

CA-IDMS[®]

Features Summary
Release 12.0



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.

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.

First Edition, July 1992

One Computer Associates Plaza, Islandia, NY 11749
All rights reserved.

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

Contents

How to Use This Document	ix
Chapter 1. Database	1-1
1.1 About this chapter	1-3
1.2 Database definition	1-4
1.2.1 Separating logical and physical database definitions	1-4
1.2.2 Support for symbolic parameters	1-8
1.2.3 Indexing options	1-10
1.2.4 Collating options	1-11
1.2.5 Schema and subschema compiler enhancements	1-13
1.3 Data dictionary enhancements	1-15
1.3.1 New dictionary structure	1-15
1.3.2 Dictionary statement enhancements and options	1-16
1.3.3 Dictionary compiler options	1-18
1.4 Exploitation of ESA dataspace	1-20
1.5 24-hour processing	1-21
1.5.1 Dynamic database file management	1-21
1.5.2 Dynamic DMCL	1-23
1.5.3 Dynamic area extension	1-24
1.5.4 Dynamic database name tables	1-25
1.5.5 Dynamic buffer acquisition	1-25
1.5.6 Dynamic buffer pool configuration	1-25
1.6 Batch processing	1-27
1.6.1 SYSIDMS parameter file	1-27
1.6.2 Local mode security	1-33
1.6.3 Loading from a load area in local mode	1-33
1.7 Lock management	1-34
1.8 Deadlock management	1-36
1.9 Utilities	1-37
1.9.1 New interface	1-37
1.9.2 Utility statements	1-37
1.9.3 Utility programs	1-39
1.9.4 Security for utilities	1-40
1.9.5 For further information	1-40
1.10 Command facility	1-41
Chapter 2. Data Communications	2-1
2.1 About this chapter	2-3
2.2 CA-IDMS database communications architecture	2-4
2.2.1 New design	2-4
2.2.2 Components of the architecture	2-5
2.2.3 Setting up your environment	2-6
2.2.4 Multiple DC/UCF region communications	2-7
2.2.5 Enhanced CICS interface	2-9
2.2.6 For further information	2-9
2.3 Operating system features	2-10
2.4 Profile support	2-12

2.4.1	Defining profiles	2-13
2.4.2	Associating profiles with users	2-15
2.4.3	Displaying and accessing attributes	2-15
2.4.4	Using profiles with nonterminal tasks	2-16
2.4.5	For further information	2-16
2.5	New numbered user exits	2-17
2.6	Query device support	2-18
2.7	DCMT and DCUF commands	2-19
2.7.1	Invoking DCMT and DCUF commands from programs	2-19
2.7.2	New and modified DCMT and DCUF commands	2-19
2.8	System generation	2-22
2.8.1	Changes to statements and parameters	2-22
2.8.2	System generation compiler enhancements	2-27
2.9	Enhanced language support	2-29
2.9.1	Runtime support	2-29
2.9.2	General precompiler changes	2-29
2.9.3	Parameters supporting VS COBOL II	2-30
2.9.4	Parameters supporting COBOL 85 (Fujitsu and Hitachi)	2-31
Chapter 3.	Security	3-1
3.1	About this chapter	3-3
3.2	Security facility features	3-4
3.3	Administering security	3-6
3.3.1	Privileges	3-6
3.3.2	Resources	3-7
3.3.3	Authorization identifiers	3-8
3.4	Granting and revoking privileges	3-9
3.4.1	Granting privileges	3-9
3.4.2	Revoking privileges	3-10
3.5	Security display facility	3-11
3.6	For further information	3-15
Chapter 4.	CA-IDMS Performance Monitor	4-1
4.1	About this chapter	4-3
4.2	General enhancements	4-4
4.3	Real-Time monitor	4-5
4.3.1	New screens	4-5
4.3.2	New fields	4-6
4.4	Application monitor	4-7
4.4.1	New screen	4-7
4.5	Interval monitor	4-8
4.5.1	New screens	4-8
4.5.2	New fields	4-9
Chapter 5.	CA-ADS and Mapping Facility	5-1
5.1	About this chapter	5-3
5.2	Integration with centralized security	5-4
5.3	CUA-style user interface	5-5
5.4	Enhanced compiler support	5-7
5.5	Mapping enhancements	5-9
5.6	Support for century date variables and built-in functions	5-15

