size 4276
  within file testdml
  from 1 for all blocks;

add area ddldclod
  primary space 1000 pages
  from page 5020001
  maximum space 5000
  page size 8196
  within file testlod
  from 1 for all blocks;
```

```
add area ddlcat
  primary space 5000 pages
  from page 5030001
  maximum space 10000 pages
  page size 8196
  within file testcat;
```

```
add area ddlcatx
  primary space 1000 pages
  from page 5040001
  maximum space 3000 pages
  page size 8196
  within file testcatx;
```

```
add area ddlcatlod
  primary space 500 pages
  from page 5045001
  maximum space 5000 pages
  page size 8196
  within file testcatl;
```

2. Modify the DMCL

3. Generate, punch, and linkedit the new DMCL:

```
generate dmcl idmsdmcl;
```

4. Define a new database name for the dictionary

```
add dbname alldbs.testdict
  segment testdict
  segment testcat
  segment sysmsg;
```

5. Generate, punch, and link the database name table:

```
generate table dbtable alldbs;
```

6. Create and format new dictionary files:

```
format segment testdict;
format segment testcat;
```

7. Populate the dictionary using the appropriate source from the installation source library.

8. Execute UPDATE STATISTICS for the new DDLCAT area:

```
update statistics for area testcat.ddlcat;
```

9. Assign appropriate authorities within the new dictionary.

### 23.3.3 Defining new system dictionaries

**Steps:** To create a system dictionary for a new system, follow these steps:

<b>Action</b>	<b>Steps</b>
Start a session in the command facility	CONNECT TO SYSTEM
Create new segments that contain these dictionary areas  DDLDDL DDLDCLOD DDLDCAT DDLDCATX DDLDCATLOD DDLDCMSG	CREATE SEGMENT
<b>Note:</b> Use segment name that are different than existing segment names.	
If you created more than one segment, create a database name table entry that contains all the segments you created	CREATE DBNAME
Add the segment(s) to the DMCL	ALTER DMCL with the ADD SEGMENT clause
Generate, punch, and link the DMCL	See Chapter 25, “Modifying Physical Database Definitions” on page 25-1
If you created more than one segment, generate, punch, and link the database name table	See Chapter 26, “Modifying Database Name Tables” on page 26-1
Format the new dictionary files	FORMAT FILE/SEGMENT
Grant appropriate administrative privileges to authorized individuals on and within the new dictionary	Refer to <i>CA-IDMS Security Administration</i>

Action	Steps
<p>Re-define the dictionary segment(s) in the new dictionary by either:</p> <ul style="list-style-type: none"> <li>■ Creating new dictionary segment(s)</li> <li>■ Punching the segment definitions from the current SYSTEM dictionary and re-adding them to the new dictionary</li> </ul> <p>Make sure the segment name of the message area in the new dictionary is SYSMMSG. Define additional segments necessary for a complete runtime environment.</p>	<ul style="list-style-type: none"> <li>■ CREATE SEGMENT</li> <li>■ PUNCH SEGMENT</li> </ul>
<p>Define a database name table that includes the database name SYSTEM; SYSTEM must identify the new dictionary segments. Add additional entries as necessary.</p>	<ul style="list-style-type: none"> <li>■ CREATE DBTABLE</li> <li>■ CREATE DBNAME</li> </ul>
<p>Create a new DMCL with associated database buffers, a journal buffer, and journal files</p>	<p>See Chapter 4, “Defining, Generating, and Punching a DMCL” on page 4-1</p>
<p>Add the new segments and associate the database name table with the new DMCL</p>	<p>ALTER DMCL</p>
<p>Generate, punch, and link the new DMCL</p>	<p>See Chapter 4, “Defining, Generating, and Punching a DMCL” on page 4-1</p>
<p>Generate, punch and link the new database name table</p>	<p>See Chapter 5, “Defining a Database Name Table” on page 5-1</p>
<p>Populate the system dictionary with the following CA-supplied definitions:</p> <ul style="list-style-type: none"> <li>Entity, class, and attribute definitions</li> <li>DC/UCF device types, tasks, and programs</li> </ul>	
<p>If the new catalog segment was defined as FOR SQL, complete its definition.</p>	<p>See 23.3.1, “Defining new catalog components” on page 23-13 earlier in this chapter.</p>

## 23.4 Establishing a default dictionary

**What is a default dictionary:** A default dictionary is the dictionary that will be accessed by CA-IDMS tools if you don't specify a dictionary by other means such as using a DCUF SET DICTNAME command or a CONNECT statement.

**Defining a default dictionary:** To define a default dictionary for your runtime environment, include a subschema mapping in the database name table associated with the runtime DMCL for the IDMSNWK subschemas. For example, the statement below establishes TESTDICT as the default dictionary for the runtime environment using the ALLDBS database name table:

```
create dbtable allpbs
  add subschema idmsnwk? maps to idmsnwk? dbname testdict;
```

## 23.5 About runtime environments

**Central version or local mode:** CA-IDMS/DB can run within a DC/UCF system as a central version or in local mode:

- Central version operations provide database services to batch or online applications. Multiple users can gain access to a database concurrently.
- Local mode operations are batch operations that do *not* run under a central version. In local mode, only one user at a time has access to a database area in update mode.

**Data sharing environment:** Data sharing is an environment in which two or more central versions operate cooperatively through the use of a coupling facility. In this environment, multiple central versions may concurrently access a database area in update mode.

**Central version runtime components:** The table below lists the components needed for a central version runtime environment:

Component	Description
System dictionary	Defines the DC/UCF system and physical database entities
DDLDCLOG	Contains central version log records when the log file for the central version is assigned to the database
DDLDCRUN	Contains runtime queue information used by CA-supplied tools and online user programs
DDLDCSCR	Contains runtime scratch information used by CA-supplied tools and online user programs
SYMSMSG.DDLDCMSG	Contains CA-supplied and user-defined messages
DDLSEC	Contains user and group information
Application dictionaries	
User databases	

### Considerations

- The segment name of the system message area must be SYMSMSG.
- The segment(s) associated with DDLDCLOG, DDLDCRUN, and DDLDCSCR must be included in the SYSTEM database name
- Each central version must have its own DDLDCLOG and DDLDCSCR. In a non-data sharing environment, each central version must also have its own DDLDCRUN area. In a data sharing environment, the DDLDCRUN area may be shared among members of a data sharing group.

- The DDLSEC area may not be necessary depending on your security implementation.

►► For more information about sharing the DDLDCRUN area, refer to *CA-IDMS System Operations*.

►► For more information about security, refer to *CA-IDMS Security Administration*.

**Local mode runtime components:** The table below lists the components needed for a local mode runtime environment:

Component	Description
System dictionary	Defines the DC/UCF system and physical database entities
SYSMSG.DDLDCMSG	Contains CA-supplied and user-defined messages
DDLSEC	Contains user and group information
DDLOCSCR	Contains runtime scratch information used by local mode CA-supplied tools and user programs issuing SQL requests
Application dictionaries	
User databases	

### Considerations

- The segment name of the system message area must be SYSMSG.
- The system dictionary and DDLSEC area may not be necessary depending on your security implementation.
  - For more information on security, refer to *CA-IDMS Security Administration*.
- At least the default dictionary should be available in local mode. Additional application dictionaries may be needed for loading subschemas and processing SQL requests.

**What follows:** What follows is a description of:

- The SYSIDMS parameter file, which describes the runtime parameters in a batch environment
- How to establish a default dictionary for your session

## 23.5.1 SYSIDMS parameter file

**About SYSIDMS parameters** A SYSIDMS parameter is a parameter that can be added to the JCL stream of a batch job running in local mode or under the central version. You can use SYSIDMS parameters to specify:

- Physical requirements of the environment, such as the DMCL and database to use at runtime
- Runtime directives that assist in application execution
- Operating system-dependent file information

For a complete list of the parameters that can be specified, see Appendix H, “SYSIDMS Parameter File” on page H-1.

**Establishing site defaults:** Site-specific defaults can be established for all SYSIDMS parameters by assembling a SYSIDMS defaults load module. If it exists, this load module is used at runtime to determine the default values for all SYSIDMS parameters. Defaults may then be overridden in an individual job step by including a SYSIDMS parameter file in the execution JCL.

**Creating a SYSIDMS defaults load module:** The following example illustrates how to code a SYSIDMS defaults load module. It is a table of 80-character constants, each of which may contain one or more SYSIDMS parameters, as described in Appendix H. A parameter and its value must be contained within a single 80-character constant, but more than one parameter may appear within a constant. The last constant must have a value of "END SYSIDMS DEFAULTS."

```
TITLE 'SYSIDMS - Build load module for SYSIDMS defaults'
SYSIDMS START 0
*****
* Code any SYSIDMS parms that you want to be part of this SYSIDMS
* defaults load module. This SYSIDMS defaults load module will be
* processed first before trying to process any SYSIDMS parms defined
* in the JCL for any IDMS batch job.
*****
      SPACE
      DC CL80'ECHO=ON DMCL=GLBLDMCL'
      DC CL80'JOURNAL=OFF'
      SPACE
* The following statement is mandatory and must be the last statement
* in the SYSIDMS defaults load module.
      DC CL80'END SYSIDMS DEFAULTS'
      END
```

**Linking a SYSIDMS defaults load module:** The load module must have both a name and an entry point of SYSIDMS. For operating systems that support XA storage, the load module can be linked as AMODE 31, RMODE ANY.

**Overriding SYSIDMS parameter defaults:** SYSIDMS default values can be overridden for an individual job step by including a SYSIDMS parameter file in the execution JCL.

In the following example, the SYSIDMS parameters included in the job stream instruct CA-IDMS/DB to use the DMCL LOCLDMCL to execute a job. DBNAME identifies EMPDB as the database to access at runtime, and the QSAM parameters instruct CA-IDMS/DB to use the IDMSQSAM look-ahead read facility when accessing EMPSEG.EMPAREA:

```
//SYSIDMS DD *
DMCL=LOCLDMCL DBNAME=EMPDB
IDMSQSAM=ON QSAMAREA=EMPSEG.EMPAREA
```

In the following example, the SYSIDMS parameters used are typical for a batch job running under the central version:

```
//SYSIDMS DD *
DBNAME=EMPDB NODENAME=SYSTEM90
```

## 23.5.2 Establishing session options

**Established at signon:** CA-IDMS establishes options for your runtime session when you signon on to a DC/UCF system or when CA-IDMS/DB issues its first database request from a batch application (in local mode or under the central version) or external teleprocessing monitor. The manner in which CA-IDMS implements the options and how they affect your session depends on the runtime environment.

**Specifying a default database or dictionary:** CA-IDMS/DB provides several ways to specify a session default dictionary or database. The methods available depend on the runtime environment.

**Online processing:** To specify a session default in an online environment, you can:

- Specify DICTNAME/DICTNODE or DBNAME/DBNODE attributes in a system or user profile
- Issue a DCUF command
- Issue a compiler SIGNON or CONNECT statement naming the dictionary and/or database from within an online CA-IDMS/DB compiler or tool (this will update the default dictionary for the runtime session)

**Batch processing:** To specify a session default dictionary for a batch central version or external teleprocessing monitor application, you can use:

- An IDMSOPTI module (for non-SQL applications only)
- A SYSCTL file
- A SYSIDMS parameter file

►► For more information about how CA-IDMS/DB determines which database or dictionary to access when provided with information by the program, IDMSOPTI module, SYSCTL file, and SYSIDMS file, refer to *CA-IDMS System Operations*

**Local mode processing:** To specify a session default for local mode, you can use:

- An IDMSOPTI module (for non-SQL applications only)
- A SYSIDMS parameter file

## 23.6 Related information

- About database name tables, see Chapter 5, “Defining a Database Name Table” on page 5-1
- About the SYSCTL file and IDMSOPTI module, refer to *CA-IDMS System Operations*
- About dictionary entities, refer to the *IDD DDDL Reference*
- About system table definitions and system record definitions, refer to the *CA-ADS Reference*
- About SYSIDMS parameter syntax, see Appendix H, “SYSIDMS Parameter File” on page H-1



# Chapter 24. Migrating from Test to Production

---

- 24.1 About migration . . . . . 24-3
- 24.2 Establishing migration procedures . . . . . 24-4
- 24.3 Implementing migration procedures . . . . . 24-5
  - 24.3.1 Step 1: Determine the types of components to migrate . . . . . 24-5
  - 24.3.2 Step 2: Determine the sequence of migration . . . . . 24-9
  - 24.3.3 Step 3: Identify the individual components . . . . . 24-11
  - 24.3.4 Step 4: Migrate the components . . . . . 24-11
- 24.4 Identification aids . . . . . 24-12
- 24.5 Migration tools . . . . . 24-15
- 24.6 General methods . . . . . 24-17
  - 24.6.1 Using the DISPLAY statement . . . . . 24-17
  - 24.6.2 Using the PUNCH statement . . . . . 24-18
  - 24.6.3 Using the mapping compiler and mapping utility . . . . . 24-22
  - 24.6.4 For SQL-defined entities . . . . . 24-23
- 24.7 Additional considerations . . . . . 24-25
  - 24.7.1 Additional tasks . . . . . 24-25



## 24.1 About migration

**Migrate definitions from one dictionary to another:** Whether you have multiple dictionaries under a single CA-IDMS/DC system or several dictionaries under separate CA-IDMS/DC systems, you probably need to migrate definitions from one dictionary to another. Typically, migration occurs when testing is complete and an application is ready for production. At that time, the database and application definitions must be moved from the test into the production environment.

**Considerations for non-SQL and SQL defined data:** The need to migrate database and applications applies to both SQL-defined and non-SQL defined databases and applications using SQL or navigational DML. Most of this chapter applies to both SQL and non-SQL equally. Text that applies specifically to one or the other will be noted.

## 24.2 Establishing migration procedures

**Considerations:** Because many of the pieces of an application, such as subschemas, maps, and dialogs, exist in both source and load module format, you must consider the following questions when you migrate from one dictionary to another:

- Should you copy or move the components?
- Should you migrate and recompile source code to produce load modules?
- Should you migrate just the load modules?

**Accessibility of the source code:** The major benefit of a complete, fully documented application is that the proper source code is accessible when needed for debugging. If a problem arises and the source code resides in a properly controlled production environment, the source code can easily be found and it will correspond exactly to the load module(s) where the problem was encountered.

**Availability of disk space:** A trade-off to migrating a fully documented application is the amount of disk space required. The space may be in one environment, such as production, or may be spread out over a number of environments, such as development, test, and production. Determining exactly how much disk space is necessary depends on whether you decide to copy the application into the production environment or simply move it.

**Redundancy:** If you choose to maintain separate copies of the application, you must contend with the trade-offs of redundancy. Often, updates to one copy must also be made to the other, and they both must be made within a short period of time in order to maintain consistency.

**Accessibility of information:** If you maintain only one copy of the application, you use a minimum amount of disk space and do not have to contend with redundancy. However, accessibility of information becomes a consideration. If the information is secured so that only one person is able to access it, procedures must be developed that allow maintenance programmers and all members of the staff to obtain reports of component definitions. At the same time, you must ensure that there is ample security so that no one can make accidental or malicious updates that would invalidate production applications.

---

## 24.3 Implementing migration procedures

**Steps:** There are essentially four steps involved in migration:

1. Determine the types of components to migrate  
Carefully examine the circumstances for dependencies and other relationships among the components involved.
2. Determine the sequence of migration  
Components that do not depend on the definitions of other components should be first on the list.
3. Identify the components  
Identify the names, version numbers, and, as appropriate, languages of the individual components that you need to migrate.
4. Migrate the components using the batch and online compilers and utilities

These steps are discussed on the following pages.

**Before you begin:** Before you begin a migration, you may want to back up all involved files. These files can include:

- Source and target DDLML, DDLDCLOD, DDLCAT, DDLCATX, and DDLCATLOD areas
- Source and target source libraries
- Source and target load libraries
- Source and target JCL procedure libraries

These backups provide coverage during the migration as well as after the migration is complete. If problems arise at any time, you can restore individual components or entire files from the backups.

### 24.3.1 Step 1: Determine the types of components to migrate

The components to be migrated should include not only what needs to be migrated but also what is **affected** by the migration. The descriptions that follow identify components typically involved in migration and how these affect other components.

**Application structure:** The application structure is saved as a load module in the DDLDCLOD area of the data dictionary; no source definitions for the application are stored in the DDLML area. The application structure is relatively autonomous. If you make changes to the application structure, you do not need to recompile any other application components.

Changes to the application structure, however, can logically affect other components, specifically dialogs. For example, if you change a response name, you will want to change the response field value of any response processes you expect to execute before

control is passed to the response. The application will execute without modifying the dialog, but it will not produce the expected results.

**Maps:** Changes to maps fall into two categories:

- Critical changes
- Noncritical changes

Critical changes update the date/time stamp. Any dialogs that use the map must be recompiled before they can be executed. Critical changes to maps include:

- Adding a data field to the map
- Deleting a data field from the map

Noncritical changes to maps do not cause the map date/time stamp to be updated and, therefore, do not affect any other application components.

**Dialogs:** Dialogs associate subschemas or access modules, maps, and process code. The dialog load module contains executable process source code. Recompiling a dialog creates a new relational command module (RCM). Any access modules that include that RCM must then be recompiled also. Recompiling a dialog does not affect any other application component.

**Process source code:** Process source code is stored in the data dictionary. Process code is compiled by the dialog compiler and becomes executable when the dialog is compiled. To have changes to process source code reflected in the dialog load module, you must recompile the dialog.

**RCMs (SQL DML applications only):** If a program/dialog containing SQL statements is recompiled, a RCM is automatically created for it and stored as a load module in the DDLDCLD area. If the program/dialog load module is copied intact to the production system, the RCM load module must also be copied.

**Subschemas (navigational DML applications only):** If you change a subschema associated with a dialog, map, or program, you do not need to recompile the dialog, map or program. If the subschema changes cause you to change the logic of a process module, you will need to recompile the dialog(s) in which the module is used. If the subschema changes affect the lengths of data elements or records or the procedural code in a program, you will need to recompile and relink the program.

**Access modules (SQL DML applications only):** Access modules must be compiled from scratch using the catalog that defines the physical database being accessed. They *cannot* be copied in load module form like other application components. A typical migration would copy the RCM load modules, apply any needed database definition changes and then create all access modules used by the application, using the CREATE ACCESS MODULE command.

**Non-SQL data definitions:** Non-SQL data is defined in records that consist of record elements. Records are either database records, which are included in a schema, or work records, which are defined through the DDDL compiler.

Changes to database records require that all subschemas that use those records be recompiled. All SQL access modules that reference those records must also be recompiled for SQL applications that access non-SQL defined databases. Changes to either database records or work records may require map and/or dialog recompilation.

Some changes to database records require some form of restructuring to incorporate those changes into the existing database.

►► For more information about modifying the schema definition of a non-SQL defined database, see Chapter 31, “Modifying Schema Entities” on page 31-1.

**SQL data definitions:** Data is defined in tables that consist of columns. Changes to these tables require that all access modules that use those tables be recreated. Depending on the definition of a particular access module, this recreation may occur automatically or may have to be initiated manually. These changes may require map and/or dialog recompilation.

Some changes to table definitions requires some form of restructuring to incorporate those changes into the existing database.

►► For more information about modifying the schema definition of an SQL defined database, see Chapter 28, “Modifying Schema, View, and Table Definitions” on page 28-1.

**Adaptive query management:** Adaptive query management is a feature of the IDMS SQL option that automatically recompiles access modules in response to certain kinds of changes in a database application. For example, if a dialog/program has been recompiled, the runtime SQL engine detects whether corresponding access modules have been recompiled to include the new RCM. If not, it automatically recompiles the access module at runtime (if the AUTO RECREATE ON option was specified when the access module was created or last altered). Adaptive query management applies to SQL DML applications that access either non-SQL or SQL-defined databases.

Adaptive query management also automatically recompiles existing access modules that access SQL-defined databases when the definitions of those databases change. Note that this does *not* happen for non-SQL defined databases. It is the responsibility of the applications administrator to manually recompile any access modules affected by changes to a non-SQL defined database.

**Edit and code tables:** Changes to stand-alone edit and code tables that are associated with a map require that the map be recompiled only if the tables are **linked** to the map. Changes to unlinked tables do not affect the map load module.

## Examples

*Example 1 Adding a data item to a screen:* Suppose users of an application request an additional data item on a screen. To determine what is affected, consider the relationship between the map and the new data item:

- For an application using navigation DML, you need to do the following if the data item is from a database record already being used by the map:
  1. Change the map to display the data item
  2. Recompile the map
  3. Recompile any dialogs that use the map

To take these actions, you need to migrate the map and the dialogs.

- For an application using navigation DML, you need to do the following if the database record is part of the subschema used by the map, but the record is not already in use by the map:
  1. Add the record to the map definition
  2. Change the map to display the data item
  3. Recompile the map
  4. Recompile any dialogs that use the map

To take these actions, you need to migrate the map and the dialogs.

- For an application using navigation DML, you need to do the following if the database record is not already part of the subschema:
  1. Add the record to the subschema
  2. Recompile the subschema
  3. Add the record to the map definition
  4. Change the map to display the data item
  5. Recompile the map
  6. Recompile any dialogs that use the map

To take these actions, you need to migrate the subschema, map, and dialogs.

- If the data item can be derived (for example, calculated) from data already available to the application, you need to:
  1. Create a work record for the map and add it to the map definition or modify the existing work record
  2. Change the map to display the data item
  3. Recompile the map
  4. Change any processes that must derive the data item
  5. Recompile any dialogs that use the map

To take these actions, you need to migrate the record, subschema, map, affected processes, and dialogs.

- If the application uses SQL DML, a work record will already have been defined to move data between the map and the SQL statements in the dialog. To add another database item to the screen, you need to:
  1. Add the item to the work record already defined for the host variables referenced in the SQL DML statements.
  2. Change the map to display the data item.
  3. Recompile the map.
  4. Make necessary changes to the SQL statements to retrieve the data item from the database.
  5. Recompile any dialogs that contain altered SQL statements and any dialogs that use the map.
  6. Recompile (using the ALTER ACCESS MODULE statement) any access modules that contain the recompiled dialogs.

*Example 2 Implementing a new application:* Suppose you implement an entirely new application based on an existing database. When the new application has been adequately tested, all of the application components need to be migrated from the test system to the production system. In addition, you must also consider what database components to migrate:

- If you have not made any changes to the structure of the database, then the existing schema and physical definitions are not affected
- Depending on the volume and type of activity involved in the new application, you may need to adjust the buffers and review the adequacy of the journals in the global DMCL
- If the application uses navigational DML and you used existing subschemas, they, too, are unaffected by the migration. However, if you created new subschemas for the application, you must migrate them.
- If the application uses SQL DML, you must migrate any RCMs and access modules that were created as part of the application.

### 24.3.2 Step 2: Determine the sequence of migration

**Can migrate load module at any time:** If you choose to migrate only load modules, the sequence in which you migrate them does not matter.

**Sequence matters for source code migration:** If you migrate any source code, the sequence is *very* important because there are dependencies among the components.

In some migrations, certain components will already be in place; in others, you will need to migrate all components. The list below shows the sequence required if all components were to be migrated.

Non-SQL database definitions	<ol style="list-style-type: none"> <li>1. Elementary data items</li> <li>2. Group level data items</li> <li>3. Database records</li> <li>4. Schemas</li> <li>5. Subschemas</li> </ol>
SQL database definitions	<ol style="list-style-type: none"> <li>1. Schemas</li> <li>2. Tables</li> <li>3. CALC keys</li> <li>4. Indexes</li> <li>5. Constraints</li> </ol>
Physical database definitions	<ol style="list-style-type: none"> <li>1. Segments</li> <li>2. Areas</li> <li>3. Files</li> <li>4. DMCL modules</li> <li>5. Database name tables</li> </ol>
Application components definitions	<ol style="list-style-type: none"> <li>1. Edit and code tables</li> <li>2. Work records for elementary data items, group level data, maps, and dialogs</li> <li>3. CA-ADS process modules</li> <li>4. Modules called by CA-ADS processes or other programs</li> <li>5. Maps</li> <li>6. CA-ADS application structures</li> <li>7. CA-ADS dialogs</li> <li>8. RCMs (for SQL DML applications only)</li> <li>9. Access modules (for SQL DML applications only)</li> </ol>
Components that can be migrated in any sequence	<ol style="list-style-type: none"> <li>1. Load modules</li> <li>2. Source code for batch and online programs</li> <li>3. CA-CULPRIT source code</li> <li>4. JCL</li> </ol>

### 24.3.3 Step 3: Identify the individual components

Having determined the types of components you need to migrate, you can begin to identify the individual occurrences. To identify them uniquely, you need both their names and version numbers. For modules, programs, and edit/code tables, you also need the name of the language in which they are programmed.

### 24.3.4 Step 4: Migrate the components

Depending on the volume of information and the configuration of your dictionaries, you can use batch or online facilities to move or copy the component definitions to their target dictionary.

**Using online compilers for migration:** If the volume of information is small and both dictionaries are under the control of the same CA-IDMS/DC or CA-IDMS/UCF system, you can use the online compilers for most of the migration.

**Using batch compilers for migration:** If the volume is large or if the dictionaries are under the control of separate CA-IDMS/DC or CA-IDMS/UCF systems, you need to use the batch compilers and utilities.

**Migrating only load modules:** If you only want to create an executable application in the production environment, you migrate just the essential load modules. Note that for SQL DML applications, the access modules must still be compiled from scratch on the production system.

**Migrating the complete application:** If you want a complete, fully documented application in the production environment, you need to:

- Migrate the source for all components
- Recompile the components
- Recompile the corresponding load modules

## 24.4 Identification aids

The descriptions below identify facilities or techniques you can use to identify the individual application components you need to migrate. To extract information on components stored in libraries or other data sets, use an appropriate operating system utility.

**IDD DISPLAY statement:** Using either the online or batch dictionary compiler, you can list the names and version numbers of entity occurrences with a simple form of the DISPLAY ALL statement. Any of the IDD entity types can be displayed.

Using an optional WHERE clause on the DISPLAY ALL statement, you can more closely select the occurrences you want displayed. With any entity types, you can qualify the occurrence name. For some entity types, there are additional selection criteria that you can specify, such as the user ID of the person who created the entity.

►► For more information on the DISPLAY ALL statement and its WHERE clause, refer to the discussion on entity-occurrence display in *IDD DDDL Reference*

**Command facility:** With either the online or batch command facility, you can:

- Display physical database definitions
- Use a SELECT statement to list, but not display the syntax of, SQL entity definitions

**IDMSRPTS:** IDMSRPTS is a utility that produces reports on information stored in the dictionary. One of its options, the Program Cross-Reference Listing, is particularly useful for migration operations if you are using program registration. The report lists all subschemas for a specified schema and all of the programs registered against those subschemas.

►► For a sample of this report and instructions on how to run the IDMSRPTS utility, refer to *CA-IDMS Utilities*.

**DREPORTs:** DREPORTs also report on information stored in the dictionary. There are some DREPORTs that summarize information for dictionary entities and some that present detailed information on these entities.

From the summary reports, you can obtain the names and version numbers of the components that need to be migrated. If you need to know whether other related components will be affected, you can run one or more of the reports that present detailed information.

►► For further information on DREPORTs, refer to *CA-IDMS Reports*.

**AREPORTs:** AREPORTs report on CA-ADS dialogs, application structures, and their associated components (such as subschemas, RCMs, maps, and processes) from the information stored in the DDLML area of the dictionary.

The complete detail report is most useful when you are planning the migration of an entire application. When planning the migration of more than one dialog, run the report that keys in on only the dialogs you need.

►► For further information on AREPORTs, refer to *CA-IDMS Reports*.

**SQL catalog:** The SQL catalog contains the definitions of all SQL-defined database entities. It also contains information on all access modules compiled using the catalog, and the tables that they reference (or records, for SQL DML applications that access non-SQL defined databases). Since the catalog is itself an SQL-defined database, SQL SELECT statements may be used to query its contents.

**Dictionary classes and attributes:** Classes and their attributes are primarily a means of extending the documentation capabilities in the dictionary. When migrating, documentation by class and attribute provides a powerful mechanism to analyze and identify the components involved. Using classes and attributes provides you with the capability to display a simple list of names or to report on the details of all components having the same attribute. For example, using the DDDL compiler, you can display all modules associated with attribute TEST within class STATUS:

display attribute test within class status with modules.

►► For more information on creating classes and attributes and display entities based on class and attribute, refer to *IDD DDDL Reference*. For more information on reporting by class and attribute, refer to *CA-IDMS Reports*.

**Naming conventions:** Naming conventions help in identifying and migrating components.

Although there are no hard-and-fast rules for designing naming conventions, there are a few factors that you should keep in mind:

- Collating sequence

Many of the DREPORTs display the components sorted in ascending order by name. If the names of all components of an application begin with the same few characters, it is easy to distinguish one application from another, but more difficult to distinguish components within an application. Likewise, if the names of all elements within a record begin with the same few characters, it is easy to distinguish one record from another in a list, but more difficult to distinguish elements within a record.

- Acceptable name lengths

The software permits names of different lengths for different components. If you want several characters of every name to identify the application, select a small

number (for example, 2 or 3) of characters for this purpose, in order to leave enough characters for other purposes.

- Consistent number of characters

Consider selecting a consistent number of characters to identify the record in which an element is placed or components within a particular application. If you choose a standard number of characters and place them in a standard position, it will be easy to sort information or to scan lists or reports for a particular item.

As an alternative to embedding an application identifier in component names, you may choose to use a class/attribute pair. This arrangement allows more characters per name for other purposes, while still providing a connection between components of the same application.

## 24.5 Migration tools

Most of the compilers and utilities you use to create database and application components also have options that support migration. The table below summarizes these tools:

<b>Component</b>	<b>Tool</b>	<b>Task code</b>	<b>Batch program</b>
Non-SQL defined schema source	Schema compiler	SCHEMA	IDMSCHEM
SQL-defined schema source	Command facility	OCF	IDMSBCF
Physical database definitions <ul style="list-style-type: none"> <li>▪ Segments, areas, files</li> <li>▪ Database name tables</li> <li>▪ DMCL source and load modules</li> </ul>	Command facility	OCF	IDMSBCF
Subschema source	Subschema compiler	SSC	IDMSUBSC
Subschema load module	DDDL compiler 1; subschema compiler	IDD; SSC	IDMSDDDL; IDMSSUBC
Definitions of: <ul style="list-style-type: none"> <li>▪ Elements</li> <li>▪ Messages</li> <li>▪ Modules</li> <li>▪ Programs</li> <li>▪ Records</li> </ul>	DDDL compiler 1	IDD	IDMSDDDL
Edit/code table source	DDDL compiler 1	IDD	IDMSDDDL
Map source	Mapping utility Mapping compiler		RHDCMPUT RHDCMAP1
Module source <ul style="list-style-type: none"> <li>▪ Copybook-style modules</li> <li>▪ CA-ADS process code</li> </ul>	DDDL compiler 1	IDD	IDMSDDDL

---

<b>Component</b>	<b>Tool</b>	<b>Task code</b>	<b>Batch program</b>
Load modules for: <ul style="list-style-type: none"><li>▪ Applications</li><li>▪ Dialogs</li><li>▪ Maps</li><li>▪ Edit/code tables</li><li>▪ RCMs</li></ul>	DDDL compiler <sup>1</sup>	IDD	IDMSDDDL
Access modules	Command facility	OCF	IDMSBCF

<sup>1</sup> All definitions that can be migrated using the DDDL compiler can also be migrated from the command facility.

---

---

## 24.6 General methods

**Tasks:** Migration generally consists of two or three tasks:

- Punching or decompiling components from a dictionary to a temporary work file or external file
- Compiling the components from the temporary work file or external file into the target dictionary
- Recompiling load modules, as necessary, in the target dictionary

The options of the schema, subschema, DDDL compilers, and command facility that you use for these tasks function identically. Different options exist in the mapping compilers and the mapping utility, and CA-ADS compilers.

The following discussions explore the methods of migration using the DISPLAY and PUNCH statement options of the schema, subschema, and DDDL compilers and the command facility, and the various parameters of the mapping compilers and the mapping utility.

**Techniques for SQL definitions and access modules:** The methods described below apply to non-SQL database definitions, physical database definitions, and RCM load modules. To migrate SQL database definitions, you need to copy stored source from the test to the production system. To do this you can store the definitions in:

- In a file that serves as input to the IDMSBCF compiler
- OCF-language modules, as described in 24.6.4, “For SQL-defined entities” on page 24-23 later in this section

**Exception for views:** You can use the DISPLAY or PUNCH techniques described below for view definitions. To obtain the view definition, select the SYNTAX column from the SYSCA.SYNTAX table.

### 24.6.1 Using the DISPLAY statement

**Use for small volumes of data:** The DISPLAY statement of the schema, subschema, and DDDL compilers, and command facility is useful for moving small volumes of information between dictionaries under the control of the same DC/UCF system. Because this technique occurs online, system resources, such as response time and storage pool space, will limit the volume you are able to migrate.

**Note:** This technique does not work for SQL database definitions and access modules unless you stored the source DDL in a module; if so, then follow the steps below by displaying the module.

**Steps:** There are four steps in the technique:

1. Sign on to the dictionary containing the components (the source dictionary)
2. Display the individual components using the AS SYNTAX clause.

This step accomplishes the task of decompiling the components to a temporary work file. If you need to modify existing components in the target dictionary, use the VERB MODIFY option of the DISPLAY statement (DISPLAY ADD is the default action):

```
display subschema empss01 as syntax verb mod.
```

3. While the components are in the compiler's work file, insert a SIGNON statement for the target dictionary into the work file as the first statement.

This step prepares for the task of compiling the components from the temporary work file into the target dictionary. At the conclusion of this step, the work file contains a SIGNON statement for the target dictionary, followed by ADD or MODIFY statements for those components you want to migrate.

**Note:** Typically the output of the previous step includes an echo of the input, so the first statement in the output is the DISPLAY statement. The DISPLAY statement is not necessary, so you can replace it with the SIGNON statement.

4. Invoke the compiler

The compiler signs you off the source dictionary, signs you on to the target dictionary, and adds or modifies the components in the work file.

**Final tasks for schemas and load modules:** This technique will copy the source to the target dictionary, but it does not automatically validate schemas or recompile load modules for subschemas and edit and code tables. You can perform these additional functions in one of two ways:

- After you compile the source into the target dictionary, establish currency on the appropriate component and issue the VALIDATE or GENERATE statement. To establish currency, issue a simple MODIFY statement for the component. For example:  

```
modify subschema empss01.  
generate.
```
- Before you compile the source into the target dictionary, edit the work file by inserting the VALIDATE or GENERATE statement after the source for the component.

## 24.6.2 Using the PUNCH statement

**Used for batch migration:** The PUNCH statement of the schema, subschema, and DDDL compilers and command facility is useful for batch migrations. If you perform the migration in batch mode, the PUNCH statement allows you to migrate larger volumes of information. It also allows you to migrate between dictionaries under the control of different DC/UCF systems.

**Writes information to file or module:** The PUNCH statement has the same options as the DISPLAY statement. However, it writes the requested information to one of two destinations: an external file or an IDD module.

---

**Note:** This technique does not work for SQL database definitions or access modules unless you stored the source DDL in an IDD module; if you did, then follow the steps below by punching the module.

**Use files or modules to accumulate large numbers of components:** The file or module provides an intermediate place to store the information you want to migrate. As a result, you can:

- Accumulate components in one or more modules over the course of several terminal sessions
- Accumulate several files of components over the course of separate executions of the batch compiler
- Edit the content of the modules or files; For example, to change the STATUS of components from TEST to PRODUCTION

**Technique 1:** This technique is very similar to the technique for the DISPLAY statement described above. Because it occurs in batch, however, you can migrate larger volumes of information.

**Steps:** The steps in this technique follow:

1. In the first execution of the compiler, sign on to the source dictionary in batch mode and punch the individual components to an external file.

In the PUNCH statement, use the AS SYNTAX clause. In addition, specify VERB MOD if you are migrating existing components. Define the file as SYSPCH in the JCL.

To avoid having to specify these clauses in every PUNCH statement, you can issue a SET OPTIONS statement before the PUNCH statements:

```
set options display as syntax verb mod.
```

2. When the job ends, edit the external file as follows:

- Insert a SIGNON statement for the target dictionary as the first statement.
- Insert the following statement after the SIGNON statement:

```
set options input 1 thru 80.
```

This step prepares for the task of compiling the components from the temporary file into the target dictionary. Be sure the SIGNON and SET OPTIONS statements start between columns 1 and 72.

- Execute the compiler a second time, using the edited file as input.

The compiler signs on the target dictionary and adds or modifies the components in the file.

**Technique 2:** With this technique, you create a dictionary module in the source dictionary to hold the components you want to migrate. You migrate the module to the target dictionary, extract the ADD or MODIFY statements for the individual components, and store or modify each of the components in the target dictionary.

*Steps:* The steps in this technique follow:

1. In batch or online mode, sign on to the source dictionary and create a module occurrence to hold the components to be moved. For example:
2. While signed on to the source dictionary, punch the components to be moved into the module using the TO MODULE and AS SYNTAX clauses:

```
add module holdit.
```

```
punch element emp-last-name
  to module holdit
  as syntax.
```

The module source for HOLDIT now consists of the ADD ELEMENT EMP-LAST\_NAME statement.

You can perform this step in batch or online mode, and you can punch more than one component to the module. If you use the SET OPTIONS statement following signon, your input appears as follows:

```
set options input 1 thru 80
  default is on
  punch to module holdit
  as syntax.
punch element emp-last-name.
.
.
.
```

This statement automatically changes an ADD to MODIFY if the entity already exists in the dictionary and punches the entity as syntax.

3. In batch mode, sign on to the source dictionary and punch the module to an external file.

The input to the compiler consists of only two statements: a SIGNON statement for the source dictionary and a PUNCH statement for the module. In the PUNCH statement, use the AS SYNTAX and TO SYSPCH clauses. Also, be sure to define the file as SYSPCH in the JCL.

At the end of this step, the external file contains only one statement: an ADD/MODIFY MODULE statement. Within the MODULE statement, however, the module source consists of the ADD or MODIFY statement for the individual components that you want to migrate.

4. Edit the external file as follows:
  - Insert a SIGNON statement for the target dictionary as the first statement
  - Insert the following statement after the SIGNON statement:

```
set options input 1 thru 80.
```
  - Insert an INCLUDE MODULE statement as the last statement.

As a result of the editing, the external file contains four statements:

- A SIGNON statement
- A SET OPTIONS statement

- An ADD MODULE or MODIFY MODULE statement
- An INCLUDE MODULE statement.

For example:

```

signon user dba password pass dictname target.
set option input 1 thru 80.
add module holdit
.
.
.
module source follows
add element emp-last-name
  version is 1
  pic is x(20)
.
.
.
msend.
include module holdit.

```

5. Execute the compiler in batch mode, using the edited file as input.

The compiler signs on to the target dictionary and adds or modifies the module. The INCLUDE statement brings the module source into the compiler's work file. The content adds or modifies the individual components to the target dictionary.

*Final tasks for schemas and load modules:* As with the DISPLAY statement, the PUNCH statement does not automatically validate the schemas or generate the load modules for subschemas and edit/code tables. To perform these function, use one of the methods described earlier in 24.6.1, “Using the DISPLAY statement” on page 24-17.

**Technique 3:** This technique combines parts of the Technique 2 presented above and parts of the online DISPLAY technique described earlier in 24.6.1, “Using the DISPLAY statement” on page 24-17. Because this technique entails an online migration, you need to moderate the volume of information you punch.

*Steps:* The steps in this technique follow:

1. In online mode, sign on to the source dictionary and create a module occurrence to hold the components to be moved.
2. While signed on to the source dictionary, punch the components to be moved to the module.

As above, direct the output of the punch to the module by including the TO MODULE clause in each PUNCH statement or in a SET OPTIONS statement. Also, specify the AS SYNTAX clause and the VERB ADD or VERB MODIFY clause, as appropriate.

3. Clear the compiler's work file.
4. Display the module.

This step brings the module (with all of its source) into the compiler's work file. In the DISPLAY statement, use the AS SYNTAX clause.

5. Edit the work file as follows:
  - Insert a SIGNON statement for the target dictionary as the first statement.
  - Insert an INCLUDE MODULE statement as the last statement.

This step prepares the work file for the task of compiling the module and then the components into the target dictionary. As a result of the editing, the work file contains three statements:

- A SIGNON statement
  - An ADD MODULE or MODIFY MODULE statement
  - An INCLUDE MODULE statement
6. Invoke the compiler

The compiler signs on to the target dictionary and adds or modifies the module. The INCLUDE statement brings the module source into the compiler's work file and executes the content of the work file. The content adds or modifies the individual components to the target dictionary.

*Final steps for schemas and load modules:* As with the other techniques, this technique does not automatically validate schemas or generate load modules for subschemas and edit/code tables. To perform these functions, use one of the methods described earlier in 24.6.1, "Using the DISPLAY statement" on page 24-17.

### 24.6.3 Using the mapping compiler and mapping utility

**Steps:** There are three steps to migrate maps between dictionaries (whether under the same CA-IDMS/DC or CA-IDMS/UCF system or not):

1. Decompile the maps from the source dictionary.

For this step, use the decompile function of the mapping utility (RHDCMPUT). You can decompile one or several maps in a single execution:

```
PROCESS=DECOMPILE
MAP=map1-name
MAP=map2-name
.
.
.
```

The output of the decompilation consists of the source form of the maps, typically stored in a temporary file.

2. Compile the maps into the target dictionary.

For this step, use the file of decompiled maps from the previous step as input to the mapping compiler (RHDCMAP1). The mapping compiler places a source description of the map in the DDL DML area of the target dictionary.

3. Generate the load modules for the maps in the target dictionary.

For this step, use either the online mapping facility or the load function of the mapping utility (RHDCMPUT). If you use the load function of the mapping utility, you can generate multiple load modules in a single execution:

```
PROCESS=LOAD
MAP=map1-name
MAP=map2-name
.
.
.
```

**Specify source and target dictionary:** The source and target dictionaries are typically part of multiple dictionary environments. Consequently, you must indicate which of the dictionaries the mapping compiler and mapping utility should run against. There are several techniques for specifying a particular dictionary in a multiple dictionary environment.

►► For more information on this, see Chapter 23, “Dictionaries and Runtime Environments” on page 23-1.

## 24.6.4 For SQL-defined entities

**SQL source cannot be displayed or punched:** The source definitions for SQL-defined data cannot be displayed or punched. Therefore, you must save the source DDL when you create the SQL definition by either:

- Submitting the statements in batch using IDMSBCF
- Including the statements in an OCF-language module

**Steps using a batch job stream:** To migrate SQL definitions using a batch job, you must have first *created* the definitions in the source dictionary using IDMSBCF. If you did, then:

1. Edit the batch file to connect to the target dictionary
2. Use the batch file as input to IDMSBCF

**Steps using an OCF-language module:** To migrate SQL definitions using an OCF-language module, you must have first *saved* the SQL DDL statements in an OCF-language module when you created the definitions in the source dictionary. If you did, then:

1. Retrieve the contents of the OCF-language module using the EDIT command.
2. Insert a CONNECT command at the top of the resulting work file that connects to the target dictionary.
3. Invoke the OCF compiler.

The compiler signs on the target dictionary and creates the SQL definitions.

►► For more information about OCF-language modules, refer to *CA-IDMS Command Facility*.

## 24.7 Additional considerations

**When to migrate:** You can perform most migration activities during regular working hours. Obviously, identifying, punching or decompiling components, and adding or modifying source definitions of components will not disturb programs or systems that are currently executing.

**Perform some tasks after system shutdown:** Depending on the specifics of the migration, you may not have to do any of it after regular working hours. To be on the safe side, however, you should plan to migrate or recompile load modules after the system has been shut down. You should also perform any restructuring operations on the production database after the system has been shut down. Note that if it is an SQL-defined database, the restructuring occurs immediately as part of the execution of the DDL statement that define the change. Therefore, you may want to delay execution of the DDL statements until system shut down.

**Making load modules available:** If you migrate or recompile new copies of existing load modules while the system is down, they automatically come into use when you bring the system back up. If you migrate or recompile existing load modules while the system is up, you can control the time at which the new load modules take effect through the NEW COPY option of the SYSTEM system generation statement or DCMT VARY PROGRAM command.

**NEW COPY options:** Using the NEW COPY option of the SYSTEM system generation statement, you can designate whether new load modules should be loaded automatically by the system or manually through explicit commands. If you choose to control loading manually, issue the DCMT VARY PROGRAM command with the NEW COPY option.

If you are migrating a new system whose tasks and programs are not enabled in the system generation, then you can migrate or recompile all of its load modules at any time. Access to the load modules will not be possible until the tasks and programs are enabled.

**Check your work:** When you have completed the mechanical migration of components, run a series of reports or issue a series of DISPLAY statements to check your work. However, to verify that the migration is complete and successful, you must test the new components in their new environment.

### 24.7.1 Additional tasks

**Updating system generation:** A new application may have an impact on system generation. Minimally, it may require a new task definition. Other system resources, such as program pool and storage pool space, may also need to be adjusted.

**Updating users:** New user IDs may have to be defined and existing user definitions reviewed.

**Updating the task application table:** If you choose to recreate and recompile an application structure in a target dictionary, the recompilation automatically updates the task application table (TAT) for that dictionary. If you choose simply to migrate the load module of an application structure, you must manually update the TAT for the target dictionary.

There are two utilities for updating the TAT:

- ADSOTATU works in online mode
- ADSOBTAT works in batch mode

►► For information on how to execute these utilities, refer to *CA-ADS Reference*.

**Backup the new files:** After you have migrated and tested the components, back up the files in the new environments.

**Cleanup:** The migration methods described throughout this chapter create copies of components. They do not physically move the components or automatically delete them from the source dictionary after the migration is complete.

If you decide to maintain a single copy of all components, you need to delete the unwanted copies. Be sure to delete all versions of both source definitions and load modules. Also be sure to delete copies of load modules from both the dictionary load areas and load libraries.

# Chapter 25. Modifying Physical Database Definitions

---

25.1	Modifications you can make	25-3
25.2	Making the changes available under the central version	25-7
25.3	Dynamic DMCL management	25-8
25.4	Changing a file's access method	25-10
25.4.1	Step 1: Expand the page size	25-10
25.4.2	Step 4: Copy the data to the new file	25-10
25.5	Increasing the size of an area	25-12
25.5.1	Increasing an area's page size	25-12
25.5.2	Extending an area's page range	25-13
25.6	Adding or dropping files associated with an area	25-14
25.7	Changing the size of a disk journal	25-15
25.8	Changing the access method of a disk journal	25-16
25.9	Related information	25-17



## 25.1 Modifications you can make

**Changes you can make and what to do:** The tables below summarize the changes you can make to physical database definitions and how to make the change. In most cases, all you need to do is:

- Alter the entity's definition
- Generate, punch, and link all DMCLs associated with the entity definition

Note, however, that if the entity is defined to the runtime DMCL, some changes affect how CA-IDMS/DB processes a request to make the modified DMCL available dynamically. The tables below identify those changes:

### Segment definition

Change you can make	How to make it
<ul style="list-style-type: none"> <li>▪ The schema reserved for defining tables and indexes within areas associated with the segment</li> <li>▪ The segment's page group</li> <li>▪ The maximum number of records or rows per page if the segment's area is empty</li> </ul>	Alter the segment's definition and generate, punch, and link all DMCLs to which the segment is defined
<ul style="list-style-type: none"> <li>▪ The maximum number of records or rows per page if the segment's area is not empty</li> </ul>	Unload and reload the segment, as described in <i>CA-IDMS Utilities</i>

### File definition

Change you can make	How to make it
<ul style="list-style-type: none"> <li>▪ The external file name</li> <li>▪ The file's allocation information, such as the data set name and disposition</li> </ul>	Alter the file's definition and generate, punch, and link all DMCLs in which the segment that contains the file is defined
<ul style="list-style-type: none"> <li>▪ The file's access method (VSAM or non-VSAM)</li> </ul>	See 25.4, "Changing a file's access method" on page 25-10 below

### Area definition

<b>Change you can make</b>	<b>How to make it</b>
To increase the size of an area, the options are:	
<ul style="list-style-type: none"> <li>■ Increase the area's page size</li> </ul>	See 25.5.1, "Increasing an area's page size" on page 25-12 below
<ul style="list-style-type: none"> <li>■ Extend the area's page range</li> </ul>	See 25.5.2, "Extending an area's page range" on page 25-13 below
<ul style="list-style-type: none"> <li>■ Change the primary number of pages assigned to the area's page range</li> </ul>	If the area is not empty, unload and reload the area as described in <i>CA-IDMS Utilities</i>
<ul style="list-style-type: none"> <li>■ Decrease the size of the area's pages</li> </ul>	If the area is not empty, unload and reload the area as described in <i>CA-IDMS Utilities</i>
<ul style="list-style-type: none"> <li>■ Increase or decrease the page reserve</li> </ul>	<ul style="list-style-type: none"> <li>■ Use an area override in the DMCL definition for special operations, such as loading a database; then remove the area override</li> <li>■ For permanent page reserve, alter the area definition; if the area is not empty, changing the page reserve affects only subsequent store and insert operations</li> <li>■ Generate, punch and link the DMCL(s) that contain the area override or the segment that contains the defined area</li> </ul>
<ul style="list-style-type: none"> <li>■ Add, modify, or drop a symbolic definition <sup>1</sup></li> </ul>	Alter the area's definition and generate, punch, and link all DMCLs in which the segment that contains the area is defined
<ul style="list-style-type: none"> <li>■ Re-assign the area to new or different files</li> </ul>	See 25.6, "Adding or dropping files associated with an area" on page 25-14

<sup>1</sup> If changing the page range of a subarea associated with a record in a non-empty area, unload and reload the area as described in *CA-IDMS Utilities*.

If changing the page range of a subarea associated with an index in a non-empty area, use the MAINTAIN INDEX utility statement to rebuild the index in the new page range as described in *CA-IDMS Utilities*.

### **DMCL definition**

<b>Change you can make</b>	<b>How to make it</b>
<ul style="list-style-type: none"> <li>■ Reassign the buffer associated with a file</li> <li>■ Associate or disassociate a database name table</li> <li>■ Add or remove a segment</li> <li>■ Change an area's startup or warmstart status</li> <li>■ Change an area's page reserve</li> <li>■ Change the external file name for a file</li> <li>■ Change the disposition for a file</li> <li>■ Change the dataspace usage for a file</li> <li>■ Change the shared cache assigned to a file</li> </ul>	Alter the DMCL definition and generate, punch, and link the DMCL

### Database buffer definitions

<b>Change you can make</b>	<b>How to make it</b>
<ul style="list-style-type: none"> <li>■ Change the buffer page size</li> <li>■ Change the buffer page count</li> <li>■ Change how the CA-IDMS/DB acquires storage for the buffer</li> <li>■ Add or remove buffers</li> </ul>	Alter the buffer definition and generate, punch, and link the DMCL with which the buffer is associated

### Journal buffer definition

<b>Change you can make</b>	<b>How to make it</b>
<ul style="list-style-type: none"> <li>■ Change the size of the journal buffer pages</li> </ul>	See 25.7, "Changing the size of a disk journal" on page 25-15
<ul style="list-style-type: none"> <li>■ Change the number of journal buffer pages</li> </ul>	Alter the definition of the journal buffer and generate, punch, and link the DMCL with which the journal buffer is associated

---

### Disk journal definition

Change you can make	How to make it
<ul style="list-style-type: none"> <li>▪ Change the external file name</li> <li>▪ Change the file's dataspace usage</li> </ul>	Alter the definition of the disk journal and generate, punch, and link the DMCL with which the disk journal file is associated
<ul style="list-style-type: none"> <li>▪ Change the number of pages in the disk journal file</li> </ul>	See 25.7, "Changing the size of a disk journal" on page 25-15 below
<ul style="list-style-type: none"> <li>▪ Change the file's access method</li> </ul>	See 25.8, "Changing the access method of a disk journal" on page 25-16 below

### Archive journal definition

Change you can make	How to make it
<ul style="list-style-type: none"> <li>▪ Change the file's block size</li> <li>▪ Change the file's external file name</li> <li>▪ Add or remove archive journal files</li> </ul>	Alter the archive file's definition and generate, punch, and link the DMCL with which the archive file is associated

### Tape journal definition

Change you can make	How to make it
<ul style="list-style-type: none"> <li>▪ Change the file's external file name</li> </ul>	Alter the tape file's definition and generate, punch, and link the DMCL with which the tape journal file is associated

### Changes you can't make:

- The synchronization stamp level associated with an area
- The segment's type (that is, SQL or NONSQL)
- The name of a segment containing SQL tables

## 25.2 Making the changes available under the central version

**Journal modifications require system to be recycled:** If you change the page size of the journal buffer or any disk journal attribute, you have to recycle the system in order to make the changes available under the central version. CA-IDMS/DB cannot implement the changes if you make the DMCL available dynamically by issuing a DCMT VARY DMCL command. Note that you can make changes to the definition of the archive journal without recycling the system.

**Other changes can be accessed dynamically:** Other changes made to the DMCL definition can be made effective by issuing a DCMT VARY DMCL NEW COPY command, provided that files can be deallocated and reallocated if necessary. The ability to deallocate and reallocate files dynamically depends on the operating system and the information provided in the file definition.

►► For more information, see Chapter 3, “Defining Segments, Files, and Areas” on page 3-1.

**Backup old DMCL:** If using VARY DMCL to implement your changes, be sure to make a copy of the old DMCL load module before issuing the VARY DMCL command. This ensures that if an abnormal termination occurs before the operation is complete, you will be able to warmstart the system using the old DMCL if necessary.

**Data sharing considerations:** In a data sharing environment, most changes to an area or its associated files will not take effect until the area is varied offline in all group members in which it is shared since most area (and associated file) characteristics must be identical across all sharing members. For a list of these characteristics, see 4.5.4, “Sharing update access to data” on page 4-15.

The recommended procedure for making shared area or file changes in the following:

- Modify and generate a new DMCL for all affected members
- Vary the area offline in all sharing members
- Vary a new copy of the altered DMCL in all affected members
- Vary the area online in all affected members

►► For more information on data sharing, refer to *CA-IDMS System Operations*.

## 25.3 Dynamic DMCL management

**Impact of changes:** When a DMCL is being varied, certain changes cause:

- Areas to be quiesced
- Files to be deallocated and reallocated

Change	Quiesce area?	Reallocate file?
Segment changes		
▪ Dropping and recreating the segment	Yes	Yes
▪ Page group	Yes	Yes
▪ Maximum number of records per page	Yes	Yes
▪ Segment's schema	No	No
Area changes		
▪ Adding an area		Allocate
▪ Dropping an area	Yes	Deallocate
▪ Primary page range	Yes	Yes
▪ Extending page range	Yes	Yes
▪ Page size	Yes	Yes
▪ Original page size	Yes	Yes
▪ Symbolic parameters	Yes	Yes
▪ Area-to-file mapping	Yes	Yes
▪ Page reserve	No	No
▪ Maximum space	No	No
File changes		
▪ Dataset name	Yes	Yes
▪ VM/ESA user id/virtual address	Yes	Yes
▪ Access method	Yes	Yes
▪ Disposition	No	Yes
▪ External name (DDNAME)	No	Yes
DMCL changes		
▪ Adding a segment		Allocate

---

<b>Change</b>	<b>Quiesce area?</b>	<b>Reallocate file?</b>
▪ Dropping a segment	Yes	Deallocate
▪ Buffer associated with a file	No <sup>1</sup>	No
▪ Dataspace usage for a file	No	Yes
▪ File's external name (DDNAME)	No	Yes
▪ File's disposition	No	Yes
▪ Area status	No	No
▪ Shared cache for a file	No	Yes

<sup>1</sup> If a file is associated with a new buffer, the area's pages are first purged from the buffer pool.

---

### Considerations

- Changing the page size of a buffer causes the buffer to be closed and re-opened with the new size. All other buffer changes (such as the number of pages) are ignored. To change these parameters while the system is active, issue a DCMT VARY BUFFER command.
- Changes to a journal buffer or disk, tape, or archive journal files either have no impact on the runtime system or are not allowed when varying a new copy of a DMCL.
- In a data sharing environment, if an area is shared, most changes to the area and its associated files will not take effect until the area is varied offline in all group members in which it is shared.

## 25.4 Changing a file's access method

**Procedure:** You can change the format of database files from non-VSAM to VSAM and vice versa. To complete this process, you need to:

1. Expand the page size of the file's area, if necessary
2. Alter the file definition to change its access method (and optionally to specify a new database name or other location information) and generate, punch, and link all DMCLs in which the file's segment is included.
3. Allocate a new VSAM or non-VSAM data set, as described in Chapter 16, "Allocating and Formatting Files" on page 16-1.
4. Make the area to be processed unavailable for update under the central version.
5. Copy the old VSAM or non-VSAM file to the new data set.
6. Make the new DMCLs and file available to the runtime environment.

Steps 1 and 4 are discussed below.

### 25.4.1 Step 1: Expand the page size

**Converting from non-VSAM to VSAM:** When you convert a non-VSAM file to VSAM, expand the area's page size first if the page size of the area is significantly smaller than the size of the VSAM control interval. The optimal page size is 8 bytes less than the VSAM control interval.

**Converting from VSAM to non-VSAM:** When you convert a VSAM file to non-VSAM, consider expanding the area's page size either before or after the conversion if the page size of the area is inefficient for the device type.

►► For optimal page sizes based on device type, refer to *CA-IDMS Database Design*. For the steps involved in expanding the page size of an area, see 25.5.1, "Increasing an area's page size" on page 25-12 later in this chapter.

### 25.4.2 Step 4: Copy the data to the new file

**Options:** To copy the data, use one of the following options:

1. Use the BACKUP and RESTORE utility statements
2. Use the IDCAMS utility

**Option 1: Backup and restore:** To use BACKUP and RESTORE to copy the database files, take the following steps:

1. Offload the data in the old file(s) using the BACKUP utility statement and the old DMCL. If all files within a multi-file area are being converted, use the AREA option on the BACKUP statement; otherwise, use the FILE option.

2. If the backup was performed with the AREA option, format the new files before executing Step 3.
3. Reload the data into the new file(s) using the RESTORE utility statement and the new DMCL. If the data was offloaded with the AREA option, restore with the AREA option; otherwise, restore with the FILE option.

**Option 2: Using IDCAMS:** The REPRO command of the IDCAMS utility can be used to copy the data between a VSAM and non-VSAM file and vice versa. If you use this approach, be sure to copy all pages (blocks) in the file in their entirety without reblocking.

►► For more information about IDCAMS, refer to the appropriate IBM documentation.

## 25.5 Increasing the size of an area

**Available options:** To increase the size of an area, you can:

1. Increase the page size of the area by using the EXPAND PAGE utility statement
2. Extend the number of pages in the area by using the EXTEND SPACE clause of the AREA statement
3. Increase the current number of pages assigned to the area by unloading and reloading the area

**Which option to use:** Both options 1 and 3 distribute free space throughout an area. While option 1 is faster (and therefore less disruptive) than option 3, it does not reorganize indexes or improve the placement of existing data which may have overflowed due to lack of space on a page. Option 1 is most effective if used before the area approaches a full condition.

Option 2 adds free space only at the end of an area. This can be useful where records or tables have a location mode of direct or are clustered around a dbkey index or an OOK record. It can also be used as a temporary means of increasing space in an area whose page size cannot be increased (due to device or VSAM restrictions).

If the area to be extended contains CALC records, these records will continue to only target to pages in the original page range. If no space is available to hold the new occurrences, they will overflow into the extended page range. The area must continue to be defined as being extended until an UNLOAD/RELOAD is performed where the new database defines the entire extended page range as the original page range. Failure to do this will result in 0326 errors when CALC retrieval is attempted.

**Procedures:** Procedures for the first two options follow.

►► For information about unloading and reloading an area, refer to *CA-IDMS Utilities*.

### 25.5.1 Increasing an area's page size

**Steps:** To increase the page size for an area, follow these steps:

1. Change the definition of the area by specifying the new page size and, if this is the first time the pages have been expanded, specify the current page size as the original page size. (The original page size must be the size of the page at the time the area was formatted.)
2. If desired, alter the definitions of the area's files to specify new dataset names and/or other location information.
3. Generate, punch, and link all DMCLs that contain the segment with which the area is associated.
4. Allocate new database files to accommodate the increased page size.

5. Make the area to be processed unavailable for update under the central version.
6. Copy and expand the files associated with the area by using the EXPAND PAGE utility statement and the old DMCL. Each file must be expanded individually.
7. Backup the expanded area.
8. Make the DMCLs and the new files available to the runtime environment.

## 25.5.2 Extending an area's page range

**Steps:** To extend the number of pages in an area, follow these steps:

1. If the additional pages being added to the area will reside in a new file, define the file. If the additional pages are being added to the last file associated with the area, alter the definition of the file to specify a new dataset name and/or other location information, if appropriate.
2. Change the definition of the area specifying the number of additional pages to add to the area by using the EXTEND SPACE clause. On the EXTEND SPACE clause, specify to which file the additional pages will be mapped by using the WITHIN FILE clause.

If the additional pages would cause the number of pages in the area to exceed the maximum space allowed, you can use the MAXIMUM SPACE clause to increase the maximum provided the page numbers are not assigned to another area that will be used in the same DMCL as the area being expanded. (If the pages have been assigned, you must use UNLOAD and RELOAD to increase the area's page range.)

3. Generate, punch, and link all DMCLs that contain the segment with which the area is associated.
4. Allocate a new database file to contain the additional pages and initialize the file using the new DMCL.
5. If the new pages are being added to the last file of the area:
  - a. Make the area to be processed unavailable for update under the central version.
  - b. Backup the area using the old DMCL.
  - c. Restore the area using the *old* DMCL, but referencing the *new* file through JCL statements.

**Note:** If the area maps to its file on a one-to-one basis it is necessary to include IDMSQSAM=ON in the RESTORE utility's SYSIDMS file.

6. Backup the expanded area.
7. Make the DMCLs and the new file available to the runtime environment.

## 25.6 Adding or dropping files associated with an area

**Types of changes:** The pages of an area can be mapped to different files provided that all the pages are accounted for. For example, two files can be combined into one file or one file can be separated into multiple files.

**Steps:** To add or remove files from an area, follow these steps:

1. Define the new files.
2. Change the definition of the area by excluding all files associated with the area and re-assigning the pages of the area to file blocks.
3. Drop all unused files.
4. Generate, punch and link all DMCLs that contain the segment with which the area is associated.
5. Allocate and format new database files.
6. Make the area to be processed unavailable for update under the central version. (If re-using some of the existing files, take the area offline to the central version.)
7. Backup the area using the AREA option of the BACKUP utility statement and the old DMCL.
8. Restore the area using the AREA option of the RESTORE utility statement and the new DMCL.
9. Make the DMCLs and the new files available to the runtime environment.

## 25.7 Changing the size of a disk journal

**Steps:** To change the size of a disk journal, follow these steps:

1. Change the size of the disk journal by either changing the size of the journal buffer page or the number of pages in the disk journal file.
2. Generate, punch, and link the DMCL.
3. Shut down the system.
4. Offload all currently used journals using the ARCHIVE JOURNAL utility statement with the ALL option and the old DMCL.
5. Allocate and format new disk journal files.
6. Restart the system with the new DMCL and the new journal files.

## 25.8 Changing the access method of a disk journal

**Steps:** You can change the access method used for a disk journal file from non-VSAM to VSAM or vice versa. To do this you must:

1. Change the definition of the disk journal file specifying the desired access method. Alter the page size of the journal buffer:
  - If changing from non-VSAM to VSAM, the page size should be 8 bytes less than the control interval size
  - If changing from VSAM to non-VSAM, choose an optional page size for the device type
2. Generate, punch, and link the DMCL.
3. Shut down the system.
4. Offload all currently used journals using the ARCHIVE JOURNAL utility statement with the ALL option and the old DMCL.
5. Allocate and format new disk journal files.
6. Restart the system with the new DMCL and the new journal files.

## 25.9 Related information

- About segment, area, and file definition, see Chapter 3, “Defining Segments, Files, and Areas” on page 3-1
- About DMCL, database buffer, journal buffer, and journal file definition, see Chapter 4, “Defining, Generating, and Punching a DMCL” on page 4-1
- About the syntax for physical database entities, see Chapter 6, “Physical Database DDL Statements” on page 6-1
- About DCMT commands, refer to *CA-IDMS System Tasks and Operator Commands*
- About utility statement syntax, refer to *CA-IDMS Utilities*
- About data sharing, refer to *CA-IDMS System Operations*



# Chapter 26. Modifying Database Name Tables

---

- 26.1 Changes you can make . . . . . 26-3
- 26.2 Procedure for modifying database name tables . . . . . 26-4
- 26.3 Related information . . . . . 26-5



## 26.1 Changes you can make

**What you can change:** You can modify the following characteristics of a database name table definition:

- What databases are associated with the database name table (through the DBNAME statement)
- What segments and/or subschema mappings are associated with a database name
- Generic subschema mappings defined to the database name table
- The MIXED PAGE GROUP BINDS option setting
- What database groups are associated with the database name table (through the DBGROUP statement)

## 26.2 Procedure for modifying database name tables

**Steps:** To modify a database name table, follow these steps:

Action	Statement
Modify the database name, database group, and/or database name table	<ul style="list-style-type: none"> <li>■ CREATE, ALTER, or DROP DBNAME</li> <li>■ CREATE, ALTER, or DROP DBGROUP</li> <li>■ ALTER DBTABLE</li> </ul>
Regenerate the database name table	GENERATE DBTABLE
Punch and link the database name table to a load library	PUNCH DBTABLE LOAD MODULE
Make the database name table available under the central version	DCMT VARY DBTABLE NEW COPY

**Example:** In the example below, the DBA adds a new database name to an existing database name table. After generating and punching the database name table load module, the DBA instructs CA-IDMS/DB to load the updated database name table:

```
create dbname allpbs.benefits
  add segment empseg
  add segment projseg
  add segment beneseg;

generate dbtable allpbs;

punch dbtable load module allpbs;
```

After link-editing the modified database name table to a load library, make it available under the central version:

```
dcmt vary dbtable allpbs new copy
```

## 26.3 Related information

- About defining database name tables and database names, see Chapter 5, “Defining a Database Name Table” on page 5-1
- For syntax and syntax rules for the DBTABLE, DBGROUP, and DBNAME statements, see Chapter 6, “Physical Database DDL Statements” on page 6-1
- About DCMT commands, refer to *CA-IDMS System Tasks and Operator Commands*
- About the PUNCH utility statement, refer to *CA-IDMS Utilities*
- About database groups and dynamic routing, refer to *CA-IDMS System Operations*



# Chapter 27. About Modifying SQL-Defined Databases

---

- 27.1 What you can modify . . . . . 27-3
- 27.2 Methods for modifying . . . . . 27-4



## 27.1 What you can modify

You can modify an SQL-defined database by:

- Adding or dropping tables
- Modifying table components
- Adding or dropping indexes and referential constraints
- Adding, modifying, or dropping schemas
- Adding or dropping views

▶▶ See Chapter 25, “Modifying Physical Database Definitions” on page 25-1 for maintaining physical definitions.

## 27.2 Methods for modifying

You can use the following methods to change an SQL-defined database:

- Single DDL statement

You use a single DDL statement to make the change. The change takes effect immediately. For example, you use a single DDL statement when adding a check constraint.

- Multiple DDL statements

You use multiple DDL statements to make the change. The particular SQL DDL statements you use depend on the type of change being made. For example, to change index characteristics (such as the area in which an index resides) requires the following SQL statements:

- DROP INDEX
- CREATE INDEX

The change takes effect upon completion of these statements.

- Combination of DML and DDL statements

You use a combination of DML and DDL statements to modify a definition. This method often involves dropping, redefining, and reloading a table to make the change.

Once the data has been reloaded, the change takes effect. For example, to drop a column from a table, you use DML or utility statements to:

1. Create a new table with the appropriate columns (DDL CREATE)
2. Copy the rows of data to the new table (DML INSERT)
3. Delete the existing table (DDL DROP)

**Choosing a modification method:** In some cases, you may choose the method to use. In other cases, the method is dictated by database factors such as whether the table contains data or whether it participates in a referential constraint.

Each modification is discussed in detail in the following chapters.

**Inform your users:** Some changes you make to the database will have a direct impact on your users. For example, if you drop a table or a view, users will no longer have access to the data.

Before you make a change such as dropping a table, you can use SELECT statements to determine where the entity to be changed is used. Specifically, look for:

- Views that reference the table
- Referential constraints in which the table participates
- Access modules that access the table

This indicates the potential impact the change may have and provides information on determining the best method to use to make the change.



# Chapter 28. Modifying Schema, View, and Table Definitions

---

28.1	Maintaining schemas	28-4
28.1.1	Dropping an existing schema	28-4
28.1.2	Modifying a schema	28-4
28.2	Maintaining views	28-5
28.2.1	Dropping a view	28-5
28.2.2	Modifying a view	28-5
28.3	Maintaining tables	28-7
28.3.1	Creating a table	28-7
28.3.2	Dropping a table	28-7
28.3.3	Adding a column to a table	28-8
28.3.4	Dropping a column from a table	28-9
28.3.5	Changing the characteristics of a column	28-10
28.3.6	Adding or removing data compression	28-10
28.3.7	Adding a new check constraint	28-10
28.3.8	Dropping a check constraint	28-11
28.3.9	Modifying a check constraint	28-11
28.3.10	Revising the estimated row count for a table	28-11
28.3.11	Changing a table's area	28-12
28.3.12	Dropping the default index associated with a table	28-12
28.4	Dropping and recreating a table	28-14
28.4.1	Method 1 — Using DDL and DML statements	28-14
28.4.2	Method 2 — Using DDL and utility statements	28-16



---

This chapter describes methods for creating, dropping, and changing schemas, views, and tables.

►► For more information on the SQL DDL statements used in the procedures in this chapter, refer to the *CA-IDMS SQL Reference*.

## 28.1 Maintaining schemas

This section describes how to:

- Drop a schema
- Change a component of a schema

### 28.1.1 Dropping an existing schema

**DROP SCHEMA statement:** To drop a schema, use an SQL DDL DROP SCHEMA statement. This removes the named schema only if no tables or views are associated with it.

**CASCADE option:** If you specify the CASCADE option, you also delete:

- The definition of each table and view associated with the named schema
- The data stored in each table associated with the schema
- The definition of each referential constraint, index, and CALC key defined on the tables associated with the named schema
- The view definition of each view derived from one or more of the tables associated with the named schema
- All privileges granted on tables dropped as a result of cascade processing

**Considerations:** If all tables and indexes on those tables are in a segment in which no other table or index from another schema resides, then you can use the FORMAT utility to erase rows and indexes before using DROP SCHEMA. This will enable more efficient execution.

**Example:** In the following example, a schema and its associated tables are dropped.  
drop schema demoempl cascade;

### 28.1.2 Modifying a schema

To modify a schema, use the SQL DDL ALTER SCHEMA statement.

**Considerations:** Changing the default area associated with the schema does not affect existing tables.

**Example:** In the following example, the schema's default area is changed.