5.7	Numeric test	5-16
5.8	Trailing sign BIF support	5-17
5.9	READY NOREADY	5-18
Chapter 6. CA-Culprit, CA-OLQ, and CA-ICMS		6-1
6.1	About this chapter	6-3
6.2	Overview	6-4
6.3	CA-Culprit	6-5
6.3.1	Double word binary support	6-5
6.4	For further information	6-6
6.5	CA-OLQ	6-7
6.5.1	Extended selection criteria	6-7
6.6	For further information	6-9
6.7	CA-ICMS	6-10
Chapter 7. CA-IDMS/DDS		7-1
7.1	About this chapter	7-3
7.2	CA-IDMS/DDS enhancements	7-4
7.3	For further information	7-6
Chapter 8. Introduction to SQL-Defined Databases		8-1
8.1	What is an SQL-defined database?	8-3
8.2	ANSI and FIPS support	8-4
8.3	Benefits and features	8-5
8.4	Components	8-7
8.5	SQL as a language	8-8
8.6	Tables, rows, columns	8-9
8.7	Schemas and views	8-10
8.8	Table operations	8-12
8.9	Integrity and constraints	8-13
8.10	Storing SQL definitions	8-15
8.11	For further information	8-16
Chapter 9. Defining a Database Using SQL		9-1
9.1	SQL data definition language	9-3
9.2	The definition process	9-4
9.3	For further information	9-7
Chapter 10. Accessing a Database Using SQL		10-1
10.1	Data manipulation with SQL	10-3
10.2	Interactive and embedded SQL	10-4
10.2.1	Interactive SQL	10-4
10.2.2	Embedded SQL	10-4
10.2.3	Dynamic SQL	10-6
10.3	CA-IDMS tools support for SQL	10-7
10.3.1	CA-ADS support	10-7
10.3.2	CA-OLQ support	10-7
10.3.3	CA-ICMS support	10-8
10.3.4	CA-Culprit support	10-8
10.4	SQL access to a non-SQL defined database	10-9

10.4.1	How to do it	10-9
10.4.2	Database requirements	10-10
10.5	For further information	10-11
Chapter 11.	SQL Extended Features	11-1
11.1	What are SQL extended features?	11-3
11.2	Database definition extensions	11-4
11.2.1	Data types	11-4
11.2.2	32-character column names	11-4
11.2.3	Database tuning extensions	11-4
11.3	Data access and manipulation extensions	11-5
11.3.1	Bulk access to tables	11-5
11.3.2	Scalar functions	11-5
11.3.3	Special registers	11-7
11.3.4	Date/time arithmetic	11-7
11.3.5	Temporary tables	11-8
11.3.6	Modular programming	11-8
11.3.7	Dynamic SQL	11-8
11.3.8	Access to non-SQL defined databases	11-8
11.4	Precompiler directive extensions	11-9
11.5	Session management extensions	11-10
11.5.1	Specifying a dictionary	11-10
11.5.2	Pseudoconversational support	11-10
11.5.3	Establishing session characteristics	11-11
11.6	Transaction management extensions	11-12
11.6.1	CONTINUE/RELEASE parameters on the COMMIT WORK statement	11-12
11.6.2	Dynamic selection of access module	11-12
11.6.3	Overriding access module defaults	11-13
Chapter 12.	CA-IDMS Access Module Creation	12-1
12.1	What is the optimizer?	12-3
12.2	Compilation strategy	12-4
12.3	How does optimization work?	12-5
12.3.1	Automatic reoptimization	12-6
12.3.2	Describing the access strategy	12-6
12.4	For more information	12-7
Chapter 13.	Administration of an SQL-Defined Database	13-1
13.1	Tuning the database	13-3
13.1.1	Indexes	13-3
13.1.2	CALC keys	13-3
13.1.3	Referential constraints	13-3
13.1.4	Clustering	13-4
13.2	Utilities in the SQL environment	13-5
13.3	Locking	13-6
13.3.1	Types of locks	13-6
13.4	Security for SQL-defined databases	13-8
13.4.1	Privileges	13-8
13.4.2	Granting privileges	13-9
13.4.3	Ownership	13-9

13.4.4 Security checking	13-9
13.5 For further information	13-11
Index	X-1

How to Use This Document

What this document is about

CA-IDMS refers to the complete line of systems software products in the IDMS product family. This document covers Release 12.0 features for these CA-IDMS products:

- CA-IDMS/DB
- CA-IDMS/DC
- CA-IDMS/UCF
- CA-IDMS Performance Monitor
- CA-ADS® and the Mapping Facility
- CA-Culprit™
- CA-OLQ®
- CA-ICMS™
- CA-IDMS/DDS
- CA-IDMS/SQL Option

The purpose of this document is to summarize new features so that you can identify areas of change and new features to use. The document is not intended to provide detailed information about how to use features. Detailed information is provided in the CA-IDMS Release 12.0 documentation set for each product. References to these documents are provided with the description of each feature.