```
alter schema demoempl
  default area demoempl.emplarea;
```

---

## 28.2 Maintaining views

This section describes how to:

- Drop a view
- Change a view definition by dropping and recreating it

### 28.2.1 Dropping a view

**DROP VIEW statement:** To drop a view, use the SQL DDL DROP VIEW statement.

**CASCADE option:** Use the CASCADE option if the view being dropped participates in any other view definitions. CASCADE directs CA-IDMS/DB to drop the named view and all views derived from the named view.

When you drop a view (*without* CASCADE), the following definitions are removed from the dictionary:

- The view
- All privileges granted on the view

If you specify CASCADE, these additional definitions are removed from the dictionary:

- All views in which the view is referenced and all views referencing those views
- All privileges granted on views dropped as a result of cascade processing

**Considerations:** You must specify CASCADE if there are views defined on the view you are dropping.

**Example:** In the following example, the view EMP\_HOME\_INFO is dropped. This also drops any views derived from this view.

```
drop view emp_home_info cascade;
```

### 28.2.2 Modifying a view

To modify a view, use the SQL DDL DROP VIEW statement to drop the view and then use the SQL DDL CREATE VIEW statement to re-add the view.

Before modifying a view, you can use the SELECT SYNTAX FROM SYSCA.SYNTAX statement to display the syntax used to create a view.

```
select syntax from sysca.syntax
  where schema=HR
  and table=EMP-SALARY;
```

►► For more information on SELECT SYNTAX, refer to the *CA-IDMS SQL Reference*.

**Example:** In the following example, the syntax for the view EMP\_HOME\_INFO is displayed using the SELECT SYNTAX statement. The view is then dropped (DROP) and re-added (CREATE) with an additional column (CITY).

This SELECT SYNTAX statement:

```
select syntax from sysca.syntax
  where schema=demoempl
  and table=emp_home_info;
```

Displays this view syntax:

```
create view emp_home_info
  as select emp_id, emp_lname, emp_fname, phone
  from employee;
```

DROP VIEW AND CREATE VIEW are used to modify the view.

```
drop view emp_home_info;
create view emp_home_info
  as select emp_id, emp_lname, emp_fname, phone, city
  from employee;
```

## 28.3 Maintaining tables

This section describes how to:

- Create or drop a table
- Create or drop a column
- Change column characteristics
- Add or remove data compression
- Create, drop, or modify check constraints
- Revise the table's estimated row count
- Change the table's area
- Drop the default index associated with the table

### 28.3.1 Creating a table

**CREATE TABLE statement:** To create a table, use the SQL DDL CREATE TABLE statement.

**Considerations:** The area in which the table's rows are to reside must be defined in the application dictionary and be accessible to the runtime environment in which the CREATE TABLE statement is issued.

### 28.3.2 Dropping a table

**DROP TABLE statement:** To drop a table, use the SQL DDL DROP TABLE statement. Use the CASCADE option if the table participates in a referential constraint or is referenced in one or more view definitions.

**No CASCADE:** When you drop a table (*without* CASCADE), the following definitions are removed from the dictionary:

- The table
- Its CALC key (if any)
- All indexes defined on the table
- All privileges granted on the table

Table rows and indexes are removed from the database.

**With CASCADE:** If you specify CASCADE, these additional definitions are removed from the dictionary:

- All referential constraints in which the table participates
- All views in which the table is referenced and all views referencing those views
- All privileges granted on views dropped as a result of cascade processing

### Considerations

*Using FORMAT to erase table rows:* If the table you want to drop is the only table in an area and its indexes (if any) also reside in areas in which no other table or index resides, you can use the FORMAT utility to drop the table more efficiently:

1. Format the area(s) containing the table and indexes
2. Drop the table

►► For information on FORMAT, refer to the *CA-IDMS Utilities* document.

*Dropping all tables in a schema:* If you want to drop all tables in a schema, use the DROP SCHEMA statement with the CASCADE option rather than dropping each table individually.

**Example:** In the following example, these entities are dropped: the BENEFITS table, its CALC key, all indexes defined on it, all privileges on it, all referential constraints in which BENEFITS participates, all views in which this table is referenced and all views referencing that view, and all privileges granted on all those views. In addition, all data will be deleted.

```
drop table demoempl.benefits cascade;
```

## 28.3.3 Adding a column to a table

**ALTER TABLE statement:** To add a column to a table, use the SQL DDL ALTER TABLE statement with the ADD COLUMN option.

The definition of the table is updated to include the new column definition, and the new column becomes the last column in the table. Table rows are not updated as part of the ALTER TABLE processing; instead, the column is added to an existing row only when that row is next updated.

### Considerations

*If the table is not empty:* If the table is not empty, you must supply a default value for the added column. You do this one of the following ways:

- By specifying that the column is to have a default value, in which case all existing rows are considered to have the default value for the new column
- By allowing the column to have a null value, in which case all existing rows are considered to have a null value for the column

►► For more information about choosing a value, refer to the *CA-IDMS SQL Reference*.

---

*Maximum row length:* Adding a column to a table might increase the length of the table row beyond the maximum allowed.

For compressed tables, the maximum is 32760. If the new column would cause this to be exceeded, the column cannot be added to the table; instead, consider creating a second table to hold the additional information.

For uncompressed tables, the maximum depends on the page size of the area in which the table resides. If the new column would cause the length of the row to be greater than (page size - 40), then do one of the following:

1. Use the EXPAND PAGE utility statement to increase the page size of the areas
  - ▶▶ For information on EXPAND PAGE, refer to the *CA-IDMS Utilities* document.
2. Compress the table
3. Create a second table to hold the new information

**Note:** The maximum length of an uncompressed row can be as much as (page size - 40); however, it is recommended that row lengths be no more than 30% of the size of the page.

*Expanding space in an area:* If an area is becoming full, consider expanding its space before adding the column. Chapter 25, “Modifying Physical Database Definitions” on page 25-1 describes methods you can use to expand an area.

*Compressed records:* If a new column in a compressed table will be used as an index key or as a referencing column, consider placing the column near the front of the table. Otherwise, the compression potential of the table will be greatly reduced.

To do this, the table must be dropped and re-added with a new column order. When you put the rows back into the table, make sure the data is in the new column order.

*Effect on programs and view definitions:* Adding a column to a table does not impact existing programs or view definitions except under the following circumstances:

- If your host language programs include SELECT \* from the table, they will receive runtime errors because of the added column
- If a view definition includes a SELECT \* from the affected table, it becomes invalid and must be dropped and recreated.

### 28.3.4 Dropping a column from a table

**Drop/add table:** In order to delete a column from a table, you must drop the table and redefine it without the column.

▶▶ See 28.4, “Dropping and recreating a table” on page 28-14 later in this chapter for the steps and considerations involved with this process.

**Considerations:** All programs referencing the column must be recompiled and all views referencing the column must be recreated.

### 28.3.5 Changing the characteristics of a column

**Drop/add table:** Characteristics of a column include data type, null option, and default value. Changing any of these characteristics requires that the table be dropped and redefined, as described in 28.4, “Dropping and recreating a table” on page 28-14 later in this chapter. When redefining the table, make the necessary changes to the column definition.

#### Considerations

- You cannot remove the NOT NULL attribute if the column participates in a unique CALC key, index, or ORDER BY clause of a referential constraint
- If you change the data type or length of a column that participates in a referential constraint, the change must be reflected in both the referenced and referencing columns.

### 28.3.6 Adding or removing data compression

**Drop/add table:** To add or remove compression, you must drop and redefine the table, as described in 28.4, “Dropping and recreating a table” on page 28-14 later in this chapter. When redefining the table, add or remove the COMPRESS clause as desired.

#### Considerations

- By removing compression, the table will occupy more space in the database and may overflow a database that is already near capacity
- By adding compression, you may incur a modest increase in CPU time during subsequent DML processing of the table

►► For more information about data compression, refer to the *CA-IDMS Presspack User Guide*.

### 28.3.7 Adding a new check constraint

**ALTER TABLE statement:** To add a new check constraint, use the SQL DDL ALTER TABLE statement with the ADD CHECK option.

#### Considerations

- Adding a check constraint will *append* the new check constraint to any check constraints currently on the table
- If current data does not conform to the new check constraint, you will receive an error when CA-IDMS/DB processes the ALTER TABLE command

**Example:** In the following example, a new check constraint is added to the BENEFITS table.

```
alter table emp.benefits
  add check ( fiscal_year > 1920 );
```

### 28.3.8 Dropping a check constraint

**ALTER TABLE statement:** To drop a check constraint, use the SQL DDL ALTER TABLE statement with the DROP CHECK option. DROP CHECK deletes *all* check constraints associated with the table.

**Example:** In the following example, all check constraints associated with the BENEFITS table are dropped.

```
alter table emp.benefits
  drop check;
```

### 28.3.9 Modifying a check constraint

To modify a check constraint, follow these steps:

1. Drop the existing check constraint, as described above
2. Add the new check constraint, as described above

**Tip:** Use SELECT SYNTAX from SYSCA.SYNTAX to display the existing check constraints before dropping it:

```
select syntax from sysca.syntax
  where schema='EMP' and
        table = 'BENEFITS';
```

**Example:**

```
alter table emp.benefits
  drop check;

alter table emp.benefits
  add check ( fiscal_year > 1930 );
```

### 28.3.10 Revising the estimated row count for a table

**ALTER TABLE statement:** To change the estimated row count on the table definition, use the SQL DDL ALTER TABLE statement with the ESTIMATED NUMBER OF ROWS option.

#### Considerations

- Changing the estimated number of rows for a table will not affect default index sizing unless you drop and re-add the index or referential constraint. The estimated number of rows is used for index calculations only if it is greater than the NUMROWS column in SYSCA.TABLE. NUMROWS is updated whenever an UPDATE STATISTICS utility statement is issued for the table or the table's area.

►► For more information about index calculations, refer to the *CA-IDMS SQL Reference*

- Changing the estimated row count may affect the access paths chosen by the access module compiler for SQL DML statements that reference the table. Unlike other table modifications, though, changing the estimated row count will not cause existing access modules that reference the table to be automatically recompiled. If recompilation of selected access modules is desired, you must use the ALTER ACCESS MODULE statement to force reoptimization.

**Note:** Estimated number of rows is used for optimized purposes only if the NUMROWS column of SYSCA.TABLE is 0.

**Example:** In the following example, the estimated row count for the EMPLOYEE table is revised.

```
alter table emp_employee
  estimated row count 750000;
```

### 28.3.11 Changing a table's area

**Drop/add table:** To change the area in which the rows of a table are stored, you must drop the table and redefine it specifying the new area.

►► See 28.4, “Dropping and recreating a table” on page 28-14 later in this chapter for the steps and considerations involved in this process.

### 28.3.12 Dropping the default index associated with a table

**ALTER TABLE statement:** To drop the default index associated with a table, use the SQL DDL ALTER TABLE statement with the DROP DEFAULT INDEX option.

#### Considerations

- Do not drop the default index on a table until the CALC key, indexes, and referential constraints in which the table participates have been defined. If no other index exists on the table, an area sweep will be initiated each time one of the above components is defined.
- Dropping the default index could change the location mode of a record.
- Default indexes can be useful whenever it is anticipated that a table will be accessed without WHERE clauses specifying index or CALC keys and without joins that might use referential relationships with other tables. In short, they are useful whenever it is anticipated that the optimizer would otherwise choose area sweeps to satisfy access requests on the table. This is particularly true when it is a sparse table, since a sweep of the default index will only access data pages that contain rows of the table; whereas, an area sweep will access every page of the area.

►► For complete information on when you would choose to drop the default index, refer to the *CA-IDMS Database Design* document.

**Example:** In the following example, the default index for the EMPLOYEE table is dropped.

```
alter table emp.employee  
  drop default index;
```

## 28.4 Dropping and recreating a table

**Considerations for dropping/adding a table:** Many types of changes can only be implemented by dropping and redefining a table. There are two major considerations involved with this process:

- Preserving the table's data
- Re-establishing the table's relationships with other tables and views

**What follows:** This section outlines two approaches that can be used to drop and recreate a table:

- Method 1 — Uses a combination of DDL and DML statements to perform the operation
- Method 2 — Uses DDL and utility statements

**Considerations:** Select the approach based on the size of the table and the importance of minimizing the time during which the table cannot be accessed. Consider the following:

- Method 1 requires there be enough space in the database to hold two copies of the data simultaneously. It also builds indexes and validates relationships as the data is being inserted into a new table, potentially requiring a large number of row locks and journal images.
- Method 2 reloads the data in local mode using the LOAD utility statement. Therefore the table and all other tables in the same area cannot be accessed while the load is taking place.

For these reasons, Method 1 is more appropriate for small tables, while Method 2 is more suited for large tables.

### 28.4.1 Method 1 — Using DDL and DML statements

**Steps:** To use a combination of DDL and DML statements to recreate a table, follow these steps:

1. Define a new table that has the same definition as the original table except for the desired changes.
2. Define the same indexes and CALC keys for the new table as for the old (unless changes in these are desired).
3. For each referential constraint in which the original table is the **referencing** table, define a similar constraint on the new table. The new constraint must be defined with a different name and if the referenced table is not empty, it must be defined as unlinked. (The unlinked constraint may also require that an index be defined on the foreign key of the new table).
4. For each referential constraint in which the original table is the **referenced** table, determine if the referencing table is empty. If it is, define a similar constraint

with a different name in which the new table is the referenced table. If the referencing table is not empty, determine if additional indexes are needed on the foreign key of the referencing table to support a similar constraint defined as unlinked. If additional indexes are required, create them now.

5. For each view in which the original table is referenced (or views of those views), display the definition syntax by selecting from SYSCA.SYNTAX. Save the resulting output so the views can be recreated later.
6. Copy the data from the original table to the new table using an INSERT statement with the SELECT option.
7. For each referential constraint in which the original table is the **referenced** table and the referencing table is not empty, define a constraint in which the new table is the referenced table. The new constraint must have a different name and be defined as unlinked.
8. Drop the original table using the CASCADE option of DROP table.
9. For each **self-referencing constraint** defined on the original table, define a similar constraint on the new table. (A self-referencing constraint is a referential constraint in which the referenced and referencing table are the same.)
10. Complete the transition to the new table as follows:
  - Define a view on the new table with the same name as the original table and including all of its columns.
  - Recreate the views whose syntax was previously saved; examine those view definitions to see if changes are required.
  - Re-specify privilege definitions on the individual table and views if access is controlled through CA-IDMS internal security.

**Guaranteeing integrity of the data:** Steps 6 through 8 should be performed within a single transaction to minimize the potential of changes to the data in the original table and any of its related tables until the entire operation is completed. To *ensure* that no changes are made between the time the data is copied and the time the table is dropped, take one of the following actions just prior to issuing the SELECT statement:

- Prohibit access to the table by explicitly dropping all views that reference it. This is effective only if all update access to the table is done through a view.
- Revoke all INSERT, UPDATE, and DELETE privileges from the table (and any matching wildcarded table names) if access is controlled through CA-IDMS internal security.
- Alter the original table and add a dummy column. This has the effect of prohibiting access to the table until the transaction has terminated.

**Recreating empty tables:** If the table to be recreated is empty, you need not define a new table. Instead, simply drop and redefine the table making the desired changes to its definition. However, be sure to take appropriate steps to preserve referential constraints, views derived from the table, and privilege definitions.

## 28.4.2 Method 2 — Using DDL and utility statements

**Steps:** To use a combination of DDL and utility statements to drop and recreate a table, take the following steps:

1. Identify all tables related through a linked constraint to the target table (the table whose definition is to be changed). Either the related tables must be unloaded and reloaded together with the target table or the constraints will become unlinked when they are redefined.
2. For each view in which the target table is referenced (or views of those views), display the definition syntax by selecting from SYSCA.SYNTAX. Save the resulting output so the views can be recreated later.
3. For each table to be unloaded, extract the data to a sequential file using either:
  - A user-written program
  - A CA-CULPRIT report

Use separate extract files for each table or place an indicator in each output record to identify the table from which the data was extracted. Be sure the data was extracted successfully before proceeding to the next step.

4. Drop the target table (specifying the CASCADE option) and delete the rows from the related tables that were unloaded by using a DELETE statement. If no other tables or indexes exist within the affected areas and all relationships are within those areas (and were unloaded), format the area before issuing the DROP and DELETE statements. Be sure to vary the areas offline to the DC/UCF system before formatting them.
5. Redefine the table making any necessary changes.
6. Redefine the indexes and CALC key on the target table.
7. Redefine the referential constraints in which the target table participates. If any of the constraints involve non-empty tables, those constraints must be defined as unlinked.
8. Reload the tables using the LOAD utility statement and the sequential file as input.

▶▶ See Chapter 21, “Loading an SQL-Defined Database” on page 21-1 for information about how to perform the load operation.
9. Complete the process as follows:
  - Recreate the views whose syntax was previously saved; examine those view definitions to see if changes are required
  - Respecify privilege definitions on the target table and its referencing views if access is controlled through CA-IDMS internal security

**Guaranteeing the integrity of the data:** You must ensure that no updates are made to any of the unloaded tables once their data has been extracted. To *ensure* that no changes are made between the time the data is extracted and the time the tables have been reloaded:

- Prohibit access to the tables by explicitly dropping all views that reference it. This is effective only if all update access to the table is done through a view.
- Revoke all INSERT, UPDATE, and DELETE privileges from the tables (and any matching wildcarded table names) if access is controlled through CA-IDMS internal security.



# Chapter 29. Modifying Indexes, CALC Keys, and Referential Constraints

---

- 29.1 Maintaining indexes . . . . . 29-4
  - 29.1.1 Creating an index . . . . . 29-4
  - 29.1.2 Dropping an index . . . . . 29-4
  - 29.1.3 Changing index characteristics/ moving an index . . . . . 29-5
- 29.2 Maintaining CALC keys . . . . . 29-6
  - 29.2.1 Creating a CALC key . . . . . 29-6
  - 29.2.2 Dropping a CALC key . . . . . 29-6
- 29.3 Maintaining referential constraints . . . . . 29-7
  - 29.3.1 Creating a referential constraint . . . . . 29-7
  - 29.3.2 Dropping a referential constraint . . . . . 29-7
  - 29.3.3 Modifying referential constraint tuning characteristics . . . . . 29-8



---

This chapter describes methods for creating, dropping, and changing indexes, CALC keys, and referential constraints.

►► For more information on the SQL DDL statements used in the procedures in this chapter, refer to the *CA-IDMS SQL Reference*.

## 29.1 Maintaining indexes

This section describes how to:

- Create or drop an index
- Change index characteristics
- Move an index from one area to another

### 29.1.1 Creating an index

To create a new index on a column or columns in a table, use the SQL DDL CREATE INDEX statement. If the index is going to map to a new area, see Chapter 3, “Defining Segments, Files, and Areas” on page 3-1 for information about defining an area.

#### Considerations

- If you specify that the index is unique, and data in the key columns is *not* unique, you will receive an error and the index will not be created.
- Each index implies additional runtime processing to handle INSERT, UPDATE, and DELETE statements for the index itself.

►► For more information about designing indexes, refer to *CA-IDMS Database Design*.

**Example:** In the following example, an index is built on the LAST\_NAME column in the BENEFITS table.

```
create index be_lname (last_name) on emp.benefits;
```

### 29.1.2 Dropping an index

To drop an index from an existing table, use the SQL DDL DROP INDEX statement.

**Considerations:** A unique index or CALC key is required on all referenced columns in a constraint and an index or CALC key must exist on all referencing (foreign key) columns in unlinked constraints. If dropping an index would violate either of these rules, the DROP will not be allowed.

**Example:** In the following example, an optional index is dropped from a table:

```
drop index xyz;
```

### 29.1.3 Changing index characteristics/ moving an index

To change index characteristics or to move an index from one area to another:

1. Create a new index with a new name using CREATE INDEX
2. Drop the old index using DROP INDEX

**Note:** Creating the new index before dropping the old one lets you modify an index involved in a referential constraint.

#### Considerations

- If changing index tuning options, remember to observe referential constraint rules.
- Adding an index with a new name, then dropping an old index does not impact program logic. Affected access modules will be automatically recompiled and, where relevant, use the new index for accessing the table.

**Example:** In the following example, a new index is created on the BENEFITS table, and the existing index, EM\_LNAME, is dropped.

```
create index emp_lname (last_name) on emp.benefits
      in area emp.emp1;
drop index em_lname;
```

## 29.2 Maintaining CALC keys

This section describes how to:

- Create a CALC key
- Drop a CALC key

### 29.2.1 Creating a CALC key

To create a CALC key for an empty table, use the SQL DDL CREATE CALC statement.

If the table is not empty, you must drop and recreate the table, adding the CALC key before reloading the table's data.

►► For the steps involved in this process, see Chapter 28, “Modifying Schema, View, and Table Definitions” on page 28-1.

**Considerations:** Only one location mode is permitted for a table. If the table is stored clustered on an index or constraint, you must drop the clustering index or constraint and re-add it as non-clustered before you can create a CALC key.

**Example:** In the following example, a unique CALC key is created for the EMPLOYEE table.

```
create unique calc key on emp.employee (emp_id);
```

### 29.2.2 Dropping a CALC key

To drop a CALC key from an empty table, use the SQL DDL DROP CALC statement.

If the table is not empty, you must drop and recreate it.

►► For the steps involved in this process, see Chapter 28, “Modifying Schema, View, and Table Definitions” on page 28-1.

**Considerations:** You can't drop a CALC key that is required for implementation of a referential constraint if no index exists to support it. If necessary, either drop the constraint or create an index to support it before dropping the CALC key.

**Example:** In the following example, the CALC key is dropped from the EMPLOYEE table.

```
drop calc key from emp.employee;
```

## 29.3 Maintaining referential constraints

This section describes how to:

- Create or drop linked or unlinked referential constraints
- Modify the tuning characteristics of referential constraints

### 29.3.1 Creating a referential constraint

To create an *unlinked* or *linked* referential constraint, use the SQL DDL CREATE CONSTRAINT statement. CA-IDMS/DB checks and rejects any invalid CREATE CONSTRAINT statements.

#### Considerations

- To create a *linked* constraint if both tables are not empty, you must drop and recreate the tables, defining the linked constraint before reloading the data.
  - ▶▶ For steps and considerations involved with this process, see Chapter 28, “Modifying Schema, View, and Table Definitions” on page 28-1.
- When adding an *unlinked constraint* on a non-empty table, CA-IDMS/DB ensures that all rows of the table satisfy the constraint. If one or more rows do not satisfy the constraint, the create will not be allowed.

**Example:** In the following example, a linked referential constraint has been created to make sure that the employee ID in the benefits table is a valid ID by checking it against the employee IDs in the employee table. The referential constraint is indexed and ordered by the fiscal year.

```
create constraint emp_benefits
  benefits (emp_id)
  references employee (emp_id)
  linked index
  order by (fiscal_year desc);
```

### 29.3.2 Dropping a referential constraint

To drop an *unlinked* referential constraint, or a *linked* referential constraint, use the SQL DDL DROP CONSTRAINT statement.

**Considerations:** If you drop a clustered constraint, the location mode will change as follows:

- If a default index exists, CA-IDMS/DB will use it as the clustering index.
- Otherwise, it uses a direct location mode which means that all new rows will be stored in the first page containing enough space to hold the row.

**Example:** In the following example, the EMP\_BENEFITS constraint is removed from the BENEFITS table:

```
drop constraint emp_benefits from benefits;
```

### 29.3.3 Modifying referential constraint tuning characteristics

To modify referential constraint tuning characteristics (for example, changing from unlinked to linked or adding an ORDER BY option) use the SQL DDL DROP CONSTRAINT statement, then re-add the constraint using the SQL DDL CREATE CONSTRAINT statement.

**Considerations:** All considerations for dropping and creating a referential constraint apply.

**Example:** In the following example, a linked referential constraint has been changed to unlinked:

```
drop constraint emp_benefits from benefits;  
create constraint emp_benefits  
  benefits (emp_id)  
  references employee (emp_id);
```

# Chapter 30. About Modifying Non-SQL Defined Databases

---

30.1	Types of modifications	30-3
30.2	Overview	30-4
30.2.1	Methods for modifying	30-4
30.2.2	Procedure for modifying the non-SQL definitions	30-5
30.2.3	RESTRUCTURE SEGMENT utility statement	30-7
30.2.4	UNLOAD/RELOAD utility statements	30-7
30.2.5	MAINTAIN INDEX utility statement	30-8



## 30.1 Types of modifications

Modification of a non SQL-defined database involves modifying any of the components you defined earlier for the schema or subschema. This includes:

- Adding or deleting schemas
- Adding, modifying, or deleting schema areas
- Adding, modifying, or deleting schema records
- Adding, modifying, or dropping indexes and sets

►► For information about modifying physical definitions, see Chapter 25, “Modifying Physical Database Definitions” on page 25-1.

## 30.2 Overview

**Changes to schemas:** In general, when you change a database, you must modify the schema code and revalidate the schema. However, changing the schema has an impact on other components of the CA-IDMS/DB environment. If you add or delete an area from a schema, you may have to add or delete that area in one or more segments and regenerate DMCLs. You will also have to modify and recompile some or all subschema definitions compiled under the original schema to reflect changes made to the schema.

If you access the non-SQL defined data through SQL, you may also need to recompile access modules and drop and recreate SQL view definitions.

The primary tool for changing a schema is the schema compiler.

**Steps to modify the schema:** The steps to make any schema modification are as follows:

1. Change and re-validate the necessary schema and subschema definitions
2. Change the actual *data* (if it exists) to fit the new database specifications using the RESTRUCTURE SEGMENT, MAINTAIN INDEX, or UNLOAD/RELOAD utility statements
3. Revise and recompile any application programs that may have been affected by the above changes
4. Test to ensure that the change has been made correctly.

**Changes to subschemas:** Subschemas identify selected areas, records, elements, and sets of the database. They also define logical records and establish security by restricting runtime access to the database.

Any time you make a change to any of the above components in your CA-IDMS/DB environment, you will have to change one or more of your subschemas.

The primary tool for changing subschemas is the subschema compiler.

### 30.2.1 Methods for modifying

Depending on the type of change you want to make to a non-SQL defined database, you would do one of the following:

- Change the definition
- Change the definition and additionally use one or more utility statements

**Basic definition change:** To change a logical database definition when there is no impact on data, you can use the schema compiler (or another compiler). This type of change takes affect without requiring a utility statement (UNLOAD/RELOAD, RESTRUCTURE SEGMENT, or MAINTAIN INDEX).

---

An example of a change in which there is no data impact is adding a new area to a schema.

**Definition change using utility statements:** For database changes that have an impact on data, you must change the database definition and additionally use an appropriate utility statement:

- **RESTRUCTURE SEGMENT** — Modifies record occurrences to fit new schema specifications. **RESTRUCTURE SEGMENT** allows you to:
  - Insert new data items anywhere in a record
  - Delete existing data items
  - Change the length and position of data items
  - Change the format of a record from fixed length to variable length or from variable length to fixed length
  - Add or remove record compression
  - Delete chained sets and add or delete set pointers
- **UNLOAD and RELOAD** — Reorganizes data when changes are made to the placement of records within the database (for example, moving a record from one area to another).
- **MAINTAIN INDEX** — Builds, rebuilds, or deletes indexes in the database. You use this utility whenever you need to make a structural change to the database involving indexes (for example, adding a new index to the database).

## 30.2.2 Procedure for modifying the non-SQL definitions

### Step 1: Copy the original schema and global subschema

1. Create a new schema which is identical to the *original* schema.
2. Create a global subschema for the new schema with a name which is different from that of any other subschema in the dictionary. Include in the subschema all areas, records, and sets associated with the schema.

### Step 2: Modify the new schema and subschema

1. Make the necessary changes to the *new* schema definition.
2. Validate the schema.
3. Regenerate the global subschema, modifying it if necessary.

### Step 3: Modify the segment and DMCL, if necessary

**Note:** You need to modify segments and DMCLs only if you add or remove an area or make other changes to the physical definition in addition to changing the schema.

1. Make the appropriate changes in the segment definition. Make sure that subareas and other symbolics are defined appropriately.

2. Generate, punch, and link all DMCLs containing the altered segment.

**Step 4: Make changes to the data**

**Note:** Not all schema changes require data changes. See Chapter 31, “Modifying Schema Entities” on page 31-1 for the steps needed in each case.

1. Backup the area or files.
2. Use the appropriate utility or user-written program to change the data.
3. Verify the change using IDMSDBAN and/or a retrieval program, CA-OLQ, or CA-CULPRIT.
4. Backup the altered areas or files.

**Step 5: Complete the change**

1. Update the *original* schema in the same way that the copy was changed.
2. Regenerate all subschemas associated with the original schema that are affected by the change, modifying them if necessary to add new areas, records, or sets.
3. Recompile all access modules affected by the change, using the ALTER ACCESS MODULE statement with the REPLACE ALL option.
4. Drop and recreate all SQL views affected by the change.
5. Make the new subschemas, DMCLs, and file available to your runtime environment.

**Considerations:** The procedure outlined above requires that changes first be made to a copy of the original schema and only after all other steps have been completed are the changes made to the original schema. This approach ensures that the original schema continues to describe the data until the altered areas are made available to runtime environment. You should use this (or a similar approach) if during the process:

- CA-OLQ, CA-CULPRIT, or dynamic SQL users will be accessing the original schema definition
- Application programs will be compiled against the original schema and must access the data before it has been changed.

If the above is not a concern or if no data changes are necessary, then the initial modifications can be made to the *original* schema rather than a copy, avoiding the necessity of replicating those changes later. (The copy of the schema is still useful if the REGENERATE SCHEMA utility statement will be used to alter data.)

### 30.2.3 RESTRUCTURE SEGMENT utility statement

**What RESTRUCTURE SEGMENT does:** The RESTRUCTURE SEGMENT utility statement modifies record occurrences to fit new schema specifications. You run RESTRUCTURE SEGMENT in local mode using a subschema associated with a schema that describes the database *before* restructuring.

RESTRUCTURE SEGMENT does not require that the database be unloaded and reloaded. Database keys remain unchanged. This means that database procedures can be executed during restructuring. For example, IDMSCOMP can be executed to compress records being changed from fixed length to variable length format.

**Steps for RESTRUCTURE SEGMENT:** To make changes using RESTRUCTURE SEGMENT, follow the procedure described in 30.2.2, “Procedure for modifying the non-SQL definitions” on page 30-5, except add the steps listed in the following table.

After ...	Do this
Modifying the schema and subschemas	Execute the schema compare utility (IDMSRSTC) to generate IDMSRSTT macro statements for use in the database restructure
Executing IDMSRSTC	Assemble the IDMSRSTT statements into a base restructuring table and use the table with the RESTRUCTURE SEGMENT utility statement; use a subschema that describes the database <i>before</i> restructuring
Executing RESTRUCTURE SEGMENT	Connect any new pointers to existing sets using the RESTRUCTURE CONNECT utility statement; use a subschema that describes the database <i>after</i> restructuring
Executing RESTRUCTURE SEGMENT	Write a program to connect pointers in new sets to existing records

►► For detailed information on IDMSRSTC, RESTRUCTURE SEGMENT, and RESTRUCTURE CONNECT, refer to *CA-IDMS Utilities*.

### 30.2.4 UNLOAD/RELOAD utility statements

**What UNLOAD and RELOAD do:** The UNLOAD and RELOAD utility statements unload existing database records and reload them into the database. UNLOADING and RELOADING involves:

1. Using the UNLOAD utility statement to offload data to an intermediate file in preparation for reloading it.
2. Using the RELOAD utility statement to store the record data into the database, build index structures, and connect related records together in set structures.

**Steps for UNLOAD/RELOAD:** To make changes using UNLOAD/RELOAD, follow the procedure described in 30.2.2, “Procedure for modifying the non-SQL definitions” on page 30-5, adding the steps listed in the following table.

After ...	Do this
Modifying the schema and subschemas	UNLOAD using subschemas that reflect both the old and new schema definitions.
Unloading the data	Use the FORMAT utility statement to initialize the files into which the data will be reloaded.

►► For detailed information about UNLOAD and RELOAD, refer to *CA-IDMS Utilities*.

### 30.2.5 MAINTAIN INDEX utility statement

**What MAINTAIN INDEX does:** The MAINTAIN INDEX utility statement allows you to build, rebuild, and delete both system-owned and user-owned indexes (indexed sets). You can also change the characteristics of an index, such as changing an index key from a compressed to an uncompressed format.

**Steps to modify indexes:** To make changes to an index, follow the procedure described in 30.2.2, “Procedure for modifying the non-SQL definitions” on page 30-5, adding the steps listed in the following table.

After ...	Do this
Modifying the schema and global subschema	<p><b>For system-owned indexes:</b></p> <p>Use MAINTAIN INDEX to build, rebuild, or delete an index.</p> <p><b>For user-owned indexes (indexed sets):</b></p> <p>Write a program that calls IDMSTBLU and passes descriptor information</p>

**Note:** Depending on the operation, you will need either a subschema reflecting the old schema, the new schema, or both.

►► For detailed information about MAINTAIN INDEX and IDMSTBLU, refer to *CA-IDMS Utilities*.

# Chapter 31. Modifying Schema Entities

---

31.1	Modifications to an unloaded database	31-4
31.2	Schema modifications	31-5
31.2.1	Deleting a schema	31-5
31.2.2	Changing schema characteristics	31-5
31.3	Area modifications	31-6
31.3.1	Adding or deleting an area	31-6
31.3.2	Changing area characteristics	31-7
31.4	Record modifications	31-8
31.4.1	Adding schema records	31-8
31.4.2	Deleting schema records	31-8
31.4.3	Changing a record's CALC key	31-9
31.4.4	Changing the DUPLICATES option on a CALC or SORT key	31-11
31.4.5	Changing the location mode of a record	31-12
31.4.6	Changing a record's area	31-13
31.4.7	Modifying record elements	31-14
31.4.8	Changing other record characteristics	31-15
31.4.9	Adding and dropping database procedures	31-16
31.5	Set modifications	31-17
31.5.1	Adding or deleting a set	31-17
31.5.2	Changing set mode	31-18
31.5.3	Adding and dropping set pointers	31-19
31.5.4	Changing set order	31-20
31.5.5	Changing set membership options	31-21
31.6	Index modifications	31-23
31.6.1	Adding or deleting system-owned indexes	31-23
31.6.2	Changing the location of an index	31-24
31.6.3	Changing index characteristics	31-24
31.6.4	Adding or deleting index pointers	31-25



---

**What this chapter contains:** This chapter describes:

- The procedure to modify a schema or schema entities when the database is empty.
- Specific procedures to add, delete, or modify schema or schema entity definitions when the database is *not* empty. These procedures include only those that require a utility to affect the change.

## 31.1 Modifications to an unloaded database

**What components are affected:** Changes to a schema or schema entity in an unloaded database affects:

- The schema
- The subschemas that reference the schema
- The access modules that reference the schema
- SQL views that reference the schema

### Steps

1. Modify the schema and any schema entities, as desired
2. Validate the schema
3. Regenerate any affected subschemas
4. Alter affected access modules using the REPLACE ALL option
5. Drop and recreate SQL views that reference the schema, as necessary

## 31.2 Schema modifications

This section describes how to:

- Delete a schema
- Change schema characteristics

### 31.2.1 Deleting a schema

**What components are affected:** When you delete a schema, the definitions of the schema and all subschemas associated with the schema are removed from the dictionary.

**Steps to delete a schema:** To delete a schema from the dictionary:

1. Delete the schema
2. Delete load modules associated with the deleted subschemas
3. Delete files that contain the data
4. Delete the segment(s) corresponding to the schema
5. Regenerate all affected DMCLs
6. Remove the segment(s) from the database name table
7. Delete SQL schema(s) referencing the non-SQL schema

**Considerations:** When you delete a schema, subschemas associated with that schema are also deleted. The subschema load modules are not deleted.

In addition, the physical database definition(s) that apply to the schema's areas are not automatically deleted. You must modify the physical database definitions to delete the areas and regenerate all affected DMCLs.

### 31.2.2 Changing schema characteristics

Schema characteristics include:

- Description
- Memo date
- Assignment rules for record IDs
- Security specifications
- User-defined information (class/attribute and user-defined comments)

**What components are affected:** When you modify characteristics of a schema, only the schema definition is affected. These characteristics do not impact critical definitions within the schema or its subschemas, so a VALIDATE statement is not required.

## 31.3 Area modifications

**Types of changes:** This section describes how to make the following area-related changes:

- Add or delete an area in an existing schema definition
- Add, remove, or change procedures associated with the area

### 31.3.1 Adding or deleting an area

**What components are affected:** Adding or deleting an area in the schema affects the schema and subschemas referencing an area to be deleted, segments describing related physical databases, and DMCLs in which those segments are included.

#### Steps to add an area

1. Modify the schema
2. Add the new area
3. Define one or more records or system-owned indexes associated with the area
4. Validate the schema
5. Add the new area to one or more subschemas
6. Format the new area using the FILE option of the FORMAT utility statement

#### Steps to delete an area

1. Modify the schema
2. Modify existing records mapping to the area to be deleted so that they map to a different area
3. Delete the area
4. Validate the schema
5. Regenerate any affected subschemas

#### Considerations

- If existing records are to reside in a new area, see 31.4.6, “Changing a record's area” on page 31-13 later in this chapter.
- If an existing index is to reside in a new area, see 31.6.2, “Changing the location of an index” on page 31-24 later in this chapter.
- After you have added an area to a schema or deleted an area from a schema, make sure you update the DMCL module appropriately.

## 31.3.2 Changing area characteristics

**What components are affected:** When you add or delete area procedures, the area and schema definitions are affected. All subschemas which include the area must be regenerated and all access modules accessing a record in the area must be altered.

**Steps to change area characteristics:** ►► See 31.1, “Modifications to an unloaded database” on page 31-4 at the beginning of this chapter for the steps to modify an empty database.

**Considerations:** Remember to respecify all database procedures in the order that they are to be called when you add, remove, or change the procedures associated with an area.

## 31.4 Record modifications

**Types of changes:** This section describes the following record-related changes:

- Adding or deleting a record in your schema
- Changing a record's CALC key
- Changing a record's location mode
- Changing a record's area
- Changing a record element
- Changing record procedures

### 31.4.1 Adding schema records

**What components are affected:** Adding schema records affects the schema.

**Steps to add a record**

1. Add the record using DDDL statements
2. Modify the schema
3. Add the record to the schema using SHARE STRUCTURE
4. Validate the schema
5. Modify any subschemas that should contain the new record
6. Add the new record to each subschema
7. Regenerate the subschemas

**Considerations:** If the record participates in a set with existing records, you must use the RESTRUCTURE SEGMENT utility statement to add pointer positions to the existing records. You must also write a program that connects the records into proper set occurrences.

►► For more information on adding a set to a schema, see 31.5.1, “Adding or deleting a set” on page 31-17 later in this chapter.

### 31.4.2 Deleting schema records

**What components are affected:** Deleting schema records affects the schema *and* the data. It also affects subschemas and access modules that reference the record and any other records connected to the record through sets. SQL views referencing the record become invalid.

**Steps to delete a record:** To delete a record from the schema where data has been loaded:

1. Write and execute a program to erase all occurrences of the record
2. Create a new schema based on the original schema omitting the record and omitting any affected sets
3. Validate the schema
4. Use the schema compare utility (IDMSRSTC) to generate the IDMSRSTT macro statements
5. Restructure the database using the RESTRUCTURE SEGMENT utility statement

**Note:** If the record does not participate in any set relationships, there is no need to restructure the database.

6. Complete the process by updating the original schema definition, regenerating subschemas, altering affected access modules, and dropping affected views

### Considerations

- If you created the record using DDDL statements, the record definition will remain in the dictionary after it has been deleted from the schema
- If you created the record using schema DDL and the record has *not* been copied into any other schema, its definition will be deleted from the dictionary after it has been deleted from the schema definition.
- If the record participates in a set relationship, you have to remove the set from the schema definition or modify the definition before validation
- Regenerating affected subschemas will remove the record from the subschema definition
- When you erase occurrences of the record, you may have to use a subschema derived from a schema where the sets in which the record is an owner have been changed to optional. This permits the member record to be disconnected from the owner record rather than being erased.

### 31.4.3 Changing a record's CALC key

**Types of changes:** You can make the following changes to a record's CALC key:

- Replace one or more elements in the CALC key
- Add or remove elements in the CALC key
- Change the picture or usage of an element in the CALC key

**What components are affected:** Both the schema record definition and the data are affected. Subschemas and access modules that reference the record are also affected.

**Steps to change the CALC key:** To change the CALC key of a schema record where data has been loaded:

1. Add a new schema based on the original schema
2. Modify the record in the new schema specifying the new CALC key
3. Validate the schema
4. Create a new global subschema
5. Unload and reload the database using the UNLOAD/RELOAD utility statements
6. Complete the process by updating the original schema definition, regenerating subschemas, and altering affected access modules

### Considerations

- UNLOAD/RELOAD clusters VIA records based on the CALC key defined in the subschema used to *unload* the data. Therefore, you need to do a second UNLOAD/RELOAD to properly cluster VIA records if the subschema used to unload the data describes the old CALC key.

As an alternative, you can use the new subschema for unloading. This ensures that the new CALC key is used to determine target pages for both the CALC record and its associated VIA records. However, the new subschema can be used to unload data only if no other changes have been made to the record (such as the record's area, set pointers, etc.).

In some cases multiple changes can be accommodated by using an intermediate schema/subschema to unload the data. For example, to change the CALC key of a record and also move it to a new area, unload the data using a subschema that describes the record's new CALC key but old area. Reload the data using a subschema describing the new CALC key and the new area.

- If the control length of the record is changing as a result of the change to the CALC key and the record is compressed or variable length, you must do one of the following:
  - Use RESTRUCTURE SEGMENT to alter the control length before unloading and reloading the data.
  - Unload and reload the data twice (using the old subschema on the first unload and the new subschema on the second)
- If a field to be added to the CALC key does not exist in the record, add the field using RESTRUCTURE SEGMENT and initialize it before unloading and reloading the data. Initialize the field using restructure or a user-written program.

►► See 31.4.7, “Modifying record elements” on page 31-14 later in this chapter for information on adding a new record element.

---

### 31.4.4 Changing the DUPLICATES option on a CALC or SORT key

**Types of changes:** You can make the following changes to the DUPLICATES option on a record's CALC or sort key:

- Duplicates first to duplicates last
- Duplicates last to duplicates first
- Duplicates not allowed to duplicates first/last
- Duplicates first/last to duplicates not allowed

**What components are affected:** The entire schema definition is affected. Depending on the change, the data may also be affected. All subschemas and access modules referencing the record are also affected.

**Steps to change the duplicates option:** See 31.1, “Modifications to an unloaded database” on page 31-4 at the beginning of this chapter for the steps to change the duplicates option from:

- Duplicates first/last to duplicates not allowed
- Duplicates not allowed to duplicates first/last

To change the duplicates option from first to last or from last to first:

1. Write a program using a subschema that specifies duplicates first for the CALC or sort key. The program must
  - Modify the CALC or sort key value to a dummy value
  - Modify the CALC or sort key value to its original value
  - Read each record that has duplicate values, using either OBTAIN CALC DUPLICATE or OBTAIN NEXT IN SET to retrieve duplicate records in the current order

This has the effect of reversing the order of the duplicate records.

2. Modify the schema
3. Modify the record changing the duplicates option
4. Validate the schema
5. Regenerate any affected subschemas
6. Alter affected access modules using the REPLACE ALL option

#### Considerations

- When you change from duplicates first or last to duplicates not allowed, make sure that no duplicate key values exist in the database.
- When changing from duplicates first to last or last to first, write a conversion program to logically reorder the record occurrences in the database.

Using the approach described above, the program must execute using a subschema specifying duplicates *first*. Therefore, it should use a subschema created either before or after the schema has been changed depending on whether the duplicates option is being changed from or to duplicates first.

### 31.4.5 Changing the location mode of a record

**Types of changes:** These are the possible location mode changes for a record in the database:

- CALC to VIA
- CALC to DIRECT
- DIRECT to CALC
- DIRECT to VIA
- VIA to CALC
- VIA to DIRECT
- VIA one set in the schema to VIA another set

**What components are affected:** The record definition and the data are affected. Subschemas and access modules referencing the record are also affected.

**Steps to change the location mode:** To change the location mode of a schema record where data has been loaded:

1. Add a new schema based on the original schema
2. Modify the record in the new schema to specify the new location mode
3. Validate the schema
4. Create a new global subschema
5. Unload and reload the database using the UNLOAD/RELOAD utility statements
6. Complete the process by updating the original schema, regenerating affected subschemas, and altering affected access modules.

#### Considerations

- If the storage mode is being changed to VIA, add the set if it does not exist
- If you change a location mode from CALC or DIRECT to VIA, or if you change the VIA location mode from VIA one set to VIA another set, you must run the UNLOAD and RELOAD utilities a second time to ensure proper clustering
- If you change a location mode from DIRECT to VIA or VIA one set to VIA another set, you can sometimes avoid a second UNLOAD/RELOAD by using the new subschema in the UNLOAD step. This technique can be used only if no set pointers or data lengths have changed. If used, the unload process may require more I/Os because UNLOAD walks the VIA set to locate the member record occurrences.

- If you change a location mode from CALC to DIRECT you will need to run an UNLOAD/RELOAD to disconnect the record from the CALC set. UNLOAD/RELOAD stores DIRECT records near their original locations. If you are not satisfied with this location, write a program to delete and store the records exactly where you want them.
- If you change a location mode from VIA to DIRECT, an UNLOAD/RELOAD is not necessary. If you are not satisfied with the locations of the records, write a program to delete and store the records where you want them.
- If you add a new set, UNLOAD/RELOAD will not connect it. However, after the RELOAD the pointer positions will exist and you can write a program to connect the members to the sets.
- If you need to add a set to the schema, make sure you add the set to the subschemas as required

### 31.4.6 Changing a record's area

**Types of changes:** You can move a record from one area to another or change the portion of an area in which a record is stored.

**Note:** If a subarea symbolic is associated with the record, you change the portion of the area in which the record is stored by changing the physical area definition and regenerating DMCLs. See Chapter 25, “Modifying Physical Database Definitions” on page 25-1 for more information.

**What components are affected:** The schema record definition and data area affected. Subschemas and access module that reference the record are also affected.

**Steps to change the record's area:** To change the area (or portion of an area) in which record occurrences are stored when data has been loaded:

1. Create a new schema based on the original schema
2. Add the area, if necessary, to the new schema
3. Modify the record in the new schema to specify the new area or subarea/offset
4. Validate the schema
5. Create a new global subschema
6. Unload and reload the database using the UNLOAD/RELOAD utility statements
7. Complete the process by updating the original schema, regenerating affected subschemas, and altering affected access modules.

#### Considerations

- If you had to add the area to the schema, you must explicitly add it to subschemas associated with the record. You must also explicitly add the area to all applicable physical database definitions and regenerate affected DMCLs.

- If you increase the page range of a record whose location mode is other than CALC, you do not need to unload and reload the data provided the new page range includes all pages of the original range.

### 31.4.7 Modifying record elements

**Types of changes:** These are changes you can make to an element within a schema record:

- Adding or removing a record element
- Changing the picture or usage mode of an element

**What components are affected:** The record definition is affected. If data has been loaded, the data may also be affected. Subschemas in which the record is included are affected as are programs compiled from those subschemas. Access modules and SQL views that reference the record are also affected.

**Steps to change the record element:** To make any of the above changes when data has been loaded:

1. Using DDDL statements, create a record with a new version number and same name having the revised structure
2. Create a new schema based on the original schema with the new record
3. Validate the schema
4. Use the schema compare utility (IDMSRSTC) to generate the IDMSRSTT macro statements
5. Restructure the database
6. Complete the process by updating the original schema, regenerating affected subschemas, altering access modules, and dropping and recreating affected SQL views

#### Considerations

- You can replace filler elements with record elements whose total length equals that of the filler element without creating a new version of the record. The new elements are immediately reflected in the schema. The next time any programs that use that schema record are compiled, the new elements appear. Affected subschemas are flagged for regeneration.
- You should initialize any 'filler' fields or fields whose picture or usage has been changed using either RESTRUCTURE SEGMENT or a user-written program.
- A record element in a schema-owned record can be replaced with elements of the same name
- If you want to maintain consistency among the record version numbers in your schema:
  1. Complete all of the steps above

2. Delete the original version of the record
3. Modify the record using DDDL statements to change its version number

You do not have to modify the schema.

- Non-structural changes can be made directly to schema-owned records using DDDL. For example, you can change the external picture of a record element even if it is associated with a schema.
  - ▶▶ For more information on the types of changes that can be made to schema-owned records, refer to *IDD DDDL Reference*
- If you change the format (picture or usage) of an element used in a CALC or sort key, additional steps may be needed to convert the data.

### 31.4.8 Changing other record characteristics

**Types of changes:** You can make the following changes to the characteristics of a record (changes other than those described previously in this chapter):

- Record ID
- Record synonyms
- VSAM type
- Minimum root and minimum fragment length
- Whether the record is compressed or uncompressed

**Note:** For information on dropping or adding a database procedure associated with a record, see 31.4.9, “Adding and dropping database procedures” on page 31-16, later in this chapter.

**What components are affected:** The record definition is affected and the data is affected if changing the record ID or compression. All subschemas and access modules that reference the record are affected. SQL views are affected only if the SQL synonym for an element is changed.

**Steps to make the change:** To modify the VSAM type, record synonyms, or minimum root and fragment lengths, follow the procedure described in 31.1, “Modifications to an unloaded database” on page 31-4 at the beginning of this chapter.

#### Considerations

- To change the record ID when data exists, write a program that offloads and reloads that data
- Optionally, UNLOAD/RELOAD can be used to reorganize existing data after changing the minimum root or minimum fragment.
- To change a record from compressed to uncompressed, you must use either UNLOAD/RELOAD or RESTRUCTURE SEGMENT

- If changes to the record elements do not affect control fields, all you need to do is issue an DDDL MODIFY RECORD statement
  - ▶▶ For more information about modifying non-IDD owned records, refer to *IDD DDDL Reference*
- If you change the SQL synonym for one or more elements then you must drop and recreate all SQL views that reference the record. You must also change all programs that refer to those elements in an SQL statement.
- If you change the VSAM type, be sure that appropriate changes, if necessary, are made to the VSAM definition using the IDCAMS utility.

### 31.4.9 Adding and dropping database procedures

**What components are affected:** If you implement a new database procedure, or change the name of an existing procedure, it will affect the schema and one or more subschemas. It *may* also require that you restructure the database, if the purpose of the procedure is to alter the physical data (for example, record compression). All subschemas and access modules that reference the record are also affected by procedure changes.

**Steps to make the change:** To add, modify, or delete database procedures that have no effect on the data, follow the procedure described in 31.1, “Modifications to an unloaded database” on page 31-4 at the beginning of this chapter.

#### Considerations

- If a new database procedure *does* affect the data, write and compile the new procedure and then use the RESTRUCTURE SEGMENT utility statement to change the existing data by specifying the new procedure in a NUPROCS macro.

If database procedures are already associated with the record, they may need to be removed from the schema and subschema before executing RESTRUCTURE SEGMENT. The existing procedures, if invoked, will be called *after* all NUPROCS procedures have been called. If, for example, the new procedure compresses the data in the record, the existing procedures may not work properly. To overcome this problem, either execute RESTRUCTURE SEGMENT using a subschema derived from an intermediate schema in which all procedures normally called before the new procedure have been removed from the record or use UNLOAD/RELOAD to add the new procedure.
- UNLOAD/RELOAD can be used to add or remove procedures that affect the data. To use UNLOAD/RELOAD, create a new schema and subschema containing the revised procedure calls. Unload the data using the old subschema and reload it using the new subschema.
- To change a database procedure for an area, all calls must be respecified.

## 31.5 Set modifications

**Types of changes:** The following set-related changes can be made:

- Add or remove a set
- Change the mode (index or chain)
- Add or remove set pointers
- Change set order
- Change membership options

### 31.5.1 Adding or deleting a set

**What components are affected:** The schema set definition and data are affected. Segments and DMCLs may also be affected if a set is indexed and a symbolic index specification needs to be added, removed, or replaced in the physical definition. Subschemas and access modules that reference either the owner or member of the set are also affected.

**Steps to add or delete a set:** To add or delete a set when data has been loaded:

1. Create a new schema based on the original schema but containing the new set or omitting the deleted set
2. Validate the schema
3. Create a global subschema for the new schema
4. Use the schema compare utility (IDMSRSTC) to generate the IDMSRSTT macro statements
5. Restructure the database using the RESTRUCTURE SEGMENT utility statement
6. If adding a set, write a program to connect member record occurrences to the appropriate owner occurrences.
7. Complete the change by updating the original schema, regenerating affected subschemas, and altering affected access modules.

#### Considerations

- Both records participating in a new set must be defined to the schema
- If you replace an existing set with a new set, do not use the AUTO parameter; specify the actual pointer positions. This eliminates the possibility that the schema compiler will identify different pointer positions than exist in the loaded database.
- When deleting an existing set from a schema and a participating record contains pointer positions for sets *beyond* the deleted set's pointer positions, you must renumber the remaining positions. You cannot leave unused pointer positions.
- If you delete a set, the set is also deleted from all subschema descriptions.

- If you delete the owner record within a set, the set is automatically deleted and both the set and deleted record are removed from all subschema descriptions.
- If you delete the member record within a set, the set remains. You receive an error on validation if there are no remaining members in the set (as in a multimember set)
- If you want a new set to be included in a subschema, you must modify the subschema and add the set to the subschema.
- Regenerating affected subschemas will remove a deleted set from all subschemas.
- When you delete a set, alter and recompile all programs that use the set

## 31.5.2 Changing set mode

**Types of changes:** You can change the mode of a set from chain to index or vice versa.

**What components are affected:** The schema set definition and data are affected. All subschemas and access modules that reference either the owner or member records are also affected.

**Steps to change from chained to indexed:** To change a chained set to an indexed set when data has been loaded:

1. Create a new schema based on the original schema
2. Modify the set in the new schema to change the set mode
3. Validate the schema
4. Create a global subschema
5. Write a program that sweeps the area, walks each set, and calls IDMSTBLU to perform a BUILD function.
6. Restructure the database if needed to remove old pointer positions and add new ones.
7. Execute MAINTAIN INDEX from SORT3 using the output from step 5 as input
8. Complete the change by updating the original schema, regenerating affected subschemas, and altering affected access modules.

**Steps to change from indexed to chained:** To change an indexed set to a chained set when data has been loaded:

1. Create a new schema based on the original schema
2. Modify the set in the new schema to change the set mode
3. Validate the schema
4. Create a global subschema
5. Write a program that sweeps the area and calls IDMSTBLU to perform a DELETE function and also produces a work file for input to step 8.

6. Use the output generated by IDMSTBLU as input to MAINTAIN INDEX and run it from SORT3 to delete each index occurrence
7. Restructure the database as needed to remove old pointers positions and add new ones
8. Sort the workfile produced by IDMSTBLU by owner key, member symbolic key, or set position.
9. Write a program to:
  - a. Read the sorted output
  - b. Obtain owner by db-key
  - c. Obtain member by db-key
  - d. Connect the member to the set
10. Complete the change by updating the original schema, regenerating affected subschemas, and altering affected access modules.

#### **Considerations for the change from indexed to chained**

- When you submit the RESTRUCTURE SEGMENT utility statement to initialize pointers (and possibly to delete pointers), you must initialize all existing pointer positions in the owner and member records that will be *re-used* for the chained set. If this is not done, you will be unable to connect the members to their owners in Step 9.
- The work file produced in Step 5 should contain the following information:
  - The dbkey of each owner record occurrence
  - The dbkey of each member record occurrence
  - The position of each member record with the set (if its necessary to maintain the same set order)
  - The sort key of the member record within the set (if the set order is changing or the order of duplicates does not have to be maintained)

### **31.5.3 Adding and dropping set pointers**

**Types of changes:** You can make the following changes to set pointers:

- Add or remove prior or owner pointers from a chain set
- Add or remove owner pointers from an indexed set

**What components are affected:** When you change the prior or owner pointers defined to a set, the schema set definition and data are affected. Subschemas and access modules that reference either the owner or member records are also affected.

**Steps to add or drop set pointers:** To add or drop set pointers when data has been loaded:

1. Create a new schema based on the original schema but containing the modified set pointers
2. Validate the schema
3. Create a global subschema
4. Use the schema compare utility (IDMSRSTC) to generate the IDMSRSTT macro statements
5. Restructure the database using RESTRUCTURE SEGMENT
6. If you add a prior or owner pointer to an existing set, fill in the pointer values using RESTRUCTURE CONNECT
7. Complete the process by modifying the original schema, regenerating affected subschemas, and altering affected access modules.

#### Considerations

- When adding or deleting pointers, do not use the AUTO parameter; specify the actual pointer positions. This eliminates the possibility the schema compiler will identify different pointer positions than exist in the loaded database.
- When deleting a pointer from a set in a schema and a participating record contains pointer positions *beyond* the deleted pointer, you must renumber the remaining positions. You cannot leave unused pointer positions.

### 31.5.4 Changing set order

**Types of changes:** You can make the following order-related changes:

- Change from SORTED to unsorted (NEXT, PRIOR, FIRST, LAST) order
- Change from unsorted to sorted
- Change one of the unsorted orders to another
- Change the sort key or collating sequence of a sorted set

**What components are affected:** When you change NEXT, PRIOR, FIRST, LAST, or SORTED specifications, the schema set definition and data are affected. Subschemas and access modules that reference either the owner or member records area are also affected.

**Steps to change set order:** Follow the steps listed in 31.1, “Modifications to an unloaded database” on page 31-4 at the beginning of this chapter if both of the following statements are true:

- You are changing a chained or an unsorted indexed set to NEXT, PRIOR, FIRST, or LAST
- It is not important to re-order existing data

**Steps to re-order chain and unsorted indexed sets:** To change the set order of a chained or unsorted indexed set, member records must be re-ordered:

1. Create a new schema based on the original
2. Modify the set to change the set order
3. Validate the schema
4. Create a global subschema
5. Write a conversion program that disconnects and re-orders (in the desired sequence) each member record occurrence
6. Complete the process by updating the original schema, regenerating affected subschemas and altering affected access modules

**Steps to re-order sorted indexed sets:** To change the sort key of a sorted indexed set or to change an indexed set from unsorted to sorted and vice versa, follow the procedure for re-ordering chain sets except replace Step 5 with the following:

1. Write a program that sweeps the area and call `IDMSTBLU` with a `REBUILD` function
2. Use the output from the step above as input to `MAINTAIN INDEX` and run `MAINTAIN INDEX` from `SORT3`

**Considerations:** When you change the set order from or to `SORTED` or when you change the sort key of a sorted set, the control length of the member record may change. If it does, and the member record is compressed or variable in length, you must use `RESTRUCTURE SEGMENT` to change the control length of existing record occurrences.

### 31.5.5 Changing set membership options

**Types of changes:** You can change a `MANDATORY` set to `OPTIONAL` and vice versa. You can also change an `AUTOMATIC` set to `MANUAL`.

**What components are affected:** When you change membership options, the schema set definition is affected. Subschemas and access modules that reference either owner or member record are also affected.

**Steps to change membership options:** To change membership options, regardless of whether or not data is loaded, follow the steps outlined in 31.1, "Modifications to an unloaded database" on page 31-4 earlier in this chapter.

**Considerations:** Changing membership options may impact existing application programs. Consider the following:

- If you change from `AUTOMATIC` to `MANUAL` or vice versa, programs that `STORE` member records may need to connect records into the set using a `CONNECT` statement or no longer issue such a `CONNECT`.

- If you change from OPTIONAL to MANDATORY, programs that DISCONNECT members from the set *must* be changed and programs that ERASE owner records may need to be changed.
- If you change from MANDATORY AUTOMATIC to any other option, programs that obtain the owner of the set may be affected (because a given member occurrence may not have an owner).

## 31.6 Index modifications

**Types of changes:** You can make the following types of changes to system-owned indexes:

- Add or remove an index
- Change the area in which an index resides
- Change index characteristics
- Change from linked to unlinked or vice versa

### 31.6.1 Adding or deleting system-owned indexes

**What components are affected:** When you add or remove a system-owned index, the schema set definition and the data are affected. All subschemas and access modules that reference the member record are also affected.

**Steps to add an index:** To add an index when data has been loaded:

1. Add a new schema based on the original schema adding the new index
2. Validate the schema
3. Create a global subschema for the new schema
4. If the index is linked, add the index pointer position to the member record using the schema compare utility (IDMSRSTC) and RESTRUCTURE SEGMENT
5. Build the index structure using the new subschema using the MAINTAIN INDEX utility statement
6. Complete the process by updating the original schema, regenerating affected subschemas, and altering affected access modules

**Steps to remove an index:** To remove an index when data has been loaded:

1. Add a new schema based on the original schema removing the index
2. Validate the schema
3. Create a global subschema for the new schema
4. Delete the index structure using an old subschema and the MAINTAIN INDEX utility statement
5. If the index is linked, remove the index pointed from the member record using the schema compare utility (IDMSRSTC) and the RESTRUCTURE SEGMENT utility statement
6. Complete the process by updating the original schema, regenerating affected subschemas, and altering affected access modules

**Considerations**

- The index you delete from the schema will automatically be deleted from any affected subschemas when you request that affected subschemas be regenerated.
- If an index is added or removed, it may change the control length of the record. If it does and the record is compressed or variable in length, you must change the control length of existing data using RESTRUCTURE SEGMENT.

## 31.6.2 Changing the location of an index

**Types of changes:** You can change the area (or portion of an area) in which the index structure resides.

**What components are affected:** The schema set definition and data are affected. Subschemas and access modules that reference the member record are also affected.

**Note:** If a subarea symbolic is associated with the index, you change the portion of the area in which the index is stored by changing the physical area definition and regenerating DMCLs. See Chapter 25, “Modifying Physical Database Definitions” on page 25-1 for more information.

**Steps to change the area:** To change the area (or portion of an area) in which an index resides when data has been loaded:

1. Add a new schema based on the original schema
2. Add the area, if necessary
3. Modify the indexed set to map to the new area or subarea/page range
4. Validate the schema
5. Create a new global subschema
6. Rebuild the index using both an old and new subschema using the MAINTAIN INDEX utility statement

**Considerations**

- If the area does not exist in the subschema, you will receive an error when you issue REGENERATE AFFECTED SUBSCHEMAS.

## 31.6.3 Changing index characteristics

**Types of changes:** You can change the following index-related characteristics:

- Key compression
- Number of entries in an SR8 record
- Index displacement
- Index key or collating sequence

**What components are affected:** The schema set definitions and data are affected. Subschema and access modules that reference the member record are also affected.

**Steps to change index characteristics:** To change index characteristics when data has been loaded:

1. Add a new schema based on the original schema modifying the set characteristics
2. Validate the schema
3. Create a global subschema
4. Rebuild the index using the MAINTAIN INDEX utility and the REBUILD option
5. Complete the process by updating the original schema, regenerating affected subschemas, and altering affected access modules

### Considerations

- If you change the index key, the control length of the member record may change. If it does, and the member record is compressed or variable in length, you must use RESTRUCTURE SEGMENT to change the control length of existing record occurrences.
- When you execute the MAINTAIN INDEX utility statement, use the REBUILD option:
  - If you change key compression, you must specify the name of an *old* subschema in the USING parameter and the name of a *new* subschema in the NEW SUBSCHEMA parameter.
  - If you change the symbolic key or the collating sequence, you must specify the *new* subschema in the USING parameter.

If both types of change are being made at once, you will need to run MAINTAIN INDEX twice, once to delete the existing index (using the old subschema) and once to build the new index (using the new subschema).

## 31.6.4 Adding or deleting index pointers

**Types of changes:** You can delete or add index pointers. The index pointer in a member record is optional for system-owned indexes.

**Adding or deleting index pointers:** To add or delete an index pointer:

1. Modify the schema specifying INDEX POSITION IS NONE
2. Add or delete the pointer position using the RESTRUCTURE SEGMENT utility statement
3. Rebuild the index using the MAINTAIN INDEX utility statement



# Chapter 32. Modifying Subschema Entities

---

- 32.1 Modifying or deleting a subschema . . . . . 32-4
  - 32.1.1 Modifying a subschema . . . . . 32-4
  - 32.1.2 Deleting a subschema . . . . . 32-4
- 32.2 Adding, modifying, or deleting a record . . . . . 32-6
- 32.3 Adding, modifying, or deleting a set . . . . . 32-7
- 32.4 Adding, modifying, or deleting an area . . . . . 32-8
- 32.5 Adding, modifying, or deleting a logical record or path group . . . . . 32-9



---

**Affect on applications associated with the subschema:** Changes you make to a subschema impact application programs associated with that subschema. In general, when you add, modify, or delete a subschema entity and regenerate the subschema, follow the appropriate procedure in the following table:

<b>If a program...</b>	<b>You should...</b>
Is associated with the subschema but does not need to access the new entity (area, record, or set)	Not have to recompile the program
Is associated with the subschema and needs access to a new entity or has access to a modified entity	<ul style="list-style-type: none"><li>■ Alter the program as needed</li><li>■ Recompile the program</li></ul>

**Regenerating the subschema:** Before you can use the subschema, you must regenerate it as described in Chapter 14, “Subschema Statements” on page 14-1.

If you want to use the subschema before the system is recycled, you must issue a DCMT VARY PROGRAM .. NEW COPY command. This statement causes the regenerated subschema to be loaded into the program pool the next time it is requested.

**Identifying programs associated with a subschema:** If your site updates the dictionary every time a program is compiled, the dictionary will contain the necessary information to identify the programs associated with a modified subschema.

If this information is stored in the dictionary, you can run IDMSRPTS Program Cross-Reference Listing report. For each program associated with a subschema, the report lists:

- Name
- Version number
- Date last compiled
- Number of times compiled
- Language

▶▶ Refer to *CA-IDMS Utilities* for complete information on IDMSRPTS.

If your site does not update the dictionary when a program is compiled, such information must be maintained manually.

## 32.1 Modifying or deleting a subschema

### 32.1.1 Modifying a subschema

**When you might want to make this change:** There are several modifications you may want to make to the subschema definition itself (other than modifications to set, area, and record definitions). These modifications include:

- Description
- Program registration
- Authorization
- Usage
- Information on transferring statistics
- Logical record currency
- Security
- User-defined information (class/attribute and user-defined comments)

**What components are affected:** The definition of the subschema as it resides in the dictionary is affected by such modifications.

**Example:** In the following example, program registration has been turned on. This requires that all programs using this subschema be registered with the named subschema in order to be compiled against it.