Who should use this document

Users of CA-IDMS products. In particular:

- Database administrators
- Data communications administrators
- Programmers
- Data dictionary administrators
- And others who are interested in the features of Release 12.0

How information is presented

This document describes the following:

- **First Half:** Describes CA-IDMS Release 12.0 Enhancements to existing CA-IDMS products
- **Second Half:** Describes the CA-IDMS/SQL Option (SQL processing product for Release 12.0)

First Half: The first half of this document introduces features as they relate to existing CA-IDMS products. The first three chapters are separated by function (database, data communications, security). Chapters on CA-IDMS Performance Monitor, CA-ADS, CA-Culprit, CA-OLQ, CA-ICMS, and CA-IDMS/DDS follow.

Discussion of each feature includes:

- A description of the feature
- Uses and benefits of a feature
- An example, if appropriate
- References to related CA-IDMS documents that provide detailed information on how to use the feature

Second Half: The second half of this document introduces the CA-IDMS/SQL Option, a new product that provides for SQL definition of a database and SQL access to both SQL and non-SQL defined databases.

Topic areas for the second half include:

- An introduction to SQL-defined databases
- Defining a database using SQL
- Accessing a database using SQL
- Discussion of SQL extended features (extensions of the ANSI standard)
- Discussion of the CA-IDMS access module compiler and optimization
- Administering an SQL-defined database

Related documentation

Use this document in conjunction with these CA-IDMS Release 12.0 documents:

Note: This is not a complete list of CA-IDMS Release 12.0 documentation. For a complete list, see "Computer Associates Documentation Pricing Guide."

- **For CA-IDMS/DB**

- *CA-IDMS Database Administration* (Volumes I and II)
- *CA-IDMS Utilities*
- *CA-IDMS Security Administration*
- *IDD DDDL Reference*
- *CA-IDMS Navigational DML Programming*
- *CA-IDMS Command Facility*
- *CA-IDMS Release 12.0 Conversion Notebook*
- CA-IDMS installation manual for your operating system
- *CA-IDMS SQL Reference*
- *CA-IDMS SQL Programming*

- **For CA-IDMS/DC and CA-IDMS/UCF**

- *CA-IDMS System Generation*
- *CA-IDMS System Operations*
- *CA-IDMS System Tasks and Operator Commands*

- **For CA-ADS**

- *CA-ADS Reference*
- *CA-ADS User Guide*
- *CA-IDMS Mapping Facility*

- **For CA-Culprit**

- *CA-CULPRIT Reference*

- **For CA-OLQ**

- *CA-OLQ Reference*

- **For CA-ICMS**

- *CA-ICMS System Administration*

- **For CA-IDMS/DDS**

- *CA-IDMS/DDS Design and Operations*

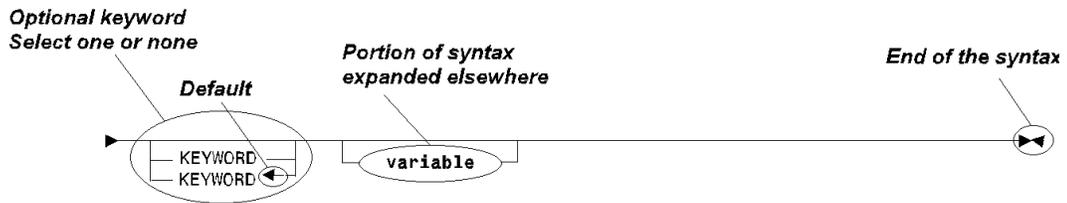
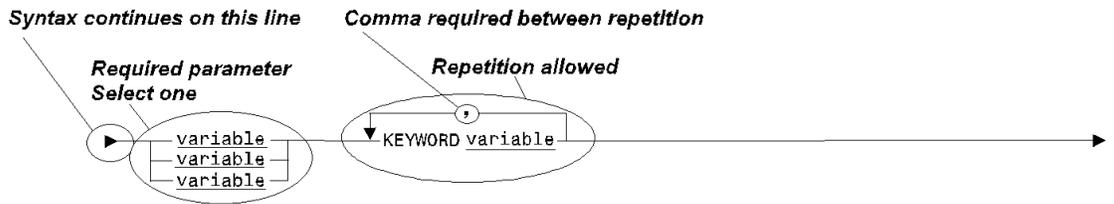
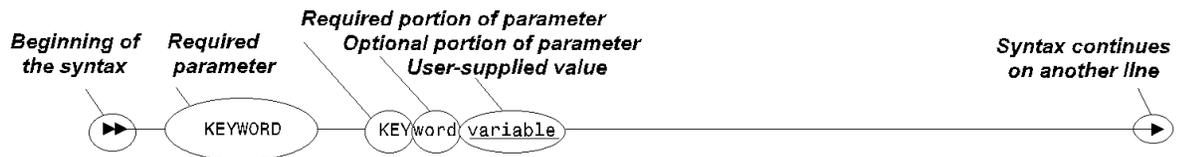
Related manuals are listed with the discussion of each feature. Where documentation is not necessary, as in the case of a feature which requires no user action, none is specified.

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.
lowercase bold	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.
▶ $\left\{ \begin{array}{l} \text{parameter} \\ \text{parameter} \end{array} \right\}$ ———▶	Shows a choice of required parameters. You must select one.
▶ $\left[\text{parameter} \right]$ ———▶	Shows an optional parameter.
▶ $\left\{ \begin{array}{l} \text{parameter} \\ \text{parameter} \end{array} \right\}$ ———▶	Shows a choice of optional parameters. Select one or none.
▶ $\left\{ \text{parameter} \right\}$ ———▶	Shows that you can repeat the parameter or specify more than one parameter.
▶ $\left\{ \text{parameter} , \text{parameter} \right\}$ ———▶	Shows that you must enter a comma between repetitions of the parameter.

Sample syntax diagram



Chapter 1. Database

1.1 About this chapter	1-3
1.2 Database definition	1-4
1.2.1 Separating logical and physical database definitions	1-4
1.2.2 Support for symbolic parameters	1-8
1.2.3 Indexing options	1-10
1.2.4 Collating options	1-11
1.2.5 Schema and subschema compiler enhancements	1-13
1.3 Data dictionary enhancements	1-15
1.3.1 New dictionary structure	1-15
1.3.2 Dictionary statement enhancements and options	1-16
1.3.3 Dictionary compiler options	1-18
1.4 Exploitation of ESA dataspace	1-20
1.5 24-hour processing	1-21
1.5.1 Dynamic database file management	1-21
1.5.2 Dynamic DMCL	1-23
1.5.3 Dynamic area extension	1-24
1.5.4 Dynamic database name tables	1-25
1.5.5 Dynamic buffer acquisition	1-25
1.5.6 Dynamic buffer pool configuration	1-25
1.6 Batch processing	1-27
1.6.1 SYSIDMS parameter file	1-27
1.6.2 Local mode security	1-33
1.6.3 Loading from a load area in local mode	1-33
1.7 Lock management	1-34
1.8 Deadlock management	1-36
1.9 Utilities	1-37
1.9.1 New interface	1-37
1.9.2 Utility statements	1-37
1.9.3 Utility programs	1-39
1.9.4 Security for utilities	1-40
1.9.5 For further information	1-40
1.10 Command facility	1-41

1.1 About this chapter

This chapter describes enhancements to database-related processing. While most of the features apply to just CA-IDMS/DB, some apply across the CA-IDMS product line. The features provide:

- Ease-of-use
- Processing flexibility
- Processing efficiency
- 24-hour processing support
- Exploitation of new technology

1.2 Database definition

Database definition topics covered in this section are:

- Separation of logical and physical database definition
- Support for symbolic parameters
- Indexing and collating options
- Schema and subschema compiler options

1.2.1 Separating logical and physical database definitions

Separation of logical and physical: Under CA-IDMS/DB, logical and physical database definitions are now separate.

What is a logical definition?: A logical database definition is made up of:

- **Schema** — The definition of the logical components of a database.
- **Subschema** — A subset of the schema (an application view).

What is a physical definition?: A physical database definition is made up of:

- **Segment(s)** — A collection of areas and files.
- **Database name table** — A table in which you assign logical database names to segments. You use logical names to access physical segments at runtime. You can also include subschema mapping rules.
- **DMCL** — A collection of physical database, journal, and buffer definitions representing a CA-IDMS/DB runtime environment. Each runtime environment is defined by a single DMCL.