```
modify subschema empss01
  program registration required is on.
generate.
```

### 32.1.2 Deleting a subschema

**When you might want to make this change:** If there is no longer a need for a subschema, you may want to delete it.

**What components are affected:** The subschema source is affected by the deletion of a subschema. The subschema load module is not affected.

#### Considerations

- When you delete a subschema, programs associated with that subschema can no longer be compiled. You must associate each program with a new subschema.
- If you have not specified DELETE IS ON in the SET OPTIONS statement, the subschema load module is not automatically deleted when you delete the subschema definition. You must explicitly delete the associated load module.

**Example:** In the following example, the subschema EMPSS01 is deleted.  
`delete subschema empss01.`

## 32.2 Adding, modifying, or deleting a record

**What components are affected:** The record definition portion of the subschema is affected by such a modification.

### Considerations

- If you include a new record in the subschema and that record participates in a mandatory automatic set, you have to include the owner of that set in the subschema (or the set itself) so that application programs using the subschema can store occurrences of the new record.

When you regenerate the subschema, you will receive a notice of an access restriction.

- If you modify a record so that some elements are omitted, you may have to modify and regenerate maps for online programs as well as the programs or dialogs themselves.
- If you add a record that is stored in an area not currently participating in this subschema, you must add that area to the subschema if a program is to access the new record. You will receive an error on generation if the area is not added.

## 32.3 Adding, modifying, or deleting a set

**What components are affected:** The set definition portion of the subschema is affected by such a modification.

### Considerations

- If you do not add the set owner and member record types to the subschema, your program cannot access the new set.
- If you add the set and the set member record type but not the owner record type, the application program will not be able to obtain the owner of the set.
- If you add the set and the set owner record type but not the member record type, the application program will not be able to walk the set.
- If you delete a set but not its owner or member record type, currency will not be maintained for that set and, although an application program can access the owner, it cannot walk a set to obtain all members. In addition, the application program cannot connect a member into the set, and, if the set is mandatory automatic, the application program cannot store a new record occurrence.

## 32.4 Adding, modifying, or deleting an area

**When you might want to make this change:** You can add areas to or delete areas from a subschema or make a modification to an existing area. Normally this is done in conjunction with adding or deleting a record or index structure stored in that area or to move records into a new area for performance reasons.

**What components are affected:** The area definition portion of the subschema is affected by such a modification.

### Considerations

- If you modify the usage mode so that a mode is no longer allowed, you may have to modify the READY mode of your program to match.
- If you modify the default usage mode, you should check programs using the subschema to see that there is no conflict.
- If you delete an area, make sure that there are no records or indexes mapping to that area still in the subschema.
- If an area is renamed or deleted, all ADS dialogs that use the subschema must be recompiled if they use neither READY ALL nor DBMS autoready.

## 32.5 Adding, modifying, or deleting a logical record or path group

**What components are affected:** Only the definition of either the logical record or a path group is affected.

### Considerations

- If you modify a logical record so that some elements are omitted, you may have to modify and regenerate maps for online programs as well as the programs themselves.
  - If you remove a path group from the subschema, you must modify and recompile any program or dialog associated with that subschema using the deleted path group.
  - If you have changed the selection criteria in a path, you need to modify the program requests in programs or dialogs associated with that subschema.
- For complete information on Logical Record Facility, refer to the *CA-IDMS Logical Record Facility*.



# Chapter 33. Space Management

---

- 33.1 About space management . . . . . 33-3
- 33.2 Database pages . . . . . 33-4
- 33.3 Database keys . . . . . 33-7
- 33.4 Area space management . . . . . 33-10
  - 33.4.1 SR1 records . . . . . 33-11
  - 33.4.2 Space management pages . . . . . 33-12



## 33.1 About space management

**Definitions of areas and pages:** A CA-IDMS database contains one or more areas. Each **database area** is a named subdivision of addressable storage in the database. A CA-IDMS area is subdivided into **database pages**. Most database pages are used to hold actual record occurrences (or rows). Some pages are reserved by CA-IDMS for space management.

**Note:** Record occurrences and rows of an SQL-defined table are stored in the same way in a CA-IDMS database. For simplification, the term record occurrence will be used to indicate both row and record occurrence, and record type to indicate both table and record type.

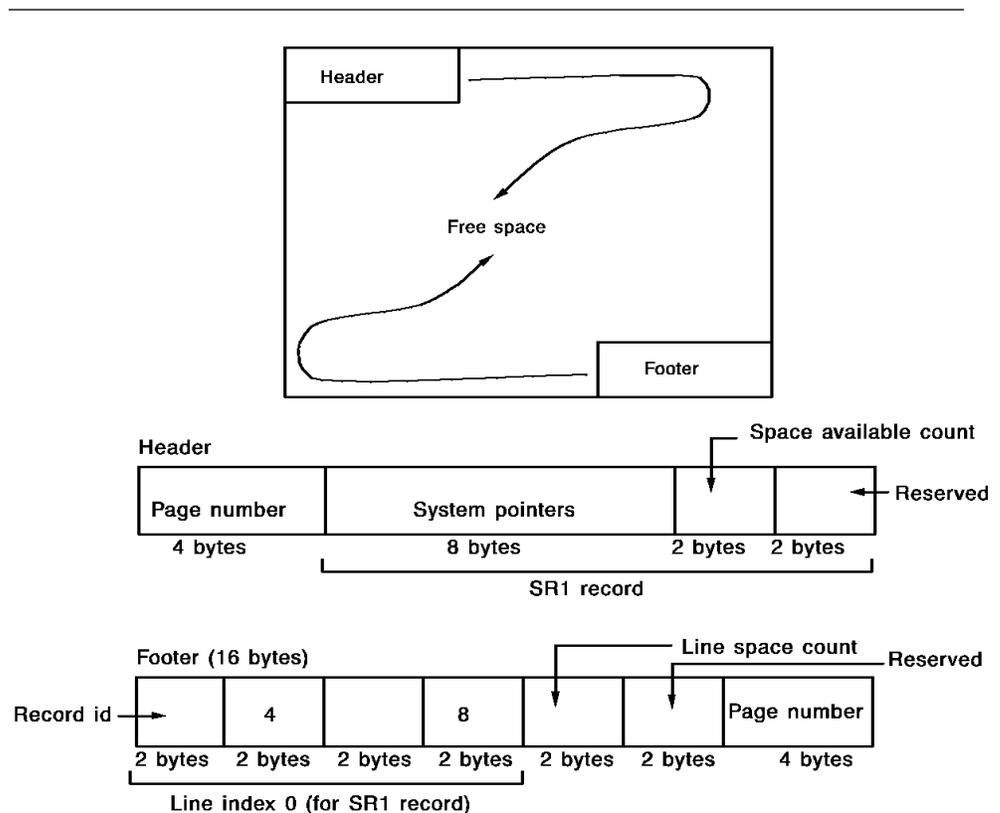
**Definition of database key:** Each record occurrence in a CA-IDMS database is uniquely identified by a **database key (db-key)** that specifies the physical location of the occurrence. Database keys are used as pointers to related record occurrences or index records.

The format of a database key can vary from database to database. The variable format of the db-key allows you to tailor space management factors to different processing requirements.

## 33.2 Database pages

**Size of database:** A database can have from 2 to 1,073,741,822 pages. Each area contains pages of equal size. Each page can contain up to 32,756 bytes of data. For details, see 33.3, “Database keys” on page 33-7 later in this chapter. Database pages are mapped to BDAM, DAM, or PAM blocks, or VSAM control intervals (for details, see Chapter 16, “Allocating and Formatting Files” on page 16-1). Each database page is identified by a unique **page number** and data transfers are accomplished one page at a time.

**Page format:** All database pages, regardless of size, have a header and footer with the same general format as shown in the diagram below. A database page always has a header at the beginning of the page and a footer at the end; free space is in the middle.



**Header:** The header occupies the first 16 bytes of each page and is formatted as follows:

- **Page number** (4 bytes) — A unique, system-assigned number of the page.

- **SR1 system record** (12 bytes) — An SR1 record is stored on each page during initialization by the FORMAT utility. Each SR1 record contains the **space available count** (that is, the number of bytes of free space on the page).

**Footer:** The footer occupies the last 16 bytes of each page and is formatted as follows:

- **Line index 0** (8 bytes) — Identifies the location and length of the SR1 system record
- **Line space count** (2 bytes) — Number of bytes used for line indexes and the footer
- **Filler** (2 bytes) — Reserved space
- **Page number** (4 bytes) — The unique system-assigned number of the page

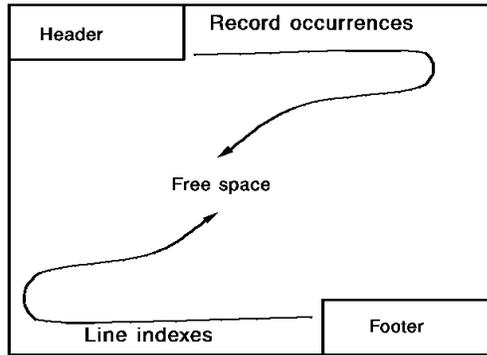
**Note:** Numeric fields maintained by CA-IDMS are in binary format, although this manual represents them as decimal numbers.

To simplify the illustrations, the page size (800 bytes) in the figures of this manual is unusually small.

**Database page layout:** Except for the header and the footer, pages are filled with the following entries:

- **Record occurrences** — The actual record occurrences are positioned on the page from top to bottom immediately following the header. Each occurrence consists of a **prefix** (containing pointers) and a **data portion**. A page can hold from 3 to 2,727 record occurrences depending on user specification (for details, see 33.3, “Database keys” on page 33-7 later in this chapter).
- **Line indexes** — The line indexes identify the locations of record occurrences on the page and are positioned on the page from bottom to top, immediately preceding the footer. A page contains one line index per record occurrence on the page. Each line index has the following format:
  - **Record id** (2 bytes) — Identification of the record type
  - **Displacement** (2 bytes) — Location of the record occurrence relative to the beginning of the page, where the first byte on the page is position 0
  - **Record length** (2 bytes) — Length of the entire record occurrence stored on this page (data plus prefix) in bytes
  - **Prefix length** (2 bytes) — Length of the prefix portion of the record in bytes

Record occurrences are added from the top down; line indexes from the bottom up. Free space is always in the middle.



**Record occurrences**



4 bytes per pointer

Size according to user specifications

**Line index (8 bytes)**

Record id	Displacement	Record length	Prefix length
2 bytes	2 bytes	2 bytes	2 bytes

---

---

## 33.3 Database keys

**Identify each record occurrence:** Each record occurrence in a CA-IDMS database is uniquely identified by a **database key (db-key)**, which indicates the occurrence's physical location in the database. A db-key is assigned when a record occurrence is stored in the database. The db-key never changes as long as the record remains in the database (that is, until the record is erased or until the database is unloaded and subsequently reloaded).

**Used as pointers:** Database keys are used as pointers to related record occurrences or index records. As such, database keys are found in the system-maintained prefixes that precede the data portion of the record occurrence. For example, a record occurrence's prefix may contain the database keys of the next, prior, and owner records of the chained set in which that occurrence is a member.

A db-key is a 4-byte (32 bit) binary number. The Database Management System (DBMS) creates a db-key for a record occurrence by concatenating the following numbers:

- **Page number** — The page on which the record occurrence is stored
- **Line number** — The position of the record occurrence's line index on the page relative to the other line indexes, where the line index for the SR1 record is line index 0

**Db-key format:** The db-key format is variable. The number of bits reserved in the db-key for the page number and line number, respectively, can vary from one physical database to another, as long as the total number of bits used is 32. You identify the db-key format to be used by specifying the maximum number of record occurrences to be stored on one database page in the CREATE SEGMENT statement.

**Default db-key format:** In the default db-key format, 24 bits are allocated for the page number and eight bits for the line number. This format allows a maximum of 16,777,214 pages in the database, with each page containing up to 255 record occurrences.

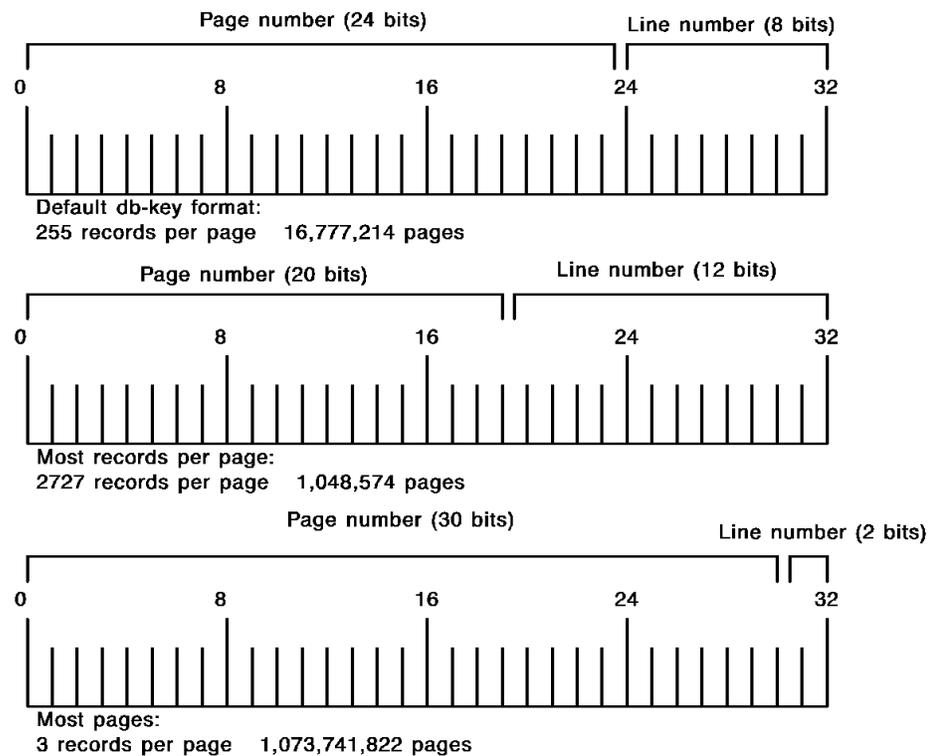
**Variable format:** The variable format of the db-key allows you to tailor space management factors to different processing requirements. For storage of small records, specify a database with many record occurrences per page and a smaller number of pages. For storage of large records, specify a database with few record occurrences per page but a large number of pages. For these different requirements, adjust the db-key format as follows:

- **To allow more record occurrences per page**, increase the number of bits for the line index. (The line number must be from 2 to 12 bits in length.)
- **To allow more pages per database**, increase the number of bits for the page number.

Note that as the number of record occurrences allowed on a page increases, the number of pages allowed in the database decreases. Conversely, the more pages in the database, the fewer occurrences each page can hold.

**Note:** The MIXED PAGE GROUP BINDS ALLOWED option for a DBNAME may be used to increase the number of records accessible in a database from a single database transaction.

The following diagram shows the db-key formats for a CA-IDMS database with three possible formats: 255 record occurrences per page (the default size); the greatest possible number of occurrences per page; and the greatest possible number of pages.



**Determining the db-key format:** Using the decimal value that you specify in the MAXIMUM RECORDS PER PAGE clause on the CREATE SEGMENT statement, CA-IDMS/DB determines the db-key format, as follows:

- **To determine the total possible number of line indexes for a page,**  
 CA-IDMS/DB adds 1 to the maximum number of record occurrences per page. (This number represents line index 0, reserved for the SR1 record.)
- **To determine the size of the line number portion of the db-key,** CA-IDMS/DB identifies the number of bits required to store the largest possible line index.

- **To determine the size of the page number portion of the db-key,**  
CA-IDMS/DB subtracts the number of bits for the line number from 32 (the total number of bits in a db-key).

For example, the default number of record occurrences per page is 255. In this case, the total number of line indexes is 256 (that is, line index 0 through 255). Since the decimal number 255 takes eight bits of storage in binary format, the line number portion of the db-key for this database is eight bits, and the page number portion is 24 bits.

**Note:** CA-IDMS uses all 32 bits of the db-key for the page number and the line number. If you want to reserve a bit in the db-key as a sign bit (that is, if you will write routines that perform arithmetic operations using the db-key sign bit), make sure that the db-keys created for your occurrences can be stored in only 31 bits.

**Conversion algorithms:** Use the following algorithms to convert a db-key into individual page and line numbers:

$$\underline{\text{dbkey-page}} = \underline{\text{dbkey} / 2^{**\text{bits-for-line}}}$$

$$\underline{\text{dbkey-line}} = \underline{\text{dbkey} - (\text{dbkey-page} * (2^{**\text{bits-for-line}}))}$$

where:

*dbkey* = the 4-byte binary database key

*dbkey-page* = the binary database page number

*dbkey-line* = the binary database line number

*bits-for-line* = the number of bits for the line number in the database key

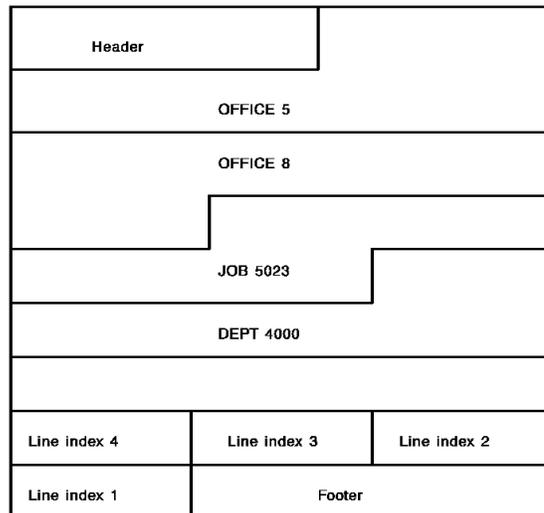
## 33.4 Area space management

**What is an area?:** A CA-IDMS database is divided into one or more areas. Each **database area** is a named subdivision of addressable database storage. Each area can contain one or more record types, according to varying processing requirements, but all occurrences of a particular record type must be in the same area.

**Managing space in an area:** To manage space in an area, CA-IDMS/DB keeps track of available space on each page. CA-IDMS reserves selected pages called **space management pages (SMPs)** for this purpose. The first page in each area is an SMP. Depending on the number and size of pages in the area, CA-IDMS may reserve additional SMPs throughout the area.

Since you frequently assign several record types to an area, data pages in these areas are typically filled with record occurrences of different record types and the occurrences' corresponding line indexes. For example, in the sample employee database, the DEPARTMENT, JOB, OFFICE, and SKILL records are all assigned to the ORG-DEMO-REGION area. Thus, occurrences of all of these record types can be stored on the same page.

**Sample page:** The drawing below shows a sample page in the ORG-DEMO-REGION. Typically, except for the header and footer, a page in an area is filled with occurrences of different record types. Page 7130 in the ORG-DEMO-REGION area contains occurrences of the OFFICE, JOB, and DEPARTMENT record types.



**Space available:** To manage space, CA-IDMS/DB keeps track of the available space on each page. The space available is maintained in the following locations:

- **SR1 records** — System records in each page's header which contain the space available count for the page
- **Space management pages (SMPs)** — One or more system-reserved pages which contain entries that indicate whether each page (in a range of pages) is empty or full

SR1 records and space management pages are discussed separately below.

### 33.4.1 SR1 records

Each database page in an area contains an SR1 record in the page header. Each occurrence of the SR1 record contains the space available count for that page. The SR1 record type is the owner of a set used by CA-IDMS/DB to keep track of CALC records (for details, see "Storing CALC records" in Chapter 35).

**SR1 record format:** The SR1 record is formatted as follows:

- **Next pointer for CALC set** (4 bytes) — Database key (next pointer) of the CALC record, targeted to that page, with the lowest CALC key
- **Prior pointer for CALC set** (4 bytes) — Database key (prior pointer) of the CALC record, targeted to that page, with the highest CALC key
- **Space available count** (2 bytes) — Number of bytes of free space remaining on the page
- **Filler** (2 bytes) — Reserved space

**Line index:** Every **line index 0** in an area identifies the location of an SR1 record and always contains the following values:

record identification = 1

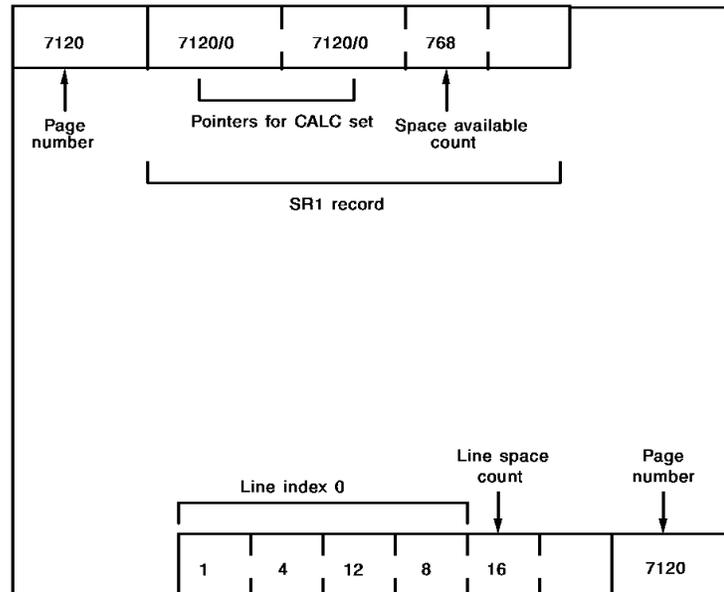
displacement = 4

record length = 12

prefix length = 8

The following diagram shows an empty page in an area. This is what a page would look like after initialization by the FORMAT utility.

Note that the **space available count** for an empty page is always the page size minus 32 (in this case,  $800 - 32 = 768$ ) and the **line space count** for an empty page is always 16. The CALC set pointers in the SR1 record on an empty page point back to the SR1 record itself since it is the only record in the set.



### 33.4.2 Space management pages

**What is a space management page?:** CA-IDMS reserves selected pages, called **space management pages (SMPs)**, to keep track of the available space on each page. These pages are filled with 2-byte items called **space management entries**. Each space management entry, depending on the entry's relative position on the page, corresponds to a page in the area. The first entry corresponds to the space management page itself, the second entry to the first page following the space management page, and so on.

**Number of pages managed by SMP:** The number of pages managed by one space management page is the page size minus 32 (header and footer) divided by 2 (two bytes per space management entry).

For example, a space management page for an area whose page size is 800 bytes holds 384 entries. The first entry is for the space management page itself. If the area contained 900 pages, the area would require three space management pages. The first space management page would be the first page in the area, the second would be the 385th page, and the third would be the 769th page.

**FORMAT utility initializes SMP entries:** For pages that will contain record occurrences, the **FORMAT utility initializes space management entries** to a value equal to the page size of the area minus the number of bytes used by the header and footer (that is, the amount of usable space on each page). The first space management entry is for the space management page and is initialized to zero. In the above

example, the space management entries for data pages would be initialized to a value of 768.

**Accessing space management pages:** After initialization, space management pages are accessed only in the following situations:

- **STORE command** — If CA-IDMS/DB cannot store a record occurrence on the target page because insufficient space exists on that page, the space management page is consulted for the next page that has sufficient space. Further, if the space available count field on the target page shows that more than 70 percent of the usable space is used, the space management page is accessed and the space management entry is changed to the actual space available. Also, if CA-IDMS/DB uses the last available line index on a page to store a record, a halfword of 2 is entered in the space management entry, indicating that the page is **logically full**.
- **ERASE command** — When the actual space available for a page is shown in the space management entry (that is, when the page is more than 70 percent full) and a record occurrence is deleted from the page, CA-IDMS/DB accesses the space management page and does one of the following:
  - If the page is still more than 70 percent full, CA-IDMS/DB moves the new space available count from the page to the space management entry.
  - If the page is now less than 70 percent full, CA-IDMS/DB reinitializes the space management entry to the value of the page size minus the length of the header and the footer (that is, the decimal value 32).

**Actual space available:** The actual space available for each page is not maintained constantly to avoid accessing the space management page each time a record is stored or erased. Instead, a page is considered empty (for space management purposes) until either of the following conditions occurs:

- A store operation for a record occurrence puts the space used over the 70 percent threshold.
- All line indexes on that page have been used (that is, the page is logically full).

A page returns to the empty status when an erase operation puts the space used back below the 70 percent threshold.

Consequently, unless a large enough page size is specified, CA-IDMS/DB might attempt to store records that will not physically fit on a page.

Suppose, for example, that a page is 60 percent full and marked as empty in the space management page, and that a record occurrence being stored is 45 percent of the page size. Using information maintained in the space management page, CA-IDMS/DB would determine that the record occurrence could fit on the page, when it could not.

To ensure that CA-IDMS/DB can successfully store all records, **specify a page size that allows CA-IDMS/DB to store the largest fixed-length record on 30 percent of the page.**

**Determining minimum page size:** Use the following algorithm to determine minimum page size:

$$\text{min-page-size} = ((\text{record-length} + 8) / 0.30) + \text{head-foot-length}$$

where:

*min-page-size* = the decimal value of the minimum page size

*record-length* = the length of the largest fixed-length record type (data plus prefix)

**8** = the length of the line index

*head-foot-length* = the maximum length of a header and footer on a page; the decimal value 32

**Reporting on area space utilization:** The PRINT SPACE utility statement reports on:

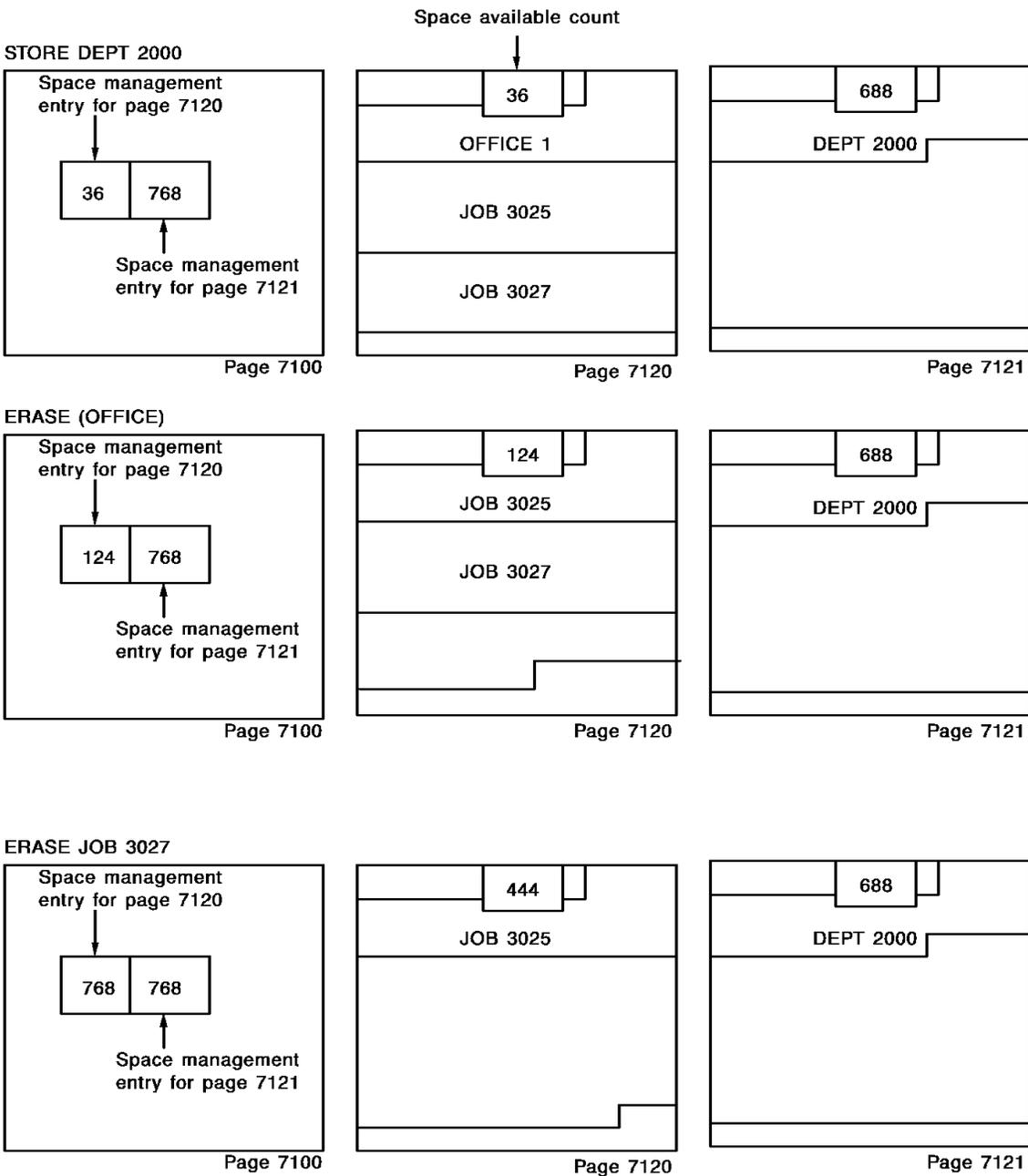
- Space utilization based on the contents of the SMPs
- With the FULL option, space utilization based on the actual contents of each database page (using the space available count)

**Use of the space management page:** The following diagram shows the use of the space management page.

CA-IDMS/DB changes the space management entry for page 7120 from 768 (the page size minus 32) to 36 (the actual number of bytes left on page 7120) after storing the JOB 3027 record. Thus, after consulting the space management page, CA-IDMS/DB knows that it cannot store the DEPT 2000 record (72 bytes long) on page 7120 because of insufficient space, and stores it on the next page.

When the OFFICE 1 record is deleted from page 7120, the page is still more than 70 percent full, so CA-IDMS/DB moves the value 124 (the actual amount of space available) to the space management entry.

When the JOB 3027 record is deleted, however, page 7120 is less than 70 percent full and the space management entry is reinitialized to 768 bytes.





# Chapter 34. Record Storage and Deletion

---

- 34.1 Record storage . . . . . 34-3
  - 34.1.1 Storing CALC records . . . . . 34-4
  - 34.1.2 Clustering records . . . . . 34-7
    - 34.1.2.1 Clustering records around a chained set . . . . . 34-7
    - 34.1.2.2 Storing records via an indexed set . . . . . 34-9
  - 34.1.3 Storing variable-length records . . . . . 34-11
  - 34.1.4 Relocated records . . . . . 34-14
- 34.2 Record deletion . . . . . 34-16
  - 34.2.1 Physical deletion . . . . . 34-16
  - 34.2.2 Logical deletion . . . . . 34-18



---

## 34.1 Record storage

**Determining the target page:** To store a record in the database, CA-IDMS/DB first determines a **target page**. Storage mode specifications govern the selection of the target page, as follows:

- In **CALC** storage mode, CA-IDMS/DB calculates the number of the target page by executing a randomizing routine against the CALC key.
- In **VIA** or **CLUSTERED** storage mode, which is used to store related record occurrences (or rows) on the same page or on as few pages as possible, CA-IDMS/DB determines the number of the target page from:
  - For non-SQL, the number of the page that contains the current record of the VIA set
  - For SQL, the referenced row of a clustered constraint
- In **DIRECT** storage mode, the user explicitly specifies the target page. (Note that if you specify the value -1, the target page is the first page assigned to the record type.)

**Storing the record occurrence:** If the target page has sufficient space to store the entire record occurrence (fixed-length uncompressed records) or the record's minimum root, CA-IDMS/DB then stores the record occurrence on the target page. If the target page does *not* have sufficient free space to store the record occurrence, CA-IDMS/DB stores the record occurrence on the next page that has sufficient space. The search for free space always proceeds in a forward (higher database key) direction. If the end of the area (or the page range assigned to the record type) is reached before space is located, the search wraps around to the beginning of the area (or the page range assigned to the record type).

After identifying the first available free page, CA-IDMS/DB performs the following operations to store a record occurrence:

- **Creates a line index** and positions it at the end of the free space or an unused line index.
- **Positions the prefix and data** (as retrieved from the program variable storage) at the beginning of the free space.

When storing a fixed-length uncompressed record, CA-IDMS/DB places the entire record occurrence on the target page. When storing a variable-length record occurrence, CA-IDMS/DB places as much of the record occurrence as possible on the target page. (For details, see 34.1.3, “Storing variable-length records” on page 34-11, later in this chapter.)

- **Updates the space available count** in the header and the **line space count** in the footer.
- **Updates the record's pointers** as follows:
  - Updates the pointers for all user sets in which the record is an automatic member

- Sets the pointers to null (-1) for all sets in which the record is a manual member
- Sets the pointers to the database key of the object record itself for all owner records (indicating an empty set)
- For SQL, sets the pointers to null (-1) for linked constraints in which the table is the referencing table if one or more columns of the foreign key are null; otherwise, sets the pointers to the db-keys of related rows
- For SQL, sets the pointers to the database key of the object row itself for linked constraints in which the table's the referenced table

- **Updates the record's CALC set pointers** (if any).
- **Updates the pointers in all other records affected** by the stored record's automatic (and CALC, if applicable) set connections.

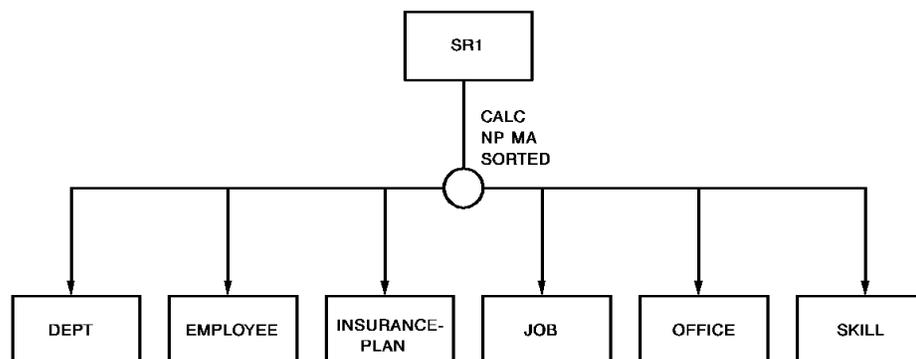
For example, if record B2 is being stored between records B1 and B3 in set A-B, B2's next pointer is set to B3's database key, while B2's prior pointer is set to B1's database key. Additionally, B1's next pointer is changed from B3's database key to B2's, and B3's prior pointer is changed from B1's database key to B2's.

### 34.1.1 Storing CALC records

**Stored on or near calculated page:** CA-IDMS/DB stores records that have a storage mode of CALC on or near the page calculated from the record's CALC key (a schema-specified symbolic key). CA-IDMS/DB uses the system-owned **CALC set** to keep track of all CALC records that randomize to a specific page. The CALC set's owner is the system-owned SR1 record type. The CALC set's members are all of the user records with a storage mode of CALC. The set is sorted in ascending sequence on the CALC key of each member record occurrence.

**Example of a system-owned CALC set:** The following diagram shows the system-owned CALC set for the sample employee database.

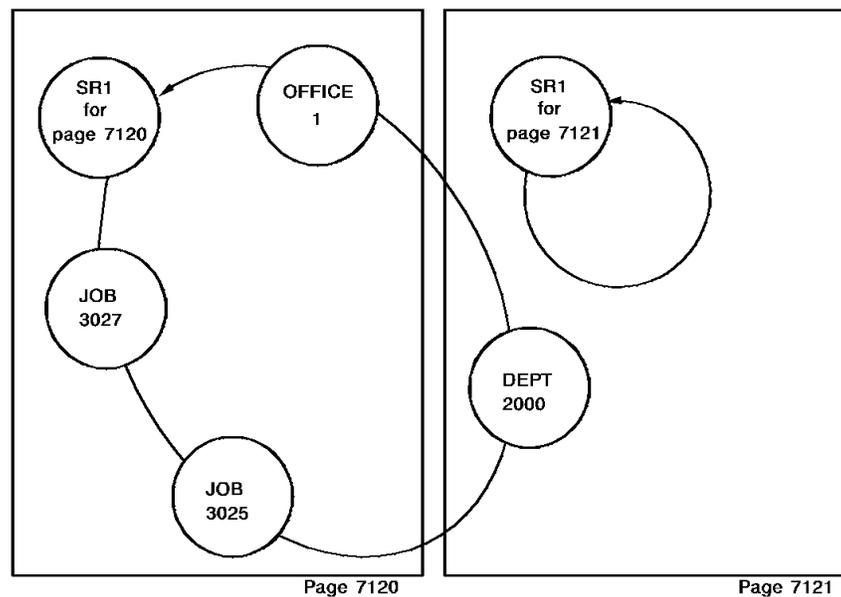
**Note:** The system-owned CALC set is an internal set. It should not be included in the user's schema or in structural diagrams.



**One system-owned CALC set per database:** There is one system-owned CALC set type per database; there is one CALC set occurrence for each page in the area. The CALC set is sorted in ascending sequence based on the CALC key of each member occurrence.

**SR1 system record:** On a page that contains record occurrences, the SR1 record on a data page owns all CALC records that randomize to that page at store time, including records that end up on another page due to overflow conditions.

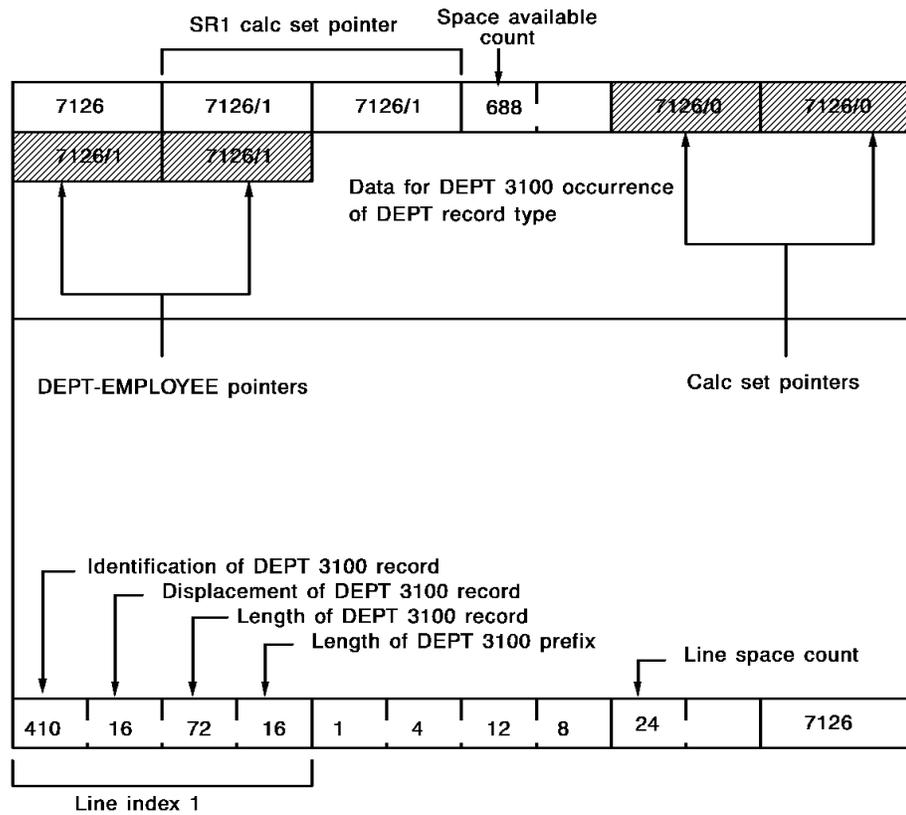
The diagram below shows the format and occurrences of the CALC set on page 7120 of the sample database. The CALC set for page 7120 includes all CALC records randomized to that page. Note that DEPT 2000 belongs to the CALC set for page 7120 even though DEPT 2000 was actually stored on page 7121 (due to lack of space on its target page).



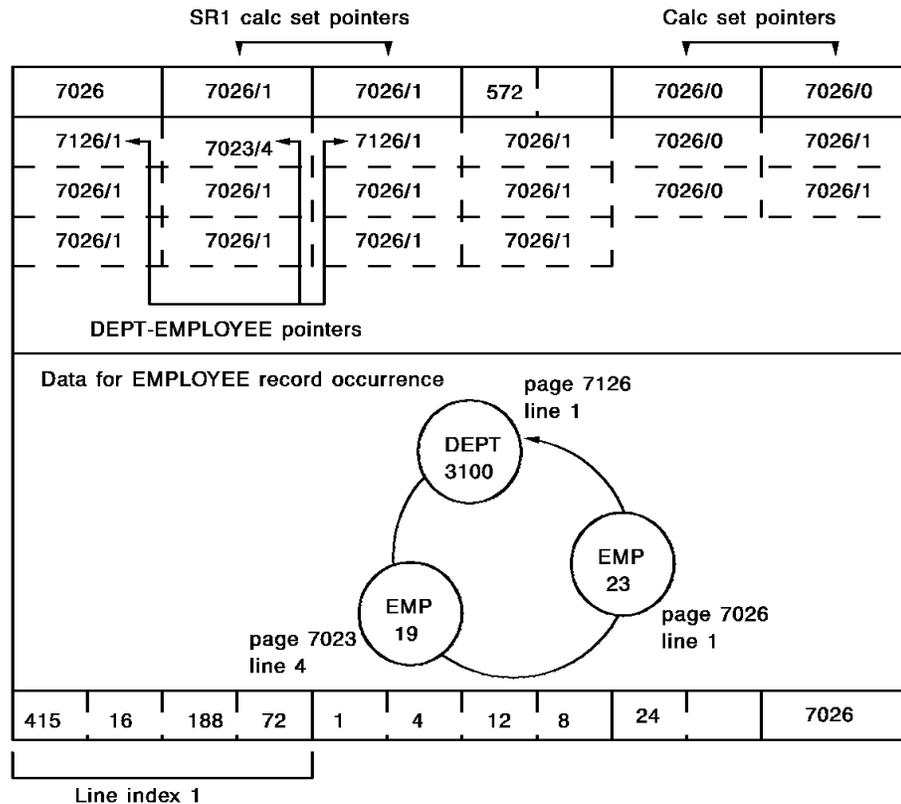
**Retrieving a CALC record:** To retrieve a record occurrence stored CALC, CA-IDMS/DB accepts from the user the value of the record's CALC key and calculates a page number from the key. CA-IDMS/DB then enters this database page on the SR1 record and follows the page's CALC chain until either the requested record is located or a record of the same type with a higher key value is located; in the latter case, CA-IDMS/DB returns an error status of 0326 (record not found) to the user.

**Storing a CALC record:** In adding the DEPT 3100 record to page 7126, CA-IDMS/DB creates a record prefix (shaded portion) that consists of pointers for the CALC set and for the DEPT-EMPLOYEE set. The prefix and data (as found in program variable storage) are positioned at the beginning of the free space. A line index is created at the end of the free space. The space available count is decremented, and the line space count is incremented.

Note that the CALC pointers in the SR1 record are updated to point to the DEPT 3100 record, while the CALC pointers in the DEPT 3100 record are set to point to the SR1 record. All other pointers in the DEPT 3100 record point back to the record itself because its DEPT-EMPLOYEE set occurrence is empty.



**Storing another CALC record:** The EMPLOYEE 23 record randomizes to and is stored on page 7026. The prefix of the EMPLOYEE 23 record supplies the following information: EMPLOYEE 23 (the only member of the CALC set on page 7008) and EMPLOYEE 19 are the only members of the DEPT-EMPLOYEE set for OFFICE 3100; EMPLOYEE 19 is next of set in the DEPT-EMPLOYEE set for DEPT 3100; all of the set occurrences that EMPLOYEE 23 owns are empty.



## 34.1.2 Clustering records

In the VIA or CLUSTERED storage mode, CA-IDMS/DB stores related records together on the same page or on as few pages as possible. A record can be clustered through a chained set (a linked clustered constraint), an indexed set (a clustering index), or an unlinked constraint (SQL only).

### 34.1.2.1 Clustering records around a chained set

**Storage strategy:** If a record has a storage mode of VIA a chained set (or CLUSTERED around a referential constraint), CA-IDMS/DB uses the location of the current record of set (always the referenced row of referential constraints) to determine where to store the new record, as follows:

- If the current record of set is a member of the set, the DBMS stores the new record as close as possible to the current record of set.
- If the current record of set is an owner of the set, CA-IDMS/DB determines where to store the member record, as follows:

---

If the members and owners in the specified set are assigned to the same page range, and if you have not specified displacement in the non-SQL schema...	CA-IDMS/DB stores the member record occurrence as close as possible to the owner
If the members and owners in the specified set are assigned to the same page range, and you have specified displacement in the non-SQL schema...	CA-IDMS/DB stores the member record occurrence as close as possible to the owner, allowing for displacement
If the members and owners in the specified set are assigned to different page ranges...	CA-IDMS/DB stores the member record occurrence as close as possible to the page (within the member record's page range) that is proportional to the location of the owner (within the owner's page range)

---

The following diagram shows how CA-IDMS/DB stores a record via a chained set. For a discussion of how CA-IDMS/DB stores a record via an indexed set, see 34.1.2.2, “Storing records via an indexed set” on page 34-9, later in this chapter.

**Example:** In this example, EMPLOYEE 23 has randomized to page 7026. EMPLOYEE 23's EMPOSITION record is stored VIA EMPLOYEE 23 on page 7026. To locate the EMPOSITION record, CA-IDMS/DB applies the randomizing routine to EMPLOYEE 23 (giving page number 7026), enters page 7026 on the SR1 record, and follows the CALC set until the EMPLOYEE 23 record is located. CA-IDMS/DB then obtains the EMPOSITION record through the EMP-EMPOSITION chain.

7026	7026/1	7026/1	524	7026/0	7026/0
- - + - - -	- - - -	- - - -	- - - -	- - - -	- - - -
- - + - - -	- - - -	7026/2	7026/2	- - - -	- - - -
- - - -	- - - -	EMP-EMPOSITION pointers		- - - -	- - - -
Data for EMPLOYEE 23 record occurrence					
7026/1	7026/1	7026/1			
EMP-EMPOSITION pointers					
Data for EMPOSITION for EMPLOYEE 23					
				420	204
				40	12
415	16	188	72	1	4
				12	8
				24	7026

### 34.1.2.2 Storing records via an indexed set

**Storage order:** Indexed sets can be used to store member records in a physical order that reflects the order of the member's db-key or symbolic key in the index, by defining the member record's storage mode as via (or clustered) an indexed set that is sorted on db-key or symbolic key.

**Determining the target page:** CA-IDMS/DB determines the target page on which to store a member occurrence via an indexed set, as follows:

---

If this is the first record occurrence stored via a user-owned index set or a system-owned index with the same page range as the member record...

CA-IDMS/DB determines the target page as follows:

- If the member or owner in the set are assigned to the same page range, CA-IDMS/DB stores the member record occurrence as close as possible to the owner record (allowing for record displacement if specified).
- If the member and owner in the set are assigned to different page ranges, CA-IDMS/DB stores the member record as close as possible to the page (within the member's page range) that is proportional to the location of the owner (within the owner's page range).

---

If this is the first record occurrence stored via a system-owned index with a separate page range from that of the member...

The target page is the low page of the member's page range

---

If other record occurrences have already been stored (that is, if the index is *not* empty)...

CA-IDMS/DB determines the target page, as follows:

- If the set is sorted by db-key, CA-IDMS/DB finds the highest db-key of a record that is already a member of the indexed set, and uses the page specified in this db-key as the target page.
- If the set is sorted by symbolic key, CA-IDMS/DB determines the target page for the new record as follows:
  - Identifies the SR8 record that will hold the symbolic key for the new record
  - Finds the db-key of the record with the preceding or following symbolic key in the index and uses the page specified in this db-key as the target page

---

**Example:** For example, the EMP-EXPERTISE set in the sample order entry database is an indexed set, and EXPERTISE records are stored in physical-sequential order based on the value of the SKILL-LEVEL field. The non-SQL schema DDL statements necessary to specify physical-sequential placement of the EXPERTISE record are as follows:

```

RECORD NAME EXPERTISE
      LOCATION MODE VIA EMP-EXPERTISE SET ...