Benefits of separating logical and physical definition: Separation of logical and physical database definition provides the following benefits:

- Fewer schemas and subschemas need to be defined and maintained. A single subschema can be used to access different databases by altering the database name on a BIND RUN-UNIT statement.
- Fewer subschemas need to be loaded into the program pool at runtime.
- Only one set of IDMSNWKx subschemas are needed to access all dictionaries in your environment.

Steps for defining the logical database: This table shows the basic definition process; some of the items in the 'What is included' column are optional.

Steps for definition	What is included
1. Define a schema.	Create a schema: <ul style="list-style-type: none"> ■ Add areas (areas no longer contain page ranges) ■ Add records ■ Add sets
2. Define a subschema.	Create a subschema: <ul style="list-style-type: none"> ■ Add areas ■ Add records ■ Add sets ■ Add logical records (and path-groups)

Steps for defining the physical database This table shows the basic definition process; some of the items in the 'What is included' column are optional.:

Steps for definition	What is included
1. Define segments.	Create segments: <ul style="list-style-type: none"> ■ Specify a page group ■ Specify maximum records per page ■ Add files, and specify: <ul style="list-style-type: none"> – External file name – Data set name – File access method ■ Add areas, and specify: <ul style="list-style-type: none"> – Page range – Page size and page reserve – Symbolic parameter values – File mapping

Steps for definition	What is included
2. Define a database name table (DBTABLE)	<p>Create a database name table:</p> <ul style="list-style-type: none"> ▪ Specify subschema rules <p>Create DBNAME entries in the table:</p> <ul style="list-style-type: none"> ▪ Identify segment(s) ▪ Specify subschema mapping rules <p>Note — Subschema mapping rules are primarily used for upward compatibility with previous releases.</p>
3. Define the DMCL .	<p>Create a DMCL:</p> <ul style="list-style-type: none"> ▪ Define buffers ▪ Define journals ▪ Assign a default buffer for the DMCL ▪ Identify a database name table for the DMCL ▪ Include segments, and specify: <ul style="list-style-type: none"> – Default buffer for the segment – Startup status – Warmstart status ▪ Add file overrides ▪ Add area overrides

Note: All physical data definition language statements are submitted to the CA-IDMS Command Facility. For more information on the CA-IDMS Command Facility, see "Command facility" later in this chapter.

Definitions stored in the data dictionary: Logical and physical definitions are stored in the data dictionary.

►► For information about new areas in the dictionary, see "Data dictionary enhancements" later in this chapter. For detailed information about dictionaries, see *CA-IDMS Database Administration*.

Examples: The following example shows the definition of a segment.

```

create segment empseg;
create file emp1 assign to emp1;
create file ins1 assign to ins1;
create file org1 assign to org1;
create area emp-region
    primary space 100 pages
        from page 802001
    page size 4276 characters
    within file emp1;
create area ins-region
    primary space 50 pages
        from page 802201
    page size 4276 characters
    subarea calc-range
        from page 802202 thru 802250
    within file ins1;

```

The following example shows the creation of the database name table ALLDBS and the definition of the DBNAME entries in the table.

```

create dbtable allpbs;

create dbname allpbs.system
    include segment system
    include segment catsys
    include segment sysmsg;

create dbname allpbs.empdb
    include segment empseg
    include segment projseg;

create dbname allpbs.testdb
    include segment tempseg
    include segment projseg;

generate dbtable allpbs;
punch dbtable load module allpbs;

```

The following example shows the modification of the DMCL IDMSDMCL to include segments and a database name table. The GENERATE and PUNCH statements store the new DMCL load module in a load area, then punch it to a system punch file for linking to a load library.

```

alter dmcl idmsdmcl
    default buffer def-buff
    dbtable allpbs
    include segment empseg
        default buffer emp-buff
        file empseg.indx-file
        buffer indx-buff
    include segment projseg
    include segment tempseg;
generate dmcl idmsdmcl;
punch dmcl load module idmsdmcl;

```

Linking logical and physical definitions at runtime: The connection between the logical definitions in the subschema and the physical definitions in the DMCL is made at runtime using information in the database name table. DBNAME entries in the table identify the segments CA-IDMS/DB is to access. Users can specify either a segment name or a database name (DBNAME) to identify the database to access at runtime.