SET NAME EMP-EXPERTISE
      ORDER SORTED
      MODE INDEX ...
      OWNER EMPLOYEE
      MEMBER EXPERTISE ...
      DESCENDING KEY SKILL-LEVEL ...

```

In this case, CA-IDMS/DB stores each EXPERTISE record as close as possible to the record with the next lower SKILL-LEVEL.

### 34.1.3 Storing variable-length records

**Types of variable-length records:** Internally, CA-IDMS/DB treats the following types of records as variable-length:

Fixed-length compressed records	Records with a fixed length that are compressed through a specified compression routine. Although the length of these record types is fixed from the point of view of user programs, compression makes them internally variable.
Variable-length records	Records (either compressed or uncompressed) the length of which depends on a variable field (that is, records that contain an OCCURS DEPENDING ON clause).

Since you cannot anticipate the total length of either of these types of records, specify, in the schema, the following information:

- **The record's minimum root** — The smallest amount of the data to be stored on the record's home page
- **The record's minimum fragment** — The smallest amount of data to be stored on any additional page

**Steps to store a variable-length record:** Using the values specified for minimum root and minimum fragment, CA-IDMS/DB performs the following steps to store a variable-length record:

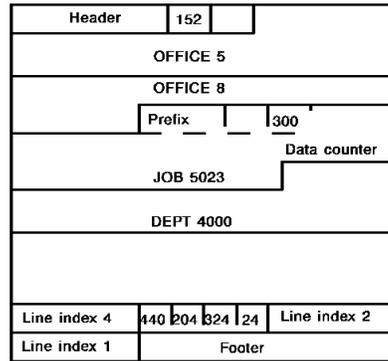
1. CA-IDMS/DB stores either the entire record or as much of the record as possible on the target page (provided that the space available is sufficient for the minimum root specification in the schema). This page, the first page on which CA-IDMS/DB stores either the entire record or a portion of the record, is referred to as the record's **home page**; the portion of the record placed on the home page is called the **root**.
2. CA-IDMS/DB stores the remainder of the record on subsequent pages, by working in a forward direction and wrapping around to the beginning of the area (or the page range assigned to the record), if necessary. Each subsequent portion of the

record that exists on a separate page is called a **fragment**. No fragment except the last one will be less than the schema minimum fragment specification.

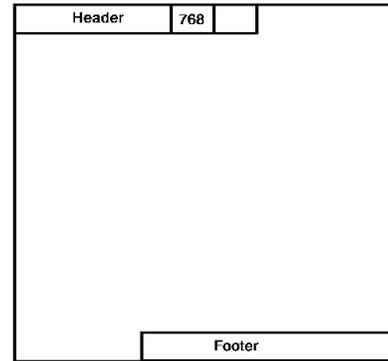
**Variable-length indicator:** In the root, CA-IDMS/DB places an extra pointer at the end of the prefix to point to the first fragment. At the beginning of the data portion of the root, CA-IDMS/DB adds a 4-byte **variable-length indicator (VLI)**. The VLI contains a 2-byte counter used to keep track of the size of the data portion of the entire record (including four-bytes for the VLI). The record-length field in the line index for a root segment contains the length of the portion of the record (prefix and data) that is stored on the home page.

**SR4 system record:** Each fragment contains a one-pointer prefix that points to the next fragment; the last fragment points back to the root. Fragments are placed on a page in the same manner as any record. A fragment is considered an **SR4 system record**; the record-id field in the line index of a fragment is always set to a value of 4.

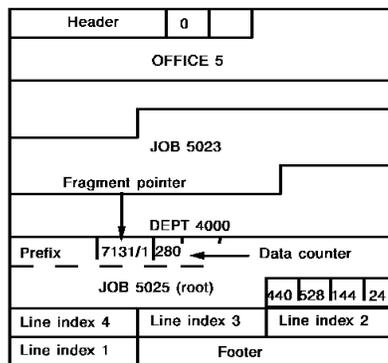
**Storing a variable-length record:** In the example below, the JOB 5023 record fits entirely on page 7130; because the JOB record is a compressed record, it is a variable-length record and CA-IDMS/DB includes a 4-byte variable-length indicator (VLI) in it, bringing the total data length of the record to 300 bytes. CA-IDMS/DB cannot store the entire JOB 5025 record on page 7130; however, the page does have sufficient space for a root. CA-IDMS/DB stores the root portion of JOB 5025 on page 7130 and includes a VLI, bringing the data portion of the entire record to 280 bytes. CA-IDMS/DB stores the remainder of the record on page 7131 as a fragment. Note that the record-id field for the last line index on page 7131 is 4, indicating that the record is a fragment.



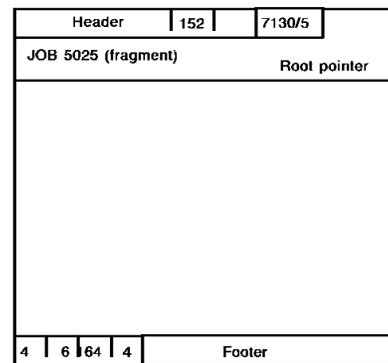
Page 7130



Page 7131



Page 7130



Page 7131

**Returning fragments to the home page:** On future accesses (GET, OBTAIN, or SELECT) of a fragmented variable-length record, CA-IDMS/DB may reduce the number of fragments. If the area is readied in update mode and the home page has sufficient space to hold the entire record, CA-IDMS/DB returns the fragments to the page. The fragments (minus fragment pointers) are concatenated to the root and physically deleted from the pages on which the fragments were located; the fragment pointer in the root is set to point to itself. Adjustments are made to the space available count in the page header and to the record length in the record's line index.

**Page reserve:** When the size of a variable-length record is increased by a DML MODIFY command, CA-IDMS/DB may create additional fragments for the record. If you anticipate a general increase in the size of variable-length records in an area, specify a **page reserve** for the area to decrease the possibility that CA-IDMS/DB will create fragments.

A page reserve sets aside a specified number of bytes on each database page in an area for modification of variable-length records. CA-IDMS/DB cannot use this reserved space to store any kind of record.

Specify an area's page reserve in the physical database definition(s) for the area using either a CREATE AREA statement or in an area override statement within the DMCL(s) that include the area's segment. An adequate page reserve is typically 30

percent of the area's size. Use the following criteria to estimate the size of the page reserve:

- The likelihood that variable-length records will be modified
- The anticipated increase in the number of variable-length records

When you specify a page reserve, you do not affect the physical structure of the database. In fact, you can vary the page reserve for an area by using (at different times) several DMCL modules with different page reserves.

### 34.1.4 Relocated records

**Records relocated because of increased size:** When increasing record sizes in areas, the RESTRUCTURE SEGMENT utility statement may occasionally relocate a record if the record no longer fits on its **home page**. Similarly, if a table has been altered to add one or more columns, CA-IDMS/DB may relocate a row when it is next updated because it will no longer fit on its original page. The dictionary migration utility (RHDCMIG1 and RHDCMIG2) may also relocate records. When a record is stored on a new page, the relocated record is considered an **SR3 system record** and the line index created for the record on the new page contains a record id of 3.

**Record identified by SR2 system record:** To preserve the integrity of the record's database key, CA-IDMS/DB leaves an 8-byte control record (an **SR2 system record**) on the home page in place of the relocated record. The SR2 system record has a record id of 2 and contains the following information about the relocated record:

- **Database key** (4 bytes) — The pointer (db-key) to the new location of the relocated record
- **Record id** (2 bytes) — The original record id of the relocated record
- **Length** (2 bytes) — The **total length** (fixed-length records) or **root length** (variable-length records) of the relocated record

**Returning relocated record to its home page:** On future accesses (GET, OBTAIN, or SELECT) of a relocated record, CA-IDMS/DB may return the relocated record to its home page. If the area is readied in update mode and the home page has sufficient space to hold the relocated record, CA-IDMS/DB returns it to the page.

In the example below, the OFFICE 1 record, increased in size by RESTRUCTURE SEGMENT, is moved from page 7120 to 7121.

Original configuration

Header		
OFFICE 1		
JOB 3025		
JOB 3027		
Line index 3	Line index 2	
Line index 1	Footer	

Page 7120

Header	
DEPT 2000	
Line index 1	Footer

Page 7121

After restructuring

Control record

Header				7121/2	450	94
JOB 3025						
JOB 3027						
Line index 3				Line index 2		
2	16	8	0	Footer		

Page 7120

Header			
DEPT 2000			
OFFICE 1			
3   88   94   16			
Line index 1	Footer		

Page 7121

## 34.2 Record deletion

**Operations performed:** To erase a record (or delete a row), CA-IDMS/DB performs the following operations:

- **Disconnects and/or erases all records owned by the object record**, depending on the nature of the ERASE DML command issued by the program (that is, ERASE, ERASE ALL, ERASE PERMANENT, or ERASE SELECTIVE).  
**Note:** When using SQL, the row must not be referenced by any other row.
- **Disconnects the object record from all indexed sets in which it participates as a member.**
- **Disconnects the object record from all chained sets (with prior pointers) in which it participates as a member.**
- **Deletes the record** either physically or logically, as follows:
  - If all chained sets in which the record participates as a member have prior pointers, CA-IDMS/DB **physically** deletes the record.
  - If any of the chained sets in which the record participates as a member do *not* have prior pointers, CA-IDMS/DB **logically** deletes the record.

**Note:** If CA-IDMS/DB has identified the prior record in each chained set (without prior pointers) in which the record participates (for example, walking the set), CA-IDMS/DB **physically** deletes the record.

**Note:** All linked clustered constraints have prior pointers.

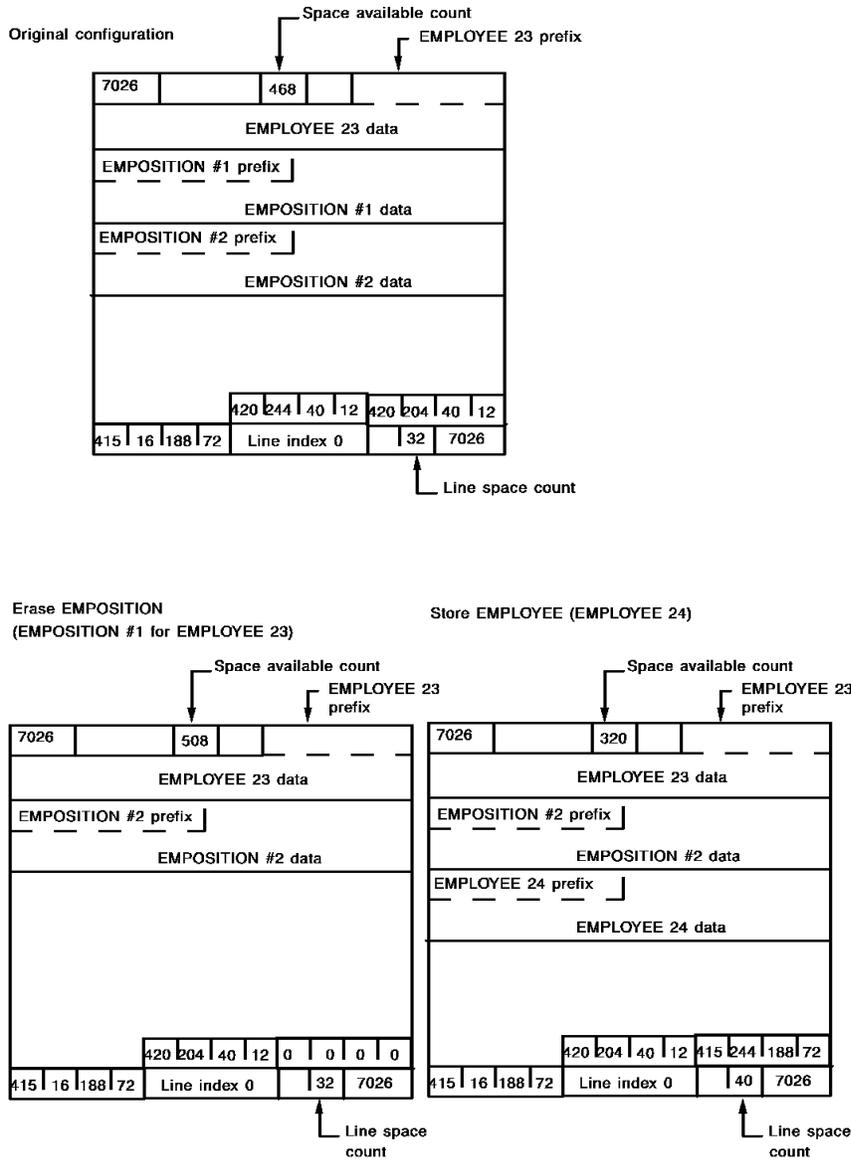
### 34.2.1 Physical deletion

**Operations performed:** CA-IDMS/DB performs the following operations to physically delete a record:

1. Removes the record's data and prefix from the database.
2. Moves all records following the deleted record on the page, so that all free space remains in the middle of the page.
3. Performs the following operations, depending on the location of the record's line index on the page:
  - If the line index is contiguous with the free space on the page (that is, if the record's line index is the last index on the page), CA-IDMS/DB removes the line index and updates the line space count in the footer.
  - If the record's line index is *not* contiguous with the free space on the page, CA-IDMS/DB sets the record's line index to zeros.
4. Updates the space available count in the header.

**Example:** In this example, the first EMPOSITION record for EMPLOYEE 23 has prior pointers. In erasing the record, CA-IDMS/DB removes the record completely (data and prefix), shifts the remaining EMPOSITION record up on the page, and sets

the line index for the deleted record to zeros. The remaining EMPOSITION record, although now physically the second record on the page, remains as line number 3. Line index 2 is reused when a new record is added to the page.



**Use of record's line index:** Line indexes cannot be shifted down because the position of the line index relative to other line indexes determines the line number, and changing a record's line number would invalidate the record's database key. Existing line indexes for physically deleted records are either reused as new records are added to the page (as shown in the diagram above or removed as further deletions make them contiguous to the free space.

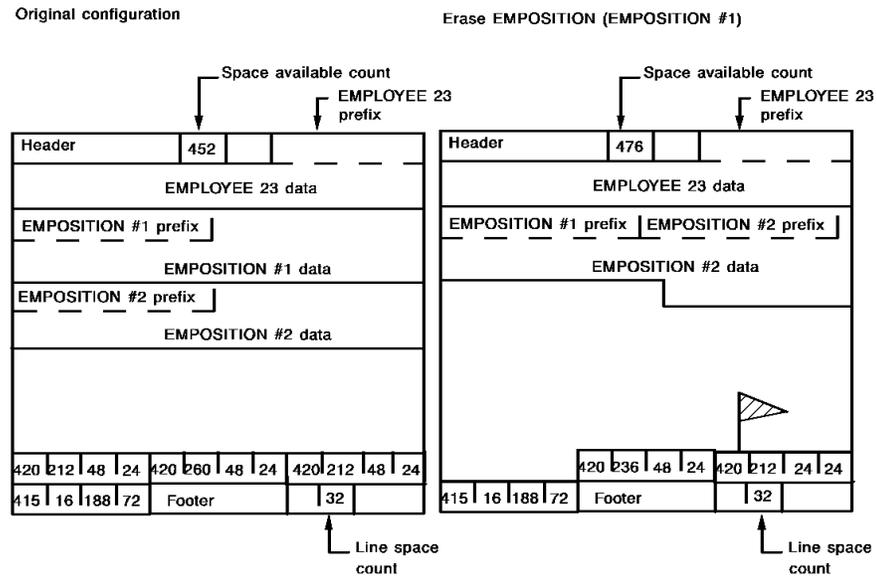
## 34.2.2 Logical deletion

**Pointers deleted:** To avoid consuming unnecessary time and I/O disconnecting records from sets without prior pointers, CA-IDMS/DB does not physically delete the record when an ERASE command is issued. Instead, the next time CA-IDMS/DB encounters a logically deleted record while walking a chained set of which the record is a member, CA-IDMS/DB disconnects the record from the set, provided that the record's area was readied in update mode. Since the record prior to the logically deleted record is still current of run unit, CA-IDMS/DB can update the record's next pointer and disconnect the logically deleted record. In order to be physically deleted, the record must have been disconnected from all sets in which the record was a member.

**Operations performed:** CA-IDMS/DB performs the following operations to logically delete a record:

- Removes the record's data from the database, but leaves the prefix
- Moves all records following the deleted record on the page, so that all free space remains in the middle of the page
- Sets the logical delete flag (the first bit) in the record id field of the record's line index
- Updates the space available count field in the header

**Example:** In the following example, assume that the EMPOSITION records do not have prior pointers in the EMP-EMPOSITION set. When erasing an EMPOSITION record, CA-IDMS/DB removes only the data and flags the record's line index. The EMPOSITION record is logically deleted. The next time CA-IDMS/DB is walking this occurrence of the EMP-EMPOSITION set in update mode and encounters the flagged record, CA-IDMS/DB physically deletes the record.



**Consideration:** Occasionally, in recovering from an error during a store operation, CA-IDMS/DB may create a logically deleted record. If CA-IDMS/DB has stored a record and is in the process of making the automatic connections when CA-IDMS/DB discovers an error condition (for example, no currency established in one of the automatic sets), CA-IDMS/DB must erase the record being stored. If one of the chained sets to which the record has already been connected has next pointers only, CA-IDMS/DB logically deletes the record.



# Chapter 35. Chained Set Management

---

- 35.1 About chained sets . . . . . 35-3
- 35.2 Chained sets . . . . . 35-4
  - 35.2.1 Connecting records to chained sets . . . . . 35-5
  - 35.2.2 Disconnecting records . . . . . 35-6
  - 35.2.3 Retrieving records . . . . . 35-7



## 35.1 About chained sets

**Physically link record occurrences together:** Chained sets can be used to physically link related record occurrences together. In a chained set, a pointer in each member record occurrence's prefix contains the db-key of the logically next occurrence of the set.

**Defining a chained set:** Define a set as chained as follows:

---

Non-SQL schema definition	MODE IS CHAIN on the SET statement.
SQL schema definition	LINKED CLUSTERED on the CONSTRAINT statement. When a constraint is implemented as a chained set, the referenced table is the <i>owner</i> of the set and the referencing table is the <i>member</i> .

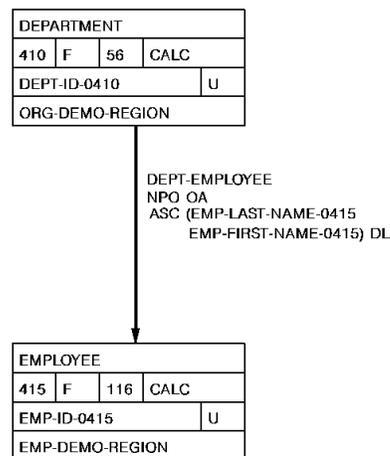
---

## 35.2 Chained sets

**Use:** A chained set is used to establish a logical relationship between two or more user-defined record types and consists of an owner record type and one or more member record types.

The following diagram uses standard CA-IDMS database notation to describe a chained set type; the diagram includes the name of the set, linkage options, membership options, sort sequence (if any), and sort key (if any).

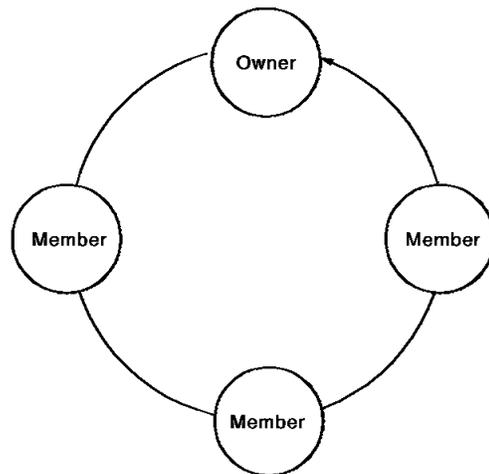
This example shows a chained set (the DEPT-EMPLOYEE set) between two user-defined record types. The owner of the DEPT-EMPLOYEE set type is the user-defined DEPARTMENT record type; the member is the EMPLOYEE record type.



**Next, prior, and owner pointers:** A **chained set occurrence** consists of one occurrence of the owner record type and any number of member record occurrences. The prefix of each record occurrence that participates in a set contains a **next** pointer (that is, the db-key of the next logical record occurrence in the set occurrence). Optionally, record occurrences can include **prior** pointers, which link records together in the logically prior direction, and **owner** pointers, which link member record occurrences to the owner occurrence.

**Note:** SQL-defined constraints implemented as a chained set always have next, prior, and owner pointers.

**Basic structure of a chained set occurrence:** A record occurrence in a chained set occurrence always contains in its prefix a next pointer that points to the logically next record occurrence in the set occurrence.

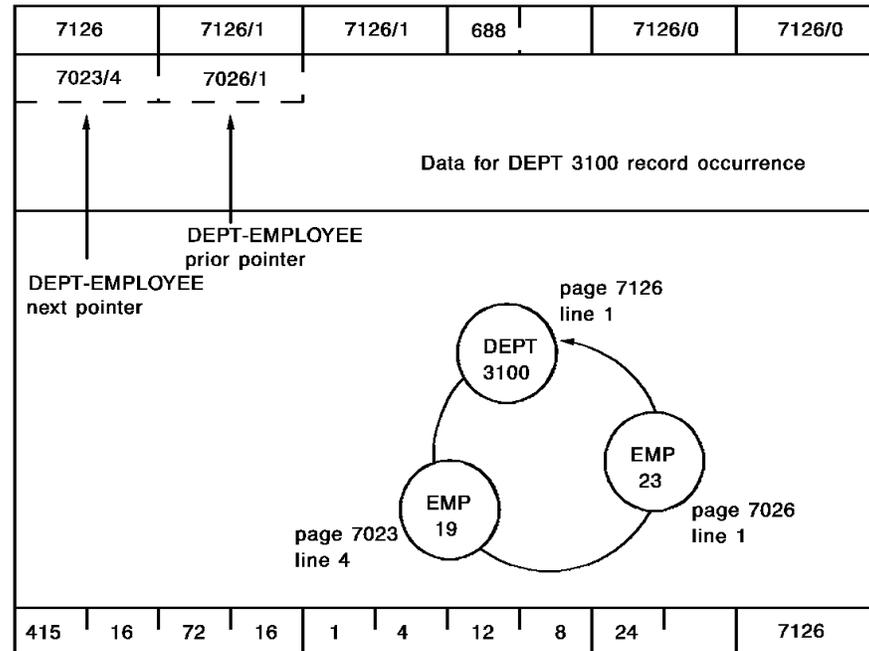


### 35.2.1 Connecting records to chained sets

**Operations performed:** CA-IDMS/DB performs the following operations to connect a record (that has previously been stored) to a chained set:

- Updates the prefix of the record being connected to reflect the record's next, prior, and owner (as applicable) pointers in the set
- Updates pointers in all other records affected by the new set connections

In the example below, EMPLOYEE 19 and EMPLOYEE 23 have been stored on pages 7023 and 7026, respectively. Connecting each to DEPT 3100 as members of the DEPT-EMPLOYEE set affects the DEPT 3100 record on page 7126. Its prefix must be updated to point to the next and prior members of the set.

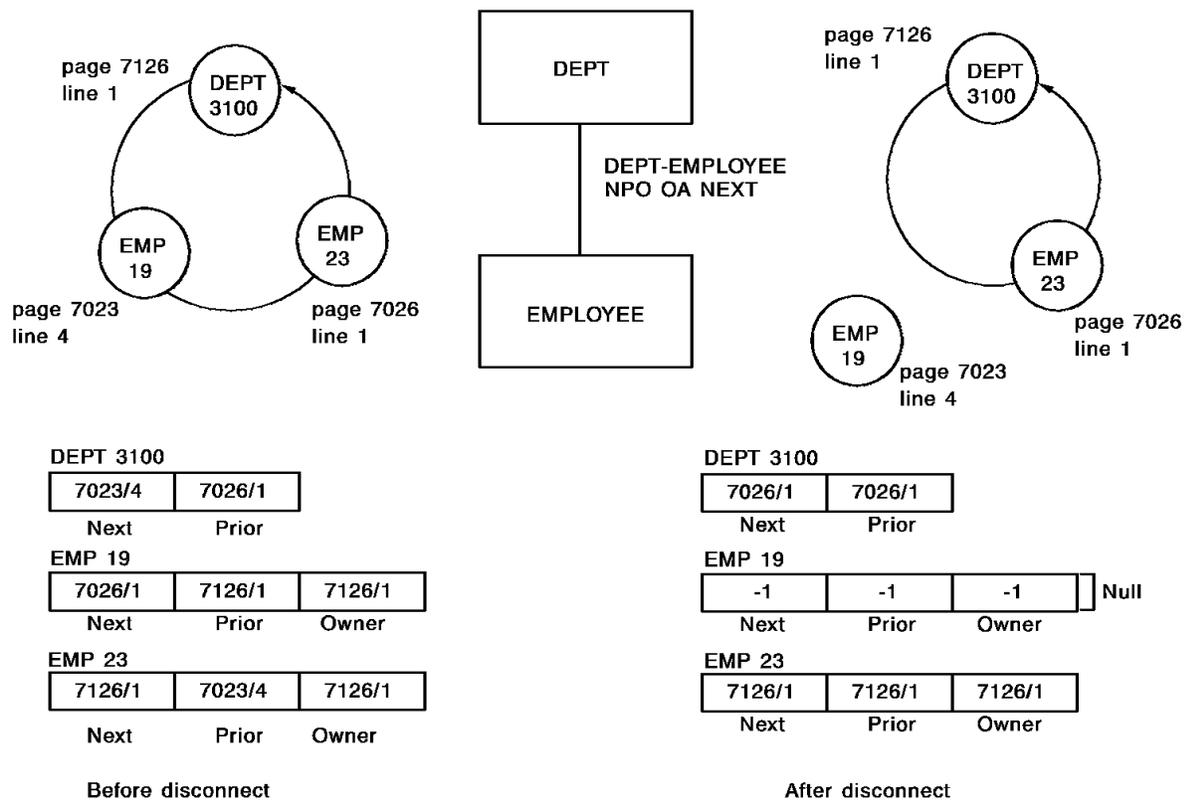


### 35.2.2 Disconnecting records

**Operations performed:** To disconnect a record occurrence from a chained set without erasing the record occurrence, CA-IDMS/DB must update pointers in the current, prior, and next records, as described below:

- For the **record being disconnected**, CA-IDMS/DB adjusts all the record occurrence's pointers to null (minus 1) for the set from which the record is being disconnected.
- For the **prior record in the chain**, CA-IDMS/DB adjusts the next pointer for the set from which the record occurrence is being disconnected so that the prior record points to the next record.
- For the **next record in the chain** (if the set has prior pointers), CA-IDMS/DB adjusts the prior pointer for the set from which the record occurrence is being disconnected so that the next record points to the prior record.

The following diagram shows disconnecting a record. The EMPLOYEE 19 record is disconnected from the DEPT-EMPLOYEE set for DEPT 3100. EMPLOYEE 19's pointers for that set are changed to null. The prior pointer in the EMPLOYEE 23 record is adjusted to point to the DEPT 3100 record, while the next pointer in the DEPT 3100 record must be adjusted to point to the EMPLOYEE 23 record.



**Adjusting the pointer:** To adjust the next pointer in the prior record, CA-IDMS/DB must access the prior record. In a set without prior pointers, however, CA-IDMS/DB must walk the entire set to access the prior record. For this reason, prior pointers are typically included in all sets to which the DISCONNECT (or ERASE) DML command might be applied.

### 35.2.3 Retrieving records

**Walking a set:** A program using navigational DML or CA-IDMS/DB in response to an SQL request can access all of the members of a chained set in the following manner: starting with the owner record occurrence, a program can use the next pointers to access each member occurrence in the chain until the program reaches the owner record again. Accessing members in a chain in this way is known as "walking a set."

Assume that the DEPT-EMPLOYEE set in the sample database is a chained set sorted by employee identification number (EMP-ID-0415). To retrieve an occurrence of the EMPLOYEE record, a program could issue the following requests:

```
MOVE '0050' TO DEPT-ID-0410
OBTAIN CALC DEPARTMENT.
OBTAIN NEXT WITHIN DEPT-EMPLOYEE SET.
```

**Processing the request:** CA-IDMS/DB processes this request as follows:

1. Using the value '0050' placed by the program in the DEPT-ID-041 0 field, CA-IDMS/DB obtains the DEPARTMENT record with an identification number of '0050'.
2. CA-IDMS/DB then finds the record occurrences pointed to by DEPARTMENT 50's next DEPT-EMPLOYEE pointer.

**Retrieving an owner:** A program can issue the following request to retrieve an occurrence of the DEPARTMENT record associated with an employee:

```
MOVE '0019' TO EMP-ID-0415.  
OBTAIN CALC EMPLOYEE.  
OBTAIN OWNER WITHIN DEPT-EMPLOYEE SET.
```

**Processing the request:** CA-IDMS/DB processes this request as follows:

1. Using the value '0019' placed by the program in the EMP-ID-0415 field, CA-IDMS/DB obtains the EMPLOYEE record with an identification number of '0019'.
2. If the DEPT-EMPLOYEE set has owner pointers, CA-IDMS/DB uses the EMPLOYEE record's owner pointer to retrieve the owning DEPARTMENT.
3. If the DEPT-EMPLOYEE set does not have owner pointers, CA-IDMS/DB uses the EMPLOYEE record's next pointer to walk the set until it retrieves the owner occurrence (that is, an occurrence of the DEPARTMENT record type).

# Chapter 36. Index Management

---

- 36.1 About indexed sets . . . . . 36-3
  - 36.1.1 Structure of indexes . . . . . 36-5
  - 36.1.2 Connecting records to indexed sets . . . . . 36-11
    - 36.1.2.1 Connecting members to unsorted indexed sets . . . . . 36-11
    - 36.1.2.2 Connecting members to sorted indexed sets . . . . . 36-14
  - 36.1.3 Disconnecting records from indexed sets . . . . . 36-15
  - 36.1.4 Retrieving indexed records . . . . . 36-16



## 36.1 About indexed sets

**Use:** Indexed sets can be used to physically link related record occurrences together or to provide alternate access to a record. In an indexed set, a pointer array associated with each owner occurrence contains the db-keys of all related member record occurrences.

**Types of indexed sets:** There are two types of indexed sets:

User-owned	The owner of the set is a user-defined record.
System-owned	The owner of the set is a system-defined SR7 record. The location mode of an SR7 record is CALC on the set name for non-SQL defined indexes or on an internally-generated name for SQL defined indexes. There is at most one occurrence of an SR7 record for each system-owned index.

**How to define an indexed set:** Use the following clauses on the SET statement to define an indexed set in a non-SQL schema definition:

User-owned	MODE IS INDEX
System-owned	MODE IS INDEX OWNER IS SYSTEM

Use the following SQL statements to implement an SQL defined constraint as an indexed set:

User-owned	Use this clause on the CONSTRAINT statement: LINKED INDEX
System-owned	Use the CREATE INDEX statement

When you implement a constraint as an indexed set, the referenced table is the *owner* of the set and the referencing table is the *member*.

**Set order:** An indexed set can have any of the following set orders: FIRST, LAST, NEXT, PRIOR, or SORTED. If it is SORTED, it can be sorted either on a user-specified symbolic key (sort key) or on the db-key of the member record occurrences.

Using SQL, the set order of a LINKED INDEX constraint is:

- SORTED, if you specify the ORDER BY clause
- Otherwise, LAST

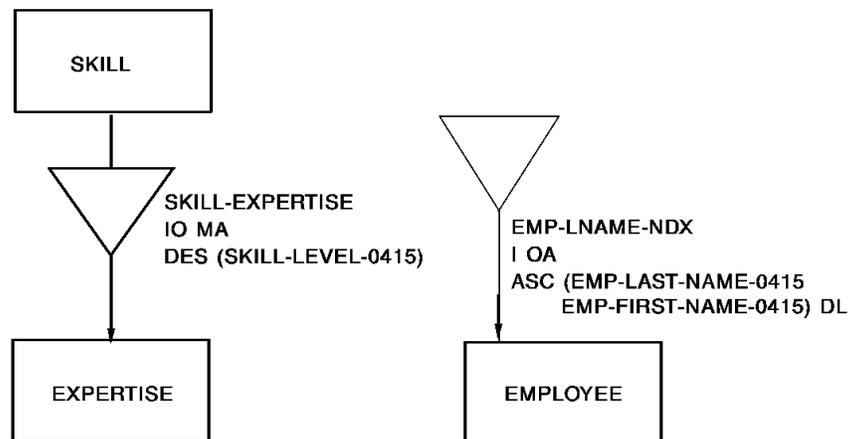
The set order of an indexed set created using the CREATE INDEX statement is:

- SORTED on a symbolic key if you specify the ORDER BY clause
- Otherwise, SORTED on db-key

**Notation:** The following diagram uses standard CA-IDMS database notation for two indexed sets, SKILL-EXPERTISE and EMP-LNAME-NDX. The descriptions of the indexed set relationships in the figure include the name of the set, linkage options, membership options, and the sort sequence and symbolic key.

The left side of the figure illustrates an indexed set (the SKILL-EXPERTISE set) between two user-defined record types. The owner of the SKILL-EXPERTISE set is the user-defined SKILL record; the member, EXPERTISE, is the indexed database record.

The right side of the figure illustrates an indexed set (the EMP-LNAME-NDX set) used to place an index on a user-defined member record type. The owner of the EMP-LNAME-NDX set is a system record, represented by a triangle; the member, EMPLOYEE, is the indexed database record.



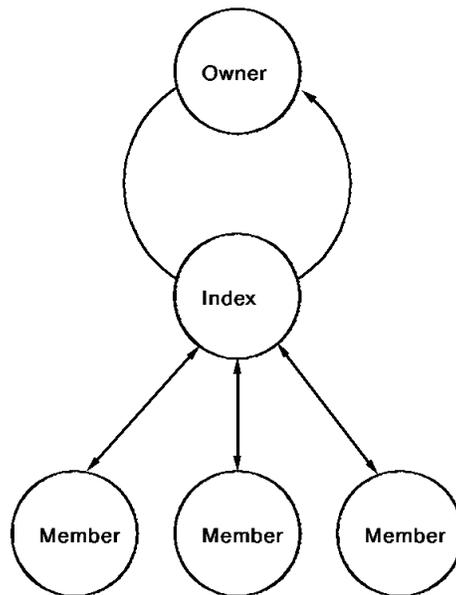
**Indexed set occurrence:** An **indexed set occurrence** consists of one occurrence of the owner record, type, an index, and any number of member record occurrences. The owner occurrence contains **next** and **prior** pointers to the index; the bottom-level of the index contains the member record occurrences' db-keys in the specified set order. If the indexed set has a user-defined owner record, each member occurrence contains an *index pointer* to the bottom-level of the index, and optionally, a pointer that links them directly to the owner occurrence. If the indexed set is system-owned, each member occurrence may *optionally* contain an index pointer.

**Note:** AN SQL defined linked constraint implemented as an indexed set always has owner pointers.

---

**Basic structure of an indexed set:** The member record occurrences in an indexed set point to the index that is chained to the owner record by next, prior, and owner pointers. The owner record contains next and prior pointers that chain it to the index.

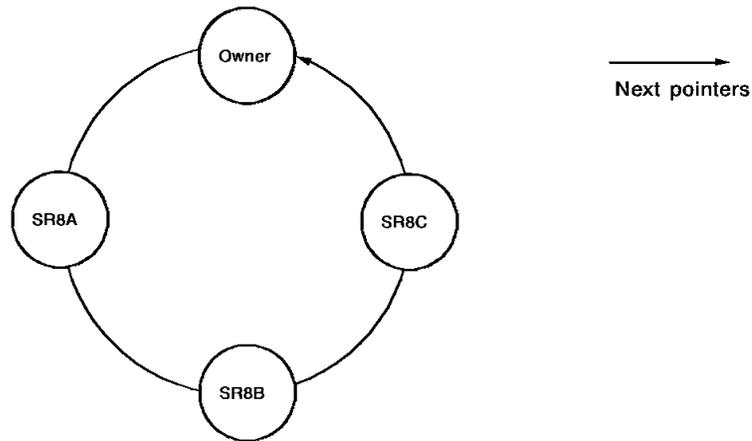
---



### 36.1.1 Structure of indexes

**Index creation:** The creation of an index is transparent to application programs. An index is created according to your specifications, but the actual creation and storage of the index is performed by CA-IDMS/DB. An index is composed of **SR8 system record** occurrences chained (by next, prior, and owner pointers) to the owner occurrence and each other.

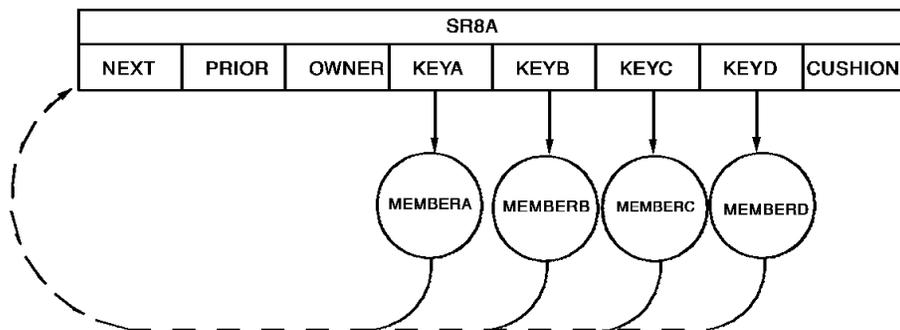
**SR8 records in an index:** Thus, an index is a chained set between the indexed set's owner record and the SR8 records. An index contains SR8 records chained by next, prior, and owner pointers to the indexed set's owner record. (Note that, for simplicity, prior and owner pointers are not included in the figure below):



Initially, the index is composed of a single SR8 member record. When the first SR8 record is full, additional SR8 records are added to the index as chained records.

**Bottom-level SR8 record and database record occurrences:** An SR8 record, shown in the following diagram, contains from 3 to 8,180 **index entries** (as specified in the schema or segment definition) and a **cushion** (that is, a field the length of the largest possible index entry).

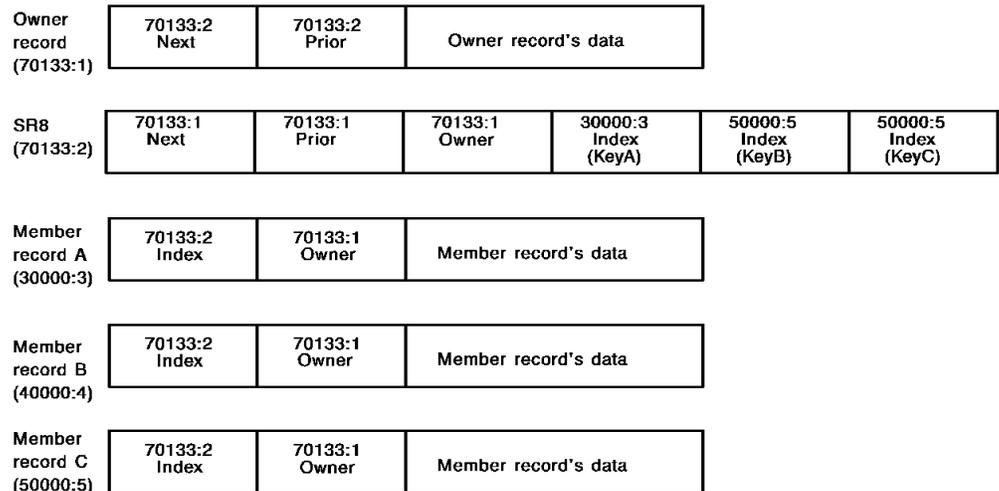
The SR8 record in the diagram contains four entries and a cushion. Each index entry contains an index pointer that points to a database occurrence that is a member of the indexed set; each member occurrence contains an index pointer that points to that SR8 record. (Note that, for simplicity, prior and owner pointers are not included in this figure.)



**Content of an index entry:** The actual content of an index entry depends on the indexed set's characteristics, as follows:

- **Unsorted set** — An index entry contains only the db-key of a member record occurrence.
- **Sorted set** — SR8 records for sorted indexed sets are arranged in levels to form a tree structure to facilitate a binary search. Consequently, an index entry contains the db-key of a member record occurrence or the db-key of another SR8 record occurrence. Additionally, for indexed sets sorted on a symbolic key, an index entry is composed of a db-key and a symbolic key. A symbolic key is a key constructed of one or more record elements (or columns) in the order specified in the schema (up to 256 bytes in length). (For a detailed discussion of indexed set structure for sorted indexed sets, see 36.1.2.2, “Connecting members to sorted indexed sets” on page 36-14, later in this chapter.)

**Example:** In this example, there is a single SR8 record chained to the indexed set's owner. The SR8 record contains three entries. Each entry contains an index pointer that points to a member database occurrence; each member occurrence contains an index pointer that points to that SR8 record. Additionally, the member occurrences contain owner pointers that point back to the set's owner.



**Indexed set with sorted set order:** For sorted indexed sets, you can specify that CA-IDMS/DB keep the index entries within the SR8 records in ascending, descending, or mixed order according to the member record's db-key or symbolic key. You can also specify whether numeric fields should be collated so that negative values are lower than positive values (natural sequence) or whether they should collate based on their bit pattern. If you specify that an indexed set be sorted on symbolic key, you can also specify whether duplicate symbolic keys are allowed or disallowed. Even if you specify that duplicate symbolic keys are allowed, CA-IDMS/DB does not store the same symbolic key more than once in the index. For example, the first time a record with a symbolic key ADAMS is added to the indexed set, CA-IDMS/DB adds the

symbolic key ADAMS to the index and associates the record occurrence's db-key to the key ADAMS. Later, if you add another record with the symbolic key ADAMS to the indexed set, CA-IDMS/DB associates the db-key of the new record to the existing symbolic key of ADAMS in the SR8 record.

**Specifying compression:** Additionally, you can specify that CA-IDMS/DB store symbolic keys in either compressed or uncompressed format. (Note that CA-IDMS/DB always strips trailing pad characters from an indexed set's symbolic keys.) If you specify compression, CA-IDMS/DB applies a 2-level compression algorithm to the symbolic key before inserting the key into the index, as follows:

- **Prefix compression** — CA-IDMS/DB compares (left to right) the symbolic key of the record being inserted into the index with adjacent symbolic keys and removes like characters. For example, if there are two symbolic keys, JOHNSON and JONES, CA-IDMS/DB stores the JOHNSON key in its entirety and stores JONES as NES.
- **Repeating character compression** — CA-IDMS/DB compresses three or more repeating single characters within each symbolic key into two bytes, and compresses 2 through 64 repeating blanks or nulls into one byte.

Specify compression of symbolic keys if the keys have either of the following characteristics:

- Commonly share the same prefix
- Contain many repeating characters (including blanks or nulls)

**How the index is organized:** To facilitate the process of locating an index entry for sorted sets, CA-IDMS/DB organizes an index for sorted records into levels. In this case, when the first (top-level) SR8 record is full, CA-IDMS/DB performs the following processing:

1. Splits the SR8 record into two parts. These two SR8 records stay at the same level.
2. Constructs a new higher level with two entries. Each entry points to one of the SR8 records created by Step 1.

CA-IDMS/DB repeats this process as the index expands. Indexes can have any number of intermediate levels. As CA-IDMS/DB adds new entries, it **splits** SR8 records and **spawns** new levels of SR8 records. An entry on one level points to an SR8 record at a lower level; the bottom-level entries point to the indexed database records themselves.

Thus, in a sorted indexed set with three levels (top, intermediate, and bottom), the index is structured as described below:

- The **top level** is made up of one SR8 record that contains index entries. Each entry is composed of a pointer to (that is, the database key of) an intermediate-level SR8 record and the highest symbolic key contained therein.

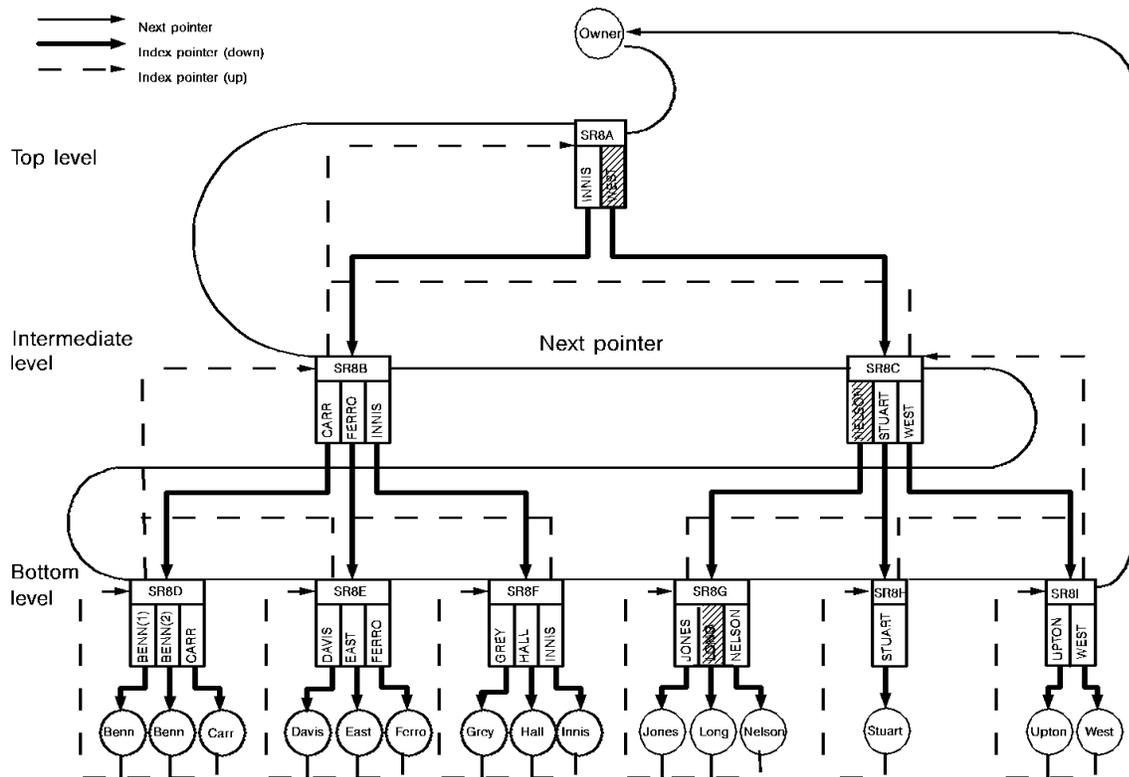
- The **intermediate level** is made up of one SR8 record for each entry in the top level. Each entry is composed of a pointer to a bottom-level SR8 record and the highest symbolic key contained therein.
- The **bottom level** is made up of one SR8 record for each entry in the intermediate level. Each entry is composed of a symbolic key and a pointer to a database record occurrence.

**Example:** For example, the sample database includes the indexed set EMP-LNAME-NDX. The EMP-LNAME-NDX set, shown in the following diagram and table shows the function of index levels and the search process. This simple index contains only three entries per SR8 record. The figure represents index and database records. (Note that, for simplicity, prior and owner pointers are not included in this figure.) The table shows the index pointers and symbolic keys.

To locate LONG in this 3-level index, CA-IDMS/DB performs the following steps:

1. Accesses the SR7 owner record by using the set name as the CALC key  
**Note:** For SQL-defined indexes, it uses a CALC key based on a number assigned to each index.
2. Accesses the top-level SR8 record by using the next pointer in the SR7 record
3. Searches this top-level SR8 record for the first entry with a symbolic key equal to or greater than LONG
4. Uses the db-key in this entry to access an intermediate-level SR8 record (that is, the NELSON/WEST SR8 record)
5. Searches this intermediate-level SR8 record (that is, the NELSON/WEST SR8 record) for the first entry equal to or greater than LONG
6. Uses the db-key in this entry to access a lower level SR8 record (that is, the JONES/NELSON SR8 record at the bottom level)
7. Searches this bottom-level SR8 record (the JONES/NELSON SR8 record) for the LONG entry
8. Uses the db-key in the LONG entry to access the requested member database record occurrence

Note that since previous processing deleted indexed records, not all of the index entries in each SR8 record are presently used (for instance, the STUART and UPTON/WEST SR8 records at the bottom level). Consequently, this index has space for expansion without spawning a new level.



In this example, each SR8 record is composed of a maximum of three entries. Each entry is composed of a symbolic key value and a db-key. The shaded entries are used to locate the LONG record in the database. In the top and intermediate levels, the db-key in each entry points to another SR8 record. In the bottom level, the db-key in each entry points to a database record. (Note that, for simplicity, prior and owner pointers are not included in this figure; also, since two employees are named BENN, there are two database member occurrences with that name.)

The entries in the 3-level index are shown below. Each entry is composed of a symbolic key and a db-key. The shaded entries are used to locate the LONG record in the database. The index entries in the top and intermediate levels point to SR8 records at the next lowest level. Only the bottom-level entry points to the database record. Note that since two employees are named BENN, there are two db-keys (one to each database member occurrence) for that symbolic key.

	SR8 db-key	SR8 index entries	
Top level SR8 records	90002:3	Innis West	90004:10 90004:57
Intermediate level SR8 records	90004:10	Carr Ferro Innis	90015:13 90016:40 90030:6
	90004:57	Nelson Stuart West	90021:3 90018:53 90030:12
Bottom-level SR8 records	90015:13	Benn Carr	721009:147 723006:105 721007:3
	90016:40	Davis East Ferro	720617:201 721592:63 722310:16
	00030:6	Grey Hall Innis	720016:31 727160:52 725921:74
	90021:3	James Long Nelson	726412:4 724263:12 727160:90
	90018:53	Stuart Upton	720039:37 720715:52
	90030:12	West	725129:2

### 36.1.2 Connecting records to indexed sets

All set orders (that is, FIRST, LAST, NEXT, PRIOR, and SORTED) are supported for indexed sets. Indexed set order determines the way CA-IDMS/DB builds the index when new member record occurrences are connected to the indexed set.

Connecting members to indexed sets ordered FIRST, LAST, NEXT, or PRIOR is discussed below, followed by a separate discussion of connecting members to indexed sets with a set order of SORTED.

#### 36.1.2.1 Connecting members to unsorted indexed sets

To connect new members to indexed sets with FIRST, LAST, NEXT, and PRIOR set order, CA-IDMS/DB inserts a new index entry between existing index entries. When one SR8 record fills, CA-IDMS/DB creates a new SR8 record; there is only one level of SR8 records in an unsorted index.

Once a request has been made to connect a member occurrence to an indexed set, CA-IDMS/DB first checks whether other entries exist. If no other entries exist, CA-IDMS/DB creates and stores the first SR8 record (containing the first entry) and connects it to the owner occurrence with next, prior, and owner pointers. The target

page for the first SR8 record is the page of the owner of the indexed set occurrence (plus displacement, if any).

**CA-IDMS/DB actions:** If other entries *do* exist, CA-IDMS/DB takes the following actions:

*Step 1:* Identifies the appropriate SR8 record and insertion point based on the set order, as follows:

- **NEXT** — The insertion point is physically after the index entry for the current SET occurrence.
- **FIRST** — The insertion point is physically the first index entry in the first SR8 record.
- **PRIOR** — The insertion point is physically before the index entry for the current SET occurrence.
- **LAST** — The insertion point is physically the last index entry in the last SR8 record.

**Note:** SQL-defined unsorted indexed constraints have an internal order of LAST.

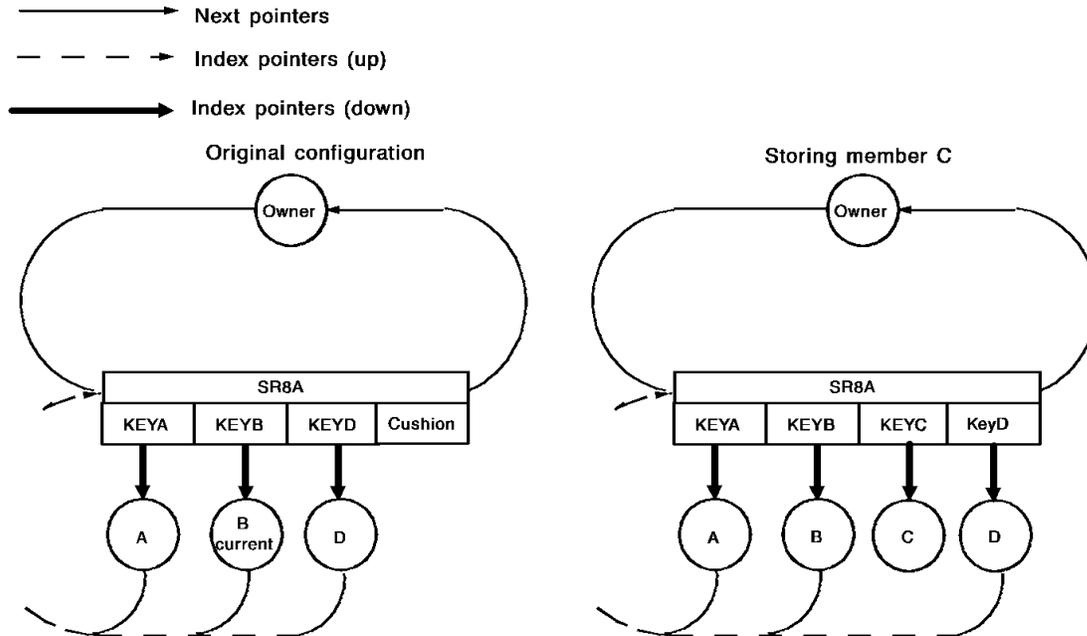
*Step 2:* Inserts the new entry into the index, as follows:

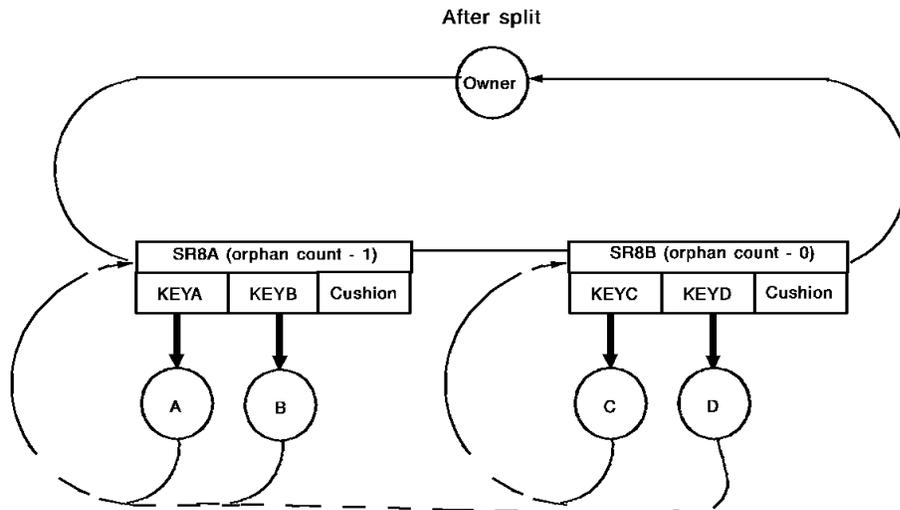
- If there is enough space in the target SR8 record for the new entry (that is, the insertion of this entry would not exceed the maximum allowable entries, and the target SR8's page has sufficient available space), CA-IDMS/DB inserts the new entry into the target SR8 record.
- If there is *not* enough space in the target SR8 for the new entry, CA-IDMS/DB inserts the new entry based on the location of the identified insertion point, as follows:
  - If the identified insertion point is physically first in the target SR8 record, CA-IDMS/DB checks whether there is enough space in the prior SR8 record:
    - If there is enough space in the prior SR8 record, CA-IDMS/DB inserts the new entry as the physically last entry in the prior SR8 record.
    - If there is not enough space in the prior SR8 record, CA-IDMS/DB splits the target SR8 record.
  - If the insertion point is physically last in the target SR8 record, CA-IDMS/DB checks whether there is enough space in the next SR8 record:
    - If there is enough space in the next SR8 record, CA-IDMS/DB inserts the new entry in the next SR8 record.
    - If there is not enough space in the next SR8 record, CA-IDMS/DB splits the target SR8 record.
  - If the insertion point is neither the physically first nor last in the target SR8 record, CA-IDMS/DB checks whether there is enough space in the next SR8 record:

- If there is enough space in the next SR8 record, CA-IDMS/DB moves the last entry to the SR8 record
- If there is not enough space in the next SR8 record, CA-IDMS/DB splits the target SR8 record.

**Index pointers for split SR8s:** When CA-IDMS/DB splits an SR8 record (Record A) into two SR8 records (Records A and B), the entries relocated to Record B point to member occurrences that still contain index pointers pointing to Record A if index pointers are maintained for the set (index pointers are optional for system-owned indexes). If index pointers are maintained, splitting Record A causes CA-IDMS/DB to set the **orphan count** in Record A equal to the number of entries moved to Record B.

**Splitting an SR8 record:** The following diagram shows splitting an SR8 record to add a member occurrence to an indexed set with a set order of NEXT. (Note that, for simplicity, prior and owner pointers are not included in this figure.)





### 36.1.2.2 Connecting members to sorted indexed sets

**Spawning:** CA-IDMS/DB organizes an index for sorted records into levels. When a top or intermediate SR8 record is full, CA-IDMS/DB **spawns** a new level through the following steps:

1. CA-IDMS/DB splits the SR8 record into two SR8 records.
2. CA-IDMS/DB constructs a new higher-level SR8 record. This new full-size SR8 record contains only two entries. Each entry points to one of the SR8 records created by Step 1.

CA-IDMS/DB determines the target page of a new SR8 record, as follows:

- If displacement has been specified and if the new record is a bottom-level SR8 record, the target page is the page of the owner of the indexed set occurrence plus displacement.
- Otherwise, the target page is the page of the owner of the indexed set occurrence.

CA-IDMS/DB repeats this process as the index expands. Indexes can have any number of intermediate levels. As CA-IDMS/DB adds new entries, it **splits** SR8 records and spawns new levels of SR8 records. An entry on one level points to an SR8 record at a lower level; the bottom-level entries point to the indexed database records themselves.

**Connecting new members:** To connect new members into a sorted index, CA-IDMS/DB first identifies the appropriate insertion point of the new entry based on the symbolic key or db-key. If this is the first entry (and, therefore, the first SR8 record), CA-IDMS/DB creates, stores and connects a new SR8 record to the owner occurrence. CA-IDMS/DB determines the target page for the new SR8 record as described above.

If this is not the first entry, CA-IDMS/DB identifies the insertion point of the new entry based on the symbolic key or db-key. Once the appropriate insertion point is identified, CA-IDMS/DB inserts the new entry into the index, as follows:

- If there is space in the target SR8 record (that is, if insertion of this entry would not exceed maximum allowable entries and the target SR8 record's page has sufficient available space, CA-IDMS/DB inserts the new entry in the target SR8 record.
- If space in the target SR8 record is insufficient for the new entry, CA-IDMS/DB attempts to move a number of entries to a prior or next SR8 record if space is available. Otherwise, a split occurs which may cause spawning depending on the available space in the higher-level SR8 records.

### 36.1.3 Disconnecting records from indexed sets

Assume that the DEPT-EMPLOYEE set in the sample database is an indexed set sorted by employee identification number (EMP-ID-0415). To disconnect an occurrence of the EMPLOYEE record, a program could issue the following requests:

```
MOVE '0019' TO EMP-ID-0415.  
FIND CALC EMPLOYEE.  
DISCONNECT EMPLOYEE FROM DEPT-EMPLOYEE.
```

**Processing the request:** CA-IDMS/DB processes these requests as follows:

1. Finds the SR8 record pointed to by EMPLOYEE record 19's index pointer.
2. Searches the SR8 record for EMPLOYEE 19's db-key. If CA-IDMS/DB finds EMPLOYEE 19's db-key, processing skips to Step 3. If CA-IDMS/DB does *not* find EMPLOYEE 19's db-key, processing continues as follows:
  - a. CA-IDMS/DB decrements the SR8 record's orphan count by 1. If the SR8 contains no entries and the orphan count is 0 CA-IDMS/DB erases the SR8 record.
  - b. CA-IDMS/DB follows SR8 records until it finds the db-key.
  - c. CA-IDMS/DB updates EMPLOYEE 19's index pointer to point to the correct SR8 record.
3. Removes EMPLOYEE 19's key entry from the bottom-level SR8 record and rewrites that SR8 record.
4. If this were an unsorted set, processing would be complete. Since this is a sorted set, if EMPLOYEE 19's symbolic key is the highest key in the SR8 record, CA-IDMS/DB passes up the key to each level in which the key is the highest and removes the entry for EMPLOYEE 19 from each successive SR8 record.

### 36.1.4 Retrieving indexed records

In contrast to locating member records of a chained set, CA-IDMS/DB locates member record occurrences in an index by searching the index. CA-IDMS/DB does *not* have to access each member record occurrence as with chain linkage.

**Types of processing:** Because CA-IDMS/DB searches the index rather than actual record occurrences, indexed sets provide a quick and efficient method for the following types of processing:

- **Random retrieval by symbolic key or generic key** — CA-IDMS/DB can retrieve individual records randomly by means of a symbolic key. CA-IDMS/DB can also retrieve a group of records by using a partial (generic) symbolic key. A string of characters, up to the length of the symbolic key, can be used as a generic key.
- **Sorted retrieval by db-key, symbolic key, or generic key** — CA-IDMS/DB can retrieve records in sorted order if the index is ordered on db-key or symbolic key. In this case, the db-keys or symbolic keys in an index are automatically maintained in sorted order and records therefore can be retrieved in ascending or descending order by db-key or symbolic key. Because records can be accessed through more than one index, they can be retrieved in more than one sort sequence.
- **Unsorted in exceptionally long sets** — To locate the db-keys of members in an indexed set, CA-IDMS/DB walks the index. Since accessing member record occurrences' db-keys in an index requires less database I/O than accessing the record occurrences themselves, CA-IDMS/DB can retrieve the db-keys of member records in exceptionally long sets more efficiently if the records are related using an index rather than a chain. This type of processing is useful for finding the *n*th record in a set or for manipulating lists of db-keys.
- **Physical sequential processing by db-key** — Member record occurrences can be clustered through an index. With this storage mode, the physical location of member records reflects the ascending or descending order of their db-key. If occurrences of a record type are to be retrieved in sequential order, clustering them through an index reduces I/O. This type of processing is most efficient when used with a stable database.

**Example when owner pointers:** Assume that the DEPT-EMPLOYEE set in the sample database is an indexed set sorted by employee identification number (EMP-ID-0415). To retrieve an occurrence of the EMPLOYEE record, a program might issue the following requests:

```
MOVE '0019' TO EMP-ID-0415.  
OBTAIN CALC EMPLOYEE.  
OBTAIN NEXT WITHIN DEPT-EMPLOYEE SET.
```

To fulfill the above request, CA-IDMS/DB performs the following processing:

*Step 1:* Using the value '0019' placed by the program in the EMP-ID-0415 field, obtains the EMPLOYEE record with an identification number of '0019'.

**Step 2:** Obtains the EMPLOYEE record with the next-highest identification number as follows:

1. Finds the SR8 record pointed to by EMPLOYEE 19's index pointer.
2. Searches the SR8 record for EMPLOYEE 19's db-key. If CA-IDMS/DB finds EMPLOYEE 19's db-key, processing skips to Step 3. If CA-IDMS/DB does *not* find EMPLOYEE 19's db-key, processing continues as follows:
  - a. Decrements the SR8 record's orphan count by 1. If the orphan count is now 0, CA-IDMS/DB erases the SR8 record if it is empty or rewrites it if it still contains entries.
  - b. CA-IDMS/DB searches the next SR8 record in the index until it finds the db-key. At this time, CA-IDMS/DB updates the EMPLOYEE record's index pointer to point to the correct SR8 record.

**Note:** CA-IDMS/DB only updates pointers and the orphan count at this time if both the area that contains the SR8 records and the area that contains the EMPLOYEE records were readied in update mode.

**Step 3:** Obtains the EMPLOYEE record whose db-key is adjacent to current of set (that is, the next EMPLOYEE record).

**Example when no owner pointers:** If the EMPLOYEE record did not have owner pointers, a program could issue the following request to retrieve an occurrence of the DEPARTMENT record:

```
MOVE '0019' TO EMP-ID-0415.
OBTAIN CALC EMPLOYEE.
OBTAIN OWNER WITHIN DEPT-EMPLOYEE SET.
```

When fulfilling the above request, the DBMS would discover the lack of an owner pointer in the set and use the EMPLOYEE record's index pointer to find the bottom-level SR8 record that contains the key for the requested EMPLOYEE record. CA-IDMS/DB will then use the owner pointer contained in that SR8 record to obtain the DEPARTMENT record.

**SR8 record currency:** When a program uses a subschema that contains records in an indexed set, CA-IDMS/DB changes SR8 record currency *only when it accesses a member record through the index*, since CA-IDMS/DB keeps track of SR8 record currency internally. When CA-IDMS/DB accesses a member record in any other manner, CA-IDMS/DB does *not* change SR8 record currency.

For example, a program might issue the following commands:

```
MOVE '0019' TO EMP-ID-0415.
OBTAIN CALC EMPLOYEE.
OBTAIN NEXT WITHIN EMP-LNAME-NDX SET.
```

CA-IDMS/DB then fulfills these requests as follows:

1. Using the value '0019' for CALC entry into the database, CA-IDMS/DB accesses the EMPLOYEE record with that identification number. EMPLOYEE 19 is now

current of run unit, record, and the EMP-LNAME-NDX set. At this point, since CA-IDMS/DB has not accessed an SR8 record, internal currency is *not* created for the SR8 structure of the index set.

2. On OBTAIN NEXT, CA-IDMS/DB accesses the SR8 record that contains the index entry for EMPLOYEE 19. At this time, CA-IDMS/DB makes this SR8 record the current record of the SR8 structure of the index set.
3. CA-IDMS/DB finds the next index entry (either in that SR8 record or the next SR8 record). The SR8 record containing that index entry is now current of the SR8 structure.
4. Using the next index entry, CA-IDMS/DB obtains the corresponding EMPLOYEE record. That EMPLOYEE record is now current of run unit, record, and the EMP-LNAME-NDX.

**RETURN and FIND commands:** When a program uses a subschema that contains records in an index, use:

- **The RETURN command** to retrieve database keys and/or symbolic keys from the index without accessing database records.
- **The FIND command** to maintain indexed set currency.

# Chapter 37. Lock Management

---

37.1	Controlling access to CA-IDMS databases	37-3
37.2	Readying areas	37-4
37.2.1	Area ready modes	37-4
37.2.2	Central version area status	37-7
37.2.3	Default ready mode using navigational DML	37-8
37.2.4	Ready modes and SQL access	37-8
37.3	Physical area locks	37-11
37.3.1	About physical area locks	37-11
37.3.2	Controlling update access	37-11
37.4	Locking within central version	37-13
37.4.1	Logical locks	37-13
37.4.2	Types of locks	37-14
37.4.3	Logical area locks	37-15
37.4.4	Area locking for SQL transactions	37-16
37.4.5	Record locks	37-18
37.4.6	System generation options affecting record locking	37-19
37.5	Locking within a data sharing group	37-21
37.5.1	Inter-CV-interest	37-21
37.5.2	Global transaction locks	37-21
37.5.3	Proxy locks	37-22
37.5.4	Page locks	37-23
37.6	Controlling access to native VSAM files	37-24
37.7	Deadlocks	37-25
37.7.1	How the system detects a deadlock	37-25
37.7.2	Global deadlock detection	37-26



## 37.1 Controlling access to CA-IDMS databases

**Factors controlling access to data:** The primary means of influencing how CA-IDMS/DB controls access to data is in the way areas are readied and the status assigned to areas within a central version. These factors ultimately determine which transactions can access and update data within a CA-IDMS database and whether or not concurrent access is controlled at the area or record occurrence level.

**What follows:** The remainder of this section discusses:

- Ready areas
- The status of areas under the central version
- Default ready modes
- Ready modes and SQL access to data

Later sections in this chapter discuss how these factors determine the types of locks CA-IDMS uses to control access to data.

## 37.2 Readying areas

### 37.2.1 Area ready modes

**Types of ready modes:** A transaction can restrict runtime operations in a database area by readying that area with a mode of update or retrieval, as follows:

---

Update	The readying transaction can both retrieve and update data within the area.
Retrieval	The readying transaction cannot update data in the area.

---

**Ready mode qualifiers:** You can qualify the specified area ready mode with a shared (default), protected, or exclusive option to prevent update or retrieval of an area by other transactions executing concurrently under the same central version or, in the case of a shared area, under other central versions that are members of the same data sharing group. The qualified ready modes are:

---

Shared update	If a transaction has readied the area in shared update mode, other transactions executing concurrently can ready the area in shared update or shared retrieval mode.
Shared retrieval	If a transaction has readied the area in shared retrieval mode, other transactions executing concurrently can ready the area in shared update, shared retrieval, protected retrieval, or protected update mode.
Protected update	If a transaction has readied the area in protected update mode, other transactions executing concurrently can ready the area in shared retrieval mode only.
Protected retrieval	If a transaction has readied the area in protected retrieval mode, other transactions executing concurrently can ready the area in shared retrieval or protected retrieval mode.
Exclusive update and exclusive retrieval	If a transaction has readied the area in exclusive update or exclusive retrieval mode, other transactions executing concurrently cannot ready the area in any mode. Exclusive retrieval is available only using navigational DML.
Transient retrieval	<p>A ready mode of transient retrieval cannot be explicitly set by application programs or access module specifications. Instead, transient retrieval is automatically used by a transaction accessing an area in a retrieval mode, if either of the following conditions apply:</p> <ul style="list-style-type: none"> <li>■ The status of the area within the central version is transient retrieval</li> <li>■ The isolation level of the SQL transaction readying the area is transient read</li> </ul> <p>If a transaction has readied an area in transient retrieval mode, other transactions executing concurrently can ready the area in any mode.</p>

---

►► Both area status and transaction isolation levels are discussed later in this section under 37.2.2, “Central version area status” on page 37-7 and 37.2.4, “Ready modes and SQL access” on page 37-8 respectively.

**Compatibility of ready modes:** The mode in which one transaction readies an area restricts the mode in which other transactions executing under the same central version or in the case of a shared area, within the same data sharing group, can ready that area. This table shows the modes in which transaction B can ready an area, depending on the mode in which transaction A has readied the area. Y(es) signifies

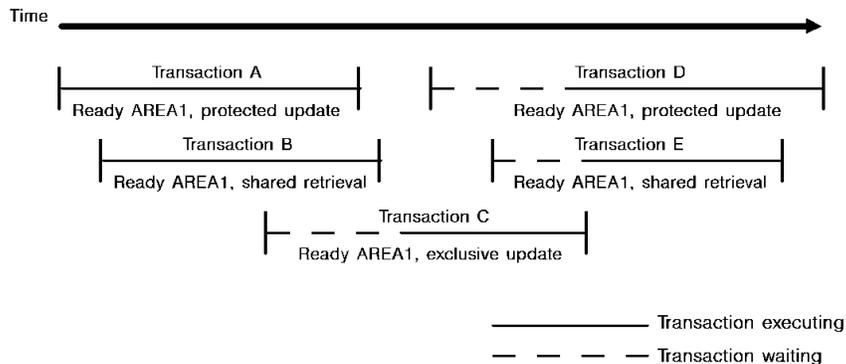
that the second transaction can ready the area in the specified mode; N(o) signifies that it cannot.

		Transaction B						
		SHARED UPDATE	SHARED RETRIEVAL	PROTECTED UPDATE	PROTECTED RETRIEVAL	EXCLUSIVE UPDATE	EXCLUSIVE RETRIEVAL	TRANSIENT RETRIEVAL
Transaction A	SHARED UPDATE	Y	Y	N	N	N	N	Y
	SHARED RETRIEVAL	Y	Y	Y	Y	N	N	Y
	PROTECTED UPDATE	N	Y	N	N	N	N	Y
	PROTECTED RETRIEVAL	N	Y	N	Y	N	N	Y
	EXCLUSIVE UPDATE	N	N	N	N	N	N	Y
	EXCLUSIVE RETRIEVAL	N	N	N	N	N	N	Y
	TRANSIENT RETRIEVAL	Y	Y	Y	Y	Y	Y	Y

**Concurrent use of an area within a central version or data sharing group:**

When a transaction cannot ready an area because of a protected or exclusive restriction, CA-IDMS/DB places the transaction in a wait state. When the restriction is lifted, the transaction can proceed.

**Example of concurrent area usage:** The following diagram shows concurrent use of an area by transactions executing under a central version or data sharing group. Concurrently, transaction A readies AREA1 in protected update mode, transaction B readies the area in shared retrieval mode, and transaction C attempts to ready the area in exclusive update mode. The system puts transaction C into a wait state until both transactions A and B terminate. Transactions D and E, attempting to ready the area, must wait until transaction C terminates.



## 37.2.2 Central version area status

**Area status and ready modes:** Each area accessible from within a central version has a status associated with it. The status of an area affects the mode in which transactions executing under the central version can ready the area:

UPDATE (or ONLINE)	Transactions executing under the central version can ready the area in any mode
RETRIEVAL	Transactions executing under the central version can ready the area in any retrieval mode (EXCLUSIVE, PROTECTED, SHARED or TRANSIENT)
TRANSIENT RETRIEVAL	Transactions executing under the central version can ready the area in any retrieval mode, but the CA-IDMS/DB automatically changes the mode to TRANSIENT RETRIEVAL
OFFLINE	Transactions executing under the central version cannot ready the area in any mode

**Establishing the area status:** The status of an area within a central version is initially established by specifications made within the DMCL used by the DC/UCF system. An area's status may subsequently be changed by DCMT commands.

*Permanent area status:* When an area's status is changed through a DCMT command, it may be designated as permanent. A permanent area status persists across both normal and abnormal system terminations until it is subsequently changed by another DCMT command or until the journal files associated with the central version are initialized. Whether or not an area's status has been designated as permanent is indicated on the output from a DCMT DISPLAY AREA command.

*At system startup:* If a permanent area status is not in effect, the first time a system is started and each time it is subsequently started after a normal shutdown, the status of the area is set to that specified in the ON STARTUP parameter of the ADD SEGMENT or ADD AREA statement within the DMCL definition. The default area status is UPDATE.

**Following an abnormal system termination:** If a permanent area status is not in effect, when restarting a system following an abnormal termination, the status of the area is set to that specified in the ON WARMSTART parameter of the ADD SEGMENT, or ADD AREA statement within the DMCL definition. The area can be set to what it was at the time of the failure (the default) or it can be set to an explicit value.

**Changing area status:** You can change the status of an area within a central version by issuing a DCMT VARY AREA or VARY SEGMENT command. In certain cases, CA-IDMS cannot change the status of the area immediately because existing transactions are accessing the area. In addition to active transactions,

longterm or notify locks held by pseudo-conversational applications may prevent the area status from being changed. If CA-IDMS cannot change the status immediately, it initiates an internal task that completes the DCMT VARY operation when no more conflicts exist. During the time it takes to complete the vary, transactions attempting to ready the area in a mode that is incompatible with the new area status receive an error.

►► For more information about the DCMT VARY AREA and VARY SEGMENT commands, refer to *CA-IDMS System Tasks and Operator Commands*.

### 37.2.3 Default ready mode using navigational DML

You can specify a **default ready mode for a database area** in a subschema definition. The specified default mode determines the mode in which the area is to be readied for programs using that subschema. If you specify a default mode for a database area, programs using the subschema do *not* have to issue a READY command for the area. Note, however, that if a program issues a READY command for one area in the subschema, the automatic readying mechanism is disabled and the program must issue a READY command for all areas to be accessed.

### 37.2.4 Ready modes and SQL access

**Factors affecting SQL lock management:** When accessing data using SQL, the way in which an area is readied depends on several factors:

- The transaction state
- The isolation level
- The requested ready mode
- The status of areas within central version

**Transaction state:** A transaction initiated using SQL has one of two states:

---

READ ONLY	Data can be read, but not updated; updates to temporary tables <i>are</i> allowed
READ WRITE	Data can be both read and updated using DML and DDL statements

---

**Default is READ WRITE:** Unless otherwise specified, the transaction state is READ WRITE. You can override the default when you define an access module or by issuing a SET TRANSACTION statement at runtime.

**Isolation level:** A transaction initiated using SQL also has one of two isolation levels:

---

CURSOR STABILITY	Guarantees read integrity. Read integrity ensures that: <ul style="list-style-type: none"><li>■ All data accessed by the transaction is in a committed state</li><li>■ The most-recently accessed row of an updatable cursor is protected from update by other transactions while it remains current</li></ul>
TRANSIENT READ	Does not guarantee read integrity. For this reason, a transaction executing under transient read is not allowed to update the database. If the isolation level of a transaction is transient read, the transaction state is automatically READ ONLY.

---

**Default is CURSOR STABILITY:** Unless otherwise specified, the isolation level of a transaction is CURSOR STABILITY. You can override the default when you define an access module or by issuing a SET TRANSACTION statement at runtime.

**Requested ready modes:** You can specify within the access module definition the modes in which CA-IDMS/DB is to ready the areas accessed by non-dynamic SQL statements embedded in an application. Otherwise, CA-IDMS/DB determines the ready mode at runtime. It also determines the ready mode at runtime for dynamic SQL statements.

**Runtime ready modes:** The ready mode in which an area is accessed at runtime depends on the requested ready mode, the transaction state, the isolation level, and the area's availability:

Transaction state	Isolation level	Area ready mode
READ ONLY	TRANSIENT READ	Transient retrieval mode; no row locks are placed.
READ ONLY	CURSOR STABILITY	Retrieval modes only.  If update modes were specified on the CREATE or ALTER ACCESS MODULE statement, CA-IDMS/DB changes them to shared retrieval. If no ready option was specified, the default is shared retrieval.
READ WRITE	CURSOR STABILITY	All areas are accessed using the mode specified on the CREATE ACCESS MODULE.  If no mode was specified, the default is: <ul style="list-style-type: none"> <li>■ Shared update in local mode and under the central version, if the area status is update</li> <li>■ Shared retrieval under the central version, if the area status is retrieval</li> </ul>

Under central version, if an area is being readied in a retrieval mode and the status of the area is transient retrieval, CA-IDMS/DB changes the ready mode to **transient retrieval**.

## 37.3 Physical area locks

### 37.3.1 About physical area locks

CA-IDMS/DB maintains a physical area lock as a flag within the first space management page of an area. It examines and sets the lock whenever the area is opened for update. This occurs when:

- A local mode transaction readies the area in an update mode
- A DC/UCF system is started in which the area status in the DMCL is UPDATE
- A DCMT VARY AREA command changes the status of the area to UPDATE

**Unlocking the physical area:** Once a physical lock is placed on an area, it remains set until:

- The local mode transaction or central version terminates normally
- Manual recovery procedures are used to roll out the effects of a failing local mode transaction
- A central version is restarted after an abnormal termination and subsequently shutdown normally
- The area status within central version is changed from update to another status

**Physical area locks and shared areas:** In a data sharing environment, the physical area lock in a shared area is set by the first member of a data sharing group to open the area for update and it is reset by the last member to relinquish control.

### 37.3.2 Controlling update access

**Purpose of physical area locks:** Physical area locks prevent concurrent update by independent transactions (that is transactions executing outside of a single central version or data sharing group) and prevent update access to an area requiring recovery of incomplete transactions.

**How locking works:** CA-IDMS/DB provides this protection as follows:

---

Local mode	<p>As each area is readied in any update mode, CA-IDMS/DB checks the lock. If the lock is set, the transaction receives an error and access to the area is not allowed.</p> <p>If the lock is not set, the local mode transaction causes the lock to be set and the space management page is rewritten immediately. If the transaction terminates abnormally (that is, without issuing a FINISH or COMMIT WORK), the lock remains set. Further update access is prevented until the area is recovered (through CA-IDMS recovery procedures).</p>
Central version	<p>At system startup, the central version checks the locks in all areas intended for update. If the physical lock is set and the area is not shared, or the area is shared but is not currently being updated by another member of the central version's data sharing group, then a warning message is displayed at the console and the area status is changed to offline. The central version proceeds without the use of that area and any transaction attempting to ready that area will receive an error. If the physical lock is subsequently removed from the area, the status of the area can be varied to update.</p>

---

---

## 37.4 Locking within central version

### 37.4.1 Logical locks

**Control access to resources:** Logical locks are used within a central version and within a data sharing group to control access to database resources by concurrently executing transactions. Before a transaction can access a resource, it places a lock on the resource which prevents other transactions from modifying or, in some cases, accessing the resource while the lock is maintained.

In a data sharing environment, CA-IDMS uses global transaction locks, maintained in a coupling facility lock structure to control inter-member access to shared resources.

For more information on global locking, see 37.5, “Locking within a data sharing group” on page 37-21 later in this chapter.

**Wait states:** If a transaction attempts to lock a resource which is locked by another transaction with a conflicting mode, the first transaction will wait until the lock is released. If the waiting transaction exceeds the internal wait interval specified at system generation, CA-IDMS aborts the transaction and rolls out its updates. If one transaction is waiting to place a lock and the transaction that holds it then waits on a lock held by the first transaction, a deadlock condition exists. CA-IDMS resolves this condition by aborting and rolling back one of the transactions.

►► For more information on detecting and resolving deadlocks, see 37.7, “Deadlocks” on page 37-25 later in this chapter.

**Hierarchical locking:** CA-IDMS/DB uses a hierarchical locking scheme in which locks are placed at area and record occurrence (row) levels. Area locks control access to the area, and by implication, all record occurrences stored within the area. Record locks control access to individual record occurrences or rows.

A transaction intending to access data within an area must first place a lock at the area level. Depending on the strength of that lock (its mode), the transaction may or may not also place locks on individual record occurrences as it retrieves or updates them.

In a data sharing environment, CA-IDMS uses a three-level hierarchy to control inter-member access to shared resources: area, proxy, and record occurrence.

For more information on this additional level, see 37.5.3, “Proxy locks” on page 37-22 later in this chapter.

## 37.4.2 Types of locks

**Lock modes:** Each logical lock has an associated lock mode. The mode of the lock determines whether the lock conflicts with other locks already held on the resource and with locks subsequently requested by other transactions.

The following types of locks (lock modes) are used for both area and record locks:

Mode	Identifier	Description
Share	S	Typically used to guarantee that no updates are made to data while a transaction is accessing it. A share lock is compatible with other share locks but not with exclusive locks. A share lock placed on an area implies a share lock on each record within the area.
Exclusive	X	Typically placed on a resource to protect transactions from accessing data that is being updated by the issuing transaction. An exclusive lock is incompatible with both share and other exclusive locks. An exclusive lock placed on an area implies an exclusive lock on all records within the area.
Null-lock	NL	A null-lock is a special type of lock which is placed on a record to signify a notify lock and on an area to signify transient retrieval access. Null-locks provide no protection against concurrent access.

**Intent locks on areas:** The following types of locks are placed only on areas:

Mode	Identifier	Description
Intent share	IS	Allows share (S) locks to be placed on records within the area.
Intent exclusive	IX	Allows exclusive (X) locks to be placed on records within the area.
Update intent exclusive	UIX	Allows exclusive locks to be placed on records within the area by the issuing transaction, but not by other transactions.

In addition to the above lock modes, the following lock mode has been provided for but is currently not used:

- **Update (U)** An update lock is placed on a resource if it might be updated after it is retrieved (in which case the lock would be upgraded to an exclusive lock).

**Compatibility of locks:** For two transactions running within the same DC/UCF system to access the same area or row concurrently, their lock types must be compatible. When two transactions attempt to set locks that are not compatible, the first transaction to set a lock causes the second transaction to wait until the resource is freed.

**Note:** CA-IDMS ensures that a transaction does not compete with itself for locks.

**Compatibility chart:** The chart presented below shows which lock modes are compatible and which are incompatible. The plus sign (+) indicates a situation in which two lock modes are compatible. The minus sign (-) indicates a situation in which two lock modes are incompatible.

	NL	IS	IX	S	U	UIX	X
NL	+	+	+	+	+	+	+
IS	+	+	+	+	+	+	-
IX	+	+	+	-	-	-	-
S	+	+	-	+	+	-	-
U	+	+	-	+	-	-	-
UIX	+	+	-	-	-	-	-
X	+	-	-	-	-	-	-

**Example:** If TRANSACTION1 holds a share (S) lock on an area, TRANSACTION2 can set a null-lock (NL), intent-share (IS), share (S), or update (U) lock on the same area.

### 37.4.3 Logical area locks

**Effect of ready mode:** In order to control concurrent access to areas within a central version, the mode in which an area is readied is translated into a logical lock on the area. As an area is readied, CA-IDMS/DB attempts to place an appropriate lock based on the ready mode. If the new lock doesn't conflict with locks already held by other transactions, access is granted to the area. If a conflict exists, the transaction is placed in a wait state until the conflicting locks are released.

**Area lock depends on area ready mode:** The type of lock (lock mode) placed on an area depends on the mode in which the area is being readied:

<b>Ready mode</b>	<b>Lock mode</b>
Transient retrieval	Intent Share (NL)
Shared retrieval	Intent Share (IS)
Shared update	Intent Exclusive (IX)
Protected retrieval	Share (S)
Protected update	Update Intent Exclusive (UIX)
Exclusive retrieval	Exclusive (X)
Exclusive update	Exclusive (X)

**When area locks are acquired:** For transactions initiated through navigational DML, CA-IDMS/DB acquires area locks when either of the following occur:

- The first non-ready DML (other than BIND RECORD) statement is issued following one or more READY statements
- The first non-bind statement is issued within a transaction using default ready modes specified by the subschema

For SQL-initiated transactions, when area locks are acquired depends on the area acquisition mode specified within an access module or in effect for dynamic SQL.

►► For more information see 37.4.4, “Area locking for SQL transactions” later in this chapter.

**Area acquisition threshold:** If a transaction is locking multiple areas at one time, and must wait to place a lock on one of the areas, CA-IDMS/DB releases the locks on all other areas before placing the transaction in a wait state. This helps to avoid deadlocks between two or more transactions trying to gain access to areas. However, it also means that another transaction can gain access to an area whose lock was released by the waiting transaction. To avoid this pre-emption, you can specify an area acquisition threshold at system generation that limits the number of times a transaction will wait on an area lock before it no longer releases other area locks.

### 37.4.4 Area locking for SQL transactions

**When area locks are acquired:** The time at which area locks are acquired for SQL transactions varies depending on the lock acquisition mode in effect. There are two lock acquisition modes:

- Preclaim
- Incremental

**On first database access:** The preclaim mode directs CA-IDMS to place locks on all areas in a transaction that use the preclaim acquisition mode as soon as the first statement that requires access to the *database* is executed.

You can use the preclaim mode to reduce the likelihood of deadlocks. A transaction that uses the preclaim option to lock an area will not wait for an area that is held by another transaction while it holds a lock on an area.

**On first area access:** The incremental mode directs CA-IDMS to delay placing a lock on an area until the first statement in the transaction that requires access to the *area* is executed.

You can use the incremental mode to increase database concurrency. A transaction that uses the incremental mode does not place a lock on an area until the area is actually required for processing. This makes the area accessible to other transactions for a longer period of time. In general, if a transaction does not always access every area in its access path, you should assign the incremental mode to those areas that are least likely to be accessed.

**Example:** Suppose a transaction needs to access three different tables, each of which is stored in a different area:

Table	Area	Acquisition mode
T1	AREA1	Preclaim
T2	AREA2	Incremental
T3	AREA3	Preclaim

Locks would be acquired in the manner shown below:

```

TRANSACTION A
-----
.
.
SELECT * FROM T1;  <----- Locks are placed on both
.                  AREA1 and AREA3.
.
SELECT * FROM T2;  <----- A lock is placed on AREA2.
.
.
SELECT * FROM T3;
.
.
.

```

## 37.4.5 Record locks

**Purpose of record locks:** Record locks are used within the central version to control concurrent access to individual record occurrences (rows). Occurrence-level record locks (in conjunction with area locks) are used to:

- Protect against concurrent update of the same record by two or more transactions
- Prevent record occurrences that are current within one transaction from being updated by another transaction

**Implicit record locks:** CA-IDMS/DB automatically places locks on records accessed by a transaction if the area in which the record resides is readied in any of the following modes:

Area ready mode	Record lock
Shared retrieval on read records	Shared (S) locks
Shared update	Shared (S) locks on read records; exclusive (X) locks on updated records
Protected update	Exclusive (X) locks on updated records

**Note:** You can use system generation options to inhibit record locking for navigational DML applications, as discussed in 37.4.6, “System generation options affecting record locking” on page 37-19 later in this chapter.

**Shared record locks:** If shared locks are being maintained, CA-IDMS/DB places one on each record as it is accessed. Shared locks are also maintained on:

- The most-recently accessed record of its type (the most-recently accessed row of each table)
- The most-recently accessed record in each set (the most-recently accessed row of each constraint or index)
- The most-recently accessed record in each area.

**Note:** Additional shared locks are maintained on the current row of each updatable cursor open within an SQL transaction.

CA-IDMS/DB releases these locks as the transaction accesses different record occurrences. These implicit record locks guarantee the integrity of the currencies used by navigational DML applications and provide the protection necessary for SQL applications executing with an isolation level of cursor stability.

►► For more information on isolation levels, see 37.2.4, “Ready modes and SQL access” on page 37-8 earlier in this chapter.

**Exclusive record locks:** If exclusive locks are being maintained, CA-IDMS/DB places them on all records altered by a DML or DDL statement until the recovery unit terminates (that is a COMMIT (CONTINUE), ROLLBACK (CONTINUE) or FINISH is issued) or until the transaction abends.

**Implicit page locks:** Implicit locks are used in a special way to control user access to pages for which the amount of available space has been altered. When the available space on a page is changed as a result of an update operation, CA-IDMS/DB places a special implicit exclusive lock on the page, allowing retrieval to continue. If a subsequent DML or DDL command from a different transaction requests further modification to available space on that page, the request is delayed until the lock is released (that is, until the recovery unit that caused the lock to be set terminates).

**Explicit record locks:** The navigational programmer can set **explicit record locks** with the DML KEEP command. The KEEP verb or the KEEP option of a FIND or OBTAIN verb places a shared lock on the record occurrence. KEEP with the EXCLUSIVE option places an exclusive lock on the record occurrence. CA-IDMS/DB holds explicit record locks until the transaction terminates or a COMMIT ALL statement is executed.

### 37.4.6 System generation options affecting record locking

Two system generation options affect whether or not CA-IDMS/DB maintains record locks for navigational DML transactions. These options are:

RETRIEVAL LOCK/NOLOCK	Specifies whether or not CA-IDMS/DB places shared locks on records in an area readied in SHARED RETRIEVAL
UPDATE LOCK/NOLOCK	Specifies whether or not CA-IDMS/DB places exclusive locks on records updated in an area readied in PROTECTED UPDATE

**Note:** These system generation parameters affect only navigational DML applications; they do not apply to SQL applications.

**Reading uncommitted data:** If RETRIEVAL NOLOCK is specified, a transaction may read uncommitted data; that is, it may read data that has been updated by another transaction before those changes have been committed or data that has been accessed by a retrieval transaction may be concurrently updated while the retrieval transaction is still active. This may result in inconsistencies in the data processed by the shared retrieval transaction. These inconsistencies may also include transient 11xx abends from the DBMS.

If UPDATE NOLOCK is specified, a transaction updating data in an area readied in PROTECTED UPDATE does not protect transactions readying the area in SHARED RETRIEVAL. As with RETRIEVAL NOLOCK, it is possible for a transaction which

has readied the area in SHARED RETRIEVAL to read a record updated by a PROTECTED UPDATE transaction before it has been committed.

Since both options affect the protection afforded shared retrieval transactions, it is typical (though not required) to set both parameters in the same way. In systems in which there is a high volume of updates, you might want to consider specifying LOCK for both.

**Note:** No inter-CV retrieval protection is provided except for shared areas accessed through members of a data sharing group. If an area is not shared, then regardless of the system lock options in effect, it is possible for a shared retrieval transaction executing in a central version whose area status is retrieval to read uncommitted data updated by another central version.

**TRANSIENT RETRIEVAL area status:** As an alternative to using system generation parameters to reduce the volume of record locks maintained, consider using a central version's area status of TRANSIENT RETRIEVAL instead. Provided the area is not updated within the central version, a status of transient retrieval can be used to eliminate the locking of records within the area.

---

## 37.5 Locking within a data sharing group

Within a data sharing group, locking is used to control inter-member access to shared resources, just as it is used to control access to resources within a central version. The basic locking scheme used within a central version is extended for data sharing in the following ways:

- Global transaction locks are used to control inter-member access
  - An additional level is introduced in the locking hierarchy
  - Page locks are used to protect database pages while they reside in a buffer pool
- For more information on data sharing, refer to *CA-IDMS System Operations*.

### 37.5.1 Inter-CV-interest

*What is inter-CV-interest:* Inter-CV-interest denotes a state in which an area is being shared by:

- At least one group member with an area status of UPDATE, and
- More than one group member with an area status of RETRIEVAL or UPDATE. Members accessing an area in TRANSIENT RETRIEVAL, have no impact on inter-CV-interest.

Conflict for the area (and the records and pages in the area) can only occur if there is inter-CV-interest in the area. This is significant because if there is no inter-CV-interest in an area, the overhead associated with controlling access to it is reduced.

Whether or not there is inter-CV-interest in an area is indicated on the output from a DCMT DISPLAY AREA command.

### 37.5.2 Global transaction locks

*What are global transaction locks?:* Global transaction locks are locks that reside within a coupling facility lock structure and are used to control inter-member access to data in shared areas. Whenever a transaction places a lock on a shared area or on a record that resides in a shared area and there is inter-CV-interest in that area, global locks ensure that no other transaction in the data sharing group is accessing the same resource in a conflicting mode.

*Managing global locks:* Global locking relies on a coupling facility lock structure to record and manage global locks. Global locks are acquired by the CA-IDMS lock manager whenever a transaction places a lock on a resource and a sufficiently strong global lock is not already held by that CV. Global locks are retained until no transaction within a CV requires a lock of that strength, at which point the global lock may be released, downgraded, or retained, depending on the resource type and whether or not there is contention for the resource between group members.

*Inter-CV-interest and global locking:* Global transaction locks are not acquired if there is no inter-CV-interest in an area. If inter-CV-interest begins because another member accesses the area in a potentially conflicting mode, global transaction locks will be acquired by every sharing member in which a transaction holds a lock on the area or any of its records.

### 37.5.3 Proxy locks

*What is a proxy lock?:* A proxy lock is a global lock used within a data sharing group to represent a lock on all the records within a page of a shared area. Proxy locks are held by members of a data sharing group and not by individual transactions.

*An additional hierarchy level:* Proxy locks represent an additional level in the locking hierarchy used by CA-IDMS to control access to data.

Normally CA-IDMS uses a two-level locking hierarchy: area and record. Before placing a lock on a record, a transaction must place a lock on the area in which the record resides. Depending on the mode of the area lock, it may be possible to avoid placing locks on individual records within the area.

For shared areas, the locking hierarchy expands to three levels: area, proxy, and record. Before a lock is placed on a record in a shared area, a lock must be held on a proxy that represents the record's page and before this can be done, a lock must be held on the area in which the record resides.

*Proxy lock modes:* A proxy can be locked in one of two modes: Share or Exclusive. At least a share lock must be held on a proxy before a transaction can place a share or null (notify) lock on a record represented by the proxy. Similarly, an exclusive proxy lock must be held before a transaction can place an exclusive lock on a record represented by the proxy.

*Managing proxy locks:* An exclusive proxy lock held by one member does not prohibit access by another member. Instead the purpose of proxy locks is to detect inter-CV contention for resources and to eliminate the use of global record locks where possible. As long as all members holding a lock on a proxy hold it in share mode, there is no contention for resources on the page and no need to globally lock individual records on that page. However, if at least two members hold a lock on a proxy and at least one of those is an exclusive lock, then there is possible contention for individual records, necessitating the use of global record locks to control access to individual records.

The acquisition and management of proxy locks is done automatically by the CA-IDMS lock manager. Application programs do not need to explicitly acquire or manage proxy locks. However, database administrators should be aware of their existence and their impact on recovery and resource utilization.

## 37.5.4 Page locks

*What is a page lock?:* A page lock is a lock that is used within a data sharing group to protect database pages while they reside in a member's local buffer pool. Page locks are only placed on pages of areas that are designated for data sharing and only if there is inter-CV interest in the area.

*Managing page locks:* The coupling facility lock structure associated with the data sharing group is used to record and manage global page locks, just as is done for global transaction locks. And just as a proxy represents all of the records on the page, it also represents the page itself. Therefore, proxy locks reduce the need to acquire and release global page locks each time a page is moved into and out of the buffer pool.

*Page lock protection:* Before a database page is read into the buffer pool, an exclusive or shared lock is placed on that page, depending on whether or not the active transaction intends to update the page. Once the lock is acquired, no other group member may place a conflicting lock on the page until the first member relinquishes its lock. This means that no other sharing member may read the page contents while another member has it locked exclusively. Page locks are held until another group member wants access to the page in a conflicting mode. Before an exclusive page lock can be released on an updated page, the page is written to the disk and to the shared cache.

## 37.6 Controlling access to native VSAM files

**Physical area locks not set:** CA-IDMS does not maintain physical area locks for areas that map to native VSAM data sets. Therefore, a combination of SHAREOPTIONS, JCL, and operational procedures is used to control updates of native VSAM data sets by CA-IDMS, local transactions, and non-CA-IDMS programs.

**DEFINE CLUSTER command:** For example, you can prevent concurrent update by specifying the parameter SHAREOPTIONS(2,3) in the DEFINE CLUSTER command during VSAM cluster definition. This parameter permits only one application program to open the data set for update, thereby preventing concurrent update of the data set by two application programs executing in different regions. Within a central version, access to native VSAM files is controlled through the use of logical locks on areas and records just as for CA-IDMS/DB database files.

**CA-IDMS/DB facilities:** You can use CA-IDMS facilities to further control access to native VSAM data sets. For example, **to protect a data set from being updated**, set the status within central version to TRANSIENT RETRIEVAL or RETRIEVAL. In this case, no application program running under the central version can ready the area in update mode.

**To ensure read integrity of the area** when accessed by transactions executing under a central version, you can use the following procedure when updating the data set using non CA-IDMS programs:

1. Vary offline the CA-IDMS area that maps to the VSAM data set by means of the DCMT VARY AREA OFFLINE command.
2. Run the job to update the VSAM data set.
3. Vary the area to retrieval access mode using the DCMT VARY AREA RETRIEVAL command, thus making the area once again available under the central version.

---

## 37.7 Deadlocks

**What is a deadlock?:** A **deadlock** is an unresolvable contention between multiple requestors for a resource. Resources are either DC/UCF system resources (such as programs and storage) or database resources (such as areas and records). CA-IDMS/DB uses different control block structures to track contention for DC/UCF system resources and database resources. Although it tracks deadlocks using different control block structures, the same deadlock detection mechanism is used to resolve deadlocks.

### 37.7.1 How the system detects a deadlock

Deadlock detection is a process performed on a time interval basis. It is carried out in four major phases:

1. **Identifying stalled tasks** — To identify tasks that are stalled, all dispatch control elements (DCEs) in the system are examined. Any DCE found stalled while waiting on an internal resource is entered into the deadlock detection matrix (DDM). All subsequent processing begins with the DCE address stored in the DDM table. This eliminates the need to scan all DCEs in the system.
2. **Identifying task dependencies** — Next, the dependencies between the stalled tasks are identified. The deadlock detection matrix is updated. For each task on which another task is waiting, a bit in the deadlock detection matrix is set to one.
3. **Identifying deadlocks** — To determine which tasks are involved in a deadlock cycle, a transformation is performed on the matrix. From this process, a pair of deadlocked tasks is identified. From this pair, a victim is selected.
4. **Selecting a victim** — The task running for the shortest period of time is chosen as the victim of the two tasks as long as:
  - The priority of the victim task is less than that of the other task
  - The victim task's wait was not entered with COND=NONE and the other task's wait was entered with COND=DEAD

The task running for the shortest period of time is chosen as the victim because it is more likely that it will have consumed fewer resources than a longer running task. As a result, less duplication of work should be required when the victim is restarted, with these exceptions:

- If the other task is of a higher priority, implying that it is of more importance
- If the victim task entered the deadlock with COND=NONE and the other task specified COND=DEAD. In this case, the task specifying COND=DEAD is chosen as the victim since COND=DEAD indicates that the task is designed to handle and recover from deadlock situations. This prevents an abend.

**Victim selection user exit:** The algorithm used to select a victim in a deadlock situation may not be optimal for your installation or applications. User exit 30 allows victims to be selected based upon specific requirements. The exit is passed the DCE addresses of each pair of deadlocked tasks and may take one of two actions:

- Choose one of the tasks as the victim task
- Return control to the deadlock detector by requesting that the default deadlock detection logic be applied

►► For a discussion of user exit 30, refer to *CA-IDMS System Operations*.

**Deadlock detection interval:** You can control the frequency with which the deadlock detection mechanism searches for deadlocked tasks using the DEADLOCK DETECTION parameter of the SYSTEM statement.

The DEADLOCK DETECTION parameter allows you to specify the amount of time that elapses before the deadlock detection mechanism searches for deadlocked tasks. Note that in an idle system, deadlock detection is also idled until new tasks are started. This eliminates CPU consumption for deadlock detection when no tasks could possibly be deadlocked.

You can use the DCMT VARY DEADLOCK command at runtime to override the system generation specification.

►► For further information on the DEADLOCK DETECTION parameter of the SYSTEM statement, refer to *CA-IDMS System Generation*. For further information on the DCMT VARY DEADLOCK command, refer to *CA-IDMS System Tasks and Operator Commands*.

## 37.7.2 Global deadlock detection

*What is a global deadlock?:* A global deadlock is a situation in which unresolvable contention exists for shared resources between tasks executing on different members within a data sharing group.

*Detecting global deadlocks:* A global deadlock is possible if at least one stalled task is waiting on a global resource. In a potential global deadlock situation, each member passes information to the one acting as the global deadlock manager. The global deadlock manager examines the information gathered from the other members and determines which tasks, if any, are deadlocked.

*Resolving global deadlocks:* If a global deadlock exists, user exits are invoked, to assist in selecting a victim task. If these exits are not provided, the task running for the shortest period or with the lowest priority is designated as the victim. Once the victim is determined, the member on which the victim is executing is directed to cancel the task.

*Global deadlock user exits:* Two user exits are used in selecting a victim in a global deadlock situation:

- **Exit #35** is invoked when a group member is collecting information about a stalled task in order to send it to the global deadlock manager. The exit provides

the opportunity for a site to collect additional information that may be relevant to the victim selection process.

- **Exit #36** is invoked by the global deadlock manager when a victim is being selected. Its function is similar to exit #30, but it is passed different parameters. Instead of being passed the DCE addresses of two deadlocked tasks, it is passed a pair of parameters for each task, one of which is the information collected by exit #35. In this way, site-specific criteria can be used in selecting a victim even though the deadlocked tasks may be executing on a group member that is different than that of the global deadlock manager.

►► For details on coding these exits, refer to *CA-IDMS System Operations*.



## Appendixes

---



# **Appendix A. Sample Physical Database Definition**

---



---

### Sample DMCL Definition

```
-- ***** */
-- */
-- GLOBAL DMCL FOR CA-IDMS/DB 15.0 BASE INSTALL */
-- */
-- */
-- ***** */

-- * DEFINE THE FILES & AREAS IN THE SYSTEM SEGMENT */
ADD SEGMENT SYSTEM
    FOR NONSQL
    PAGE GROUP 0
    MAXIMUM RECORDS PER PAGE 255
    ;
ADD FILE SYSTEM.DCDML ASSIGN TO DCDML
    DSNAME "SYSTEM.DDLDDL";
ADD FILE SYSTEM.DCLOD ASSIGN TO DCLOD
    DSNAME "SYSTEM.DDLDCLOD";
ADD FILE SYSTEM.DCLOG ASSIGN TO DCLOG
    DSNAME "SYSTEM.DDLDCLOG";
ADD FILE SYSTEM.DCRUN ASSIGN TO DCRUN
    DSNAME "SYSTEM.DDLDCRUN";
ADD FILE SYSTEM.DCSCR ASSIGN TO DCSCR
    DSNAME "SYSTEM.DDLDCSCR";
ADD AREA SYSTEM.DDLDDL
    PRIMARY SPACE 1000 PAGES FROM PAGE 1001
    PAGE SIZE 4276
    WITHIN FILE DCDML FROM 1 FOR ALL BLOCKS ;
ADD AREA SYSTEM.DDLDCLOD
    PRIMARY SPACE 100 PAGES FROM PAGE 3001
    PAGE SIZE 4276
    WITHIN FILE DCLOD FROM 1 FOR ALL BLOCKS ;
ADD AREA SYSTEM.DDLDCLOG
    PRIMARY SPACE 4000 PAGES FROM PAGE 30001
    PAGE SIZE 4276
    WITHIN FILE DCLOG FROM 1 FOR ALL BLOCKS ;
ADD AREA SYSTEM.DDLDCRUN
    PRIMARY SPACE 1000 PAGES FROM PAGE 40001
    PAGE SIZE 2676
    WITHIN FILE DCRUN FROM 1 FOR ALL BLOCKS ;
ADD AREA SYSTEM.DDLDCSCR
    PRIMARY SPACE 2000 PAGES FROM PAGE 50001
    PAGE SIZE 2676
    WITHIN FILE DCSCR FROM 1 FOR ALL BLOCKS ;
-- * DEFINE THE CATSYS SEGMENT */
ADD SEGMENT CATSYS
    FOR NONSQL
    PAGE GROUP 0
    MAXIMUM RECORDS PER PAGE 255
    ;
```

---

```

ADD FILE CATSYS.DCCAT ASSIGN TO DCCAT
   DSNAME "CATSYS.DCCAT";
ADD FILE CATSYS.DCCATL ASSIGN TO DCCATL
   DSNAME "CATSYS.DCCATL";
ADD FILE CATSYS.DCCATX ASSIGN TO DCCATX
   DSNAME "CATSYS.DCCATX";
ADD AREA CATSYS.DDLCCAT
   PRIMARY SPACE 300 PAGES FROM PAGE 1
   PAGE SIZE 4276
   WITHIN FILE DCCAT FROM 1 FOR ALL BLOCKS ;
ADD AREA CATSYS.DDLCCATX
   PRIMARY SPACE 100 PAGES FROM PAGE 801
   PAGE SIZE 4276
   WITHIN FILE DCCATX FROM 1 FOR ALL BLOCKS ;
ADD AREA CATSYS.DDLCCATL
   PRIMARY SPACE 50 PAGES FROM PAGE 901
   PAGE SIZE 4276
   WITHIN FILE DCCATL FROM 1 FOR ALL BLOCKS ;
-- * DEFINE THE SYSMMSG SEGMENT */
ADD SEGMENT SYSMMSG
   FOR NONSQL
   PAGE GROUP 0
   MAXIMUM RECORDS PER PAGE 255
   ;
ADD FILE SYSMMSG.DCMSG ASSIGN TO DCMSG
   DSNAME "SYSMMSG.DDLDCMSG";
ADD AREA SYSMMSG.DDLDCMSG
   PRIMARY SPACE 4000 PAGES FROM PAGE 10001
   PAGE SIZE 4276
   WITHIN FILE DCMSG FROM 1 FOR ALL BLOCKS ;
-- * DEFINE THE SYSLOC SEGMENT */
ADD SEGMENT SYSLOC
   FOR NONSQL
   PAGE GROUP 0
   MAXIMUM RECORDS PER PAGE 255
   ;
ADD FILE SYSLOC.DCLSCR ASSIGN TO DCLSCR;
--
ADD AREA SYSLOC.DDLCLSCR
   PRIMARY SPACE 2000 PAGES FROM PAGE 55001
   PAGE SIZE 2676
   WITHIN FILE DCLSCR FROM 1 FOR ALL BLOCKS ;

```

---

```

-- * DEFINE THE SYSDIRL SEGMENT *                               */
ADD  SEGMENT SYSDIRL
      FOR NONSQL
      PAGE GROUP 0
      MAXIMUM RECORDS PER PAGE 255
      ;
ADD  FILE SYSDIRL.DIRLDB ASSIGN TO DIRLDB
      DSNAME "SYSDIRL.DDLDDL";
ADD  AREA SYSDIRL.DDLDDL
      PRIMARY SPACE 2000 PAGES FROM PAGE 5001
      PAGE SIZE 4276
      WITHIN FILE DIRLDB FROM 1 FOR ALL BLOCKS ;
ADD  FILE SYSDIRL.DDLDCLOD ASSIGN TO DIRLLOD
      DSNAME "SYSDIRL.DDLDCLOD";
ADD  AREA SYSDIRL.DDLDCLOD
      PRIMARY SPACE 10 PAGES FROM PAGE 4001
      PAGE SIZE 4276
      WITHIN FILE DIRLLOD FROM 1 FOR ALL BLOCKS ;
-- * DEFINE THE SYSUSER SEGMENT *                               */
ADD  SEGMENT SYSUSER
      FOR NONSQL
      STAMP BY AREA;
ADD  FILE SYSUSER.SECDD ASSIGN TO SECDD
      DSNAME "SYSUSER.DDLDCSEC";
ADD  AREA SYSUSER.DDLSEC
      PRIMARY SPACE 500 PAGES FROM PAGE 48001
      PAGE SIZE 4276
      WITHIN FILE SECDD FROM 1 FOR ALL BLOCKS ;
-- * DEFINE THE APPLDICT SEGMENT *                               */
ADD  SEGMENT APPLDICT
      FOR NONSQL
      PAGE GROUP 0
      MAXIMUM RECORDS PER PAGE 255
      ;
ADD  FILE APPLDICT.DICTDB ASSIGN TO DICTDB
      DSNAME "APPLDICT.DDLDDL";
ADD  FILE APPLDICT.DLODDB ASSIGN TO DLODDB
      DSNAME "APPLDICT.DDLDCLOD";
ADD  AREA APPLDICT.DDLDDL
      PRIMARY SPACE 2000 PAGES FROM PAGE 60001
      PAGE SIZE 4276
      WITHIN FILE DICTDB FROM 1 FOR ALL BLOCKS ;

```

---

```

ADD      AREA APPLDICT.DDLDCLOD
          PRIMARY SPACE 500 PAGES   FROM PAGE 70001
          PAGE SIZE 4276
          WITHIN FILE DLODDDB FROM 1 FOR ALL BLOCKS ;
-- * DEFINE THE EMPDEMO SEGMENT *           */
ADD      SEGMENT EMPDEMO
          FOR NONSQL
          PAGE GROUP 0
          MAXIMUM RECORDS PER PAGE 255
          ;
ADD      FILE EMPDEMO.EMPDEMO ASSIGN TO EMPDEMO
          DSNAME "EMPDEMO.EMPDEMO";
ADD      FILE EMPDEMO.INSDEMO ASSIGN TO INSDEMO
          DSNAME "EMPDEMO.INSDEMO";
ADD      FILE EMPDEMO.ORGDEMO ASSIGN TO ORGDEMO
          DSNAME "EMPDEMO.ORGDEMO";
ADD      AREA EMPDEMO.EMP-DEMO-REGION
          PRIMARY SPACE 50 PAGES   FROM PAGE 75001
          PAGE SIZE 4276
          WITHIN FILE EMPDEMO FROM 1 FOR ALL BLOCKS ;
ADD      AREA EMPDEMO.INS-DEMO-REGION
          PRIMARY SPACE 25 PAGES   FROM PAGE 75101
          PAGE SIZE 4276
          WITHIN FILE INSDEMO FROM 1 FOR ALL BLOCKS ;
ADD      AREA EMPDEMO.ORG-DEMO-REGION
          PRIMARY SPACE 25 PAGES   FROM PAGE 75151
          PAGE SIZE 4276
          WITHIN FILE ORGDEMO FROM 1 FOR ALL BLOCKS ;
-- * DEFINE THE ASFDICT SEGMENT *           */
ADD      SEGMENT ASFDICT
          FOR NONSQL
          PAGE GROUP 0
          MAXIMUM RECORDS PER PAGE 255
          ;
ADD      FILE ASFDICT.ASFDMML ASSIGN TO ASFDML
          DSNAME "ASFDICT.DDLDMML";
ADD      FILE ASFDICT.ASFLOD ASSIGN TO ASFLOD
          DSNAME "ASFDICT.DDLDCLOD";
ADD      FILE ASFDICT.ADEFN ASSIGN TO ADEFN
          DSNAME "ASFDICT.ASFDEFN";
ADD      FILE ASFDICT.ADATA ASSIGN TO ADATA
          DSNAME "ASFDICT.ASFDATA";
ADD      AREA ASFDICT.DDLDMML
          PRIMARY SPACE 2000 PAGES  FROM PAGE 80001
          PAGE SIZE 4276
          WITHIN FILE ASFDML FROM 1 FOR ALL BLOCKS ;

```

---

```

ADD    AREA ASFDICT.DDLDCLOD
        PRIMARY SPACE 2000 PAGES    FROM PAGE 83001
        PAGE SIZE 4276
        WITHIN FILE ASFL0D FROM 1 FOR ALL BLOCKS ;
ADD    AREA ASFDICT.IDMSR-AREA
        PRIMARY SPACE 1000 PAGES    FROM PAGE 85001
        PAGE SIZE 4276
        WITHIN FILE ASFDEFN FROM 1 FOR ALL BLOCKS ;
ADD    AREA ASFDICT.IDMSR-AREA2
        PRIMARY SPACE 2000 PAGES    FROM PAGE 88001
        PAGE SIZE 4276
        WITHIN FILE ASFDATA FROM 1 FOR ALL BLOCKS ;
-- * DEFINE THE SYSSQL SEGMENT *          */
ADD    SEGMENT SYSSQL
        FOR SQL
        STAMP BY AREA;
ADD    FILE SYSSQL.SQLDD ASSIGN TO SQLDD
        DSNAME "SYSSQL.DDL0AT";
ADD    FILE SYSSQL.SQLXDD ASSIGN TO SQLXDD
        DSNAME "SYSSQL.DDL0ATX";
ADD    FILE SYSSQL.SQLOD ASSIGN TO SQLOD
        DSNAME "SYSSQL.DDL0ATLOD";
ADD    AREA SYSSQL.DDL0AT
        PRIMARY SPACE 2000 PAGES    FROM PAGE 20001
        PAGE SIZE 4276
        WITHIN FILE SQLDD FROM 1 FOR ALL BLOCKS ;
ADD    AREA SYSSQL.DDL0ATX
        PRIMARY SPACE 500 PAGES     FROM PAGE 25001
        PAGE SIZE 4276
        WITHIN FILE SQLXDD FROM 1 FOR ALL BLOCKS ;
ADD    AREA SYSSQL.DDL0ATLOD
        PRIMARY SPACE 500 PAGES     FROM PAGE 28001
        PAGE SIZE 4276
        WITHIN FILE SQLOD FROM 1 FOR ALL BLOCKS ;
-- * DEFINE THE SQLDEMO SEGMENT *        */
ADD    SEGMENT SQLDEMO
        FOR SQL
        STAMP BY AREA;
ADD    FILE SQLDEMO.EMPLDEMO ASSIGN TO EMPLDEMO
        DSNAME "SQLDEMO.EMPLDEMO";
ADD    FILE SQLDEMO.INFODEMO ASSIGN TO INFODEMO
        DSNAME "SQLDEMO.INFODEMO";
ADD    FILE SQLDEMO.INXDDEMO ASSIGN TO INXDDEMO
        DSNAME "SQLDEMO.INXDDEMO";

```

---

```

ADD      AREA SQLDEMO.EMPLAREA
          PRIMARY SPACE 100 PAGES FROM PAGE 77001
          PAGE SIZE 4276
          WITHIN FILE EMPLDEMO FROM 1 FOR ALL BLOCKS ;
ADD      AREA SQLDEMO.INFOAREA
          PRIMARY SPACE 50 PAGES FROM PAGE 77201
          PAGE SIZE 4276
          WITHIN FILE INFODEMO FROM 1 FOR ALL BLOCKS ;
ADD      AREA SQLDEMO.INDXAREA
          PRIMARY SPACE 50 PAGES FROM PAGE 77301
          PAGE SIZE 4276
          WITHIN FILE INDXDEMO FROM 1 FOR ALL BLOCKS ;
-- * DEFINE THE PROJSEG SEGMENT *
ADD      SEGMENT PROJSEG
          FOR SQL
          STAMP BY AREA;
ADD      FILE PROJSEG.PROJDEMO ASSIGN TO PROJDEMO
          DSNAME "PROJSEG.PROJDEMO";
ADD      AREA PROJSEG.PROJAREA
          PRIMARY SPACE 50 PAGES FROM PAGE 77401
          PAGE SIZE 4276
          WITHIN FILE PROJDEMO FROM 1 FOR ALL BLOCKS ;

ADD      DMCL R150DMCL;

ADD      BUFFER R150DMCL.DEFAULT_BUFFER
          PAGE SIZE 4276
          LOCAL MODE BUFFER PAGES 20 OPSYS STORAGE
          CV MODE BUFFER INITIAL PAGES 30
             MAXIMUM PAGES 50 ;

ADD      BUFFER R150DMCL.LOG_BUFFER
          PAGE SIZE 4276
          LOCAL MODE BUFFER PAGES 5 IDMS STORAGE
          CV MODE BUFFER INITIAL PAGES 5
             MAXIMUM PAGES 5 ;

ADD      JOURNAL BUFFER R150DMCL.JNL_BUFFER
          PAGE SIZE 2004
          BUFFER PAGES 5 ;

```

---

```

ADD    DISK JOURNAL    R150DMCL.J1JRNL
      FILE SIZE 1000
      ASSIGN TO J1JRNL;

ADD    DISK JOURNAL    R150DMCL.J2JRNL
      FILE SIZE 1000
      ASSIGN TO J2JRNL;

ADD    DISK JOURNAL    R150DMCL.J3JRNL
      FILE SIZE 1000
      ASSIGN TO J3JRNL;

ADD    DISK JOURNAL    R150DMCL.J4JRNL
      FILE SIZE 1000
      ASSIGN TO J4JRNL;

ADD    ARCHIVE JOURNAL R150DMCL.SYSJRNL
      BLOCK SIZE 19068
      ASSIGN TO SYSJRNL;

MODIFY DMCL R150DMCL
      DEFAULT BUFFER DEFAULT_BUFFER
      DBTABLE R150DBTB
      INCLUDE SEGMENT SYSTEM
        FILE SYSTEM.DCSCR
        BUFFER DEFAULT_BUFFER
        FILE SYSTEM.DCLOG
        BUFFER LOG_BUFFER
        AREA SYSTEM.DDLDCLOD
        ON STARTUP SET STATUS TO UPDATE
      INCLUDE SEGMENT SYSLOC
        FILE SYSLOC.DCLSCR
        BUFFER DEFAULT_BUFFER
      INCLUDE SEGMENT SYMSG
        ON STARTUP SET STATUS TO RETRIEVAL
      INCLUDE SEGMENT SYSUSER
        ON STARTUP SET STATUS TO UPDATE
      INCLUDE SEGMENT APPLDICT
        ON STARTUP SET STATUS TO UPDATE
      INCLUDE SEGMENT SYSDIRL
        ON STARTUP SET STATUS TO UPDATE
      INCLUDE SEGMENT ASFDICT
      INCLUDE SEGMENT SYSSQL
      INCLUDE SEGMENT SQLDEMO
      INCLUDE SEGMENT PROJSEG
      INCLUDE SEGMENT EMPDEMO
      INCLUDE SEGMENT CATSYS
      ;

```



## **Appendix B. Sample SQL Database Definition**

---



---

### Sample Database Definition

```
CREATE SCHEMA DEMOEMPL;  
SET SESSION CURRENT SCHEMA DEMOEMPL;
```

```
CREATE TABLE          BENEFITS  
  (FISCAL_YEAR        UNSIGNED NUMERIC(4,0)          NOT NULL,  
   EMP_ID             UNSIGNED NUMERIC(4,0)          NOT NULL,  
   VAC_ACCRUED        UNSIGNED DECIMAL(6,2)          NOT NULL WITH DEFAULT,  
   VAC_TAKEN          UNSIGNED DECIMAL(6,2)          NOT NULL WITH DEFAULT,  
   SICK_ACCRUED       UNSIGNED DECIMAL(6,2)          NOT NULL WITH DEFAULT,  
   SICK_TAKEN         UNSIGNED DECIMAL(6,2)          NOT NULL WITH DEFAULT,  
   STOCK_PERCENT      UNSIGNED DECIMAL(6,3)          NOT NULL WITH DEFAULT,  
   STOCK_AMOUNT       UNSIGNED DECIMAL(10,2)         NOT NULL WITH DEFAULT,  
   LAST_REVIEW_DATE   DATE                          ,  
   REVIEW_PERCENT     UNSIGNED DECIMAL(6,3)          ,  
   PROMO_DATE         DATE                          ,  
   RETIRE_PLAN        CHAR(6)                        ,  
   RETIRE_PERCENT     UNSIGNED DECIMAL(6,3)          ,  
   BONUS_AMOUNT       UNSIGNED DECIMAL(10,2)         ,  
   COMP_ACCRUED       UNSIGNED DECIMAL(6,2)          NOT NULL WITH DEFAULT,  
   COMP_TAKEN         UNSIGNED DECIMAL(6,2)          NOT NULL WITH DEFAULT,  
   EDUC_LEVEL         CHAR(06)                       ,  
   UNION_ID           CHAR(10)                       ,  
   UNION_DUES         UNSIGNED DECIMAL(10,2)         ,  
   CHECK ( (RETIRE_PLAN IN ('STOCK', 'BONDS', '401K') ) AND  
            (EDUC_LEVEL IN ('GED', 'HSDIP', 'JRCOLL', 'COLL',  
                             'MAS', 'PHD') ) ) )  
IN SQLDEMO.EMPLAREA;
```

```
CREATE TABLE          COVERAGE  
  (PLAN_CODE          CHAR(03)                        NOT NULL,  
   EMP_ID             UNSIGNED NUMERIC(4,0)          NOT NULL,  
   SELECTION_DATE     DATE                          NOT NULL WITH DEFAULT,  
   TERMINATION_DATE   DATE                          ,  
   NUM_DEPENDENTS     UNSIGNED NUMERIC(2,0)          NOT NULL WITH DEFAULT)  
IN SQLDEMO.EMPLAREA;
```

```

CREATE TABLE DEPARTMENT
  (DEPT_ID          UNSIGNED NUMERIC(4,0)      NOT NULL,
   DEPT_HEAD_ID    UNSIGNED NUMERIC(4,0)      ,
   DIV_CODE         CHAR(03)                 NOT NULL,
   DEPT_NAME       CHAR(40)                 NOT NULL)
IN SQLDEMO.INFOAREA;

```

```

CREATE TABLE DIVISION
  (DIV_CODE        CHAR(03)                 NOT NULL,
   DIV_HEAD_ID     UNSIGNED NUMERIC(4,0)     ,
   DIV_NAME        CHAR(40)                 NOT NULL)
IN SQLDEMO.INFOAREA;

```

```

CREATE TABLE EMPLOYEE
  (EMP_ID          UNSIGNED NUMERIC(4,0)      NOT NULL,
   MANAGER_ID      UNSIGNED NUMERIC(4,0)      ,
   EMP_FNAME       CHAR(20)                 NOT NULL,
   EMP_LNAME       CHAR(20)                 NOT NULL,
   DEPT_ID         UNSIGNED NUMERIC(4,0)     NOT NULL,
   STREET          CHAR(40)                 NOT NULL,
   CITY            CHAR(20)                 NOT NULL,
   STATE           CHAR(02)                 NOT NULL,
   ZIP_CODE        CHAR(09)                 NOT NULL,
   PHONE           CHAR(10)                 ,
   STATUS          CHAR                     NOT NULL,
   SS_NUMBER       UNSIGNED NUMERIC(9,0)     NOT NULL,
   START_DATE      DATE                     NOT NULL,
   TERMINATION_DATE DATE                     ,
   BIRTH_DATE      DATE                     ,
   CHECK ( ( EMP_ID <= 8999 ) AND (STATUS IN ('A', 'S', 'L', 'T') ) ) )
IN SQLDEMO.EMPLAREA;

```

```

CREATE TABLE INSURANCE_PLAN
  (PLAN_CODE       CHAR(03)                 NOT NULL,
   COMP_NAME       CHAR(40)                 NOT NULL,
   STREET          CHAR(40)                 NOT NULL,
   CITY            CHAR(20)                 NOT NULL,
   STATE           CHAR(02)                 NOT NULL,
   ZIP_CODE        CHAR(09)                 NOT NULL,
   PHONE           CHAR(10)                 NOT NULL,
   GROUP_NUMBER    UNSIGNED NUMERIC(4,0)     NOT NULL,
   DEDUCT          UNSIGNED DECIMAL(9,2)     ,
   MAX_LIFE_BENEFIT UNSIGNED DECIMAL(9,2)   ,
   FAMILY_COST     UNSIGNED DECIMAL(9,2)   ,
   DEP_COST        UNSIGNED DECIMAL(9,2)   ,
   EFF_DATE        DATE                     NOT NULL)
IN SQLDEMO.INFOAREA;

```

```

CREATE TABLE          JOB
(JOB_ID              UNSIGNED NUMERIC(4,0)          NOT NULL,
 JOB_TITLE           CHAR(20)                      NOT NULL,
 MIN_RATE            UNSIGNED DECIMAL(10,2)        ,
 MAX_RATE            UNSIGNED DECIMAL(10,2)        ,
 SALARY_IND          CHAR(01)                      ,
 NUM_OF_POSITIONS   UNSIGNED DECIMAL(4,0)         ,
 EFF_DATE            DATE                          ,
 JOB_DESC_LINE_1     VARCHAR(60)                   ,
 JOB_DESC_LINE_2     VARCHAR(60)                   ,
 CHECK ( SALARY_IND IN ('S', 'H') ) )
IN SQLDEMO.INFOAREA;

CREATE TABLE          POSITION
(EMP_ID              UNSIGNED NUMERIC(4,0)          NOT NULL,
 JOB_ID              UNSIGNED NUMERIC(4,0)          NOT NULL,
 START_DATE          DATE                          NOT NULL,
 FINISH_DATE        DATE                          ,
 HOURLY_RATE        UNSIGNED DECIMAL(7,2)         ,
 SALARY_AMOUNT      UNSIGNED DECIMAL(10,2)        ,
 BONUS_PERCENT      UNSIGNED DECIMAL(7,3)         ,
 COMM_PERCENT       UNSIGNED DECIMAL(7,3)         ,
 OVERTIME_RATE      UNSIGNED DECIMAL(5,2)         ,
 CHECK ( (HOURLY_RATE IS NOT NULL AND SALARY_AMOUNT IS NULL)
 OR (HOURLY_RATE IS NULL AND SALARY_AMOUNT IS NOT NULL) ) )
IN SQLDEMO.EMPLAREA;

CREATE SCHEMA DEMOPROJ;

SET SESSION CURRENT SCHEMA DEMOPROJ;

CREATE TABLE          ASSIGNMENT
(EMP_ID              UNSIGNED NUMERIC(4,0)          NOT NULL,
 PROJ_ID            CHAR(10)                       NOT NULL,
 START_DATE         DATE                          NOT NULL,
 END_DATE           DATE                          )
IN PROJSEG.PROJAREA;

```

```

CREATE TABLE CONSULTANT
(CON_ID UNSIGNED NUMERIC(4,0) NOT NULL,
CON_FNAME CHAR(20) NOT NULL,
CON_LNAME CHAR(20) NOT NULL,
MANAGER_ID UNSIGNED NUMERIC(4,0) NOT NULL,
DEPT_ID UNSIGNED NUMERIC(4,0) NOT NULL,
PROJ_ID CHAR(10) ,
STREET CHAR(40) ,
CITY CHAR(20) NOT NULL,
STATE CHAR(02) NOT NULL,
ZIP_CODE CHAR(09) NOT NULL,
PHONE CHAR(10) ,
BIRTH_DATE DATE ,
START_DATE DATE NOT NULL,
SS_NUMBER UNSIGNED NUMERIC(9,0) NOT NULL,
RATE UNSIGNED DECIMAL(7,2) ,
CHECK ( (CON_ID >= 9000 AND CON_ID <= 9999) ) )
IN PROJSEG.PROJAREA;

CREATE TABLE EXPERTISE
(EMP_ID UNSIGNED NUMERIC(4,0) NOT NULL,
SKILL_ID UNSIGNED NUMERIC(4,0) NOT NULL,
SKILL_LEVEL CHAR(02) ,
EXP_DATE DATE )
IN PROJSEG.PROJAREA;

CREATE TABLE PROJECT
(PROJ_ID CHAR(10) NOT NULL,
PROJ_LEADER_ID UNSIGNED NUMERIC(4,0) ,
EST_START_DATE DATE ,
EST_END_DATE DATE ,
ACT_START_DATE DATE ,
ACT_END_DATE DATE ,
EST_MAN_HOURS UNSIGNED DECIMAL(7,2) ,
ACT_MAN_HOURS UNSIGNED DECIMAL(7,2) ,
PROJ_DESC VARCHAR(60) NOT NULL)
IN PROJSEG.PROJAREA;

CREATE TABLE SKILL
(SKILL_ID UNSIGNED NUMERIC(4,0) NOT NULL,
SKILL_NAME CHAR(20) NOT NULL,
SKILL_DESC VARCHAR(60) )
IN PROJSEG.PROJAREA;

```

---

```
CREATE UNIQUE CALC KEY ON DEMOEMPL.DEPARTMENT(DEPT_ID);
CREATE UNIQUE CALC KEY ON DEMOEMPL.DIVISION(DIV_CODE);
CREATE UNIQUE CALC KEY ON DEMOEMPL.EMPLOYEE(EMP_ID);
CREATE UNIQUE CALC KEY ON DEMOEMPL.INSURANCE_PLAN(PLAN_CODE);
CREATE UNIQUE CALC KEY ON DEMOEMPL.JOB(JOB_ID);
CREATE UNIQUE CALC KEY ON DEMOPROJ.CONSULTANT(CON_ID);
CREATE UNIQUE CALC KEY ON DEMOPROJ.PROJECT(PROJ_ID);
CREATE UNIQUE CALC KEY ON DEMOPROJ.SKILL(SKILL_ID);

CREATE UNIQUE INDEX AS_EMPROJ_NDX ON
    DEMOPROJ.ASSIGNMENT(EMP_ID, PROJ_ID);

CREATE UNIQUE INDEX EX_EMPSKILL_NDX ON
    DEMOPROJ.EXPERTISE(EMP_ID, SKILL_ID);

CREATE INDEX CO_CODE_NDX ON DEMOEMPL.COVERAGE(PLAN_CODE)
    IN SQLDEMO.INDXAREA;

CREATE INDEX DE_CODE_NDX ON DEMOEMPL.DEPARTMENT(DIV_CODE);
CREATE INDEX DI_HEAD_NDX ON DEMOEMPL.DIVISION(DIV_HEAD_ID);
CREATE INDEX DE_HEAD_NDX ON DEMOEMPL.DEPARTMENT(DEPT_HEAD_ID);
CREATE INDEX EM_MANAGER_NDX ON DEMOEMPL.EMPLOYEE(MANAGER_ID)
    IN SQLDEMO.INDXAREA;
```

---

```

CREATE INDEX EM_NAME_NDX ON DEMOEMPL.EMPLOYEE(EMP_LNAME, EMP_FNAME)
    IN SQLDEMO.INDXAREA;

CREATE INDEX EM_DEPT_NDX ON DEMOEMPL.EMPLOYEE(DEPT_ID)
    IN SQLDEMO.INDXAREA;

CREATE INDEX IN_NAME_NDX ON DEMOEMPL.INSURANCE_PLAN(COMP_NAME)
    COMPRESSED;

CREATE INDEX PO_JOB_NDX ON DEMOEMPL.POSITION(JOB_ID)
    IN SQLDEMO.INDXAREA;

CREATE INDEX CN_NAME_NDX ON DEMOPROJ.CONSULTANT(CON_LNAME,CON_FNAME);

CREATE CONSTRAINT EMP_BENEFITS
    DEMOEMPL.BENEFITS (EMP_ID) REFERENCES
    DEMOEMPL.EMPLOYEE (EMP_ID)
    LINKED CLUSTERED
    ORDER BY (FISCAL_YEAR DESC);

CREATE CONSTRAINT INSPLAN_COVERAGE
    DEMOEMPL.COVERAGE (PLAN_CODE) REFERENCES
    DEMOEMPL.INSURANCE_PLAN (PLAN_CODE)
    UNLINKED;

CREATE CONSTRAINT EMP_COVERAGE
    DEMOEMPL.COVERAGE (EMP_ID) REFERENCES
    DEMOEMPL.EMPLOYEE (EMP_ID)
    LINKED CLUSTERED
    ORDER BY ( PLAN_CODE) UNIQUE;

CREATE CONSTRAINT DIVISION_DEPT
    DEMOEMPL.DEPARTMENT (DIV_CODE) REFERENCES
    DEMOEMPL.DIVISION (DIV_CODE)
    UNLINKED;

```

---

```

CREATE CONSTRAINT EMP_DEPT_HEAD
  DEMOEMPL.DEPARTMENT (DEPT_HEAD_ID) REFERENCES
  DEMOEMPL.EMPLOYEE (EMP_ID)
  UNLINKED;

CREATE CONSTRAINT EMP_DIV_HEAD
  DEMOEMPL.DIVISION (DIV_HEAD_ID) REFERENCES
  DEMOEMPL.EMPLOYEE (EMP_ID)
  UNLINKED;

CREATE CONSTRAINT DEPT_EMPLOYEE
  DEMOEMPL.EMPLOYEE (DEPT_ID) REFERENCES
  DEMOEMPL.DEPARTMENT (DEPT_ID)
  UNLINKED;

CREATE CONSTRAINT MANAGER_EMP
  DEMOEMPL.EMPLOYEE (MANAGER_ID) REFERENCES
  DEMOEMPL.EMPLOYEE (EMP_ID)
  UNLINKED;

CREATE CONSTRAINT SKILL_EXPERTISE
  DEMOPROJ.EXPERTISE (SKILL_ID) REFERENCES
  DEMOPROJ.SKILL (SKILL_ID)
  LINKED CLUSTERED;

CREATE CONSTRAINT EMP_POSITION
  DEMOEMPL.POSITION (EMP_ID) REFERENCES
  DEMOEMPL.EMPLOYEE (EMP_ID)
  LINKED CLUSTERED
  ORDER BY (JOB_ID) UNIQUE;

CREATE CONSTRAINT JOB_POSITION
  DEMOEMPL.POSITION (JOB_ID) REFERENCES
  DEMOEMPL.JOB (JOB_ID)
  UNLINKED;

CREATE CONSTRAINT PROJECT_ASSIGN
  DEMOPROJ.ASSIGNMENT (PROJ_ID) REFERENCES
  DEMOPROJ.PROJECT (PROJ_ID)
  LINKED CLUSTERED;

CREATE CONSTRAINT PROJECT_CONSULT
  DEMOPROJ.CONSULTANT (PROJ_ID) REFERENCES
  DEMOPROJ.PROJECT (PROJ_ID)
  LINKED INDEX ORDER BY (PROJ_ID);

ALTER TABLE DEMOEMPL.COVERAGE
  DROP DEFAULT INDEX;

ALTER TABLE DEMOEMPL.DEPARTMENT
  DROP DEFAULT INDEX;

ALTER TABLE DEMOEMPL.DIVISION
  DROP DEFAULT INDEX;

ALTER TABLE DEMOEMPL.EMPLOYEE
  DROP DEFAULT INDEX;

```

---

```
ALTER TABLE DEMOEMPL.INSURANCE_PLAN
  DROP DEFAULT INDEX;
```

```
ALTER TABLE DEMOEMPL.POSITION
  DROP DEFAULT INDEX;
```

```
ALTER TABLE DEMOPROJ.ASSIGNMENT
  DROP DEFAULT INDEX;
```

```
ALTER TABLE DEMOPROJ.CONSULTANT
  DROP DEFAULT INDEX;
```

```
ALTER TABLE DEMOPROJ.EXPERTISE
  DROP DEFAULT INDEX;
```

```
CREATE VIEW DEMOEMPL.EMP_VACATION
  (EMP_ID, DEPT_ID, VAC_TIME)
AS SELECT E.EMP_ID, DEPT_ID, SUM(VAC_ACCRUED) - SUM(VAC_TAKEN)
  FROM DEMOEMPL.EMPLOYEE E, DEMOEMPL.BENEFITS B
  WHERE E.EMP_ID = B.EMP_ID
  GROUP BY DEPT_ID, E.EMP_ID;
```

```
CREATE VIEW DEMOEMPL.OPEN_POSITIONS
  (JOB_ID, JOB_NAME, OPEN_POS)
AS SELECT J.JOB_ID, J.JOB_TITLE,
  (J.NUM_OF_POSITIONS - COUNT(P.JOB_ID))
  FROM DEMOEMPL.JOB J, DEMOEMPL.POSITION P
  WHERE P.FINISH_DATE IS NULL AND P.JOB_ID = J.JOB_ID
  PRESERVE DEMOEMPL.JOB
  GROUP BY J.JOB_ID, J.JOB_TITLE, J.NUM_OF_POSITIONS
  HAVING (J.NUM_OF_POSITIONS - COUNT(P.JOB_ID)) > 0;
```

```
CREATE VIEW DEMOEMPL.EMP_HOME_INFO
AS SELECT EMP_ID, EMP_LNAME, EMP_FNAME, STREET, CITY, STATE,
  ZIP_CODE, PHONE
  FROM DEMOEMPL.EMPLOYEE;
```

```
CREATE VIEW DEMOEMPL.EMP_WORK_INFO
AS SELECT EMP_ID, MANAGER_ID, START_DATE, TERMINATION_DATE
  FROM DEMOEMPL.EMPLOYEE;
```

# **Appendix C. Sample Non-SQL Database Definition**

---



---

### Sample Database Schema Definition

```
ADD SCHEMA NAME IS EMPSCHM VERSION IS 100
SCHEMA DESCRIPTION IS 'EMPLOYEE DEMO DATABASE'
COMMENTS 'INSTALLATION: COMMONWEATHER CORPORATION'
.
ADD AREA NAME IS EMP-DEMO-REGION
.
ADD AREA NAME IS ORG-DEMO-REGION
.
ADD AREA NAME IS INS-DEMO-REGION
.
ADD RECORD NAME IS COVERAGE
SHARE STRUCTURE OF RECORD COVERAGE VERSION IS 100
RECORD ID IS 0400
LOCATION MODE IS VIA EMP-COVERAGE SET
WITHIN AREA INS-DEMO-REGION
OFFSET 5 PAGES FOR 45 PAGES
.
ADD RECORD NAME IS DENTAL-CLAIM
SHARE STRUCTURE OF RECORD DENTAL-CLAIM VERSION IS 100
RECORD ID IS 0405
LOCATION MODE IS VIA COVERAGE-CLAIMS SET
WITHIN AREA INS-DEMO-REGION
OFFSET 5 PAGES FOR 45 PAGES
MINIMUM ROOT LENGTH IS 130 CHARACTERS
MINIMUM FRAGMENT LENGTH IS RECORD LENGTH
.
ADD RECORD NAME IS DEPARTMENT
SHARE STRUCTURE OF RECORD DEPARTMENT VERSION IS 100
RECORD ID IS 0410
LOCATION MODE IS CALC USING DEPT-ID-0410
DUPLICATES NOT ALLOWED
WITHIN AREA ORG-DEMO-REGION
OFFSET 5 PAGES FOR 45 PAGES
.
ADD RECORD NAME IS EMPLOYEE
SHARE STRUCTURE OF RECORD EMPLOYEE VERSION IS 100
RECORD ID IS 0415
LOCATION MODE IS CALC USING EMP-ID-0415
DUPLICATES NOT ALLOWED
WITHIN AREA EMP-DEMO-REGION
OFFSET 5 PAGES FOR 95 PAGES
.
ADD RECORD NAME IS EMPOSITION
SHARE STRUCTURE OF RECORD EMPOSITION VERSION IS 100
RECORD ID IS 0420
LOCATION MODE IS VIA EMP-EMPOSITION SET
WITHIN AREA EMP-DEMO-REGION
OFFSET 5 PAGES FOR 95 PAGES
.
```

---

```
ADD RECORD NAME IS EXPERTISE
SHARE STRUCTURE OF RECORD EXPERTISE VERSION IS 100
RECORD ID IS 0425
LOCATION MODE IS VIA          EMP-EXPERTISE SET
WITHIN AREA EMP-DEMO-REGION
OFFSET 5 PAGES FOR 95 PAGES
.
ADD RECORD NAME IS HOSPITAL-CLAIM
SHARE STRUCTURE OF RECORD HOSPITAL-CLAIM VERSION IS 100
RECORD ID IS 0430
LOCATION MODE IS VIA          COVERAGE-CLAIMS SET
WITHIN AREA INS-DEMO-REGION
OFFSET 5 PAGES FOR 45 PAGES
.
ADD RECORD NAME IS INSURANCE-PLAN
SHARE STRUCTURE OF RECORD INSURANCE-PLAN VERSION IS 100
RECORD ID IS 0435
LOCATION MODE IS CALC        USING INS-PLAN-CODE-0435
                             DUPLICATES NOT ALLOWED
WITHIN AREA INS-DEMO-REGION
OFFSET 1 PAGE FOR 4 PAGES
.
ADD RECORD NAME IS JOB
SHARE STRUCTURE OF RECORD JOB VERSION IS 100
RECORD ID IS 0440
LOCATION MODE IS CALC        USING JOB-ID-0440
                             DUPLICATES NOT ALLOWED
WITHIN AREA ORG-DEMO-REGION
OFFSET 5 PAGES FOR 45 PAGES
MINIMUM ROOT LENGTH IS CONTROL LENGTH
MINIMUM FRAGMENT LENGTH IS RECORD LENGTH
CALL IDMSCOMP BEFORE STORE
CALL IDMSCOMP BEFORE MODIFY
CALL IDMSDCOM AFTER GET
.
```

---

```

ADD RECORD NAME IS NON-HOSP-CLAIM
SHARE STRUCTURE OF RECORD NON-HOSP-CLAIM VERSION IS 100
RECORD ID IS 0445
LOCATION MODE IS VIA          COVERAGE-CLAIMS SET
WITHIN AREA INS-DEMO-REGION
OFFSET 5 PAGES FOR 45 PAGES
MINIMUM ROOT LENGTH IS      248 CHARACTERS
MINIMUM FRAGMENT LENGTH IS  RECORD LENGTH
.
ADD RECORD NAME IS OFFICE
SHARE STRUCTURE OF RECORD OFFICE VERSION IS 100
RECORD ID IS 0450
LOCATION MODE IS CALC        USING OFFICE-CODE-0450
                             DUPLICATES NOT ALLOWED
WITHIN AREA ORG-DEMO-REGION
OFFSET 5 PAGES FOR 45 PAGES
.
ADD RECORD NAME IS SKILL
SHARE STRUCTURE OF RECORD SKILL VERSION IS 100
RECORD ID IS 0455
LOCATION MODE IS CALC        USING SKILL-ID-0455
                             DUPLICATES NOT ALLOWED
WITHIN AREA ORG-DEMO-REGION
OFFSET 5 PAGES FOR 45 PAGES
.
ADD RECORD NAME IS STRUCTURE
SHARE STRUCTURE OF RECORD STRUCTURE VERSION IS 100
RECORD ID IS 0460
LOCATION MODE IS VIA          MANAGES SET
WITHIN AREA EMP-DEMO-REGION
OFFSET 5 PAGES FOR 95 PAGES
.
ADD SET NAME IS COVERAGE-CLAIMS
ORDER IS LAST
MODE IS CHAIN LINKED TO PRIOR
OWNER IS COVERAGE
    NEXT DBKEY POSITION IS AUTO
    PRIOR DBKEY POSITION IS AUTO
MEMBER IS HOSPITAL-CLAIM
    NEXT DBKEY POSITION IS AUTO
    PRIOR DBKEY POSITION IS AUTO
    MANDATORY AUTOMATIC
MEMBER IS NON-HOSP-CLAIM
    NEXT DBKEY POSITION IS AUTO
    PRIOR DBKEY POSITION IS AUTO
    MANDATORY AUTOMATIC
MEMBER IS DENTAL-CLAIM
    NEXT DBKEY POSITION IS AUTO
    PRIOR DBKEY POSITION IS AUTO
    MANDATORY AUTOMATIC
.

```

---

```
ADD SET NAME IS DEPT-EMPLOYEE
ORDER IS SORTED
MODE IS CHAIN LINKED TO PRIOR
OWNER IS DEPARTMENT
    NEXT DBKEY POSITION IS AUTO
    PRIOR DBKEY POSITION IS AUTO
MEMBER IS EMPLOYEE
    NEXT DBKEY POSITION IS AUTO
    PRIOR DBKEY POSITION IS AUTO
LINKED TO OWNER
    OWNER DBKEY POSITION IS AUTO
OPTIONAL AUTOMATIC
ASCENDING KEY IS ( EMP-LAST-NAME-0415
                    EMP-FIRST-NAME-0415 )
DUPLICATES LAST
```

```
.
ADD SET NAME IS EMP-COVERAGE
ORDER IS FIRST
MODE IS CHAIN LINKED TO PRIOR
OWNER IS EMPLOYEE
    NEXT DBKEY POSITION IS AUTO
    PRIOR DBKEY POSITION IS AUTO
MEMBER IS COVERAGE
    NEXT DBKEY POSITION IS AUTO
    PRIOR DBKEY POSITION IS AUTO
LINKED TO OWNER
    OWNER DBKEY POSITION IS AUTO
MANDATORY AUTOMATIC
```

```
.
ADD SET NAME IS EMP-EMPOSITION
ORDER IS FIRST
MODE IS CHAIN LINKED TO PRIOR
OWNER IS EMPLOYEE
    NEXT DBKEY POSITION IS AUTO
    PRIOR DBKEY POSITION IS AUTO
MEMBER IS EMPOSITION
    NEXT DBKEY POSITION IS AUTO
    PRIOR DBKEY POSITION IS AUTO
LINKED TO OWNER
    OWNER DBKEY POSITION IS AUTO
MANDATORY AUTOMATIC
```

---

```

ADD SET NAME IS EMP-EXPERTISE
ORDER IS SORTED
MODE IS CHAIN LINKED TO PRIOR
OWNER IS EMPLOYEE
    NEXT DBKEY POSITION IS AUTO
    PRIOR DBKEY POSITION IS AUTO
MEMBER IS EXPERTISE
    NEXT DBKEY POSITION IS AUTO
    PRIOR DBKEY POSITION IS AUTO
    LINKED TO OWNER
        OWNER DBKEY POSITION IS AUTO
MANDATORY AUTOMATIC
DESCENDING KEY IS (SKILL-LEVEL-0425 )
    DUPLICATES FIRST
.
ADD SET NAME IS EMP-NAME-NDX
ORDER IS SORTED
MODE IS INDEX BLOCK CONTAINS 40 KEYS
OWNER IS SYSTEM
    WITHIN AREA EMP-DEMO-REGION
    OFFSET 1 PAGE FOR 4 PAGES
MEMBER IS EMPLOYEE
    INDEX DBKEY POSITION IS AUTO
    OPTIONAL AUTOMATIC
    ASCENDING KEY IS ( EMP-LAST-NAME-0415
                        EMP-FIRST-NAME-0415 )
        COMPRESSED
        DUPLICATES LAST
.
ADD SET NAME IS JOB-EMPOSITION
ORDER IS NEXT
MODE IS CHAIN LINKED TO PRIOR
OWNER IS JOB
    NEXT DBKEY POSITION IS AUTO
    PRIOR DBKEY POSITION IS AUTO
MEMBER IS EMPOSITION
    NEXT DBKEY POSITION IS AUTO
    PRIOR DBKEY POSITION IS AUTO
    LINKED TO OWNER
        OWNER DBKEY POSITION IS AUTO
OPTIONAL MANUAL
.

```

---

```

ADD SET NAME IS JOB-TITLE-NDX
ORDER IS SORTED
MODE IS INDEX BLOCK CONTAINS 30 KEYS
OWNER IS SYSTEM
    WITHIN AREA ORG-DEMO-REGION
    OFFSET 1 PAGE FOR 4 PAGES
MEMBER IS JOB
    INDEX DBKEY POSITION IS AUTO
    OPTIONAL AUTOMATIC
    ASCENDING KEY IS ( TITLE-0440 )
    DUPLICATES NOT ALLOWED
.
ADD SET NAME IS MANAGES
ORDER IS NEXT
MODE IS CHAIN LINKED TO PRIOR
OWNER IS EMPLOYEE
    NEXT DBKEY POSITION IS AUTO
    PRIOR DBKEY POSITION IS AUTO
MEMBER IS STRUCTURE
    NEXT DBKEY POSITION IS AUTO
    PRIOR DBKEY POSITION IS AUTO
    LINKED TO OWNER
    OWNER DBKEY POSITION IS AUTO
    MANDATORY AUTOMATIC
.
ADD SET NAME IS OFFICE-EMPLOYEE
ORDER IS SORTED
MODE IS INDEX BLOCK CONTAINS 30 KEYS
OWNER IS OFFICE
    NEXT DBKEY POSITION IS AUTO
    PRIOR DBKEY POSITION IS AUTO
MEMBER IS EMPLOYEE
    INDEX DBKEY POSITION IS AUTO
    LINKED TO OWNER
    OWNER DBKEY POSITION IS AUTO
    OPTIONAL AUTOMATIC
    ASCENDING KEY IS ( EMP-LAST-NAME-0415
                        EMP-FIRST-NAME-0415 )
    COMPRESSED
    DUPLICATES LAST
.
ADD SET NAME IS REPORTS-TO
ORDER IS NEXT
MODE IS CHAIN LINKED TO PRIOR
OWNER IS EMPLOYEE
    NEXT DBKEY POSITION IS AUTO
    PRIOR DBKEY POSITION IS AUTO
MEMBER IS STRUCTURE
    NEXT DBKEY POSITION IS AUTO
    PRIOR DBKEY POSITION IS AUTO
    LINKED TO OWNER
    OWNER DBKEY POSITION IS AUTO
    OPTIONAL MANUAL
.

```

---

```

ADD SET NAME IS SKILL-EXPERTISE
ORDER IS SORTED
MODE IS INDEX BLOCK CONTAINS 30 KEYS
OWNER IS SKILL
    NEXT DBKEY POSITION IS AUTO
    PRIOR DBKEY POSITION IS AUTO
MEMBER IS EXPERTISE
    INDEX DBKEY POSITION IS AUTO
    LINKED TO OWNER
        OWNER DBKEY POSITION IS AUTO
MANDATORY AUTOMATIC
DESCENDING KEY IS ( SKILL-LEVEL-0425 )
    DUPLICATES FIRST
.
ADD SET NAME IS SKILL-NAME-NDX
ORDER IS SORTED
MODE IS INDEX BLOCK CONTAINS 30 KEYS
OWNER IS SYSTEM
    WITHIN AREA ORG-DEMO-REGION
    OFFSET 1 PAGE FOR 4 PAGES
MEMBER IS SKILL
    INDEX DBKEY POSITION IS AUTO
    OPTIONAL AUTOMATIC
    ASCENDING KEY IS ( SKILL-NAME-0455 )
    DUPLICATES NOT ALLOWED
.
VALIDATE
.

```

**Sample Database Subschema Definition**

```

ADD SUBSCHEMA NAME IS EMPSS01
    OF SCHEMA NAME IS EMPSCHM VERSION 100
COMMENTS 'THIS IS THE COMPLETE VIEW OF EMPSCHM'
.
ADD AREA NAME IS EMP-DEMO-REGION
.
ADD AREA NAME IS INS-DEMO-REGION
.
ADD AREA NAME IS ORG-DEMO-REGION
.
ADD RECORD NAME IS COVERAGE
.
ADD RECORD NAME IS DENTAL-CLAIM
.
ADD RECORD NAME IS DEPARTMENT
.
ADD RECORD NAME IS EMPLOYEE
.
ADD RECORD NAME IS EMPOSITION
.

```

---

```
ADD RECORD NAME IS EXPERTISE
.
ADD RECORD NAME IS HOSPITAL-CLAIM
.
ADD RECORD NAME IS INSURANCE-PLAN
.
ADD RECORD NAME IS JOB
.
ADD RECORD NAME IS NON-HOSP-CLAIM
.
ADD RECORD NAME IS OFFICE
.
ADD RECORD NAME IS SKILL
.
ADD RECORD NAME IS STRUCTURE
.
ADD SET COVERAGE-CLAIMS
.
ADD SET DEPT-EMPLOYEE
.
ADD SET EMP-COVERAGE
.
ADD SET EMP-EXPERTISE
.
ADD SET EMP-NAME-NDX
.
ADD SET EMP-EMPOSITION
.
ADD SET JOB-EMPOSITION
.
ADD SET JOB-TITLE-NDX
.
ADD SET MANAGES
.
ADD SET OFFICE-EMPLOYEE
.
ADD SET REPORTS-TO
.
ADD SET SKILL-EXPERTISE
.
ADD SET SKILL-NAME-NDX
.
GENERATE
.
```

# Appendix D. Native VSAM Considerations

---

D.1 Native VSAM data set structures . . . . .	D-4
D.2 CA-IDMS/DB native VSAM definitions . . . . .	D-5
D.2.1 Schema definition . . . . .	D-5
D.2.2 DMCL definition . . . . .	D-6
D.3 DML functions with native VSAM . . . . .	D-8



---

CA-IDMS/DB can access information from native VSAM data sets. A native VSAM data set is one that is defined to VSAM and contains VSAM records. Although a native VSAM data set is not structured as a CA-IDMS/DB database, it can be accessed as if it were.

**Note:** Native VSAM files are different from *database* files that have VSAM as an access method.

This appendix describes:

- Native VSAM data set structures
- Schema, segment, and DMCL considerations
- DML functions that can be used to access native VSAM structures

## D.1 Native VSAM data set structures

You can structure a native VSAM data set as a key-sequenced data set (KSDS), an entry-sequenced data set (ESDS), or a relative record data set (RRDS). The following table lists the characteristics of each data set structure.

### Native VSAM data set structures

VSAM structure <sup>1</sup>	Access method	Record format	
KSDS	<ul style="list-style-type: none"><li>■ Prime index</li><li>■ Alternate indexes</li></ul>	Fixed or variable	Spanned or nonspanned
ESDS	<ul style="list-style-type: none"><li>■ Relative byte address (RBA)</li><li>■ Alternate indexes</li></ul>	Fixed or variable	Spanned or nonspanned
RRDS	Relative record number	Fixed	Nonspanned

<sup>1</sup> KSDS = key-sequenced data set  
ESDS = entry-sequenced data set  
RRDS = relative record data set

---

## D.2 CA-IDMS/DB native VSAM definitions

To use native VSAM files in a CA-IDMS environment, you define native VSAM structures in the schema, segment, and DMCL. Schema, segment, and DMCL definitions are discussed separately below.

### D.2.1 Schema definition

The schema establishes the correspondences between the logical characteristics of the native VSAM file and CA-IDMS/DB, as follows:

**A file:** A CA-IDMS *file* represents a VSAM cluster or path:

- A KSDS with a prime index or alternate indexes or both
- An ESDS with or without alternate indexes
- An RRDS

An *area* represents a KSDS, ESDS, or RRDS data component. You map one area to each VSAM file; each area must have a unique page range. The page range is a function of the VSAM data set structure. For information on how to determine the page range, see "AREA statements" in Chapter 6, "Physical Database DDL Statements" on page 6-1.

**A schema record:** A schema *record* represents a VSAM data record:

- All VSAM data records can be represented as a record with a location mode of VSAM
- A KSDS with a prime index or alternate indexes and an ESDS with an alternate index can be represented as a CALC record

**A schema set:** A schema *set* represents the index of a VSAM data set and related records. These sets are sorted and maintained through the following types of native VSAM data set structures:

- A KSDS with a prime or alternate index
- An ESDS with an alternate index

You define sets with alternate indexes in the schema to allow record occurrences with duplicate sort keys. These record occurrences are retrieved in the order in which the records were stored, regardless of the order in which the set is searched. For sorted sets that do not allow duplicate sort keys, you can use any index to maintain the set.

Native VSAM sets allow you to code application programs that:

- Access a specific record directly by sort key
- Access records by generic sort key

- Process the native VSAM file in sort-key sequence from the start of the file or from a specified starting point.

**Relationships between VSAM and CA-IDMS/DB structures:** The schema, AREA, RECORD, SET, and SET statements, and the segment statements needed to represent native VSAM structures are listed in the following table.

VSAM structure	CA-IDMS/DB structure	DDL statement
KSDS ESDS RRDS PATH	File	CREATE FILE <u>segment.file-name</u>
Data component: KSDS, ESDS, RRDS	Area	Schema definition: ADD AREA NAME IS <u>area-name</u> ... Segment definition: CREATE AREA <u>segment.area-name</u> ...
VSAM data record	Record	ADD RECORD NAME IS <u>record-name</u> LOCATION MODE IS VSAM VSAM TYPE IS <u>type</u> WITHIN AREA <u>area-name</u> .
VSAM data record: KSDS with prime or alternate index  ESDS alternate index	CALC record	ADD RECORD NAME IS <u>record-name</u> LOCATION MODE IS VSAM CALC USING <u>calc-element-name</u> DUPLICATES ARE <u>duplicates-option</u> VSAM TYPE IS <u>type</u> WITHIN AREA <u>area-name</u> .
KSDS prime or alternate index  ESDS alternate index	Set (sorted by prime or alternate key)	ADD SET NAME IS <u>set-name</u> MODE IS VSAM INCLUDE MEMBER IS <u>record-name</u> MANDATORY AUTOMATIC ASCENDING KEY IS <u>sort-key-name</u> .

## D.2.2 DMCL definition

The DMCL module establishes the correspondences at runtime between physical characteristics of the database and the native VSAM files. You describe native VSAM files to be accessed by CA-IDMS/DB in the DMCL BUFFER statements with the following:

- PAGE CONTAINS specifies the size of the largest control interval of any native VSAM file associated with the buffer
- BUFFER CONTAINS specifies the number of I/O buffers in the buffer pool to be used to transfer records between memory and auxiliary storage

- NATIVE VSAM specifies that the buffer pool is for use exclusively with native VSAM files

## D.3 DML functions with native VSAM

To access information from a native VSAM data set, CA-IDMS/DB converts DML statements issued by the application program into record-level (not control-interval) VSAM macro variations (for example, ACB, RPL) and passes control to VSAM. No changes have to be made to the VSAM data set. A local run unit or central version appears to VSAM as a single application that opens VSAM data clusters, activates VSAM paths using local-shared resources (LSR) or nonshared resources (NSR), accesses data records, and closes the clusters and paths.

The following table lists different VSAM structures and the CA-IDMS/DB DML functions that can be used to access the VSAM structures.

### **DML functions for native VSAM data set access**

<b>CA-IDMS/DB DML statement</b>	<b>VSAM structure</b>
STORE last within area	ESDS
STORE direct by db-key	RRDS
STORE physical sequential	KSDS
ERASE	KSDS or RRDS
FIND/OBTAIN FIRST/NEXT WITHIN SET	KSDS or ESDS with a primary index or alternate indexes
FIND/OBTAIN LAST/PRIOR WITHIN SET	KSDS or ESDS with a primary index or alternate indexes
FIND/OBTAIN WITHIN SET USING SORT KEY	KSDS or ESDS
FIND/OBTAIN FIRST/NEXT WITHIN AREA	KSDS, ESDS, or RRDS
FIND/OBTAIN LAST/PRIOR WITHIN AREA	KSDS, ESDS, or RRDS
FIND/OBTAIN CALC	KSDS or ESDS with a primary index or alternate indexes
FIND/OBTAIN CALC DUPLICATE	KSDS or ESDS with a primary index or alternate indexes
FIND/OBTAIN DB-KEY	ESDS or RRDS
MODIFY, changing CALC key or sort key	KSDS or ESDS with a primary index or alternate indexes
MODIFY, without changing CALC key or sort key	KSDS, ESDS, or RRDS
MODIFY, changing record length	KSDS
ROLLBACK following STORE (without restore and rollforward)	KSDS or RRDS
ROLLBACK following ERASE (without restore and rollforward)	KSDS or RRDS
ROLLBACK following MODIFY (without restore and rollforward)	KSDS or RRDS



# Appendix E. Batch Compiler Execution JCL

---

E.1 Overview of batch compilation	E-4
E.2 OS/390 JCL	E-7
E.2.1 Schema compiler	E-7
E.2.2 Subschema compiler	E-8
E.3 VSE/ESA JCL	E-10
E.3.1 =COPY facility	E-10
E.3.2 Schema compiler	E-10
E.3.3 Subschema compiler	E-12
E.3.4 IDMSLBLS procedure	E-14
E.4 CMS commands	E-20
E.4.1 Schema compiler	E-20
E.4.2 Subschema compiler	E-21
E.5 BS2000/OSD JCL	E-23
E.5.1 =COPY facility	E-23
E.5.2 Schema compiler	E-23
E.5.3 Subschema compiler	E-25



---

This appendix contains the following:

- An overview of batch compilation
- JCL/commands you need to execute non-SQL schema and subschema statements under the central version or in local mode

## E.1 Overview of batch compilation

**Local mode considerations:** The DDL compilers can be run under the central version or in local mode. If the central version has access to the DDLML or DDLDCLOUD area of the dictionary in update usage mode, an attempt to execute a compiler in local mode without specifying USAGE RETRIEVAL in a SIGNON statement will terminate with an error code of 0966.

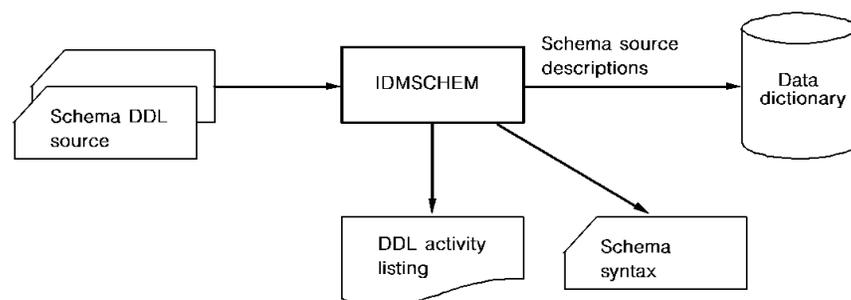
To ensure the integrity of the dictionary, either journal local mode compilations or back up the dictionary *before each local mode compilation*. The central version uses automatic recovery procedures to ensure the integrity of the dictionary.

**Naming the dictionary and system:** In an operating environment with multiple dictionaries, the name of the dictionary to be accessed (and/or the DC/UCF system on which it resides) can be specified through SYSIDMS parameters or on the compiler SIGNON statement or the command facility CONNECT statement. If specified in both places, the SIGNON or CONNECT specification takes precedence. If specified in the SYSIDMS file, the name of the dictionary is specified using DICTNAME and the DC/UCF system on which it resides is specified using DICTNODE.

►► For more information about the SYSIDMS parameter file, see Chapter 23, “Dictionaries and Runtime Environments” on page 23-1.

**Compiling a non-SQL defined schema:** To compile a schema in batch mode, execute the program IDMSCHEM. Input and output are as follows:

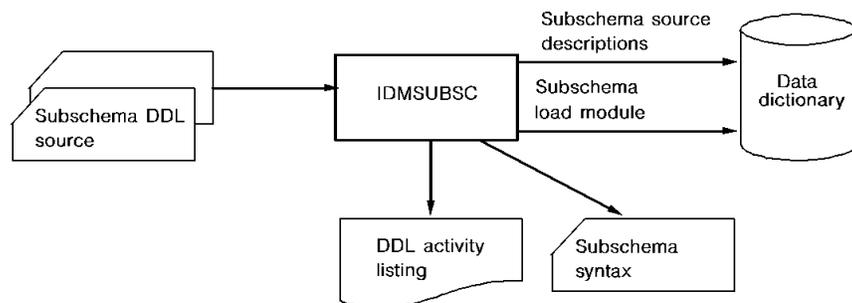
Input	Schema source statements
Output	<ul style="list-style-type: none"> <li>■ A source description of the schema stored in the dictionary</li> <li>■ A Schema Compiler Activity List</li> <li>■ A card image file containing schema syntax, if the source input contains a PUNCH statement</li> </ul>



►► To compile an SQL-defined schema, you submit SQL DDL statements through the CA-IDMS Command Facility. For sample job streams, refer to the *CA-IDMS Command Facility* document.

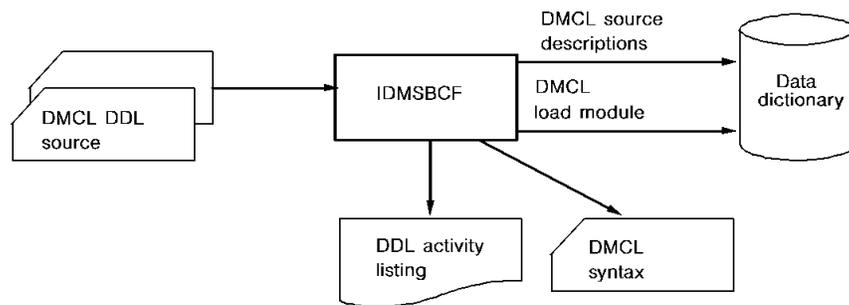
**Compiling a subschema:** To compile a subschema in batch mode, execute the program IDMSUBSC. Input and output are as follows:

Input	Subschema source statements
Output	<ul style="list-style-type: none"> <li>■ A source description of the subschema stored in the dictionary</li> <li>■ A Subschema Compiler Activity List</li> <li>■ A subschema load module stored in the dictionary load area (DDLDCLOD), if the source input contains a GENERATE statement</li> <li>■ A card image file containing schema syntax, if the source input contains a PUNCH statement</li> </ul>



**Defining segments, DMCLs, and database name tables:** To define a DMCL in batch mode, execute the program IDMSBCF (the CA-IDMS Command Facility). Input and output are as follows:

Input	Segments, DMCL, and database name table source statements as described in Chapter 6, “Physical Database DDL Statements” on page 6-1.
Output	<ul style="list-style-type: none"> <li>■ Segment, DMCL, and database name table source descriptions stored in the dictionary</li> <li>■ A DMCL or database name table load module, if the source contains a GENERATE statement</li> <li>■ A command facility activity listing</li> <li>■ A card image file containing DDL syntax or DMCL or database name table object code, if the source input contains a punch statement</li> </ul>



►► For sample IDMSBCF job streams, refer to the *CA-IDMS Command Facility* document.

## E.2 OS/390 JCL

This section provides the OS/390 JCL you need to run the schema and subschema compilers (central version and local mode).

### E.2.1 Schema compiler

#### IDMSCHEM — central version IDMSCHEM (OS/390)

```
//SCHEMA EXEC PGM=IDMSCHEM,REGION=1024K
//STEPLIB DD DSN=idms.dba.loadlib,DISP=SHR
// DD DSN=idms.loadlib,DISP=SHR
//sysctl DD DSN=idms.sysctl,DISP=SHR
//dcmsg DD DSN=idms.sysmsg.ddldcmsg,DISP=SHR
//SYSLST DD SYSOUT=A
//SYSPCH DD DSN=&&PCH, DISP=(NEW,PASS,DELETE),
           DCB=(RECFM=FB,BLKSIZE=9040,LRECL=80),
           SPACE=space-specification,
           UNIT=unit
//SYSIDMS DD *
Input SYSIDMS parameters, as required
/*
//SYSIPT DD *
Schema DDL source statements
```

#### Note:

The SYSPCH DD statement is required only if the DDL specifies PUNCH TO SYSPCH.

idms.dba.loadlib	Dataset name of the load library containing the DMCL and database name table load modules
idms.loadlib	Dataset name of the load library containing the CA-IDMS executable modules
sysctl	DDname of the SYSCTL file
idms.sysctl	Dataset name of the SYSCTL file
dcmsg	DDname of the message (DDLDCMSG) area
idms.sysmsg.ddldcmsg	Dataset name of the message (DDLDCMSG) area
space-specification	Space specification for the punch file
unit	Symbolic device name

**IDMSCHEM — local mode:** To execute the schema compiler in local mode, remove the SYSCTL DD statement and replace it with:

```
//dictdb    DD DSN=idms.appldict.ddldml,DISP=SHR
//dloddb    DD DSN=idms.appldict.ddldclod,DISP=SHR
//sysjrn1   DD DSN=idms.tapejrn1,DISP=(NEW,KEEP),
//          UNIT=tape
Additional journal file assignments, as required
```

**Note:** Include the dloddb DD statement only if the DDL contains the REGENERATE statement.

dictdb	Ddname of the data dictionary DDLDDL area
dloddb	Ddname of the data dictionary load area
idms.appldict.ddldclod	Dataset name of the tape journal file
idms.appldict.ddldml	Dataset name of the data dictionary DDLDDL area
sysjrn1	Ddname of the tape journal file; must be appropriate for the DMCL module being used
tape	Symbolic device name for the tape journal file

## E.2.2 Subschema compiler

### IDMSUBSC — central version IDMSUBSC (OS/390)

```
//SUBSCHEM EXEC PGM=IDMSUBSC,REGION=1024K
//STEPLIB    DD DSN=idms.dba.loadlib,DISP=SHR
//          DD DSN=idms.loadlib,DISP=SHR
//sysctl     DD DSN=idms.sysctl,DISP=SHR
//dcmmsg     DD DSN=idms.sysmsg.ddldcmmsg,DISP=SHR
//SYSLST    DD SYSOUT=A
//SYSPCH    DD DSN=&&PCH, DISP=(NEW,KEEP,DELETE),
              DCB=(RECFM=FB,BLKSIZE=9040,LRECL=80),
              SPACE=space-specification,
              UNIT=unit
//SYSIDMS   DD *
Input SYSIDMS parameters, as required
/*
//SYSIPT    DD *
Subschema DDL source statements
```

**Note:**

The SYSPCH DD statement is required only if the DDL specifies PUNCH TO SYSPCH.

---

idms.dba.loadlib	Dataset name of the load library containing the DMCL and database name table load modules
idms.loadlib	Dataset name of the load library containing the CA-IDMS executable modules
sysctl	DDname of the SYSCTL file
idms.sysctl	Dataset name of the SYSCTL file
dcmsg	DDname of the message (DDLDCMSG) area
idms.sysmsg.ddldcmsg	Dataset name of the message (DDLDCMSG) area
space-specification unit	See IDMSCHEM job stream

---

**IDMSUBSC — local mode:** To execute the subschema compiler in local mode, remove the SYSCTL DD statement and replace it with:

```
//dictdb      DD DSN=idms.appldict.ddldml,DISP=SHR
//dloddb     DD DSN=idms.appldict.ddldclod,DISP=SHR
//sysjrn1    DD DSN=idms.tapejrn1,DISP=(NEW,KEEP),
//           UNIT=tape
Additional journal file assignments, as required
```

**Note:** Include the dloddb DD statement only if the DDL contains the REGENERATE statement.

---

dictdb	DDname of the dictionary DDLDDL area
idms.appldict.ddldml	Dataset name of the dictionary DDLDDL area
dloddb	DDname of the dictionary load area
idms.appldict.ddldclod	Dataset name of the dictionary load area
sysjrn1	DDname of the tape journal file; must be appropriate for the DMCL module being used
idms.tapejrn1	Dataset of the tape journal file
tape	Symbolic device name

---

## E.3 VSE/ESA JCL

The following VSE/ESA information is presented in this section:

- =COPY facility for input parameter statements
- VSE/ESA JCL to run the schema and subschema compilers (central version and local mode)

### E.3.1 =COPY facility

**Purpose:** Under VSE/ESA, some or all of the input parameter statement to be submitted to a DDL compiler can be stored as a member in a source statement library. To copy the library member into the job stream, you use the =COPY IDMS statement.

The =COPY IDMS statement identifies the library member and is coded in the JCL along with other input parameter statements (if any) to be submitted to the DDL compiler. Multiple =COPY statements can be submitted.

=COPY IDMS statements and input parameter statements can be intermixed in the JCL. The input parameters are submitted to the compiler in the order in which they occur, whether they are coded directly in the JCL or copied in through the =COPY facility.

#### Syntax

```
▶▶ — =COPY IDMS ————┐ member-name —————▶<
                        └─ A. ← ───────────┘
                        └─ sublibrary-id. ─┘
```

#### Parameters

##### *A/sublibrary-id*

Identifies the source statement sublibrary that includes the member identified by *member-name*. The default is A.

##### *member-name*

Identifies the source statement library member that contains the input parameter statements to be submitted to the compiler.

**Note:** If the input parameter statements are stored as a member in a private source statement library, the DLBL file type for the library must be specified as DA.

### E.3.2 Schema compiler

**IDMSCHEM — central version IDMSCHEM (VSE/ESA)**

```
// EXEC PROC=IDMSLBS
// UPSI      b   If specified in the IDMSOPTI module
// DLBL      idmspch,'temp.ddl',0
// EXTENT    sys020,nnnnn,,,ssss,llll
// ASSGN     sys020,DISK,VOL=nnnnn,SHR
// EXEC      IDMSCHEM
Optional SYSIDMS parameters
/*
Schema DDL source statements
/*
```

Include the DLBL, EXTENT, and ASSGN statements for IDMSPCH only if the DDL specifies PUNCH TO SYSPCH. Refer to *CA-IDMS System Operations* for details.

**Overriding IDMSOPTI:** At installation, you can define a SYSCTL procedure that overrides the IDMSOPTI specifications for central version operations.

►► For information on the SYSCTL procedure, refer to *CA-IDMS Installation and Maintenance — VSE/ESA*.

---

IDMSLBS	Name of the procedure provided at installation that contains the file definitions for CA-IDMS dictionaries and databases.  ►► For a complete listing of IDMSLBS, see E.3.4, “IDMSLBS procedure” on page E-14, later in this appendix.  IDMSLBS references SYSIDMS, the input file you can use to specify runtime parameters, such as DMCL or dictionary name.  ►► For information on SYSIDMS parameters, refer to <i>CA-IDMS Database Administration</i> or <i>CA-IDMS Navigational DML Programming</i> .
b	Appropriate UPSI switch, 1-8 characters, as specified in the IDMSOPTI module
idmspch	Filename of the punched output (from IDMSPCH)
temp.ddl	File ID of the punched output (from IDMSPCH)
nnnnn	Serial number of the disk volume
ssss	Starting track (CKD) or block (FBA) of disk extent
llll	Number of the tracks (CKD) or blocks (FBA) of disk extent
sys020	Logical unit assignment of the punched output

---

**IDMSCHEM — local mode:** To execute the schema compiler in local mode, remove the UPSI specification, and include the following statements before EXEC IDMSCHEM:

```
// TLBL      sysjrn1,'idms.tapejrn1',,nnnnnn,,f
// ASSGN     sys009,TAPE,VOL=nnnnnn
```

sysjrn1	File name of the tape journal file
idms.tapejrn1	File ID of the tape journal file
nnnnnn	Volume serial number
f	File number of the tape journal file
sys009	Logical unit assignment for the tape journal file

### E.3.3 Subschema compiler

**IDMSUBSC — central version IDMSUBSC (VSE/ESA)**

```
// EXEC PROC=IDMSLBLS
// UPSI      b          If specified in the IDMSOPTI module
// DLBL      idmspch,'temp.ddl',0
// EXTENT    sys020,nnnnnn,,ssss,1111
// ASSGN     sys020,DISK,VOL=nnnnnn,SHR
// EXEC      IDMSUBSC
Optional SYSIDMS parameters
/*
Subschema DDL source statements
/*
```

Include the DLBL, EXTENT, and ASSGN statements for IDMSPCCH only if the DDL specifies PUNCH TO SYSPCH. To route punched output to a sequential disk file or to a tape file, use SYSIDMS file parameters to override the default characteristics, if necessary. Refer to *CA-IDMS System Operations* for details.

**Overriding IDMSOPTI:** At installation, you can define a SYSCTL procedure that overrides the IDMSOPTI specifications for central version operations.

►► For information on the SYSCTL procedure, refer to *CA-IDMS Installation and Maintenance — VSE/ESA*.

---

IDMSLBLS	<p>Name of the procedure provided at installation that contains the file definitions for CA-IDMS dictionaries and databases.</p> <p>►► For a complete listing of IDMSLBLS, see E.3.4, “IDMSLBLS procedure” on page E-14, later in this appendix.</p> <p>IDMSLBLS references SYSIDMS, the input file you can use to specify runtime parameters, such as DMCL or dictionary name.</p> <p>►► For information on SYSIDMS parameters, refer to <i>CA-IDMS Database Administration</i> or <i>CA-IDMS Navigational DML Programming</i>.</p>
b	Appropriate UPSI switch, 1-8 characters, as specified in the IDMSOPTI module
idmspch	Filename of the punched output (from IDMSPCH)
temp.ddl	File ID of the punched output (from IDMSPCH)
sys020	Logical unit assignment of the punched output disk extent
nnnnnn	Volume serial number
ssss	Starting track (CKD) or block (FBA) of the disk extent
llll	Number of the tracks (CKD) or blocks (FBA) of the disk extent
sysctl	Filename of the SYSCTL file
idms.sysctl	File ID of the SYSCTL file
sys008	Logical unit assignment of the SYSCTL file

---

**IDMSUBSC — local mode:** To execute the subschema compiler in local mode, remove the UPSI specification, and include the following statements before EXEC IDMSUBSC:

```
// TLBL      sysjrn1,'idms.tapejrn1',,nnnnnn,,f
// ASSGN     sys009,TAPE,VOL=nnnnnn
```

**Note:** These variables are described under the local mode discussion for the IDMSCHEM job stream.

### E.3.4 IDMSLBLS procedure

IDMSLBLS is a procedure that contains file definitions for the dictionaries, sample databases, disk journal files, and SYSIDMS file provided during installation.

You can tailor the following IDMSLBLS procedure (provided at installation) to reflect the filenames and definitions in use at your site. Reference IDMSLBLS as shown in the previous VSE/ESA JCL job stream.

```
* ----- LIBDEFS -----
// LIBDEF *,SEARCH=idmslib.sublib
// LIBDEF *,CATALOG=user.sublib
/* ----- LABELS -----
// DLBL idmslib,'idms.library',2099/365
// EXTENT ,nnnnnn,,,ssss,1500
// DLBL dccat,'idms.system.dccat',2099/365,DA
// EXTENT SYSnnn,,nnnnnn,,,ssss,31
// ASSGN SYSnnn,DISK,VOL=,nnnnnn,SHR
// DLBL dccatl,'idms.system.dccatlod',2099/365,DA
// EXTENT SYSnnn,,nnnnnn,,,ssss,6
// ASSGN SYSnnn,DISK,VOL=,nnnnnn,SHR
// DLBL dccatx,'idms.system.dccatx',2099/365,DA
// EXTENT SYSnnn,,nnnnnn,,,ssss,11
// ASSGN SYSnnn,DISK,VOL=,nnnnnn,SHR
// DLBL dcdbl,'idms.system.dddbl',2099/365,DA
// EXTENT SYSnnn,,nnnnnn,,,ssss,101
// ASSGN SYSnnn,DISK,VOL=,nnnnnn,SHR
// DLBL dclob,'idms.system.ddlclod',2099/365,DA
// EXTENT SYSnnn,,nnnnnn,,,ssss,21
// ASSGN SYSnnn,DISK,VOL=,nnnnnn,SHR
// DLBL dclog,'idms.system.ddlclod',2099/365,DA
// EXTENT SYSnnn,,nnnnnn,,,ssss,401
// ASSGN SYSnnn,DISK,VOL=,nnnnnn,SHR
// DLBL dcrun,'idms.system.ddlcrun',2099/365,DA
// EXTENT SYSnnn,,nnnnnn,,,ssss,68
// ASSGN SYSnnn,DISK,VOL=,nnnnnn,SHR
// DLBL dcscr,'idms.system.ddlccscr',2099/365,DA
// EXTENT SYSnnn,,nnnnnn,,,ssss,135
// ASSGN SYSnnn,DISK,VOL=,nnnnnn,SHR
// DLBL dcmsg,'idms.sysmsg.ddlccmsg',2099/365,DA
// EXTENT SYSnnn,,nnnnnn,,,ssss,201
// ASSGN SYSnnn,DISK,VOL=,nnnnnn,SHR
// DLBL dclocscr,'idms.sysloc.ddlocscr',2099/365,DA
```

```
// EXTENT SYSnnn,nnnnnn,,ssss,6
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL dir1db,'idms.sysdir1.dd1dm1',2099/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,201
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL dir1lod,'idms.sysdir1.dd1dc1od',2099/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,2
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL empdemo,'idms.empdemo1',2099/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,11
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL insdemo,'idms.insdemo1',2099/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,6
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL orgdemo,'idms.orgdemo1',2099/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,6
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL empldem,'idms.sqldemo.empldemo',2099/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,11
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL infodem,'idms.sqldemo.infodemo',2099/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,6
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL projdem,'idms.projseq.projdemo',2099/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,6
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL indxdem,'idms.sqldemo.indxdemo',2099/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,6
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
```

```

// DLBL   sysctl,'idms.sysctl',2099/365,SD
// EXTENT SYSnnn,nnnnnn,,ssss,2
// ASSGN  SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL   secdd,'idms.sysuser.ddlsec',2099/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,26
// ASSGN  SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL   dictdb,'idms.appldict.ddldml',2099/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,51
// ASSGN  SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL   dloddb,'idms.appldict.ddldclod',2099/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,51
// ASSGN  SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL   sqldd,'idms.syssql.ddlcat',2099/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,101
// ASSGN  SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL   sqllod,'idms.syssql.ddlcatl',2099/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,51
// ASSGN  SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL   sqlxdd,'idms.syssql.ddlcatx',2099/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,26
// ASSGN  SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL   asfdml,'idms.asfdict.ddldml',2099/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,201
// ASSGN  SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL   asflod,'idms.asfdict.asflod',2099/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,401
// ASSGN  SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL   asfdata,'idms.asfdict.asfdata',2099/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,201
// ASSGN  SYSnnn,DISK,VOL=nnnnnn,SHR

// DLBL   ASFDEFN,'idms.asfdict.asfdefn',2099/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,101
// ASSGN  SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL   j1jrnl,'idms.j1jrnl',2099/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,54
// ASSGN  SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL   j2jrnl,'idms.j2jrnl',2099/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,54
// ASSGN  SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL   j3jrnl,'idms.j3jrnl',2099/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,54
// ASSGN  SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL   SYSIDMS,'#SYSIPT',0,SD
/+
/*

```

---

<i>idmslib.sublib</i>	Name of the sublibrary within the library containing CA-IDMS modules
<i>user.sublib</i>	Name of the sublibrary within the library containing user modules
<i>idmslib</i>	Name of the file containing CA-IDMS modules
<i>idms.library</i>	ID associated with the file containing CA-IDMS modules

---

---

<i>SYSnnn</i>	Logical unit of the volume for which the extent is effective
<i>nnnnnn</i>	Volume serial identifier of appropriate disk volume
<i>ssss</i>	Starting track (CKD) or block (FBA) of disk extent
<i>dccat</i>	Filename of the system dictionary catalog (DDLDCAT) area
<i>idms.system.dccat</i>	ID of the system dictionary catalog (DDLDCAT) area
<i>dccatl</i>	Filename of the system dictionary catalog load (DDLDCATLOD) area
<i>idms.system.dccatlod</i>	ID of the system dictionary catalog load (DDLDCATLOD) area
<i>dccatx</i>	Name of the system dictionary catalog index (DDLDCATX) area
<i>idms.system.dccatx</i>	ID of the system dictionary catalog index (DDLDCATX) area
<i>dcddl</i>	Name of the system dictionary definition (DDLDCDDL) area
<i>idms.system.ddldddl</i>	ID of the system dictionary definition (DDLDCDDL) area
<i>dcclod</i>	Name of the system dictionary definition load (DDLDCCLOD) area
<i>idms.system.ddldclod</i>	ID of the system dictionary definition load (DDLDCCLOD) area
<i>dclog</i>	Name of the system log area (DDLDCLOG) area
<i>idms.system.ddldclog</i>	ID of the system log (DDLDCLOG) area
<i>dcrun</i>	Name of the system queue (DDLDCRUN) area
<i>idms.system.ddldcrun</i>	ID of the system queue (DDLDCRUN) area
<i>dcscr</i>	Name of the system scratch (DDLDCSCR) area
<i>idms.system.ddldcscr</i>	ID of the system scratch (DDLDCSCR) area
<i>dcmsg</i>	Name of the system message (DDLDCMSG) area
<i>idms.sysmsg.ddldcmsg</i>	ID of the system message (DDLDCMSG) area
<i>dclscr</i>	Name of the local mode system scratch (DDLDCSCR) area
<i>idms.sysloc.ddlocscr</i>	ID of the local mode system scratch (DDLDCSCR) area
<i>dirldb</i>	Name of the IDMSDIRL definition (DDLDCDDL) area
<i>idms.sysdirl.ddldddl</i>	ID of the IDMSDIRL definition (DDLDCDDL) area

---

---

dirllod	Name of the IDMSDIRL definition load (DDLDCLOD) area
idms.sysdirl.dirllod	ID of the IDMSDIRL definition load (DDLDCLOD) area
empdemo	Name of the EMPDEMO area
idms.empdemo1	ID of the EMPDEMO area
insdemo	Name of the INSDEMO area
idms.insdemo1	ID of the INSDEMO area
orgdemo	Name of the ORGDEMO area
idms.orgdemo1	ID of the ORGDEMO area
empldem	Name of the EMPLDEMO area
idms.sqldemo.empldemo	ID of the EMPLDEMO area
infodem	Name of the INFODEMO area
idms.sqldemo.infodem	ID of the INFODEMO area
projdem	Name of the PROJDEMO area
idms.projseg.projdemo	ID of the PROJDEMO area
indxdem	Name of the INDXDEMO area
idms.sqldemo.indxdemo	ID of the INDXDEMO area
sysctl	Name of the SYSCTL file
idms.sysctl	ID of the SYSCTL file
secdd	Name of the system user catalog (DDLSEC) area
idms.sysuser.ddlsec	ID of the system user catalog (DDLSEC) area
dictdb	Name of the application dictionary definition area
idms.appldict.ddldml	ID of the application dictionary definition (DDLML) area
dloddb	Name of the application dictionary definition load area
idms.appldict.ddldclod	ID of the application dictionary definition load (DDLDCLOD) area
sqldd	Name of the SQL catalog (DDLCCAT) area
idms.syssql.ddlcat	ID of the SQL catalog (DDLCCAT) area
sqllod	Name of the SQL catalog load (DDLCCATL) area
idms.syssql.ddlcatl	ID of SQL catalog load (DDLCCATL) area
sqlxdd	Name of the SQL catalog index (DDLCCATX) area

---

---

idms.syssql.ddlcatx	ID of the SQL catalog index (DDL CATX) area
asfdml	Name of the asf dictionary definition (DDL DML) area
idms.asfdict.ddldml	ID of the asf dictionary definition (DDL DML) area
asflod	Name of the asf dictionary definition load (ASFLOD) area
idms.asfdict.asflod	ID of the asf dictionary definition load (ASFLOD) area
asfdata	Name of the asf data (ASF DATA) area
idms.asfdict.asfdata	ID of the asf data area (ASF DATA) area
ASFDEFN	Name of the asf data definition (ASFDEFN) area
idms.asfdict.asfdefn	ID of the asf data definition area (ASFDEFN) area
j1jrnl	Name of the first disk journal file
idms.j1jrnl	ID of the first disk journal file
j2jrnl	Name of the second disk journal file
idms.j2jrnl	ID of the second disk journal file
j3jrnl	Name of the third disk journal file
idms.j3jrnl	ID of the third disk journal file
SYSIDMS	Name of the SYSIDMS parameter file

---

## E.4 CMS commands

This section provides the CMS commands to run the schema and subschema compilers (under the central version and in local mode).

### E.4.1 Schema compiler

#### IDMSCHEM — central version IDMSCHEM (CMS)

```
FILEDEF SYSLST PRINTER
FILEDEF SYSPCH DISK sypch output a (RECFM F LRECL 80
FILEDEF SYSIPT schema ddl a (RECFM F LRECL ppp BLKSIZE nnn
FILEDEF SYSIDMS DISK syidms parms a (RECFM F LRECL ppp BLKSIZE nnn
FILEDEF sysctl DISK sysctl idms a
EXEC IDMSFD
OSRUN IDMSCHEM
```

**Note:** Include the SYSPCH statement only if the DDL specifies PUNCH TO SYSPCH.

sypch output a	File name, type, and mode of the output punch file
schema ddl a	File name, type, and mode of the file that contains the schema DDL statements
ppp	Record length of file
nnn	Block size of file
sysidms parms a	File name, type, and mode of the file that contains the SYSIDMS parameters
sysctl	File name of the SYSCTL file
sysctl idms a	File name, type, and mode of the SYSCTL file
IDMSFD	Exec which defines all FILEDEFs, TXTLIBs, and LOADLIBs required by the system

**IDMSCHEM — local mode:** To execute the schema compiler in local mode, specify local mode in one of the following ways:

- Link IDMSCHEM with an IDMSOPTI program that specifies local execution mode
- Specify \*LOCAL\* as the first input parameter in the SYSIPT file
- Modify the OSRUN command:  
OSRUN IDMSCHEM PARM='\*LOCAL\*'

**Note:** This option is valid only if the OSRUN command is issued from a System Product interpreter or an EXEC2 file.

**Creating the SYSIPT file:** To create the SYSIPT file, enter these CMS commands:

---

```
XEDIT sysipt data a (NOPROF
INPUT
.
.
.
Schema source statements
.
.
.
FILE
```

**Editing the SYSIPT file:** To edit the SYSIDMS parameter file, enter these CMS commands:

```
XEDIT sysidms parms a (NOPROF
INPUT
.
.
.
SYSIDMS parameters
.
.
.
FILE
```

## E.4.2 Subschema compiler

### IDMSUBSC — central version IDMSUBSC (CMS)

```
FILEDEF SYSLST PRINTER
FILEDEF SYSPCH DISK syspch output a (RECFM F LRECL 80
FILEDEF SYSIPT subscheddl a (RECFM F LRECL ppp BLKSIZE nnn
FILEDEF SYSIDMS DISK syidms parms a (RECFM F LRECL ppp BLKSIZE nnn
FILEDEF sysctl DISK sysctl idms a
EXEC IDMSFD
OSRUN IDMSUBSC
```

**Note:** Include the SYSPCH statement only if the DDL specifies PUNCH TO SYSPCH.

---

subsch ddl a	File name, type, and mode of the file that contains the subschema DDL statements
syspch output a	File name, type, and mode of the output punch file
ppp	Record length of file
nnn	Block size of file
sysidms parms a	File name, type, and mode of the file that contains the SYSIDMS parameters
sysctl	File name of the SYSCTL file
sysctl idms a	File name, type, and mode of the SYSCTL file
IDMSFD	Exec which defines all FILEDEFS, TXTLIBs, and LOADLIBs required by the system

---

**IDMSUBSC — local mode:** To execute the subschema compiler in local mode, specify local mode in one of the following ways:

- Link IDMSUBSC with an IDMSOPTI program that specifies local execution mode
- Specify \*LOCAL\* as the first input parameter in the SYSIPT file
- Modify the OSRUN command:  
OSRUN IDMSUBSC PARM='\*LOCAL\*'

**Note:** This option is valid only if the OSRUN command is issued from a System Product interpreter or an EXEC2 file.

►► For information on creating a SYSIPT file or on editing the SYSIDMS file, see the information provided with the IDMSCHEM jobstream.

## E.5 BS2000/OSD JCL

The following BS2000/OSD information is presented in this section:

- =COPY facility for copying source code from a source statement library
- BS2000/OSD JCL to run the schema and subschema compilers (under the central version and in local mode)

### E.5.1 =COPY facility

**Purpose:** BS2000/OSD users can copy source code into the DDL compilers from a source statement library by means of the =COPY facility.

#### Syntax

```
▶— =COPY IDMS library-linkname.member-name —————><
```

#### Parameters

*library-linkname*

Specifies the linkname of the source library.

*member-name*

Specifies the name of the library member containing the source to be copied.

### E.5.2 Schema compiler

#### IDMSCHEM — central version IDMSCHEM (BS2000/OSD)

```
/ADD-FILE-LINK L-NAME=CDMSLIB,F-NAME=idms.dba.loadlib
/ADD-FILE-LINK L-NAME=CDMSLIB1,F-NAME=idms.loadlib
/ADD-FILE-LINK L-NAME=CDMSLODR,F-NAME=idms.loadlib
/ADD-FILE-LINK L-NAME=sysctl,F-NAME=idms.sysctl,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=SYSIDMS,F-NAME=idms.sysidms
/ASSIGN-SYSOPT TO=temp.punch
/ASSIGN-SYSDTA TO=*SYSCMD
/START-PROG *MOD(ELEM=IDMSCHEM,LIB=idms.loadlib,RUN-MODE=*ADV)
Schema DDL source statements
```

**Note:** The ASSIGN-SYSOPT command is required only if the DDL specifies PUNCH TO SYSPCH.

---

idms.dba.loadlib	Filename of the load library containing the DMCL and database name table load modules
idms.loadlib	Filename of the load library containing CA-IDMS executable modules
sysctl	Linkname of the SYSCTL file
idms.sysctl	Filename of the SYSCTL file
idms.sysidms	Filename of the SYSIDMS parameters file
temp.punch	Filename of the temporary file containing results of the PUNCH statement

---

**IDMSCHEM — local mode:** To execute the schema compiler in local mode, remove the SYSCTL statement, and replace it with:

```
/ADD-FILE-LINK L-NAME=dictdb,F-NAME=idms.appldict.ddldml,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=dloddb,F-NAME=idms.appldict.ddldclod,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=sysjrn1,F-NAME=idms.sysjrn1
```

---

dictdb	Linkname of the application dictionary DDLDDL area
idms.appldict.ddldml	Filename of the application dictionary DDLDDL area
dloddb	Linkname of the application dictionary load area
idms.appldict.ddldclod	Filename of the application dictionary load area
sysjrn1	Linkname of the tape journal file; must be appropriate for the DMCL being used
idms.sysjrn1	Filename of the journal file

---

### E.5.3 Subschema compiler

#### IDSSUBSC — central version IDMSUBSC (BS2000/OSD)

```

/ADD-FILE-LINK L-NAME=CDMSLIB,F-NAME=idms.dba.loadlib
/ADD-FILE-LINK L-NAME=CDMSLIB1,F-NAME=idms.loadlib
/ADD-FILE-LINK L-NAME=CDMSLODR,F-NAME=idms.loadlib
/ADD-FILE-LINK L-NAME=sysctl,F-NAME=idms.sysctl,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=SYSIDMS,F-NAME=idms.sysidms
/ASSIGN-SYSOPT TO=temp.punch
/ASSIGN-SYSDTA TO=*SYSCMD
/START-PROG *MOD(ELEM=IDMSUBSC,LIB=idms.loadlib,RUN-MODE=*ADV)
Subschema DDL source statements

```

**Note:** The ASSIGN-SYSOPT command is required only if the DDL specifies PUNCH TO SYSPCH.

idms.dba.loadlib	Filename of the load library containing the DMCL and database name table load modules
idms.loadlib	Filename of the load library containing CA-IDMS executable modules
sysctl	Linkname of the SYSCTL file
idms.sysctl	Filename of the SYSCTL file
idms.sysidms	Filename of the SYSIDMS parameters file
temp.punch	Filename of the temporary file containing results of the PUNCH statement

**IDSSUBSC — local mode:** To execute the subschema compiler in local mode, remove the SYSCTL FILE command and replace it with:

```

/ADD-FILE-LINK L-NAME=dictdb,F-NAME=idms.appldict.ddldml,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=dloddb,F-NAME=idms.appldict.ddldclod,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=sysjrn1,F-NAME=idms.sysjrn1

```

dictdb	Linkname of the application dictionary DDL DML area
idms.appldict.ddldml	Filename of the application dictionary DDL DML area
dloddb	Linkname of the application dictionary load area
idms.appldict.ddldclod	Filename of the application dictionary load area
sysjrn1	Linkname of the tape journal file; must be appropriate for the DMCL being used
idms.sysjrn1	Filename of the journal file



## **Appendix F. System Record Types**

---



---

**System record types for space management:** CA-IDMS/DB maintains the following nine system record types for space management:

---

<b>Type</b>	<b>Record ID</b>	<b>Description</b>
SR1	1	Participates as owner in the system-owned CALC set; members are all user record types with a storage mode of CALC; occurs once for each page in a standard database area as bytes 5 through 16 in the header
SR2	2	Replaces records relocated by the RESTRUCTURE SEGMENT, and the migration utility (RHDCMIG1 and RHDCMIG2), and SQL processing following the addition of a column to a table; eight bytes in length
SR3	3	Identifies a user record as having been relocated; the actual user-designated record identification can be found in the relocated record's corresponding SR2 record
SR4	4	Identifies fragments of variable-length records; the actual user-designated record identification can be found in the line index of the root portion of the record
SR5	5	Holds the area-level synchronization stamp for SQL-defined segments and acts as an owner for the table-level synchronization stamp records (SR9s)
SR6	6	Appears in the subschema tables for excluded owner or member record definitions in set relationships; never occurs in the database
SR7	7	Participates as owner in an index; stores CALC under the indexed set's name; occurs once for each indexed set in the database that does not have a user-defined owner record (for details, see Chapter 36, "Index Management" on page 36-1)
SR8	8	Contains index entries that point to lower level SR8 records or to an indexed set's member database record occurrences; chained by next, prior, and owner pointers to the owner record occurrence of an indexed set (for details, see Chapter 36, "Index Management" on page 36-1)
SR9	9	Holds the table identifier and synchronization stamp for each table in the area

---

# Appendix G. User-Exit Program for Schema and/or Subschema Compiler

---

G.1	When a user exit is called . . . . .	G-4
G.2	Rules for writing the user-exit program . . . . .	G-5
G.3	Control blocks and sample user-exit programs . . . . .	G-7
G.3.1	User-exit control block . . . . .	G-7
G.3.2	SIGNON Element Block . . . . .	G-7
G.3.3	SIGNON Block . . . . .	G-8
G.3.4	Entity control block . . . . .	G-8
G.3.5	Card-image control block . . . . .	G-9
G.4	Sample user-exit program for Schema and/or Subschema Compilers . . . .	G-10



---

This appendix presents the procedures for coding a user-exit program, which is called by the schema compiler and/or subschema compiler to:

- Perform security checks
- Enforce entity-occurrence naming conventions
- Maintain an audit trail of dictionary activity

A common user-exit program can be coded to be shared by the schema compiler and subschema compiler, or a specialized user-exit program can be coded for each or for only one of the compilers. The rules and procedures governing the user-exit program are the same for all compilers that use it.

## G.1 When a user exit is called

The user-exit module is called by the applicable compiler when it encounters any of these four points:

- **SIGNON/SIGNOFF/COMMIT**

After the signon procedure is complete and the compiler's security checks have been passed, or immediately after signoff or COMMIT processing.

- **Major command**

After an ADD, MODIFY, DELETE, DISPLAY or PUNCH request has been issued. The program is invoked just after the applicable compiler has identified the entity that is the object of the request and has successfully checked authorization requirements. Object entities can be any standard schema or subschema entity type.

- **Card image**

After each input statement (card image) is passed to the user-exit control block after the statement has been:

- Scanned and printed on the applicable Compiler Activity List
- Displayed on the terminal
- Written to the print file (online compiler interface only)

The administrator can build an audit trail of accesses and updates to the dictionary.

- **End of converse**

When one of the following occurs, you can perform a termination activity, such as a write-to-log:

- Pressing [Enter] during an online compiler session
- A batch run of the compiler processes a SIGNOFF statement
- A batch run of the compiler detects an end-of-file condition

---

## G.2 Rules for writing the user-exit program

This section describes the rules that apply to writing the user-exit program.

### ■ Language

You can write the user-exit module in any language that supports OS calling conventions. However, it is recommended that you write user-exit modules in Assembler to allow the online compiler to remain reentrant.

**Note:** User-exit modules cannot be CA-ADS dialogs.

### ■ Versions

You can code and maintain separate versions of user-exit modules for the batch and online compilers, or you can code modules that can be executed both in batch mode and online.

### ■ Macros

The user-exit facility supports all CA-IDMS/DC macros for exits to be used with the online compilers. For exits to be used with the batch compilers, the only CA-IDMS/DC macros supported are: #WTL, #ABEND, #GETSTG, #FREESTG, #LOAD, and #DELETE; under VSE/ESA, the only valid form of #DELETE is EPADDR=.

### ■ Run units

You can start a run unit within an exit; however, you should ensure that the run unit does not deadlock with the applicable compiler run unit. If a user-exit run unit will access a dictionary area, the run unit should ready the object area in a retrieval usage mode.

### ■ Entry point

User-exit modules must have an entry point of IDDUXIT and must be linked with:

- IDMSCHEM (the batch schema compiler) and IDMSCHDC (the CA-IDMS/DC version of the online schema compiler)
- IDMSUBSC (the batch subschema compiler) and IDMSUBDC (the CA-IDMS/DC version of the online subschema compiler)

**Note:** These compilers will not be dynamically loaded.

### ■ Interface

User exits written in COBOL to run under the applicable online compiler require a user-exit interface, written in Assembler with an entry point of IDDUXIT, to be link-edited with IDMSCHDC or IDMSUBDC. This interface should issue a #LINK to the COBOL program (with an entry point other than IDMSCHDC or IDMSUBDC) to isolate it from IDMSCHDC or IDMSUBDC, which is storage-protected.

### ■ Register conventions

User-exit modules are called using the following OS register conventions:

R15	Entry point of module IDDUXIT
R14	Return address
R13	18 fullword SAVEAREA
R1	Fullword parameter list

#### ■ Parameters 3 and 4

For all four types of user exits, parameter 1 points to a user-exit control block and parameter 2 points to a SIGNON element block. The information addressed in parameters 3 and 4 varies based on the type of user exit, as follows:

- For the **SIGNON/SIGNOFF/COMMIT** and **end-of-conversation** exits, parameter 3 points to a SIGNON block.
- For the **major command user exit**, parameter 3 points to an entity control block.
- For the **card-image** user exit, parameter 3 points to a card-image control block.
- For **all user exits except the card-image user exit**, parameter 4 is reserved for use by the applicable compiler and should be defined as a PIC X(80) field in the user-exit module.
- For the **card-image user exit**, parameter 4 points to the input card image, which is defined as a PIC X(80) field.

The user-exit control blocks are described separately later in this appendix.

#### ■ Information modification

With the exception of the fields identified within the user-exit control block described below, a user-exit module should not modify any of the information passed.

#### ■ Return codes

On return from a user-exit module, you must set a return code and, optionally, specify a message ID and message text to be issued by the applicable compiler, as follows:

Code	Compiler action
0	No message is issued; compiler continues with normal processing.
1	An informational message is issued; compiler continues with normal processing.
4	A warning message is issued; compiler continues with normal processing.
8	An error message is issued; compiler initiates error processing.

## G.3 Control blocks and sample user-exit programs

This section presents the formats of these five control blocks:

- User-exit control block
- SIGNON element block
- SIGNON block
- Entity control block
- Card-image control block

### G.3.1 User-exit control block

The following table shows how to define the user-exit control block:

Field	Usage	Size	Picture	Description
1	Char	8	X(8)	Compiler name: IDMSCHEM or IDMSUBSC
2	Char	8	X(8)	Compiler start date: mm/dd/yy
3	Char	8	X(8)	Compiler start time: hhmmssmm
4	Binary	4	S9(8) COMP	User field initialized to 0 (for use by reentrant modules as a pointer to their work area)
5	Binary	4	S9(8) COMP	User return code (described below)
6	Char	8	X(8)	Message ID returned by user, in the range DC900000 through DC999999, or any 6-digit number; blank if no message is returned
7	Char	80	X(80)	Message text returned by user

### G.3.2 SIGNON Element Block

The following table shows how to define the SIGNON element block:

Field	Usage	Size	Picture	Description
1	Binary	1	X	Length of user ID for #WTLs (not addressable by COBOL)
2	Char	32	X(32)	SIGNON user ID

### G.3.3 SIGNON Block

The following table shows how to define the SIGNON block.

Field	Usage	Size	Picture	Description
1	Char	16	X(16)	SIGNON, SIGNOFF, COMMIT or END-OF-CONVERSE statement
2	Char	8	X(8)	SIGNON dictionary name
3	Char	8	X(8)	SIGNON node name
4A	CHAR	32	X(32)	User ID
5	Binary	2	S9(4)	DDL DML area usage mode: 36=UPDATE; 38=PROTECTED UPDATE; 37=RETRIEVAL
6	Binary	2	S9(4)	DDL D CLOD area usage mode
7	Binary	2	S9(4)	DDL D CMSG area usage mode
8	Binary	10	X(10)	Reserved

**Note:** Each bit in flag 0 and flag 1 must be tested separately. More than one bit may be on at any one time.

### G.3.4 Entity control block

The following table shows how to define the entity control block.

Field	Usage	Size	Picture	Description
1	Char	16	X(16)	Major command (ADD, MODIFY, DELETE, DISPLAY, or PUNCH)
2	Char	32	X(32)	Entity type
3	Char	40	X(40)	Entity occurrence
4	Binary	2	S9(4)	Entity version number or number of records requested
5	Char	64	X(64)	Additional Qualifier
6	Char	32	X(32)	PREPARED BY user ID
7	Char	32	X(32)	REVISED BY user ID

### G.3.5 Card-image control block

The following table shows how to define the card-image control block:

<b>Field</b>	<b>Usage</b>	<b>Size</b>	<b>Picture</b>	<b>Description</b>
1	Char	16	X(16)	Compiler 'CARD IMAGE' command
2	Binary	2	S9(8)	Input low-card column
3	Binary	2	S9(8)	Input high-card column

## G.4 Sample user-exit program for Schema and/or Subschema Compilers

The following sample user-exit program can be used to enforce naming conventions for elements. The source code for this program can be found in the installation source library under member name IDDSUXIT.

```

*****
IDDUXIT  TITLE 'NAMING CONVENTION CHECKER'
*****
*
*
* PROGRAM NAME : IDDUXIT
*
* DATE          : 03/01/96
*
*
* DESCRIPTION   : THIS IS AN EXAMPLE OF A USER EXIT.  THIS PROGRAM
*                 SHOWS HOW A SHOP COULD CHECK THE ENTITY NAMES FOR
*                 A SHOP STANDARD.  ANY VIOLATIONS OF THE NAMING
*                 CONVENTION ARE TREATED AS AN ERROR AND THE ACTION
*                 (ADD, MOD, DEL) IS NOT ALLOWED.
*****
IDDUXIT  CSECT      #REGEQU
*****
*          SET UP ADDRESSABILITY
*****
          STM      R14,R12,12(R13)      SAVE CALLERS REGISTERS
          LR       R12,R15
          USING    IDDUXIT,R12
          L        R4,12(R1)            GET THE
          L        R3,8(R1)             CORRECT
          L        R2,4(R1)            PARAMETER
          L        R1,0(R1)            ADDRESSES
*
IDDUXITR DS      0H                    BASE THE CONTROL BLOCKS
*
          USING    UXITCB,R1           USER EXIT CONTROL BLOCK
          MVC      UXITRCDE,F0         ZERO OUT THE RETURN CODE
          MVC      UXITMID(8),BLANKS   BLANK OUT THE MESSAGE ID
          MVC      UXITMTXT(80),BLANKS BLANK OUT THE MESSAGE
*

```

```

*****
* INTERROGATE THE MAJOR COMMAND *
*****
*
* SPACE
UXIENTY EQU *
        USING UXITECB,R3          ENTITY CONTROL BLOCK
*
        CLC UXITEVRB,UXICSON      IS IT AN SIGNON?
        BE  USIGNON              YES, CHECK THE USER NAME
*
        CLC UXITEVRB,UXICARD      IS IT AN CARD IMAGE EXIT?
        BE  UCARD                YES, CHECK THE CARD
*
        CLC UXITEVRB,UXICADD      IS IT AN ADD?
        BE  UXIECHK              YES, CHECK THE ENTITY-NAME
*
        CLC UXITEVRB,UXICMOD      IS IT A MODIFY?
        BE  UXIECHK              YES, CHECK THE ENTITY-NAME
*
        CLC UXITEVRB,UXICDEL      IS IT A DELETE?
        BE  UXIECHK              YES, CHECK THE ENTITY-NAME
*
        MVC UXITMID(8),ELSEID     MOVE IN 'ELSE' MESSAGE ID
        MVC UXITMTXT(80),ELSEMSG  MOVE IN 'ELSE' MESSAGE
        B   UXIEBYE
*
*****
* CHECK THE CARD IMAGE *
*****
*
* SPACE
UCARD EQU *
*
        MVC UXITMID(8),CARDID     FILL IN THE MESSAGE ID
        MVC UXITMTXT(80),CARDMSG  FILL IN THE MESSAGE TEXT
        B   UXIEBYE              BACK TO THE COMPILER
*

```

## G.4 Sample user-exit program for Schema and/or Subschema Compilers

```

*****
*      CHECK THE USER NAME FOR ME      *
*****
*
      SPACE
USIGNON EQU  *
*
      USING UXITSEB,R2      SIGNON ELEMENT BLOCK
      USING UXITSB,R3      SIGNON BLOCK
*
      CLC UXITUSER(3),WHOME  IS IT ME
      BE  UXIEDC            YES GO CHECK FOR DC NAME
*                               NO, GO TO JAIL, GO DIRECTLY TO
*                               JAIL, DO NOT PASS GO DO NOT
USNAME EQU  *                               COLLECT $200.
      MVC UXITRCDE,F8      FILL IN THE RETURN CODE
      MVC UXITMID(8),NOSNID FILL IN THE MESSAGE ID
      MVC UXITMTXT(80),NOSNMSG FILL IN THE MESSAGE TEXT
      B   UXIEBYE          BACK TO THE COMPILER
*
UXIEDC EQU  *
      TM  UXITFLG1,UXIT1DC  ARE WE RUNNING DC
      BZ  UXIEBYE          NO, SKIP DC ID CHECK
*
      CLC UXITUSER,UXITIUSR  IS THE USER THE SAME AS DC
      BE  UXIEBYE          YES, OK LET IT PASS
*                               NO, DON'T LET THEM SIGNON
      MVC UXITRCDE,F8      FILL IN THE RETURN CODE
      MVC UXITMID(8),NODCID FILL IN THE MESSAGE ID
      MVC UXITMTXT(80),NODCMSG FILL IN THE MESSAGE TEXT
      B   UXIEBYE          BACK TO THE COMPILER
*
*****
*      CHECK THE ENTITY-NAME FOR VALID NAMING CONVENTION      *
*****
*
      SPACE
UXIECHK EQU  *
      USING UXITECB,R3      ENTITY CONTROL BLOCK
*
      CLC UXITENME(3),NAMECHK DOES THE NAME FOLLOW THE RULES?
      BE  UXIEBYE          YES, LET THIS ONE PASS.
*                               NO, RETURN AN ERROR
*
      MVC UXITRCDE,F8      FILL IN THE RETURN CODE
      MVC UXITMID(8),NONOID FILL IN THE MESSAGE ID
      MVC UXITMTXT(80),NONOMSG FILL IN THE MESSAGE TEXT
*

```

```

*****
*           RETURN BACK TO THE COMPILER                               *
*****
*
      SPACE
UXIEBYE EQU *
          LM   R14,R12,12(R13)      RELOAD CALLER'S REGISTERS
          BR   R14                  RETURN TO CALLER
          EJECT
*****
*           CONSTANTS AND LITERALS                                     *
*****
UXICADD DC   CL16'ADD              '
UXICMOD DC   CL16'MODIFY           '
UXICDEL DC   CL16'DELETE          '
UXICSON DC   CL16'SIGNON          '
UXICARD DC   CL16'CARD IMAGE      '
NAMECHK DC   CL3'XYZ'
WHOME   DC   CL3'XYZ'
WKLEN   DC   F'100'
NONOID  DC   CL8'DC999001'
NONOMSG DC   CL80'NAMING CONVENTION VIOLATED - ACTION NOT ALLOWED'
NOSNID  DC   CL8'DC999002'
NOSNMSG DC   CL80'SIGNON ERROR - USER NOT ALLOWED ACCESS'
NODCID  DC   CL8'DC999003'
NODCMMSG DC  CL80'SIGNON ERROR - USER NAME NOT DC USER NAME'
CARDID  DC   CL8'DC999004'
CARDMSG DC   CL80'MESSAGE PRODUCED BY CARD IMAGE EXIT      '
ELSEID  DC   CL8'DC999005'
ELSEMSG DC   CL80'MESSAGE PRODUCED BY CARD IMAGE EXIT      '
BLANKS  DC   CL80' '
F0      DC   F'0'          NORMAL RETURN CODE - NO ERRORS
F2      DC   F'1'          INFORMATION MESSAGE
F4      DC   F'4'          WARNING MESSAGE
F8      DC   F'8'          ERROR MESSAGE
*

```

## G.4 Sample user-exit program for Schema and/or Subschema Compilers

---

```
*****
*          USER EXIT CONTROL BLOCK          *
*****
UXITCB   DSECT
UXITCPLR DS   CL8           COMPILER NAME 'IDMSCHEM' OR 'IDMSUBSC'
UXITDATE DS   CL8           COMPILER START DATE MM/DD/YY
UXITTIME DS   CL8           COMPILER START TIME HHMMSSMM
UXITWORK DS   F             USER FULLWORD INITIALIZED TO 0
UXITRCDE DS   0F           RETURN CODE RETURNED BY USER
          DS   XL3           UNUSED
UXITRC   DS   X
UXITRC00 EQU X'00'         NORMAL RETURN CODE - NO ERRORS
UXITRC01 EQU X'01'         INFORMATION MESSAGE
UXITRC04 EQU X'04'         WARNING MESSAGE
UXITRC08 EQU X'08'         ERROR MESSAGE
UXITMID  DS   CL8           USER MESSAGE ID RETURNED BY USER
UXITMTXT DS   CL80          USER MESSAGE TEXT RETURNED BY USER
UXITCBLN EQU *-UXITCB     USER EXIT CONTROL BLOCK LENGTH
*
*****
*          USER EXIT SIGNON ELEMENT BLOCK   *
*****
UXITSEB  DSECT
UXITIDLN DS   X             LENGTH OF USERID FOR #WTL'S
UXITUSER DS   CL32          USER ID
          DS   0A           ROUND UP TO FULLWORD
UXITSNLN EQU *-UXITSEB     LENGTH OF SIGNON ELEMENT
*
```

G.4 Sample user-exit program for Schema and/or Subschema Compilers

```

*****
*          USER EXIT SIGNON BLOCK          *
*****
UXITSB   DSECT
UXITTYPE DS   CL16          VERB
UXITDICT DS   CL8           DICTIONARY NAME
UXITNODE DS   CL8           NODE NAME
UXITIUSR DS   CL32          USER ID

UXITIPSW DS   CL8           USER'S PASSWORD
UXITFLG0 DS   CL1           ENVIRONMENT FLAG
UXIT0DOS EQU  X'80'         COMPILER RUNNING UNDER VSE/ESA
UXIT0MEN EQU  X'40'         RUNNING UNDER 'MENU' MODE
UXITFLG1 DS   CL1           ENVIRONMENT FLAG
UXIT1LCL EQU  X'80'         RUNNING IN INTERNAL SUBROUTINE MODE
UXIT1DC EQU   X'40'         COMPILER RUNNING UNDER DC
                               RESERVED FOR FUTURE FLAGS
                               RESERVED
UXITDMLM DS   H             DDLML USAGE MODE
*                               36=UPDATE
*                               37=PROTECTED UPDATE
*                               38=RETRIEVAL
UXITLODM DS   H             DDLDCLOD USAGE MODE
UXITMSGM DS   H             DDLDCMSG USAGE MODE
                               DS   CL10          RESERVED
UXITSLEN EQU  *-UXITSB     LENGTH OF USER EXIT SIGNON BLOCK
*
*****
*          USER EXIT ENTITY CONTROL BLOCK  *
*****
UXITECB  DSECT
UXITEVRB DS   CL16          VERB
UXITENTY DS   CL32          ENTITY-TYPE
UXITENME DS   CL40          ENTITY NAME
UXITEVER DS   H             VERSION
UXITEADQ DS   CL64          ADDITIONAL QUALIFIER
UXITPREP DS   CL32          PREPARED BY USER NAME
UXITREV DS   CL32          REVISED BY USER NAME
UXITELEN EQU  *-UXITECB     LENGTH OF USER EXIT ENTITY CONTROL BLK
*
*****
*          END OF EXIT                    *
*****
END

```



# Appendix H. SYSIDMS Parameter File

---

H.1 Parameter Summary . . . . . H-3  
H.2 Parameter Descriptions . . . . . H-6



## H.1 Parameter Summary

### Debugging and abend control parameters

Parameter	CV	Batch	OS/ 390	VSE/ ESA	VM/ ESA	BS2000 /OSD	MSP/ EX
ABEND_ON_ DEADLOCK	X		X	X	X	X	X
ABENDTRACE		(1)					
ABENDTRACE_ ENTRIES		(1)					
ABENDTRACE_ _SUBSCHEMA _DISPLAY		(1)					
ABENDTRACE_ _VIBSNAP		(1)					
AREA_VALIDATION_ _MSGS	X		X	X	X	X	X
DB_DEADLOCK_ _DUMP	X		X	X	X	X	X
DC_DEADLOCK_ _DUMP	X		X	X	X	X	X
DC_DEADLOCK_0029	X		X	X	X	X	X
DEADLOCK_ABEND_ _ERUS	X		X	X	X	X	X
DEADLOCK_ABEND_ _0029	X		X	X	X	X	X
DEADLOCK_DETAILS	X		X	X	X	X	X
DMLTRACE		X	X	X	X	X	X
ECHO	X	X	X	X	X	X	X
PROCTRACE		X	X	X	X	X	X
QSAMTRACE		X	X	X	X	X	X
SQLTRACE		X	X	X	X	X	X

(1) The CA-OPTIMIZER/II product must be installed to use this parameter.

### Performance-related parameters

Parameter	CV	Batch	OS/ 390	VSE/ ESA	VM/ ESA	BS2000 /OSD	MSP/ EX
BUFFER_PURGE		X	X	X	X	X	X
BUFFERSTAT		X	X	X	X	X	X
DLBLMOD		X		X			
FILE_BUF		X	X	X	X	X	X
IDMSQSAM		X	X	X	X	X	X
PREFETCH	X	X	X	X	X	X	X
PREFETCH_BUF		X	X	X	X	X	X
QSAMAREA		X	X	X	X	X	X
QSAMBUF#		X	X		X		X
QSAM#BUF		X				X	
QSAMTRACE		X	X	X	X	X	X
SQL_INTLSORT		X	X	X	X	X	X

#### File-related parameters

Parameter	CV	Batch	OS/ 390	VSE/ ESA	VM/ ESA	BS2000 /OSD	MSP/ EX
DLBLMOD		X		X			
FILE_BUF		X	X	X	X	X	X
LENGTH_PAGE		X	X	X	X	X	X
LIST		X				X	
LOCAL_DYNAMIC _ALLOCATION		X	X	X	X	X	X
MULTIDSN		X		X			
OVERPRINT		X				X	
ROLLBACK3490		X	X	X	X		X
SYS_MSG		X	X	X	X	X	X
UPPER		X	X	X	X	X	X
<i>VSE/ESA file parameters</i>		X		X			
WIDTH_PAGE		X	X	X	X	X	X

#### Connection and environment parameters

Parameter	CV	Batch	OS/ 390	VSE/ ESA	VM/ ESA	BS2000 /OSD	MSP/ EX
CVMACH		X			X		
CVNUM		X			X		
CVRETRY		X	X	X	X		X
DBNAME		X	X	X	X	X	X
DICTIONAME		X	X	X	X	X	X
DICTIONODE		X	X	X	X	X	X
DMCL	X	X	X	X	X	X	X
LANG		X	X	X	X	X	X
LOCAL		X	X	X	X	X	X
NODENAME		X	X	X	X	X	X
REREAD_SYSCTL		X	X	X	X	X	X
SYSCTL		X	X	X	X	X	X

#### Miscellaneous runtime directives

Parameter	CV	Batch	OS/ 390	VSE/ ESA	VM/ ESA	BS2000 /OSD	MSP/ EX
DC_SCRATCH		X	X	X	X	X	X
DCNAME	X		X				
DMCL	X	X	X	X	X	X	X
DSGROUP	X		X				
JOURNAL		X	X	X	X	X	X
JRNLDTS	X		X	X	X	X	X
LOADAREA		X	X	X	X	X	X
LOCAL_NOJOURNAL _RETRIEVAL		X	X	X	X	X	X
LOCALPUR		X	X	X	X	X	X
MSGDICT		X	X	X	X	X	X
PARM		X	X	X	X	X	X
SORTSIZE		X	X	X	X	X	X

---

## H.2 Parameter Descriptions

---

Parameter	Description
ABEND_ON_DEADLOCK	<p>Forces the abnormal termination of a task that encounters a database resource deadlock. In normal CV operations, a database resource deadlock results in control being returned to the application program with an indication that a deadlock occurred. This parameter causes the task to be abended instead.</p> <p><b>Note:</b> It is meaningful only in the SYSIDMS file associated with a central version.</p>
ABENDTRACE=ON/OFF	<p>Activates the tracing of various pieces of IDMS data when using CA-OPTIMIZER/II. This parameter is meaningful only in the SYSIDMS file associated with a batch job.</p> <p><b>Note:</b> The CA-OPTIMIZER/II product must be installed to use this parameter.</p>
ABENDTRACE_ENTRIES= <i>nnn</i>	<p>Overrides the default number of entries being traced by ABENDTRACE. This parameter is meaningful only in the SYSIDMS file associated with a batch job.</p> <p><b>Note:</b> The CA-OPTIMIZER/II product must be installed to use this parameter.</p>
ABENDTRACE_SUBSCHEMA_DISPLAY=ON	<p>Activates the display of information from the subschema in use at the time of abend when using ABENDTRACE. This parameter is meaningful only in the SYSIDMS file associated with a batch job.</p> <p><b>Note:</b> The CA-OPTIMIZER/II product must be installed to use this parameter.</p>

---

Parameter	Description
ABENDTRACE_VIBSNAP=ON	<p>Causes the dump of the VIB at the time of abend when using ABENDTRACE. This parameter is meaningful only in the SYSIDMS file associated with a batch job.</p> <p><b>Note:</b> The CA-OPTIMIZER/II product must be installed to use this parameter.</p>
AREA_VALIDATION_MSGS=ON/OFF	<p>ON causes the informational messages DB347042 and DB347043 to be displayed on the JES log during startup and shutdown for each area being shared in a SYSPLEX data sharing environment. If you are sharing many areas this can cause the JES log to be congested.</p> <p>OFF is the default.</p> <p><b>Note:</b> This parameter is only applicable in a SYSPLEX data sharing environment.</p>
BUFFER_PURGE	<p>Causes updated pages to be written to the database whenever the number of buffers containing such pages exceeds 1/4 of the number of pages in the buffer pool. This parameter may improve the performance of local mode update jobs that do not issue frequent COMMITs, since it will make buffers available for the use of prefetch. It has meaning only for local mode batch jobs.</p>
BUFFERSTAT	<p>Produces a report containing buffer pool I/O statistics that can be used for tuning purposes. The report will be written to SYSLST at the end of the job. This parameter has meaning only for local mode batch jobs.</p> <p>See the table at the end of this table for a description of the fields in the report.</p>
CVMACH= <i>cms-machine-name</i> (VM/ESA only)	<p>Specifies the virtual machine in which the DC/UCF system is executing.</p>

---

<b>Parameter</b>	<b>Description</b>
CVNUM= <i>nnn</i> (VM/ESA only)	Specifies the number of the central version that is accessible by CMS and is used to route database requests through the IDMSVMCF facility; <i>nnn</i> must be an integer in the range from 0 through 255.
CVRETRY=ON/OFF	ON indicates that the following message is displayed on the operator console when the CA-IDMS central version is not active:  CV <i>nnn</i> NOT ACTIVE. REPLY RETRY OR CANCEL.  ON is the default.
DB_DEADLOCK_DUMP	Specifies that a dump will be produced for a task that is abended due to a database resource deadlock. This parameter is used in conjunction with the ABEND_ON_DEADLOCK parameter. If not specified, no dump will be produced when a task is abended due to a database deadlock.
DBNAME= <i>database-name</i>	For non-SQL applications, specifies the name of the database to access at runtime. <i>database-name</i> is either a segment name or a DBNAME defined in a database name table. For SQL applications, it has no impact.
DC_DEADLOCK_NODUMP	Specifies that a dump not be produced for a task that is abended due to a DC resource deadlock. This parameter overrides the DUMP/NODUMP sysgen parameter.
DC_DEADLOCK_0029	Specifies that tasks that encounter a DC resource deadlock be abended with a code of 0029 rather than a code of DEAD.
DC_SCRATCH=ON/OFF	ON allows local jobs to use the Central Version's scratch area (DDLDCSCR) when a local scratch area (DDLOCSCR) is not defined in the DMCL. OFF is the default.

---

---

<b>Parameter</b>	<b>Description</b>
DCNAME= <i>member-name</i> (OS/390 only)	<p>Specifies the member name of the system within a data sharing group. This name also becomes the system (node) name, overriding the value specified in the system definition. member-name must be a 1-8 character name consisting of characters A-Z, 0-9, \$, #, or @.</p> <p><b>Note:</b> This parameter is only applicable in a data sharing environment.</p>
DEADLOCK_ABEND_ERUS	<p>Specifies that ERUS tasks that encounter a database resource deadlock be abnormally terminated. This parameter is meaningful only if the ABEND_ON_DEADLOCK parameter is also specified.</p>
DEADLOCK_ABEND_0029	<p>Specifies that tasks that are abended due to database resource deadlocks use a code of 0029 rather xx29, where "xx" represents the major code of the DML request that was being issued at the time of the abend. This parameter has meaning only if the ABEND_ON_DEADLOCK parameter is also specified.</p>
DEADLOCK_DETAILS=ON/OFF	<p>ON specifies that more detail be provided in a deadlock situation. The default is OFF.</p>
DICTNAME= <i>dictionary-name</i>	<p>Specifies a dictionary to use when loading a subschema from a load area. For dictionary-related tools like CA-IDMS compilers and precompilers, IDMSBCF, etc., dictionary-name specifies the dictionary to access at run time. For SQL applications, dictionary-name specifies the name of the dictionary to connect to at run time.</p>

---

---

<b>Parameter</b>	<b>Description</b>
DICTNODE= <i>dictionary-node-name</i>	For SQL applications and dictionary-related tools under the central version, specifies the name of the DC/UCF system that controls the dictionary to access at run time. For applications running in local mode, this parameter is not applicable.
DLBLMOD=ON/OFF (VSE/ESA only)	ON specifies that the DLBL type in the disk label will be changed from 'DA' to 'SD' when sequential processing (IDMSQSAM) is activated. After the disk labels are processed as 'SD' during the QSAM file OPEN, the DLBLs are changed back to 'DA' to allow random BDAM processing. OFF is the default.
DMCL= <i>dmcl-name</i>	Specifies the name of the DMCL load module to use in local mode. IDMSDMCL is the default.
DMLTRACE=ON/OFF	ON activates a trace facility that traces all navigational DML requests made by an application. OFF is the default.
DSGROUP= <i>data-sharing-group-name</i> (OS/390 only)	Specifies the name of the data sharing group of which this system is a member. All CA-IDMS systems that are members of the same group must specify the same group name. The data-sharing-group-name must be a 1-8 character name consisting of characters A-Z, 0-9, \$, #, or @. Names that begin with SYS or UNDESIG are reserved and cannot be used. Names that begin with A-I may be in use by the operating system and should be avoided.  <b>Note:</b> This parameter is only applicable in a data sharing environment.
ECHO=ON/OFF	Indicates whether SYSIDMS parameters are displayed on the JES log. OFF is the default.

---

Parameter	Description
FILE_BUF= <i>dname=nnnnn</i>	<p>Allows users to increase the number of pages in a buffer used by a specific file for a local mode job without having to change the DMCL. In CV, a DCMT command can be used to alter the number of pages in a buffer. The FILE_BUF parameter provides a similar capability for local mode jobs. If specified, the number of pages in the buffer pool associated with the specified file is increased by <i>nnnnn</i>.</p> <p>This parameter can be used to tune PREFETCH processing by allowing the local mode user to increase the number of pages in a specific buffer for a job and thereby maximize the benefit of prefetch processing.</p>
IDMSQSAM=ON/OFF	<p>ON activates the IDMSQSAM facility (sequential access for look-ahead database reads). OFF is the default.</p>
JOURNAL=ON/OFF	<p>Specifies whether journaling will be performed in local mode. OFF specifies that local mode journaling will not be performed, even if there are tape journals defined in the DMCL. ON is the default.</p>
JRNLDTS= <i>yyyy-mm-dd-hh.mm.ss.nnnnnn</i>	<p>This parameter provides a way to bypass a date time stamp mismatch problem between the DMCL and the journal files. The <i>yyy-mm-dd-hh.mm.ss.nnnnnn</i> is the date time stamp on the journal file. This should only be used if you know that the reason for the mismatch will not cause a problem. Inappropriate use of this parameter may cause database corruption.</p>

<b>Parameter</b>	<b>Description</b>
LANG=xxxxxxxxxxxxxxxx	Sets an alternate environment for DBCS support. This parameter is useful for local mode batch jobs and is equivalent to issuing the DCUF SET LANG command for online users. The language environment name specified can be a maximum of 19 characters long.
LENGTH_PAGE=nnn	Specifies the maximum number of lines to be printed on a page. nnn must be in the range from 10 through 32,767. The default is 60.
LIST=SYSLST/SYSOUT/BOTH (BS2000/OSD only)	Specifies whether output is written to SYSLST, SYSOUT, or both. The default is SYSLST.
LOADAREA=ON/OFF	Specifies whether the dictionary load (DDLDCLOD) area is to be accessed when loading a module. If OFF is specified, the dictionary load area will not be accessed. You should specify OFF only when all load modules are linked into an OPSYS load library. ON is the default.
LOCAL=ON/OFF	Specifies whether a batch job is to execute in local mode. If ON is specified, all requests are processed locally even if an IDMSOPTI is link-edited with the program, or a SYSCTL file is specified in the JCL. OFF is the default.
LOCAL_DYNAMIC_ALLOCATION= ON/OFF	OFF directs a local IDMS batch job to ignore any DSN information defined in the DMCL for database files, and requires that the DSN information be included in the JCL in order to access a database file. ON is the default.
LOCAL_NOJOURNAL_RETRIEVAL	Specifies that local batch jobs not journal RETRIEVAL ONLY transactions.

---

<b>Parameter</b>	<b>Description</b>
LOCALPUR=ON/OFF	<p>ON forces the purging of the local mode buffer pool whenever a transaction terminates.</p> <p>This parameter addresses a change in the way local mode buffers are handled (between 10.21 and later releases). In release 10.21 a local mode job that had multiple transactions (run units) would have separate buffer pools for each transaction (and each transaction would have no knowledge of the others). When a transaction terminated its buffer pool would be purged. Starting in release 12.0, a local job with multiple transactions will have just one buffer pool shared by all transactions. When a transaction terminates the buffer remains unchanged until the last transaction terminates at which time the shared buffer pool is purged. To make the system purge the common buffer pool when each transaction terminates (and therefore mimic what happened in release 10.21), use the parm LOCALPUR=ON (OFF is the default).</p> <p><b>Note:</b> This parameter should only be specified if a compatibility problem is encountered, since there can be performance implications in specifying LOCALPUR=ON.</p>
MSGDICT=ON/OFF	<p>Specifies whether the dictionary message (SYSMSG) area is to be accessed in order to retrieve the text of messages. If OFF is specified, the dictionary message area is not accessed. OFF should be specified only when using a DMCL that does not contain the SYSMSG segment, such as during installation. ON is the default.</p>

---

---

<b>Parameter</b>	<b>Description</b>
MULTIDSN=ON/OFF (VSE/ESA only)	ON specifies that tape files may span multiple volumes. At the end of a tape reel, EOF (end of file) or EOV (end of volume) prompts the user to specify an END OF JOB or an END OF VOLUME condition. The default, OFF, specifies that END OF JOB is automatically the condition at the end of a tape reel.
NODENAME= <i>nodename</i>	For non-SQL applications running under the central version, identifies the DC/UCF system to bind to at run time.
OVERPRINT=YES/NO (BS2000/OSD only)	Specifies whether the overprint facility is used when writing to SYSLST. The default is YES.
PARM= <i>'parameter-string'</i>	Allows you to specify parameters typically specified in a JCL EXEC PARM statement. The format is the same as the IBM PARM parameter on the EXEC JCL statement. parameter-string can contain any 1 through 256 character parameter and can be specified on multiple lines.
PREFETCH=ON/OFF	OFF overrides the default ON and prevents IDMS from prefetching database pages, the normal processing when an area or index sweep is detected. Specify OFF for a local batch job to prevent prefetching database pages for the job step. Specify OFF in the SYSIDMS file associated with a central version to prevent prefetching pages for all transactions running with the central version.
PREFETCH_BUF= <i>nnnnn</i>	Specifies the minimum number of pages in a buffer pool that must be present before IDMS will use prefetch processing for non-area sweep requests. This parameter applies to both local and Central Version environments.

---

Parameter	Description
PROCTRACE=ON/OFF	ON activates a trace of key user blocks that participate in an SQL PROCEDURE call. OFF is the default.
QSAMAREA= <i>qsam-area-name</i>	Specifies the physical area in the DMCL for which the IDMSQSAM facility will do look-ahead reads. If this parameter is omitted and the IDMSQSAM=ON parameter is specified, the look-ahead reads will be performed on the first area that is accessed by the transaction.
QSAMBUF#= <i>nnn</i> (OS/390 and MSP/EX only)	Specifies the number of buffers to use when the IDMSQSAM facility is active. <i>nnn</i> can be from 1 to 255. QSAMBUF# enables you to set the number of QSAM buffers to be used without having to code JCL for the file being processed by IDMSQSAM. If QSAMBUF# is not specified, the number of buffers is determined by the DCB parameter BUFNO= <i>nnn</i> , or defaults to 5 buffers.
QSAM#BUF= <i>nnn</i> (BS2000/OSD only)	Specifies the number of buffers to use for IDMSQSAM simulation.
QSAMTRACE=ON/OFF	ON activates a trace of all the IDMSQSAM look-ahead I/O reads. This trace shows the name of the file(s) being accessed by IDMSQSAM, each RBN that is read using QSAM or BDAM (DAM/EXCP), and a summary of the number of RBN's read through QSAM and BDAM. It also shows the area being accessed and the number of OPSYS QSAM buffers being used as determined by the JCL. OFF is the default.

---

<b>Parameter</b>	<b>Description</b>
REREAD_SYSCTL=ON/OFF	<p>ON directs local mode operations to reread the SYSCTL file for each new transaction. This allows you to</p> <ol style="list-style-type: none"><li>1. Include a SYSCTL in a batch job step's JCL.</li><li>2. Start a transaction that will execute under central version, based on the contents of the SYSCTL file.</li><li>3. Deallocate the SYSCTL file defined in the JCL.</li><li>4. Start another transaction to execute in local mode.</li></ol> <p>OFF is the default.</p>
ROLLBACK3490	<p>Enables the ROLLBACK utility to process archive files residing on devices that do not support backward read, such as disk and 3490E devices.</p>
SORTSIZE=ON/OFF	<p>Directs whether or not IDMS utilities generate the SIZE= sort parameter card. Some sort packages cannot handle the SIZE= parameter. The default is ON which means that the SIZE= sort parameter will be generated.</p>
SQL_INTLSORT=ON/OFF	<p>Allows you to force the internal IDMS sort to be used in local mode. If ON is specified, an internal SORT rather than an operating system SORT will be performed on SQL commands issued in a local batch job that contain an ORDER BY clause. In many cases, an internal SORT is faster than an operating system SORT when you are not dealing with a large amount of data. OFF is the default, indicating an operating system SORT will be used.</p>
SQLTRACE=ON/OFF	<p>ON activates a trace facility of all the SQL database requests made by an application. OFF is the default.</p>

---

Parameter	Description
SYS_MSG=UPLow/UPPER	<p>UPPER directs IDMS to translate the text of internal #WTL messages to uppercase before being displayed at the output destination. The default is UPLow. This allows the text of an internal #WTL message issued by CA software to be displayed in mixed case letters.</p> <p>Specify UPPER under these conditions:</p> <ul style="list-style-type: none"> <li>■ In local batch jobs to translate any internal #WTL messages issued by CA software to uppercase for that job step.</li> <li>■ In the SYSIDMS file associated with a Central Version to translate any internal #WTL messages issued by CA software to uppercase for that CV region.</li> </ul>
SYSCTL= <i>ddname</i>	<p>Specifies an alternate ddname for the SYSCTL file (other than the default ddname of SYSCTL).</p>
UPPER=INPUT/OUTPUT/BOTH/OFF	<p>Specifies whether input and/or output files will be converted to uppercase:</p> <ul style="list-style-type: none"> <li>■ INPUT — Converts SYSIPT input files to uppercase.</li> <li>■ OUTPUT — Converts SYSLST output files to uppercase.</li> <li>■ BOTH — Converts both SYSIPT input files and SYSLST output files to uppercase.</li> <li>■ OFF (the default) — Does not convert SYSIPT input files or SYSLST output files to uppercase.</li> </ul>
USERCAT=ON/OFF	<p>Specifies whether the user catalog is to be accessed. Specify OFF only when formatting the user catalog or when the DMCL does not have access to a user catalog. ON is the default.</p>

Parameter	Description
WIDTH_PAGE= <i>nnn</i>	Specifies a maximum number of characters to be printed on a SYSLST output line. <i>nnn</i> must be an integer in the range from 71 to 132. The default is 132.
XA_SCRATCH=ON/OFF	Specifies whether scratch space will be allocated out of XA storage or not. OFF, the default, indicates that a scratch file will be used.

**BUFFERSTAT report field descriptions:** The following table gives the descriptions for the fields displayed on the report produced by the BUFFERSTAT SYSIDMS parameter.

Field	Description
*** Buffer Name ***	The name of the buffer from the DMCL which has been opened during the processing of this job. Only those buffers which are open or have had some I/O activity against them will appear in this report.
*** Pages ***	The number of pages allocated to the buffer. This is the actual number of pages 'in use' by this buffer for this job. This number is the total from the a) DMCL Local Mode Buffer Pages <i>nnn</i> , or b) JCL DCB=BUFNO= <i>nnn</i> , or c) SYSIDMS Parm FILE_BUF= <i>ddname=nnn</i> .
*** Prefetch Minimum ***	The minimum number of buffers 'in use' needed to allow Prefetch to operate from Random access verbs (ie., non-area sweep processing).
DB Page Requests	The number of times IDMSDBMS requests a database page from the buffer pool by calling IDMSDBIO.
Sequential Area Request	Of the DB Page requests in the 'DB Page Requests' count, how many were GET/NEXT/PRIOR in AREA verbs.

<b>Field</b>	<b>Description</b>
Random Request	Of the DB Page requests in the 'DB Page Requests' count, how many were not GET/NEXT/PRIOR in AREA verbs.
Found in Buffer	Of the DB Page requests in the 'DB Page Requests' count, how many DB pages were already in the buffer pool and did not require an I/O.
Not Found in Buffer	Of the DB Page requests in the 'DB Page Requests' count, how many DB pages were not in the buffer pool and therefore required an I/O.
Buffer 'hit' Ratio	Calculated as 'Found in Buffer' times 100, divided by the DB Page Requests value.
Found in Pref Buffer	Found in Prefetch Buffer. Of the 'DB Page Requests' count, how many DB pages were found in the buffer pool that had been read by a Prefetch Read.
Found in Cache	Of the 'DB Page Requests' count, how many DB pages were found in the shared cache.
Total DB Pages Read	The total number of DB pages read by both Prefetch and standard I/O. This is not the number of I/O's or EXCP's, but the number of DB pages read into a buffer as a standalone I/O, 'Start I/O - Reads', or as part of a Prefetch I/O 'Pages Read - Prefetch'.
DB Pg Req:Tot Pages Read	The number of 'DB Page Requests' count divided by the 'Total DB Pages Read' count.
DB Pg Req:Strt I/O Read	The number of 'DB Page Requests' count divided by the 'Start I/O Reads' count.

<b>Field</b>	<b>Description</b>
NonPrefetch I/O Rqst	The number of standard of non-Prefetch I/O requests. This is the number of the 'DB Pages Request' count which are 'Not found in Buffer' and were not eligible for Prefetch. Prefetch was either not allowed, turned off, or the request was started by a 'Random Request' for which the '*** Prefetch Minimum ***' is higher than the number of '*** Pages ***'.
Start I/O - Reads	The number of standard or non-Prefetch I/O reads. This number is the result of 'DB Page Requests' which were 'Not found in Buffer' and Prefetch is turned off, or the request was started by a 'Random Request' for which the '*** Prefetch Minimum ***' is higher than the number of '*** Pages ***' count, or the request is not eligible for Prefetch processing. Each of these will show up as 1 EXCP.
Start I/O - Writes	The number of writes started against the database. Each of these will show up as 1 EXCP.
Read forces Write	In order to read a DB page into the buffer, a DB page had to be written out to the database first (based on the least recently used algorithm). When this occurs, Prefetch is effectively turned off.
Prefetch Requests	With Prefetch operating, the number of DB page requests 'Not found in Buffer', which were eligible for Prefetch processing.
Sequential Area Request	Of the 'Prefetch Requests' count, how many were GET NEXT/PRIOR in AREA type verbs.
Random Request	Of the 'Prefetch Requests' count, how many were not GET NEXT/PRIOR in AREA type verbs.

<b>Field</b>	<b>Description</b>
Pref Req Denied:Buffers	Of the 'Prefetch Requests' count, how many did not use Prefetch due to too many buffer pages with the 'must write switch' on (over 1/2 the number of pages in the buffer pool).
Start I/O - Prefetch	The number of the 'Prefetch Requests' count that actually "Start an I/O" or "Start Subchannel". Each of these will show up as 1 EXCP.
Pages Read - Prefetch	The number of DB pages that were "carried" with every 'Start I/O - Prefetch'. This number plus the 'Start I/O - Reads' will equal the 'Total DB Pages Read' count.
Pref Strt I/O:Pref Req	The 'Start I/O - Prefetch' count divided by the 'Prefetch Requests' count.
Pref Pages:Pref Strt I/O	The 'Pages Read - Prefetch' count divided by the 'Prefetch Requests' count.
Pref Pages:Pref Strt I/O	The 'Pages Read - Prefetch' count divided by the 'Start I/O - Prefetch' count. This value shows how effective the Prefetch operation is. Compare this number to the "pages per track" to see how effective each Prefetch I/O is. If this number is around 3/1 or less, you probably will not see enough improvement in performance to warrant using Prefetch.

For each file in use, there will be a set of counts:

- Filename — The DDname of the file using this buffer. Only the files that are open will show up in this report.
- Pgs read — The number of DB pages read from this file. This is not the number of I/Os or EXCPs.
- Written — The number of DB pages written to this file. This is the number of I/Os or EXCPs.
- In buffer — The number of 'DB Page Requests' that were found in the buffer pool which this file maps to.

- In prefetch — The number of 'DB Page Requests' that were found in the buffer pool which this file maps to due to Prefetch.

**VSE/ESA file parameters**

<b>Parameter</b>	<b>Description</b>
FILENAME= <i>file-name</i>	Specifies the name of the file whose attributes are to be overridden by the following SYSIDMS parameters.
BLKSIZE= <i>block-size</i>	Specifies the block size for a file. BLKSIZE and BLOCKS are mutually exclusive parameters.
BLOCKS= <i>block-count</i>	Specifies a blocking factor for a file. BLKSIZE and BLOCKS are mutually exclusive parameters.
DEVADDR=SYSxxx	Specifies a device address for a tape file (SYSIPT, SYSLST, SYSRDR, SYSPCH, or SYSlogical-unit-number).
FILABL=NO	Specifies a no-label option for a tape file. FILABL=STD is the default.
FILETYPE= <i>file-type</i>	Specifies a file type of tape, disk or file independent.
REWIND=YES/NO/UNLOAD	Specifies the position of a tape file when it is opened or closed. REWIND=UNLOAD is the default.

# Index

---

## Special Characters

=COPY facility E-10, E-23

## A

ABEND\_ON\_DEADLOCK SYSIDMS parameter H-6

ABENDTRACE SYSIDMS parameter H-6

ABENDTRACE\_ENTRIES SYSIDMS parameter H-6

ABENDTRACE\_SUBSCHEMA\_DISPLAY

parameter H-6

ABENDTRACE\_VIBSNAP SYSIDMS parameter H-7

ABRT journal record 18-6

access modules

migrating 24-6

statistics, monitoring 22-14

access, restricting for DML programs

area 14-18

record 14-23

set 14-29

ADD operation 11-3

defaults 11-4

effect on areas 14-19

effect on non-SQL schema 13-12

effect on records 13-35

effect on sets 14-30

effect on subschema 14-12

interpreted as MODIFY 10-20, 13-13, 14-13

ADSOBTAT utility program 24-26

ADSTATU utility program 24-26

AFTER procedure 15-21

AFTR journal record 18-6

alignment, boundary 13-50

ALL clause

compiler operations for a user 12-13

compiler operations for public access 13-9, 14-10

in path-group ERASE 14-46

ALL COMMENT TYPES clause 13-34

ALLOWED

for DML functions 14-18, 14-23, 14-29

in PUBLIC ACCESS clause 13-9, 14-10

alphabetic data 13-57, 13-59

alphanumeric data 13-49, 13-56, 13-59

ALTER operation 11-5

AMODE clause 14-65

application configurations 1-12

application control block 15-8—15-10

application dictionary

components 23-5

defining 23-14

definition 1-7

description 23-3

application program information block 15-8, 15-10

application structure, migrating 24-5

archive journal file

defining 6-22—6-25

dropping 6-25

multiple 18-9

ARCHIVE JOURNAL statement 6-22—6-25

area

dictionary 23-3

area acquisition threshold 37-16

area control block 15-8, 15-11

AREA journal record 18-6

area locks

for SQL transactions 37-16

status 37-7

when acquired 37-16

area procedures 15-8

area ready modes

*See also* ready modes

default 37-8

types 37-4—37-10

AREA statement 13-15—13-20

AREA statement (non-SQL schema)

copying 8-8

definition procedure 8-8

AREA statement (physical database) 6-26—6-42

AREA statement (subschema) 14-17—14-20

ADD/MODIFY/DELETE syntax 14-17

definition procedure 8-19

area-to-file mapping 3-6

AREA\_VALIDATION\_MSGS SYSIDMS

parameter H-7

areas

space management 33-10—33-12

space management page 33-12

space management page (SMP) 33-15

symbolics 3-8—3-9

areas (subschema)

access restrictions 14-18

ready mode 14-18

readying 14-19

areas, non-SQL schema

adding/deleting 31-6

- 
- areas, non-SQL schema (*continued*)
    - calls needed for compression 13-20, 13-41
    - changing characteristics 31-7
    - name 13-16
    - qualification 13-16
    - ready mode, for database procedures 13-17
  - areas, physical
    - adding pages 6-42
    - AREA statement 6-26—6-42
    - contiguity of pages 6-36
    - definition 2-4, 3-4
    - dropping 6-42
    - file blocks 6-38
    - increasing size 6-36, 25-12
    - locks 37-11—37-12
    - mapping to files 6-41
    - offsets 6-36
    - override specification 6-77
    - page range, extending 25-13
    - page ranges 6-36—6-37
    - page size, increasing 25-12
    - physical device blocking 6-37—6-38
    - restrictions, native VSAM 6-39
    - synchronization stamp 3-9, 6-38
  - areas, subschema
    - adding/modifying/deleting 32-8
  - AREPORTs 24-13
  - AS SYNTAX/COMMENTS clause
    - setting the session default for 10-27
  - ASCENDING KEY clause
    - COBOL elements 13-52
    - sorted sets 13-83
  - Assembler Language
    - element names 13-54
    - record names 13-37
  - ASSIGN RECORD IDS clause 13-8
  - asterisk
    - significance in non-SQL DDL statements 9-8
  - asterisk with plus sign
    - impact on ECHO 10-21
    - impact on LIST 10-22
    - in output of DISPLAY statements 10-27
    - significance in non-SQL DDL statements 9-8
  - attributes, used in migration 24-13
  - authority
    - to access entity descriptions 11-10
    - to access schema descriptions 12-13, 13-13
    - to access subschema descriptions 14-14
  - AUTHORIZATION clause 14-8
  - AUTO
    - for assigning pointer positions 13-78, 13-81—13-82
  - AUTO (*continued*)
    - for assigning record IDs 13-27
  - automatic
    - assignment of record IDs 13-8, 13-26
    - assignment of set pointers 13-78, 13-81—13-82
    - change of hyphen to underscore for PL/I 13-37
    - changes in PUBLIC ACCESS 14-15
    - deletion of load module 13-12
    - deletion of set 13-36, 13-85, 14-25, 14-30
    - deletion of subschema area 13-19
    - generation of ON clauses 14-56
    - generation of ON clauses (table) 14-59
    - inclusion of correct synonym 13-27, 13-52
  - AUTOMATIC set removal option 13-83
- B**
- backup
    - definition 19-3
    - following normal system shutdown 19-5
    - for local mode jobs 19-10
    - procedures 19-4
  - base element, in REDEFINES clause 13-47
  - batch compilation
    - non-SQL schema E-4
    - subschema E-5
  - Batch Shared Resources Subsystem (Batch LSR) 17-5
  - BEFORE procedure 15-20
  - BFOR journal record 18-6
  - BEGIN journal record 18-6
  - BINARY element usage 13-49
  - BIND statement 6-54—6-55
  - BIT element usage 13-49
  - bit elements 13-60
  - BLANK WHEN ZERO clause (schema elements) 13-51, 13-59
  - blank, in non-SQL DDL statements 9-8
  - BLKSIZE SYSIDMS parameter H-22
  - BLOCKS SYSIDMS parameter H-22
  - boolean expression
    - order of evaluation 12-6
    - syntax 12-4
  - boundary alignment 13-50
  - BS2000/OSD JCL
    - non-SQL schema compiler E-23
    - subschema compiler E-25
  - BUFFER statement 6-43—6-48
  - buffer utilization ratio 22-6
  - BUFFER\_PURGE SYSIDMS parameter H-7
  - buffers
    - acquisition 6-47
-

---

buffers (*continued*)

- changing characteristics of 17-7
  - database 4-8
  - default 6-48
  - definition 4-8
  - dropping 6-48
  - for database load 20-5, 21-7
  - for native VSAM files 6-45
  - in DMCL 2-5
  - in hyperspace 17-5
  - incrementing through JCL 17-4
  - management 17-3—17-12
  - native VSAM file considerations D-6
  - nonshared resource buffer pools 6-45
  - number of 17-3
  - page count, central version 6-46, 6-47
  - page count, local mode 6-45
  - page size 6-44, 17-5
  - sizing 17-3
  - statistics, monitoring 22-6—22-7
  - storage acquisition method 17-5
  - tuning 17-8
- buffers, journal
- See* journal buffer
- BUFFERSTAT SYSIDMS parameter H-7
- BUFNI, VSAM buffer pool specification 6-45
- BUILD phase 21-8

## C

### CA-IDMS

- application environments 1-12
  - central version operations 1-3—1-4
  - components 1-3
  - database definition 1-11—1-12
  - database design 1-11
  - dictionaries 1-7
  - installation 1-10
  - loading the database 1-12
  - local mode operations 1-4
  - logical database definition 1-8
  - physical database definition 1-8
  - runtime components 1-10
  - security 1-9
  - types of operation 1-3
- CA-IDMS command facility 1-13, 7-4
- CA-IDMS/DB
- components 1-7—1-12
- CA-IDMS/DB VSAM file
- and LSR buffer management 17-5

### CALC

- control element 13-27
  - element name 13-28
  - location mode 13-27
  - set 33-11
  - storage mode 34-4, 34-7
- CALC keys 13-28—13-29
- changing 31-9
  - changing the DUPLICATES option 31-11
  - creating for a table 29-6
  - defining 7-8
  - dropping from a table 29-6
  - unique 7-8
- CALC record, loading 20-4
- CALL clause
- in area specification 13-17
  - in record specification 13-33
  - order of execution 13-41
- CALL statements
- generated by PROCEDURE NAME 13-41
  - specifying 15-16
- calls to database procedures
- before and after DML statements 15-4
  - location of 15-4
- card-image
- control block G-9
  - user exit G-4
- CASCADE option
- for tables 28-7
  - for views 28-5
- catalog
- areas 23-3
  - defining 23-13
  - schemas 23-11
- central version 1-3—1-4
- binding programs 15-15
  - buffers 17-3
  - journaling 2-5, 18-3
  - recovery, automatic 19-14
  - runtime components 23-20
  - warmstart 19-14
- central version operations
- area lock status 37-7
  - handling of physical area lock 37-12
  - lock management 37-13
- CHAIN set mode 13-77
- chained reads 17-9
- and the read driver 17-11
- chained sets
- connecting records to 35-5—35-6
  - database notation 35-4

---

chained sets (*continued*)  
     disconnecting records from 35-6—35-7  
     pointers 35-4  
     reordering 31-21  
     retrieving chained records 35-7—35-8

character  
     decimal point 10-20  
     quote 10-24

check constraints 28-10

checkpoints, journal 18-6

CKPT journal record 18-6

CLASS clause 13-10, 14-11

classes, used in migration 24-13

CLEAR  
     in path-group ON 14-57  
     logical-record variable-storage 14-34, 14-35

CLUSTERED storage mode  
     CLUSTERED storage mode  
     index 34-9—34-11  
     introduction to 34-7  
     linked relationship 34-7—34-9

CMS commands  
     non-SQL schema compiler E-20  
     subschema compiler E-21

COBOL  
     condition names 13-48, 13-59  
     element names 13-54  
     record names 13-37

COBOL DML precompiler 13-60

code tables, migrating 24-7

coding considerations  
     for non-SQL schema and subschema compilers 9-7

colon, in non-SQL DDL statements 9-8

columns  
     adding to an existing table 28-8  
     changing characteristics of 28-10  
     dropping 28-9

comma  
     as decimal point 10-20, 12-4, 13-58  
     in non-SQL DDL statements 9-8

command facility 7-4

comment keys  
     assigning text 14-15

comments  
     displaying options as 10-27

COMMENTS clause 9-14  
     for record elements 13-54  
     in logical-record display 14-35  
     in non-SQL schema display 13-11  
     in non-SQL schemas 14-11  
     in record display 13-34

COMMENTS clause (*continued*)  
     in subschema display 14-11  
     logical records 14-35  
     setting DISPLAY/PUNCH default 10-27  
     setting sequence numbers for 10-24

COMMIT user exit G-4

COMMIT, specifying database procedures for 13-17

communication  
     between programs and procedures 15-15

comparison operators, in boolean expression 12-4

compiler functions 8-6

compiler-directive statements  
     DISPLAY/PUNCH ALL 6-12—6-21, 10-4—10-10  
     DISPLAY/PUNCH IDD 10-11—10-13  
     INCLUDE 10-14—10-15  
     overview 10-3  
     SET OPTIONS 10-16—10-32  
     SIGNOFF 10-33  
     SIGNON 10-34—10-37  
     types of 9-16

compilers 9-3

compiling schemas and subschemas E-4—E-25

compress/decompression procedures 13-32, 13-33

COMPRESSED index entries 13-84

compressed record 13-31—13-33, 13-39—13-40

compression 13-31—13-33  
     calls needed for area 13-20, 13-41

compression, data 7-6

COMPUTATIONAL-n (COMP-n) element usage 13-49

COMPUTE, in path group 14-45

COMT journal record 18-6

condition name  
     assigning a value to a 13-48  
     defined 13-59

CONDITION-NAME element usage 13-50

CONNECT  
     DML restriction 14-23, 14-29  
     in path group 14-46  
     specifying database procedures for 13-33

constraint  
     *See* referential constraint

CONTAINS option of boolean expression 6-13, 10-6, 12-5

control element  
     CALC 13-27  
     sort 13-83  
     variable-length records 13-51  
     VSAM CALC 13-29

CONTROL LENGTH, in MINIMUM ROOT LENGTH clause 13-31

---

---

COPY ELEMENTS substatement 13-69—13-71  
   mixing with element substatement 13-70  
 copying  
   areas 13-16  
   sets 13-76  
 CREATE operation 11-3  
 CREATION responsibility, documenting 12-14  
 CULPRIT HEADER clause, in ELEMENT  
   substatement 13-53  
 CULPRIT HEADERS clause 10-25  
 currency  
   establishing, for non-SQL schemas and  
     subschemas 8-30  
   non-SQL schema 13-4  
   physical database DDL entities 6-8  
   subschema 14-4  
 CURSOR STABILITY isolation level 37-8  
 cushion 36-6  
 CVMACH SYSIDMS parameter H-7  
 CVNUM SYSIDMS parameter H-8  
 CVRETRY SYSIDMS parameter H-8

**D**

DASD block I/O file access method 16-5  
 data characteristic table (DCT) 13-32  
 data compression 7-6, 28-10, 31-15  
   specifying 13-31—13-33  
 data decompression  
   specifying 13-31—13-33  
 data items 13-56  
 data portion, record occurrence 33-5  
 data types 13-48  
   (table) 13-60  
   alphabetic 13-57, 13-59  
   alphanumeric 13-49, 13-56, 13-59  
   external floating point 13-57—13-60  
   fixed decimal 13-57  
   numeric 13-49  
   numeric edited 13-58—13-60  
 database  
   areas 33-10—33-15  
   key 33-7—33-9  
   pages 33-4—33-6  
 database access  
   SQL applications 5-5  
 database definition procedure  
   non-SQL 8-3  
   SQL-defined 7-3  
 database definition, about 1-11  
   database design 1-11  
   database files, definition of 3-3  
   database key 33-3  
     definition 33-7  
     for VSAM ESDS files 6-37  
     for VSAM KSDS files 6-37  
     format 33-7—33-9  
     variable format 6-97  
   database loading, non-SQL  
     considerations 20-4  
     procedure using FASTLOAD 20-6  
     techniques for large databases 20-5  
     using user-written program 20-7—20-10  
   database loading, SQL  
     BUILD phase 21-8  
     considerations 21-7  
     data types 21-11  
     full load 21-13  
     input file 21-10—21-11  
     multiple tables 21-10, 21-11  
     null values 21-11  
     options 21-5  
     performance, enhancing 21-9  
     phased load 21-13—21-15  
     procedures 21-12—21-19  
     process for 21-3  
     segmented load 21-15—21-16  
     stepped load 21-16—21-19  
     table columns 21-10  
   database monitoring  
     *See* monitoring  
   database name table  
     DBNAME statement 5-3, 6-52—6-57  
     DBTABLE statement 6-58—6-62  
     default dictionary 5-9  
     defining 5-13  
     definition 2-5  
     generating 5-13  
     modifying 26-3—26-4  
     restrictions 6-55  
     segments, specifying 5-5  
     subschema mapping 5-13  
   database procedure blocks 15-8  
   database procedure calls  
     in relation to DML statements 15-4  
     in relation to error conditions 15-4  
     location of 15-4  
   database procedures  
     adding/dropping 31-16  
     AFTER procedure 15-21  
     BEFORE procedure 15-20

---

---

database procedures (*continued*)
 

- calling 13-19
- calling non-reentrant or non-assembler 15-17
- changing calls 13-20
- coding 15-7—15-15
- common uses of 15-5—15-6
- compression/decompression 15-5
- data collection 15-6
- data validation 15-5
- definition 15-3
- example 15-22—15-24
- executing 15-20—15-21
- IDMSNVLR procedure 15-6
- invoking 15-16
- link editing 15-16
- ON-ERROR procedure 15-20
- privacy/security 15-5
- program/procedure communication 15-15—15-16
- under central version 15-15
- updating, deleting 13-40
- variable-length native VSAM records 15-6
- when no information is passed 15-16

database record field name
 

- assigning 13-47
- specifying in path group 12-8, 14-58

date 13-8

db-key
 

- See* database key

DB\_DEADLOCK\_DUMP SYSIDMS parameter H-8

DBCS edited data
 

- picture format 13-57

DBGROUP statement 6-49—6-51

DBKEY
 

- See also* database key
- as control element for sorted sets 13-84

DBNAME
 

- in JCL E-4

DBNAME statement 6-52—6-57

DBNAME SYSIDMS parameter H-8

DBNAME table
 

- See* database name table

DBTABLE statement 6-58—6-62

DC\_DEADLOCK\_0029 SYSIDMS parameter H-8

DC\_DEADLOCK\_NODUMP SYSIDMS parameter H-8

DC\_SCRATCH SYSIDMS parameter H-8

DCMT commands
 

- for database buffers 17-4
- for journaling 18-13
- for the read driver 17-11
- PREFETCH option, for chained reads 17-10

DCNAME SYSIDMS parameter H-9

DCTABLE NAME clause 13-32

DDDL compiler 1-13

DDL CAT area 23-3

DDL CATLOD area 23-3

DDL CLOD area 9-17, 23-3

DDL CLOG area 23-20

DDL CMSG area 23-3, 23-20

DDL CRUN area 23-20

DDL CSCR area 23-20

DDL DML area 9-17, 23-3

DDL OCSR area 23-21

DDL SEC area 23-20

deadlock detection interval 37-26

DEADLOCK\_ABEND\_0029 SYSIDMS parameters H-9

DEADLOCK\_ABEND\_ERUS SYSIDMS parameter H-9

DEADLOCK\_DETAILS SYSIDMS parameter H-9

deadlocks 37-25—37-27

decimal point character
 

- in boolean expression 12-4
- in PICTURE clauses 13-58
- setting the character for 10-20

DECIMAL-POINT clause 10-20

decompression 13-31—13-33

DEFAULT clause 10-20

default dictionary
 

- defining 23-19
- specifying 23-23

default index on a table 28-12

DEFAULT USAGE clause 14-19

DELETE clause
 

- in SET OPTIONS statement 10-21

DELETE operation 11-6
 

- allowed/disallowed for a user 12-13
- effect on areas 14-19
- effect on load modules 14-65
- effect on non-SQL schema 13-12
- effect on records 13-36, 14-25
- effect on sets 14-30
- effect on subschema 14-13
- for public access 13-9, 14-10

DELETE RECORD
 

- effect on sets 14-25

DELETION responsibility, documenting 12-14

delimiter, end-of-file

DERIVED FROM clause 13-9

DESCENDING KEY clause
 

- COBOL elements 13-52
- sorted sets 13-83

---

DEVADDR SYSIDMS parameter H-22

device blocking 6-37

dialogs, migrating 24-6

dictionaries

- See also* application dictionary
- See also* system dictionary
- DDDL compiler 1-13
- default 5-9, 6-61, 23-19, 23-23
- defining 23-14—23-18
- definition 1-7, 23-3
- definitions, CA-supplied 23-8—23-12
- logical components 23-4
- logical definitions 23-9
- message area 23-5
- modules 23-11
- nondatabase structures 23-11
- physical components 23-3
- protocols 23-11
- schemas, non-SQL 23-9
- segments 23-5
- subschemas 23-9

dictionary

- displaying options 10-27
- entities, display 10-11
- node, specifying for compilation 10-35
- record types in logical records 14-34

dictionary load utility, IDMSDIRL 23-9

DICTNAME SYSIDMS parameter H-9

DICTNODE SYSIDMS parameter H-10

DIRECT location mode 13-28

DIRECT storage mode 34-3

disallowing DML functions

- area ready modes 14-18
- record access functions 14-23
- set access functions 14-29

DISCONNECT

- DML restriction 14-23, 14-29
- in path group 14-46
- specifying database procedures for 13-33

DISK JOURNAL statement 6-63—6-66

disk journals

- considerations 4-11
- defining 6-63—6-66
- modifying the access method 25-16
- modifying the size 25-15

DISPLACEMENT

- of index keys 13-78
- of VIA set members 13-29

DISPLAY, in element USAGE clause 13-50

DISPLAY/PUNCH ALL statement 6-12—6-21, 10-4—10-10

DISPLAY/PUNCH ALL statement (*continued*)

- entity options (table) 10-7

DISPLAY/PUNCH IDD statement 10-11—10-13

- entity options (table) 10-11

DISPLAY/PUNCH operations

- allowed/disallowed for a user 12-13
- for public access 13-9, 14-10
- locations of output 11-8

DISPLAY/PUNCH SCHEMA statement 14-67—14-69

DISPLAY/PUNCH statements

- defaults 11-10
- displayed as syntax 10-27
- displayed at comments 10-27
- effect on load modules 14-65
- setting the session defaults for 10-24
- setting the session defaults for (table) 10-31
- used in migration 24-17, 24-18

DLBLMOD SYSIDMS parameter H-10

DMCL

- central version 4-4
- components of 2-4
- default 2-4
- defining 4-19—4-23
- definition 2-3, 4-3
- DMCL statement 6-67—6-81
- DMCL, central version 4-4
- DMCL, local mode 4-4
- dynamic management 25-8—25-9
- external file names 6-80
- identifying to runtime system 4-5
- local mode 4-4
- making accessible at runtime 4-22—4-23
- segments in central version DMCL 4-13
- segments in local mode DMCL 4-13

DMCL statement 6-67—6-81

DMCL SYSIDMS parameter H-10

DML functions, allowing/disallowing

- area ready modes 14-18
- record 14-23
- set 14-29

DML programs, setting default area ready mode for 14-8

DML statements

- in relation to database procedure calls 15-4

DMLTRACE SYSIDMS parameter H-10

DO, in path-group ON 14-56

documentational clauses

- COMMENTS 13-10, 13-53, 14-11, 14-35
- CULPRIT HEADER 13-53
- INCLUDE/EXCLUDE class-name 13-10, 14-11
- MEMO DATE 13-8

---

---

documentational clauses (*continued*)

- OLQ HEADER 13-53
- RESPONSIBLE FOR 12-14
- SCHEMA DESCRIPTION 13-8
- SUBSCHEMA DESCRIPTION 14-8

DREPORTs 24-12

driver, read and write 17-11

DROP operation 11-6

DSEG journal record 18-6

DSGROUP SYSIDMS parameter H-10

DUPLICATES

- clause for CALC record types 13-28
- clause for sorted sets 13-85
- clause for VSAM CALC record types 13-29

## E

EACH, in path-group FIND/OBTAIN 14-47, 14-51—14-54

ECHO clause 10-21

ECHO SYSIDMS parameter H-10

edit tables, migrating 24-7

EJECT format statement 9-18

element

- examples of definition 13-64—13-68
- in logical records 14-33
- in subschema views 14-23
- in subschema views (table) 14-26
- levels 13-56
- multiply-occurring 13-50—13-51
- name 13-47
- nesting 13-48, 13-56
- observing language conventions 13-54
- PICTURE clause 13-60
- storage characteristics 13-60
- USAGE clause 13-60

ELEMENT substatement 13-44—13-68

- COPY ELEMENTS syntax 13-69
- minimum 13-56
- mixing with COPY ELEMENTS 13-70
- qualification 13-69
- required clauses 13-64
- syntax 13-44

ELEMENT SYNONYM NAME clause (schema elements) 13-52

ELEMENT, in path-group SELECT clause 14-44

elementary item, defined 13-56

ELEMENTS clause

- and VIEW ID clause (table) 14-26
- logical-record specification 14-33
- records 14-23

ELEMENTS clause (*continued*)

- specifying fields 14-26

end-of-converse user exit G-4

end-of-file indicator 10-22

ENDJ journal record 18-6

entity

- control block G-8
- type, defined 9-7

entity occurrence

- defined 9-7
- naming conventions G-3

EOF clause 10-22

ERASE

- specifying database procedures for 13-33

ERASE command

- DML restriction 14-23
- in path group 14-46
- path group 14-43
- space management considerations 33-13

error messages

- displayed without line numbers 10-23

ESDS

- CALC keys 13-29
- database key construction 6-37
- location mode 13-29

estimated row count in tables 28-11

EVALUATE, in path group 14-47

EXCLUDE clauses

- ALL CALLS (areas) 13-19
- ALL CALLS (record) 13-34
- class-name 13-10, 14-11
- MEMBER 13-81
- RECORD SYNONYM 13-27
- USER 13-9, 14-10

exclusive lock mode 37-14

exclusive ready modes 37-4

exclusive record locks 37-19

EXCLUSIVE, area ready mode

- restricting DML programs from using 14-18
- setting as DML default 14-19

EXCP file access method 16-5

expansions for complex parameters

- See* parameter expansion

explicit record locks 37-19

explicit version number 10-20—10-21

exponent, in PICTURE clause 13-57

external floating point data 13-57—13-60

---

---

## F

FASTLOAD utility statement 20-3—20-6  
FIELDNAME-EQ, in logical record SELECT clause 14-44  
FIELDNAME, in logical record SELECT clause 14-44  
figurative constant, in VALUE clause 13-49  
FILABL SYSIDMS parameter H-22  
file override specification 6-75—6-77  
FILE statement 6-82—6-88  
FILE\_BUF parameter H-11  
FILENAME SYSIDMS parameter H-22  
files 16-3—16-12

- access method, modification 25-10—25-11
- access to native VSAM 16-6
- access to VSAM database files 16-5
- accessing 16-3
- adding and dropping 25-14
- blocks 6-38
- CA-IDMS/DB access 16-5
- characteristics 16-8—16-9
- creating 16-7
- data set name 3-10
- device types 16-7
- disk devices 16-7
- dropping 6-87, 6-88
- dynamic file allocation 6-87
- formatting 16-10
- input load file 21-10—21-11
- journal 4-9
- journal, block size 6-101
- maximum page size 16-7
- native VSAM 16-11
- preallocated, defining 6-88
- specifications 6-34—6-35
- types 16-4

FILETYPE SYSIDMS parameter H-22  
FILLER element 13-47, 14-26  
FIND command

- DML restriction 14-23, 14-29
- specifying database procedures for 13-33
- with indexed record 36-18

FIND, in path group

- CURRENT options 14-48
- EACH/EACH PRIOR 14-52
- FIRST/LAST/NEXT/PRIOR 14-52
- OWNER 14-51
- using indexed set 14-50
- WHERE CALCKEY = 14-47
- WHERE DBKEY = 14-49
- WITHIN SET WHERE SORTKEY = 14-53

FINISH, specifying database procedures for 13-17  
FIRST

- DUPLICATES option for CALC record types 13-28
- DUPLICATES option for sorted sets 13-85
- in path-group FIND/OBTAIN 14-47, 14-52—14-54
- set order 13-76, 36-12

fixed decimal data 13-57  
fixed-compressed record 13-31, 13-39, 34-11

- minimum coding requirements 13-41

fixed-length record 13-39, 13-41  
FIXED, VSAM record length specification 13-31  
footer 33-5  
FOR PROGRAM clause, to transfer subschema

- statistics 14-9

foreign key, control length 13-87  
format control statements

- EJECT statement 9-18
- SKIP statement 9-18

FORMAT utility statement

- purpose 33-12
- to erase table rows 28-8

formatting files 16-10  
FORTRAN

- element names 13-55
- record names 13-37

fragmentation 13-39  
fragmented record 13-31, 34-11  
full load, SQL-defined database 21-6, 21-13

## G

GENERATE statement 14-62

- procedure 8-24

GET

- DML restriction 14-23
- in path group 14-55
- specifying database procedures for 13-33

group item, defined 13-56

## H

header 33-4  
HEADER, for batch listings 10-22  
headers

- CA-CULPRIT 10-25, 13-11
- CA-OLQ 10-25, 13-11
- in compiler listing 10-22

HIGHEST version 10-20  
home page 34-11

- 
- I/O, reducing 17-9—17-11, 22-15
  - IDD record
    - in logical records 14-34
    - sharing the structure of an 13-38
  - IDD source module
    - inclusion in DDL input 10-14
  - IDMS
    - See* CA-IDMS
  - IDMS buffer storage 17-5
  - IDMS statistics block 15-8, 15-11
  - IDMSAJNX user exit 18-12
  - IDMSCHEM compiler 8-6, E-7, E-10, E-20, E-23
  - IDMSCOMP procedure 13-31—13-33, 13-39, 15-5
  - IDMSCPLX user exit 18-12
  - IDMSDCOM procedure 13-31—13-33, 13-39, 15-5
  - IDMSDIRL utility program 23-9
  - IDMSJNL2 user exit 18-12
  - IDMSLBLE procedure E-11
  - IDMSNTWK schema 23-9
  - IDMSNVLR database procedure 15-6
  - IDMSQSAM SYSIDMS parameter H-11
  - IDMSRPTS utility program 24-12
  - IDMSTBLU utility program 30-8
  - IDMSUBSC compiler 8-6, E-12, E-21, E-25
  - IF NOT, in path group 14-55
  - IF, in path group 14-55
  - implicit page locks 37-19
  - implicit record locks 37-18
  - IN ERROR status 11-7
  - INCLUDE clauses
    - class-name 13-10, 14-11
    - MEMBER 13-81
    - RECORD SYNONYM 13-27
    - USER 13-9, 14-10
  - INCLUDE statement 10-14—10-15
  - incremental lock acquisition mode 37-17
  - index
    - definition of 36-5
    - levels 36-8
    - pointer 13-81
    - set mode 13-77
    - spawning and splitting 36-8, 36-14—36-15
    - structure of 36-5—36-11
  - index entry
    - for sorted set 36-7
    - for unsorted set 36-7
    - number of 36-6
  - INDEXED BY clause (COBOL) 13-52
  - indexed elements, COBOL 13-52
  - indexed set
    - changing to chained 31-18
    - compressed entries 13-84
    - connecting records to 36-11—36-15
    - DBKEY as sort control element for 13-84
    - defining 36-3
    - disallowed specifications for 13-77
    - disconnecting records from 36-15
    - end.disallowed specifications for 13-82
    - location mode for 13-28
    - member 13-81
    - mode 13-77
    - notation 36-4
    - owned by system record 13-79
    - pointer defaults 13-78, 13-81, 13-86
    - pointers 13-81, 36-4
    - purposes of 36-3
    - reordering 31-21
    - retrieving indexed records 36-16—36-18
    - set order 36-3, 36-12
    - sorted 36-7, 36-14—36-15
    - sorted retrieval 36-16
    - structure of 36-5—36-11
    - types 36-3
    - unsorted 36-7, 36-11—36-14
  - indexes
    - See also* system-owned indexes
    - changing 29-5
    - creating 29-4
    - defining 7-9
    - displacement 6-33—6-34
    - dropping 29-4
    - dropping default indexes 7-12
    - dropping table's default 28-12
    - moving 29-5
    - specifications 6-33—6-34
    - statistics, monitoring 22-8—22-9
    - unique 7-9
    - unlinked 13-86
  - indexes, non-SQL
    - adding/deleting pointers 31-25
    - changing characteristics 31-25
    - changing the index area 31-24
  - INPUT COLUMNS clause 10-22
  - input format
    - non-SQL schema and subschema compilers 9-9
    - specifying columns for 10-22
  - input range 10-22
  - insertion options 13-83
-

---

installation defaults  
  for session options (table) 10-29  
  online compiler task codes 9-4  
intent locks 37-14  
isolation levels 37-8  
ITERATE, in path-group ON 14-57

## J

JCL  
  BS2000/OSD E-23  
  CMS commands E-20  
  OS/390 E-7  
  VSE/ESA E-10  
journal buffer  
  defining 6-89—6-92  
  definition 4-9  
  dropping 6-91  
  number of pages 4-11  
  page size 4-10, 6-90  
  writes to files 18-5  
JOURNAL BUFFER statement 6-89—6-92  
journal files  
  device types 16-7  
  disk devices 16-7  
  for the runtime environment 2-5  
  record types 18-5  
  types 4-9  
  under the central version 18-3  
journal fragment interval 18-14  
journal record entries 18-5  
JOURNAL SYSIDMS parameter H-11  
journaling  
  archive journal block size 6-25, 6-80  
  ARCHIVE JOURNAL utility statement 18-9—18-11  
  archive journals 6-22—6-25  
  block size 6-23  
  changing the disk journal file size 25-15  
  journal file, incomplete 19-40  
  local mode 18-4  
  modifications 25-7  
  multiple archive journals 6-24  
  offloading 18-9—18-11  
  performance 18-13—18-15  
  procedures 18-3—18-16  
  record types 18-5  
  reports 18-12  
  requirements 6-24  
  statistics 22-5—22-6  
  to disk device 4-11, 6-63—6-66, 19-40  
  to tape device 6-99—6-102, 19-40

journaling (*continued*)  
  under the central version 18-3  
  user exits 18-12  
JREPORTs 18-12  
JRNLDTS SYSIDMS parameter H-11  
JSEG journal record 18-6  
JUSTIFY RIGHT clause (schema elements) 13-51

## K

KEEP  
  DML restriction 14-23, 14-29  
  in path-group FIND/OBTAIN 14-47, 14-54  
  path-group DML command 14-56  
KEYLEN, VSAM buffer pool specification 6-45  
KEYWORD in logical record SELECT clause 14-45  
keywords, defined 9-12  
KSDS  
  CALC keys 13-29  
  database key construction 6-37  
  DUPLICATES option 13-29  
  location mode 13-29  
  set mode 13-77

## L

LANG SYSIDMS parameter H-12  
LAST  
  DUPLICATES option for CALC record types 13-28  
  DUPLICATES option for sorted sets 13-85  
  in path-group FIND/OBTAIN 14-52  
  set order 13-76, 36-12  
LEADING sign placement for element 13-51  
LENGTH\_PAGE SYSIDMS parameter H-12  
level-88 item 13-48—13-59  
level-number clause (schema elements) 13-47  
line index 33-5  
line space count 33-5, 33-11  
linked constraints 29-7  
linked index constraint, order 36-3  
LINKED TO OWNER for set member 13-82  
LINKED TO PRIOR in chained sets 13-77  
LIST 10-22  
LIST SYSIDMS parameter H-12  
listings from compilers  
  contents of 9-18  
  format control statements 9-18  
  to reports on schema/subschema definitions 8-32  
literal, in VALUE clause 13-49  
load area 23-3

---

load module

- 24-bit mode 14-66
- at runtime 9-17
- automatic deletion 10-21, 13-12
- making available to runtime system 24-25
- migrating 24-11
- object module addressing 14-65
- residency mode 14-65
- storing 9-17
- subschema 13-93, 14-62, 14-63
- version 13-93, 14-62

LOAD MODULE statement 14-63—14-66

- module residency mode 14-65
- name 14-64
- object module address mode 14-65

LOADAREA SYSIDMS parameter H-12

local mode 1-4—1-5

- buffers 17-3
- compiling batch non-SQL DDL E-4
- DMCL 2-4
- executing SQL DDL 7-4
- handling of physical area lock 37-12
- journaling 2-5, 18-4
- recovery 19-40—19-41
- runtime components 23-21
- session defaults 23-24

LOCAL SYSIDMS parameter H-12

LOCAL\_DYNAMIC\_ALLOCATION SYSIDMS parameter H-12

LOCAL\_NOJOURNAL\_RETRIEVAL SYSIDMS parameter H-12

LOCALPUR SYSIDMS parameter H-13

location mode

- changing 31-12

LOCATION MODE clause

- in record display 13-34
- schema record specification 13-27

lock acquisition mode 37-16

lock management 37-3—37-27

- area lock status 37-7
- area ready modes 37-4—37-10
- deadlock detection interval 37-26
- deadlocks 37-25—37-27
- for SQL access 37-8
- isolation levels 37-8
- lock compatibility table 37-15
- native VSAM considerations 37-24
- page locks 37-19
- physical area locks 37-11—37-12
- record locks 37-18
- statistics, monitoring 22-9—22-14

lock management (*continued*)

- under the central version 37-13

locks, intent 37-14

locks, logical

- and area ready modes 37-15
- compatibility table 37-15
- modes 37-14

logical and physical database separation 2-7

logical database definition 1-8

logical end-of-file indicator 10-22

logical record

- about 8-21—8-22
- access restrictions 14-58
- adding/modifying/deleting 32-9
- database records in 14-34
- definition procedure 8-21
- dictionary records in 14-34
- documenting 14-36
- error detection in 14-34, 14-35
- in program variable storage 14-58
- name 14-33
- path group 14-38
- ready mode for 14-8
- record role in 14-34
- when to modify 14-35

logical record elements

- defining 14-33
- sequence in program storage 14-35

logical record facility (LRF)

- securing the subschema 14-15

LOGICAL RECORD statement 14-32—14-37

logical-record field name, in path group 12-9

logically-deleted records 34-18

LONG-POINT element usage 13-50

LOWEST version 10-20

LR CURRENCY clause 14-9

LR subschema usage mode 14-8

LSR, VSAM buffer pools 6-45

## M

MAINTAIN INDEX utility statement 30-5, 30-8

major command user exit G-4

MANDATORY set removal option 13-83

mantissa, in PICTURE clause 13-57

MANUAL set removal option 13-83

maps, migrating 24-6, 24-22

mask, in boolean expression 12-6

master terminal commands

- See* DCMT commands

---

---

MATCHES option of boolean expression 6-14, 10-6, 12-6

maximum records per page 3-7

MEMBER clause 13-81

MEMO DATE clause 13-8

message area 23-3

messages, compiler display 10-23

migration

- components 24-5—24-9
- components, identification methods 24-12—24-14
- considerations 24-25
- facilities 24-11
- procedures 24-4—24-11, 24-17—24-24
- sequence 24-9—24-10
- task application table 24-26
- technique for SQL definitions 24-23
- tools 24-15—24-16

MINIMUM FRAGMENT clause

- (figure) 13-40
- applied to fixed-length records 13-39
- compressed records 13-40
- default 13-40
- example 13-40
- schema specification 13-31

minimum fragment length, changing 31-15

MINIMUM ROOT clause

- (figure) 13-40
- applied to fixed-length records 13-39
- compressed records 13-40
- default 13-40
- example 13-40
- schema specification 13-31

minimum root, changing 31-15

MIXED subschema usage mode 14-8

MODE clause (sets) 13-77

mode, 24-bit 14-66

MODIFY operation 11-5

- allowed/disallowed for a user 12-13
- DML restriction 14-23
- effect on non-SQL schema 13-12
- effect on records 13-35
- effect on sets 14-30
- effect on subschema 14-13
- for public access 13-9, 14-10
- in path group 14-56
- path group 14-43
- specifying database procedures for 13-33

monitoring

- access modules 22-14
- buffer statistics 22-6—22-7
- facilities 22-4

monitoring (*continued*)

- I/O 22-15—22-16
- index efficiency 22-8—22-9
- journal statistics 22-5—22-6
- locking 22-9—22-14
- schedule 22-3
- space management statistics 22-7—22-8
- SQL processing 22-14

MSGDICT SYSIDMS parameter H-13

MULTIDSN SYSIDMS parameter H-14

multiline input, for non-SQL schema/subschema compilers 9-9

multiply-occurring elements 13-50—13-51

- using subscripts for 12-9

## N

naming conventions

- physical database statements 6-7—6-8

native VSAM file D-9

- accessing 16-6
- buffer pool specification 6-45
- considerations D-3—D-9
- data set structure D-4
- definition 16-11
- disallowed specifications for 13-77—13-79, 13-81—13-85
- DML functions D-8—D-9
- location mode 13-29
- lock management 37-24
- record type 13-31
- recovery 19-47
- restrictions 6-39
- set duplicates option 13-85
- set insertion option 13-83
- set member 13-81
- set mode 13-77
- set order 13-77, 13-83
- set pointer defaults 13-78, 13-81, 13-86
- set removal option 13-83
- variable-length record 15-6

NEXT

- in path-group FIND/OBTAIN 14-47
- in path-group ON 14-57
- pointer 13-77
- set order 13-76, 36-12

NEXT HIGHEST version 10-21, 13-8

NEXT LOWEST version 10-21, 13-8

NO ECHO clause 10-21

NO HEADER, for batch listings 10-22

---

NO LIST 10-22  
 NO PROMPT, for TTY devices 10-23  
 NO RESET logical record currency 14-10  
 NOCLEAR, logical record variable-storage 14-34, 14-35  
 node, specifying for compilation 10-35  
 NODENAME SYSIDMS parameter H-14  
 NODENAME, in JCL E-4  
 non-SQL database definition  
   procedure 8-3—8-33  
   sample C-3  
   segment planning 3-5  
 non-SQL DDL statements  
   =COPY facility E-10  
   coding 9-7  
   components 9-7  
   end of statement delimiter 9-8  
   option delimiters 9-8  
   required delimiters 9-8  
 non-SQL defined databases  
   access through SQL application 5-5  
   modification methods 30-4  
   modification procedure 30-5  
   types of modifications 30-3  
 non-SQL schema  
   changing schema characteristics 31-5  
   changing the definition of 30-4  
   compiler 8-6  
   compiler listings 8-32  
   compiling, batch E-4  
   components 8-7  
   currency 8-30—8-31, 13-4  
   definition 8-7—8-17  
   deleting 31-5  
   dictionary 23-9  
   modification procedure 30-4  
   modifying when empty 31-4  
   modifying when not empty 31-5—31-25  
   name 13-7  
   native VSAM considerations D-5  
   sample definition C-3  
   security checking 8-25—8-29  
   validation 13-92  
   version 13-7  
 non-SQL schema and subschema compilers  
   batch compiling 9-6  
   coding comment text 9-14—9-15  
   coding entity-occurrence names 9-12  
   coding input non-SQL DDL statements 9-7—9-11  
   coding keywords 9-12  
   coding user-supplied values 9-13  
   comments 9-8  
   non-SQL schema and subschema compilers (*continued*)  
     contents of listings 9-18  
     ending a session 9-4  
     error handling 9-10—9-11  
     format control for listings 9-18  
     input format 9-9  
     load modules generated 9-17  
     output 9-17  
     recovering a session 9-4  
     source generated 9-17  
     starting a session 9-4  
   non-SQL schema compiler 1-13  
     automatic load module deletion 13-12  
     batch execution E-4  
     BS2000/OSD JCL E-23  
     CMS commands E-20  
     compiler-directive statements 10-3—10-37  
     copying source code into E-10  
     OS/390 JCL E-7  
     session options 10-16  
     status conditions 11-7  
     VSE/ESA JCL E-10  
   non-SQL schema DDL  
     all entity occurrence display 10-7  
     area 13-15  
     IDD entity display options 10-11  
     record 13-21  
     SCHEMA statements 13-4  
     schema validation 13-92  
     set 13-72  
     subschema regeneration in 13-93  
   non-SQL SCHEMA statements 13-4—13-14  
     order of presentation 13-3  
     syntax 13-7  
 NONE  
   compiler operations for public access 13-10, 14-10  
 NONE, as user responsibility 12-14  
 nonnumeric literal, in VALUE clause 13-49  
 nonshared resource (NSR) buffer pools 6-45  
 NONSPANNED, VSAM control interval  
   specification 13-31  
 NOT ALLOWED  
   DUPLICATES option for CALC record types 13-28  
   DUPLICATES option for sorted sets 13-85  
   DUPLICATES option for VSAM CALC record  
     types 13-29  
   for DML functions 14-18, 14-23, 14-29  
 NSR, VSAM buffer pools 6-45  
 NULL  
   for default area ready mode 14-19  
   for non-SQL schema comments 13-10

---

---

NULL (*continued*)

- for record fragment length 13-32
- for record root length 13-32
- for schema comments 14-11
- for subschema record priority 14-24
- for VSAM file device types 13-31
- null string, in non-SQL DDL statements 9-14
- null values, loading 21-11
- null-lock lock mode 37-14
- numeric data 13-49
- numeric edited data 13-58—13-60
- numeric literal, in VALUE clause 13-49

## O

- OBTAIN, in path group
  - considerations 14-57
  - CURRENT options 14-48
  - EACH/EACH PRIOR 14-52
  - FIRST/LAST/NEXT/PRIOR 14-52
  - OWNER 14-51
  - syntax 14-43
  - using indexed set 14-50
  - WHERE CALCKEY = 14-47
  - WHERE DBKEY = 14-49
  - WITHIN SET WHERE SORTKEY = 14-53
- OCCURS clause (schema elements) 13-50
- OCCURS DEPENDING ON clause 13-61
- OCCURS DEPENDING ON clause (schema elements) 13-51
- OF SCHEMA clause
  - in ADD/MODIFY/DELETE operations 13-16, 13-76, 14-7, 14-9
  - in COPY ELEMENTS substatement 13-69
  - to qualify areas 13-16
  - to qualify records 13-69
  - to qualify sets 13-76
  - to qualify subschema 14-7, 14-9
- OFFLINE area status 37-7
- OFFSET clause 13-38
- offsets 6-31—6-33, 13-38, 13-87
- OLQ HEADER clause
  - in ELEMENT substatement 13-53
  - in non-SQL schema display 13-11
- OLQ HEADERS clause 10-25
- ON clause
  - automatic generation of 14-56
  - automatic generation of (table) 14-59
  - in path group 14-56
  - in path group (table) 14-59

- ON LR-ERROR clause 14-34
- ON LR-NOT-FOUND clause 14-35
- ON-ERROR procedure 15-20
- ONLINE area status 37-7
- online compilation
  - installation default task codes for 9-4
  - prompt for TTY devices 10-23
  - redisplay of input 10-21—10-23
- OPSYS buffer storage 17-5
- optimization of subschema tables
  - PRIORITY clause 14-26
- optional clauses, defined 9-7
- OPTIONAL set removal option 13-83
- ORDER clause (sets) 13-76
- orphan count 36-13
- OS/390 JCL
  - non-SQL schema compiler E-7
  - subschema compiler E-8
- OUTPUT LINE SIZE clause 10-23
- OVERPRINT SYSIDMS parameter H-14
- OWNER clause
  - in ADD/MODIFY/DELETE SET statement 13-79
  - schema specification 13-78
- OWNER pointer 13-82

## P

- PACKED element usage 13-50
- page
  - empty 33-13
  - home 34-11
  - layout 33-5
  - location of records 33-5
  - maximum number of records on 6-96
- page footer 33-5
- page groups
  - assigning 6-96
  - definition 3-6
  - for dictionary segments 23-7
  - when to use 3-6
- page header 33-4
- page locks 37-19
- page number 33-4
- page ranges
  - defining 3-6, 6-36
  - extending 6-36, 25-13
- page reserve 3-8, 34-13
  - about 34-13
  - area overrides 4-13
  - changing 4-13
  - for database load 20-4, 21-7

- 
- page size
    - buffer 6-44
    - calculating 33-13
    - increasing 25-12
    - journal buffer 4-10
  - pages
    - for displacement of index keys 13-78
    - for displacement of VIA set members 13-29
    - for record placement within area 13-30—13-80
  - parameter expansion
    - boolean-expression 12-4
    - conditional expression 6-12, 10-4
    - db-record-field 12-8
    - lr-field 12-9
    - mask comparison 6-12, 10-4
    - module-specification 12-10
    - user-options-specification 12-13
    - user-specification 12-12
    - value comparison 6-12, 10-5
    - version-specification 12-15
  - PARM SYSIDMS parameter H-14
  - PASSWORD clause 12-12
  - password, when to specify 13-13
  - PATH (VSAM)
    - CALC keys 13-29
    - set mode 13-77
  - path group
    - adding/modifying/deleting 32-9
    - boolean expression in 12-4
    - considerations 14-57
    - database record field name in 12-8
    - database record name in 14-58
    - logical-record field name in 12-9
  - PATH-GROUP statement 14-38—14-60
    - definition procedure 8-22
    - required use of role names 14-36
    - terminating 14-58
  - PERCENT, for record placement within area 13-30—13-31, 13-80
  - percentage offsets 13-38, 13-87
  - period
    - as decimal point 10-20, 12-4, 13-58
    - in non-SQL DDL statements 9-7
  - PERMANENT, in path-group ERASE 14-46
  - phased load, SQL-defined database 21-13
  - physical database
    - See also* areas, physical
    - See also* buffers
    - See also* DMCL
    - See also* files
    - See also* segments
  - physical database (*continued*)
    - areas 3-4
    - buffers, database 4-8
    - buffers, journal 4-9—4-11
    - data set name 3-10
    - database files 3-3
    - database name table 2-5, 5-3—5-15
    - defining 3-3
    - definition 1-8, 2-3
    - DMCL 2-5, 4-3
    - DMCL, central version 4-4
    - journal 4-12
    - journal files 4-9—4-12
    - journals, disk 4-11
    - limits 6-103
    - page groups 3-6
    - records per page 3-7
    - sample A-3
    - segments 4-13
    - segments, defining 3-12—3-14
    - statement summary 6-5
    - symbolics 3-8—3-9
  - physical database statements
    - ARCHIVE JOURNAL 6-22—6-25
    - AREA 6-26—6-42
    - BUFFER 6-43—6-48
    - currency 6-8
    - DBGROUP 6-49—6-51
    - DBNAME 6-52—6-57
    - DBTABLE 6-58—6-62
    - DISK JOURNAL 6-63—6-66
    - DISPLAY/PUNCH 6-10
    - DMCL 6-67—6-81
    - FILE 6-82—6-88
    - JOURNAL BUFFER 6-89—6-92
    - keywords 6-6
    - naming conventions 6-7—6-8
    - SEGMENT 6-93—6-98
    - separators 6-6
    - statement summary 6-3
    - TAPE JOURNAL 6-99—6-102
    - values 6-6
    - verb synonyms 6-6
  - physical definitions
    - access method, changing 25-16
    - area size, increasing 25-12
    - DMCL, dynamic management 25-8—25-9
    - file access method, changing 25-10—25-11, 25-16
    - files, adding or dropping 25-14
    - journal file, changing the size 25-15
    - journal modifications 25-7
-

---

physical definitions (*continued*)  
  modifying 25-3—25-17  
  page range, extending 25-13  
  page size, increasing 25-12  
physical device blocking 6-37—6-38  
physical sequential retrieval 36-16  
PICTURE clause (schema elements) 13-48, 13-59  
  (table) 13-60  
PICTURE formats for data 13-56—13-58  
PL/I  
  element names 13-55  
  record names 13-37  
pointer assignments 13-85, 13-86  
POINTER element usage 13-50  
pointer positions 13-86—13-87  
pointers  
  *See also* database key  
  adding/dropping 31-19  
  changing 13-87  
  resolved by VALIDATE 13-92  
preclaim lock acquisition mode 37-17  
PREFETCH SYSIDMS parameter 17-10, H-14  
PREFETCH\_BUF SYSIDMS parameter H-14  
prefix 33-5  
  compression 36-8  
  length 33-5  
PREPARED BY clause  
  in ADD/MODIFY/DELETE operations 12-12, 13-8,  
  14-8  
  populated by SIGNON 10-23  
  setting the session default for 10-23  
  when to use 12-11  
PRINT SPACE utility statement 33-14  
PRIOR  
  in path-group FIND/OBTAIN 14-53  
  pointer 13-77—13-81  
  set order 13-76, 36-12  
PRIORITY clause (subschemas records) 14-23  
privacy/security options  
  *See* lock management  
procedure control block 15-8  
PROCEDURE NAME clause 13-33  
procedures  
  *See* database procedures  
PROCTRACE SYSIDMS parameter H-15  
production environment 1-12  
program authorization 14-8  
program pools  
  determined by residency mode 14-66  
PROGRAM REGISTRATION clause 14-8

program view of subschema 14-22—14-23  
  (figure) 14-24  
programs  
  associated with a modified subschema 32-3  
  communication with procedures 15-15  
  readying areas 14-19  
  recompiling after subschema modification 32-3  
  transferring statistics 14-14  
PROMPT, for TTY devices 10-23  
protected ready modes 37-4  
PROTECTED, area ready mode  
  compiling in 10-35  
  restricting DML programs from using 14-18  
  setting as DML default 14-19  
PUBLIC ACCESS clause  
  assigning to a user 12-13  
  automatic changes in 14-15  
  clause 13-9, 13-11  
  syntax 14-10  
PUNCH operation  
  effect on load modules 14-66  
  location of output 11-8  
  setting the session defaults for 10-23

## Q

QSAMAREA SYSIDMS parameter H-15  
QSAMBUF# SYSIDMS parameter H-15  
QSAMTRACE SYSIDMS parameter H-15  
quotation marks  
  in comments 9-13, 13-53  
  in expressions 9-13  
  in user text 9-13  
  setting the character for 10-24  
  using 9-13  
QUOTE clause 10-24

## R

RCM  
  *See* relational command module (RCM)  
read driver 17-11  
READ ONLY transaction state 37-8  
READ WRITE transaction state 37-8  
reads, chained 17-9  
READY  
  restricting for DML programs 14-18  
  specifying database procedures for 13-17  
  specifying database procedures for (table) 13-18  
  specifying defaults for 14-19

---

ready mode

- defaults for subschema areas 14-20
- restricting for DML programs 14-18
- setting default for DML programs 14-19
- subschema 14-8

ready modes

- See also* area ready modes
- and logical locks 37-15
- area 37-4—37-10
- default 37-8
- specifying database procedures for 13-17
- specifying database procedures for (table) 13-18
- specifying for dictionary 10-35

record

- chained sets
- connecting to chained set 35-5—35-6
- connecting to indexed set 36-11—36-15
- defining 35-3
- disconnecting from chained set 35-6—35-7
- disconnecting from indexed set 36-15
- erasing 34-16—34-19
- fixed-length compressed 34-11
- fragment 34-11—34-12
- logical deletion 34-18
- physical deletion 34-16
- relocated 34-14—34-15
- retrieving from chained set 35-7—35-8
- retrieving from indexed set 36-16—36-18
- root 34-11
- storing 34-3—34-15
- variable-length 34-11—34-14

record (non-SQL schema)

- assigning to an area 13-30, 13-79
- compressed 13-31—13-33, 13-39
- copying 13-26
- examples of definition 13-41—13-43
- fixed-compressed 13-39
- fixed-length 13-39, 13-41
- location mode 13-27
- modifying schema-built records 13-35
- modifying size 13-39
- name 13-24
- observing language conventions 13-36
- prefix 13-78, 13-81—13-82, 13-89
- structure 13-21, 13-24
- unused 13-35
- using synonyms 13-37
- variable-length 13-31—13-39, 13-51
- variable-length (figure) 13-40

record (subschema)

- access restrictions 14-23

record (subschema) (*continued*)

- priority 14-23
- view 14-22—14-23
- view (table) 14-26

record control block 15-8, 15-14

record deletion, logical 34-18

record deletion, physical 34-16

record description

- ELEMENTS and VIEW ID clauses 14-25

record elements

- modifying 31-14

RECORD entity type

- compression/decompression procedure 13-33
- data characteristic table 13-32

RECORD ID clause

- in record display 13-34
- non-SQL schema specification 13-26

record IDs

- assigning 13-26
- changing 31-15
- in line index 33-5

record length

- calculating 33-5
- in MINIMUM FRAGMENT LENGTH clause 13-32
- in MINIMUM ROOT LENGTH clause 13-32

record locks 37-18

- on subschema record 14-47—14-54
- on subschema records 14-56

record occurrence

- components 33-5
- on database page 33-5

record occurrence block 15-8, 15-15

record procedures 15-8

RECORD statement (non-SQL schema) 13-21—13-43

- clauses required for ADD 13-42
- COPY ELEMENTS substatement 8-11
- definition procedure 8-9—8-15
- ELEMENT substatement 8-12
- OFFSET clause 13-38
- SHARE DESCRIPTION clause 8-11
- SHARE STRUCTURE clause 8-10

RECORD statement (subschema) 14-21—14-27

- definition procedure 8-20
- syntax 14-22

record synonyms

- changing 31-15

record-descriptor word (RDW) 15-6

records per page 3-7

records, non-SQL schema

- adding 31-8
- changing data compression 31-15

---

---

records, non-SQL schema (*continued*)

- changing the area 31-13
- changing the CALC key 31-9
- changing the location mode 31-12
- deleting 31-8

records, subschema

- adding/modifying/deleting 32-6

recovery

- central version 19-14
- definition 19-3
- due to system failure 19-14
- due to transaction failure 19-16
  - end.manual 19-46
- from database file I/O error 19-33—19-36
- from journal file I/O error 19-37—19-39
- journal file, incomplete 19-40
- journaling to disk device 19-40
- journaling to tape device 19-40
- local mode 19-40—19-41
- mixed mode 19-42—19-46
- native VSAM files 19-47
  - start.manual 19-18
- warmstart 19-14
  - when warmstart fails 19-31

recovery unit 18-5

REDEFINES clause (schema elements) 13-47

referential constraint

- changing tuning characteristics of 29-8
- creating 7-10, 29-7
- dropping 29-7

REGENERATE statement 13-93—13-94

- effect on subschemas 13-93
- syntax 13-93

regeneration

- of a subschema after modification 32-3
- using the schema compiler 13-93
- using the subschema compiler 13-93

REGISTERED FOR clause 12-13, 14-14, 14-16

registration

- for all operations 14-16
- for an operation 14-15
- of user 14-14
- program 14-8
- replacing 14-14

REGISTRATION OVERRIDE clause 10-24

registration override security 8-26

relational command module (RCM), migrating 24-6

RELOAD utility statement 30-5, 30-7

relocated record 34-14—34-15

removal options 13-83

repeating character compression 36-8

repeating data items 13-50—13-51

reports, journaling 18-12

REREAD\_SYSCTL SYSIDMS parameter H-16

RESET logical record currency 14-9

residency mode 14-66

RESPONSIBLE FOR clause 12-14, 14-14

restricting DML programs

- area ready modes 14-18
- record access 14-23
- set access 14-29

restricting records to page ranges 13-30

restructure

- identifying base schema for 13-9

RESTRUCTURE SEGMENT utility statement 30-5, 30-7

retrieval

- physical sequential 36-16
- random 36-16
- sorted 36-16
- unsorted 36-16

RETRIEVAL area status 37-7

retrieval ready mode 37-4

RETRIEVAL, area ready mode

- compiling in 10-35
- restricting DML programs from using 14-18
- setting as DML default 14-19

RETURN

- specifying database procedures for 13-33

RETURN command 36-18

- in path-group ON 14-57

REVISED BY clause

- in ADD/MODIFY/DELETE operations 12-12, 13-8, 14-8
- populated by SIGNON 10-23
- setting the session default for 10-23

REWIND SYSIDMS parameter H-22

RHDCMAP1 mapping compiler 24-22

RHDCMPUT utility program 24-22

RMODE clause 14-65, 14-66

ROLE clause 12-9, 14-34, 14-45, 14-52, 14-58

rollback, automatic

- due to transaction failure 19-16

ROLLBACK, specifying database procedures for 13-17

ROLLBACK3490 SYSIDMS parameter H-16

root, of record 34-11

rows of tables, estimating 7-6

RPG II

- element names 13-55
- record names 13-37

---

---

RRDS  
 location mode 13-29

RTSV journal record 18-6

runtime  
 database name table 6-61  
 session options 23-23—23-24

runtime system  
*See also* DMCL  
 identifying the DMCL 4-5

**S**

SAM file access method 16-5

SAME AS clause 8-8  
 area 13-16, 13-19  
 set 13-76, 13-87

Schema Compiler Activity List  
 specifying the width of 10-23  
 suppressing the header on 10-22

schema compiler, non-SQL  
*See* non-SQL schema compiler

SCHEMA DESCRIPTION clause 13-8

SCHEMA statements  
 definition procedure 8-7

schema-built records  
 modifying 13-35

schema, SQL  
*See* SQL schema

security  
 and CA-IDMS 1-9  
 non-SQL schema 13-9  
 overriding 10-24  
 registration override 8-26  
 subschema 14-10, 14-15  
 through IDD user exits G-3

security checking  
 non-SQL schema and subschema compilers 8-25

SEGMENT statement 6-93—6-98

segmented load, SQL-defined database 21-6, 21-15

segments  
 defining 3-12—3-14  
 definition 2-3, 3-3  
 dictionary 23-5  
 in central version DMCL 4-13  
 in local mode DMCL 4-13  
 planning 3-5  
 specifying in database name table 5-5  
 using area overrides 4-13  
 using file overrides 4-13

SELECT clause (logical-record path groups) 14-43  
 considerations 14-57

SELECTIVE, in path-group ERASE 14-46

semicolon, in non-SQL DDL statements 9-8

SEPARATE CHARACTER sign placement for  
 element 13-51

SEQUENCE clause 10-24

session options  
 displaying 10-27  
 runtime 23-23—23-24  
 setting 10-16

session options, installation defaults (table) 10-29

set  
 access restrictions 14-29  
 automatic deletion of 13-36, 13-85, 14-25, 14-30  
 examples of definition 13-88—13-91  
 explicit deletion of 13-85, 14-30  
 indexed 36-18  
 insertion options 13-83  
 linkage 13-78, 13-81  
 member 13-81  
 mode 13-77  
 order 13-76, 36-12  
 owner 13-79  
 pointer defaults (table) 13-86  
 pointers 13-78, 13-81—13-82, 31-19  
 qualification 13-76  
 removal options 13-83

set membership options  
 changing 31-21

set modes, changing 31-18

SET OPTIONS  
 FOR DISPLAY/PUNCH (table) 10-31

SET OPTIONS statement 10-16—10-32  
 available options 10-16  
 default values 8-7

DELETE clause 13-12  
 installation defaults for (table) 10-29  
 syntax 10-16

SET statement (non-SQL schema) 13-72—13-91  
 ADD/MODIFY/DELETE syntax 13-72  
 clauses required for ADD 13-85, 13-86  
 definition procedure 8-15  
 SAME AS clause 8-16

SET statement (subschema) 14-28—14-31  
 definition procedure 8-20

sets  
*See also* chained sets  
*See also* indexed sets  
 deleting records 8-15

sets, non-SQL schema  
 adding/deleting 31-17  
 changing membership options

---

---

sets, non-SQL schema (*continued*)

- changing the mode 31-18
- changing the order 31-20

sets, subschema

- adding/modifying/deleting 32-7

SHARE clause

- record specification 13-24
- SHARE DESCRIPTION 13-26
- SHARE STRUCTURE 13-38

SHARE DESCRIPTION clause

- difference from SHARE STRUCTURE clause 13-38
- position of clause 13-38

share lock mode 37-14

SHARE STRUCTURE clause

- difference from SHARE DESCRIPTION clause 13-38
- in non-SQL schema display 13-11

shared ready modes 37-4

shared record locks 37-18

SHARED, area ready mode

- compiling in 10-35
- restricting DML programs from using 14-18
- setting as DML default 14-19

SHORT-POINT element usage 13-49

SIGN clause (schema elements) 13-51

SIGNOFF statement 10-33

SIGNOFF user exit G-4

SIGNON G-8

- block G-8
- element block G-7
- user exit G-4

SIGNON statement 10-34—10-37

- security for 10-34
- syntax 10-34

SKIP format statement 9-18

SMP

- See* space management page (SMP)

sort control element 13-83

sort element name 13-83

SORT keys

- changing the DUPLICATES option 31-11

SORTED set order 13-77, 13-83, 36-16

sorted sets 13-83

SORTSIZE SYSIDMS parameter H-16

source statements, appending 12-11

space available count 33-5, 33-11

space management 33-3

- statistics, monitoring 22-7—22-8

space management entry 33-12

space management page (SMP) 33-15

- space management page (SMP) 33-12
  - use in lock management 37-11
- space, in non-SQL DDL statements 9-8
- SPANNED, VSAM control interval specification 13-31
- spawning, index records 36-8, 36-14—36-15
- splitting, index records 36-8, 36-14
- SQL applications 5-5
- SQL database definition 7-3—7-15
  - migrating entities 24-7, 24-23
  - sample B-3
- SQL DDL
  - embedded in application programs 7-4
- SQL schema
  - creating 7-4
  - dropping 28-4
  - modifying 28-4
- SQL transactions
  - area locks 37-16
  - lock management 37-8
- SQL-defined data
  - segment planning 3-5
- SQL-defined database
  - loading 21-3—21-20
  - modification methods 27-4
  - types of modifications 27-3
- SQL\_INTLSORT parameter H-16
- SQLTRACE SYSIDMS parameter H-16
- SR1 system record F-3
  - definition of F-4
  - location on page 33-4
  - use of 33-11—33-12
- SR2 system record 34-14, F-3
- SR3 system record 34-14, F-3
- SR4 system record 34-12, F-3
- SR5 system record F-3
- SR6 system record F-3
- SR7 system record 36-3, F-3
  - definition of F-4
- SR8 system record F-3
  - currency 36-17
  - definition of F-4
  - format of 36-6
  - orphan count 36-13
  - purpose of 36-5
  - splitting 36-8, 36-14
- SR9 system record F-3
- stamps, synchronization 6-38
- statistics
  - monitoring 22-5—22-14
- status conditions 11-7

---

---

stepped load, SQL-defined database 21-6, 21-16

storage mode

- CALC 34-4—34-7
- CLUSTERED 34-7—34-11
- DIRECT 34-3
- discussion of 34-3
- VIA 34-7—34-11, 36-16

STORE

- specifying database procedures for 13-33

STORE command

- DML restriction 14-23
- in path group 14-57
- path group 14-43
- space management considerations 33-13

STRNO, VSAM buffer pool specification 6-45

sublibrary, using copied code from E-10

subroutines

- See* database procedures

subschema

- access restrictions 14-18, 14-23, 14-29
- compiler 8-6
- compiler listings 8-32
- compiling, batch E-4
- components 8-18
- considerations for modifying 32-3
- currency 8-31, 14-4
- currencys 8-30
- definition 8-18—8-24
- deleting 32-4
- deleting areas 13-19, 32-8
- documenting revisions 14-16
- elements 14-23, 14-33
- generation 13-93, 14-62
- load module 10-21, 13-93, 14-62
- mapping 6-61
- migrating 24-6
- modifying 32-3—32-9
- name 14-7
- qualification 14-7, 14-9
- ready mode 14-8
- record priority 14-23
- regeneration 13-93
- requirements for database load 20-5
- sample definition C-9
- security 14-15
- set, modifying and deleting 32-7
- status 8-23
- storing load modules 9-17
- validation 14-61
- view of record 14-22—14-23
- view of record (figure) 14-24

subschema (*continued*)

- view of record (table) 14-26

subschema compiler 1-13

- BS2000/OSD JCL E-25
- CMS commands E-21
- compiler-directive statements 10-3—10-37
- copying source code into E-10
- OS/390 JCL E-8
- session options 10-16
- status conditions 11-7
- VSE/ESA JCL E-12

Subschema Compiler Activity List

- specifying the width of 10-23
- suppressing the header on 10-22

subschema DDL

- all entity occurrence display 10-7
- area 14-17
- IDD entity display options 10-11
- load module generation in 14-62
- logical record 14-32
- path group 14-38
- record 14-21
- set 14-28
- subschema 14-4
- subschema validation 14-62

SUBSCHEMA DESCRIPTION clause 14-8

subschema load modules

- at runtime 9-17
- storing 9-17

subschema mapping

- See* database name table

SUBSCHEMA statement 14-4—14-16

- definition of program use 14-13
- definition procedure 8-18
- minimum statement 14-15

subschema validation

- after ADD and MODIFY operations 14-61

subschemas

- dictionary 23-9

symbolic key

- compression 36-8
- duplicate 36-7

symbolics 3-8—3-9

- specifications 6-31—6-33
- subareas 6-31—6-33
- symbolic index 6-33

synchronization stamps 3-9

SYNCHRONIZED clause 13-59

SYNCHRONIZED clause (schema elements) 13-50

synonym

- displaying 13-12, 13-35

---

---

synonym (*continued*)  
  element 13-52  
  in shared records 13-25  
  record 13-27  
syntax format  
  for non-SQL schema and subschema compilers 9-7  
SYNTAX, setting DISPLAY/PUNCH default 10-27  
SYS\_MSG SYSIDMS parameter H-17  
SYSCA catalog schema 23-11  
SYSCTL SYSIDMS parameter H-17  
SYSIDMS parameter file 23-21—23-23, E-4  
SYSIDMS parameters  
  ABEND\_ON\_DEADLOCK H-6  
  ABENDTRACE H-6  
  ABENDTRACE\_ENTRIES H-6  
  ABENDTRACE\_SUSCHEMA\_DISPLAY H-6  
  ABENDTRACE\_VIBSNAP H-7  
  AREA\_VALIDATION\_MSGS H-7  
  BLKSIZE H-22  
  BLOCKS H-22  
  BUFFER\_PURGE H-7  
  BUFFERSTAT H-7  
  CVMACH H-7  
  CVNUM H-8  
  CVRETRY H-8  
  DB\_DEADLOCK\_DUMP H-8  
  DBNAME H-8  
  DC\_DEADLOCK\_0029 H-8  
  DC\_DEADLOCK\_NODUMP H-8  
  DC\_SCRATCH H-8  
  DCNAME H-9  
  DEADLOCK\_ABEND\_0029 H-9  
  DEADLOCK\_ABEND\_ERUS H-9  
  DEADLOCK\_DETAILS H-9  
  described 23-22  
  DEVADDR H-22  
  DICTNAME H-9  
  DICTNODE H-10  
  DLBLMOD H-10  
  DMCL H-10  
  DMLTRACE H-10  
  DSGROUP H-10  
  ECHO H-10  
  FILABL H-22  
  FILE\_BUF H-11  
  FILENAME H-22  
  FILETYPE H-22  
  IDMSQSAM H-11  
  JOURNAL H-11  
  JRNLDTS H-11  
  LANG H-12

SYSIDMS parameters (*continued*)  
  LENGTH\_PAGE H-12  
  LIST H-12  
  LOADAREA H-12  
  LOCAL H-12  
  LOCAL\_DYNAMIC\_ALLOCATION H-12  
  LOCAL\_NOJOURNAL\_RETRIEVAL H-12  
  LOCALPUR H-13  
  MSGDICT H-13  
  MULTIDSN H-14  
  NODENAME H-14  
  OVERPRINT H-14  
  PARM H-14  
  PREFETCH 17-10, H-14  
  PREFETCH\_BUF H-14  
  PROCTRACE H-15  
  QSAMAREA H-15  
  QSAMBUF# H-15  
  QSAMTRACE H-15  
  REREAD\_SYSCTL H-16  
  REWIND H-22  
  ROLLBACK3490 H-16  
  SORTSIZE H-16  
  SQL\_INTLSORT H-16  
  SQLTRACE H-16  
  SYS\_MSG H-17  
  SYSCTL H-17  
  UPPER H-17  
  USERCAT H-17  
  WIDTH\_PAGE H-18  
  XA\_SCRATCH H-18  
SYSTEM catalog schema 23-11  
system dictionary  
  components 23-5  
  defining 23-16  
  definition 1-7  
  description 23-3  
system generation parameters  
  for lock management 37-19  
system records F-3—F-4  
system-owned index 36-3  
  *See also* indexed sets  
  adding/deleting 31-23  
  defining 13-79, 36-3

## T

table  
  adding a check constraint 28-10  
  adding a column 28-8  
  adding/removing data compression 28-10

---

table (*continued*)

- changing column characteristics 28-10
- changing its area 28-12
- creating 7-6, 28-7
- dropping 28-7
- dropping a check constraint 28-11
- dropping a column 28-9
- dropping and recreating 28-14
- dropping the default index 28-12
- modifying check constraints 28-11
- revising the estimated row count 28-11
- synchronization stamp 3-9

TAPE JOURNAL statement 6-99—6-102

tape journals

- defining 6-99—6-102
- in local mode 18-4

task application table (TAT) 24-26

task codes for online compilation 9-4

test environment 1-12

TEXT clause

- in schema-attribute association 13-10
- in schema-user association 12-14
- in subschema-attribute association 14-11

TIME journal record 18-6

TRAILING sign placement for element 13-51

transaction state (SQL) 37-8

TRANSFER STATISTICS clause 14-9, 14-14

TRANSIENT READ isolation level 37-8

TRANSIENT RETRIEVAL area status 37-7, 37-20

transient retrieval read mode 37-4

tuning

- buffers 17-8
- referential constraints 29-8

## U

UNCOMPRESSED index entries 13-84

unlinked constraints 29-7

unlinked index 13-86

UNLOAD utility statement 30-5, 30-7

UNORDERED

- DUPLICATES clause for VSAM CALC record types 13-29
- DUPLICATES option for sorted sets 13-85

unused record 13-35

UPAM file access method 16-5

UPDATE area status 37-7

update currency 11-4—11-5

UPDATE operation

- allowed/disallowed for a user 12-13
- for public access 13-9, 14-10

update ready mode 37-4

UPDATE responsibility, documenting 12-14

update-intent-exclusive lock 37-14

UPDATE, area ready mode

- compiling in 10-35
- restricting DML programs from using 14-18
- setting as DML default 14-19

UPPER SYSIDMS parameter H-17

USAGE clause

- area specification 14-8
- element specification 13-59—13-60
- element specification (table) 13-61

USAGE MODE

- for database areas 14-19
- in ADD/MODIFY/DELETE AREA statement 14-18

USER clause

- in SIGNON statement 10-34
- to access a secured dictionary 10-34

user exits

- card image G-3
- end of conversation G-3
- IDMSAJNX 18-12
- IDMSCPLX 18-12
- IDMSJNL2 18-12
- major command G-3
- SIGNON/SIGNOFF/COMMIT G-3
- WTOEXIT 18-9, 18-12

user ID

- when to specify 13-13

USER journal record 18-6

user responsibility 12-14

user-defined comments 14-11

user-owned index 36-3

- See also* indexed sets
- defining 36-3

user-specification clause 14-14

USERCAT SYSIDMS parameter H-17

USERS

- in non-SQL schema display 13-12

utilities

- ARCHIVE JOURNAL 18-9—18-11
- FASTLOAD 20-3—20-6
- FORMAT 33-12
- IDMSDIRL 23-9
- MAINTAIN INDEX 30-5, 30-8
- PRINT JOURNAL 18-12
- RELOAD 30-5, 30-7
- RESTRUCTURE SEGMENT 30-5, 30-7
- UNLOAD 30-5, 30-7

---

---

## V

VALID status 11-7  
VALIDATE operation 11-7  
VALIDATE statement 13-92, 14-61  
    effect on subschemas 14-61  
    for error checking 14-61  
    purpose 8-23  
    schema status 8-16  
    syntax 13-92, 14-61  
    validate procedure 8-17, 8-23  
    verifying schema relationships 8-16  
VALUE clause (schema elements) 13-48  
variable format of database keys 6-97  
    maximum number of records on 6-98  
variable-length indicator (VLI) 34-12  
variable-length record 13-31—13-39, 13-51,  
    34-11—34-14  
    (figure) 13-40  
VARIABLE, VSAM record length specification 13-31  
verb 9-7  
VERB, setting the session default for 10-26  
VERSION clauses  
    DEFAULT FOR EXISTING VERSION 10-20  
    DEFAULT FOR NEW VERSION 10-21  
    for subschema load modules 13-93, 14-62  
    in GENERATE statement 14-62  
    in REGENERATE statement 13-93  
version number  
    automatic assignment 13-35  
    explicit 10-20—10-21  
    HIGHEST 10-20  
    LOWEST 10-20  
    NEXT HIGHEST 10-21  
    NEXT LOWEST 10-21  
VIA location mode 13-28  
VIA storage mode  
    introduction to 34-7  
    via a chained set 34-7—34-9  
    via an indexed set 34-9—34-11, 36-16  
view  
    creating 7-13  
    dropping 28-5  
    modifying a 28-5  
    updatable 7-13  
VIEW ID clause  
    and ELEMENT clause (table) 14-26  
    subschemas specification 14-22  
VLI indicator 34-12  
VSAM CALC  
    control element 13-29

VSAM CALC (*continued*)  
    element name 13-29  
    location mode 13-29  
VSAM file access method 16-5  
VSAM file, CA-IDMS/DB  
    *See* CA-IDMS/DB VSAM file  
VSAM location mode 13-29  
VSAM set mode 13-77  
VSAM TYPE clause 13-31  
    in BUFFER statement 6-45  
    in record display 13-34  
VSAM types  
    changing 31-15  
VSAM, native  
    *See* native VSAM file  
VSE/ESA  
    file name 3-11  
    sublibrary, using copied code from E-10  
VSE/ESA JCL E-10  
    non-SQL schema compiler E-10  
    subschemas compiler E-12

## W

walking a set 35-7  
warmstart  
    failure, recovery for 19-31  
    recovery, automatic 19-14—19-17  
WHERE clause  
    in DISPLAY/PUNCH ALL statement 6-12, 6-13,  
    10-4, 10-5  
    in DML program, relation to path-group SELECT  
    clause 14-43  
    in path-group FIND/OBTAIN 14-48—14-51, 14-52,  
    14-53, 14-55  
    valid options (table) 6-15, 10-8  
WIDTH\_PAGE SYSIDMS parameter H-18  
WITH/ALSO WITH/WITHOUT clause  
    setting the session default for 10-24  
    table of options 10-31  
WITHIN AREA clause  
    schemas records 13-30  
    schemas sets 13-79  
write driver 17-11  
WTOEXIT user exit 18-9, 18-12

## X

XA\_SCRATCH SYSIDMS parameter H-18