Examples: In the following example, the BIND statement includes a database name. This automatically includes all segments defined in that DBNAME entry in the database name table.

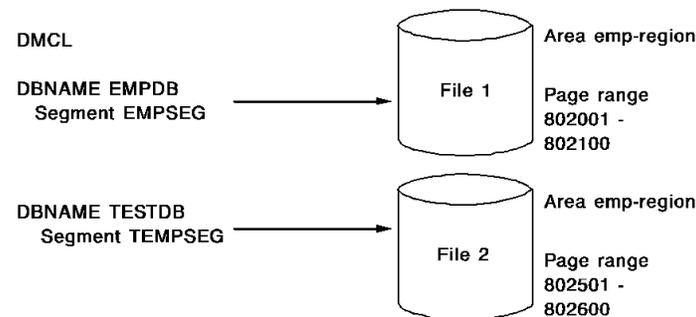
```
bind run-unit for empss01
    dbname empdb
```

In the following example, the BIND statement specifies a segment name rather than the DBNAME the segment is associated with.

```
bind run-unit for empss01
    dbname tempseg
```

The following example shows a physical representation of how you can use different database names to access different physical databases.

The area emp-region has been defined to both segments. In segment EMPSEG, area emp-region occupies a page range of 802001 - 802100 while in segment TEMPSEG area emp-region occupies a page range of 802501 - 802600. By binding to a different database name (that includes the segment), a program can access a different physical location.



For further information:

- **On the separation of logical and physical database definition**, see "Defining Physical Databases" in Volume I, Part Two of *CA-IDMS Database Administration*
- **On physical database definition**, see Volume I, Part Two, "Physical Database Definition", of *CA-IDMS Database Administration*
- **On logical database definition**, see Volume I, Part Four, "Non-SQL Database Definition", of *CA-IDMS Database Administration*

1.2.2 Support for symbolic parameters

Support for symbolic parameters: Symbolic parameters allow you to specify hard values in the physical database definition, and point to the values from the more generic logical definition. Advantages of symbolic parameters:

- The values can differ between physical databases (segments), even though a single schema definition is used.

- The values can be changed without regenerating the subschema.

Symbolic parameters are supported and defined under the logical schema definition of records and sets. The parameters are resolved at runtime using values defined as part of the segment definition.

These symbolic parameters are discussed in this section:

- Subareas
- Symbolic displacement
- Symbolic index sizing specification

Symbolic names for subareas: A subarea represents a subset of an area's page range; the subarea holds record occurrences or index structures.

Example: This example shows the use of a symbolic name for the subarea specification in the logical definition, and then the resolution of the subarea symbol in the physical definition.

Logical

```
modify record employee
    within area emp-demo-region subarea empl-range.
```

Physical

```
alter area empseg.emp-demo-region
    subarea empl-range
    offset 1 page for 100 percent;
```

Symbolic displacement for VIA sets: VIA set displacement represents how far member records of a VIA set are stored from the owner record of the set.

Example: This example shows the specification for symbolic displacement in the logical definition, and then the resolution of the symbol in the physical definition.

Logical

```
modify record coverage
    location mode is via emp-coverage set
    displacement using coverage-disp.
```

Physical

```
alter area empseg.emp-demo-region
    symbolic displacement coverage-disp 5 pages;
```

Symbolic index specification: You can use a multi-valued symbolic parameter to represent:

- The size of an internal index record (SR8)
- The displacement of bottom-level index records from the rest of the index

You can specify SR8 size and displacement values or you can provide syntax that directs CA-IDMS/DB to calculate the values based on the length of the key, the estimated number of index entries, and physical information (like the size of the page).

Example: This example shows the index symbol specification in the logical definition. The example then shows two different ways of resolving the symbol values in the physical definition. In Physical #1, values are supplied by the user. In Physical #2, the BASED ON clause directs CA-IDMS/DB to calculate the values.

Logical

```
modify set emp-coverage
    mode is index using emp-coverage.
```

Physical #1

```
alter area empseg.emp-demo-region
    symbolic index emp-coverage
        block contains 30 keys
        displacement 10 pages;
```

Physical #2

```
alter area empseg.emp-demo-region
    symbolic index emp-coverage
        based on sorted key length 10 for 100 records;
```

For further information:

- **On including symbolic parameters in RECORD and SET statements**, see "Schema Statements" in Volume I, Part Four, of *CA-IDMS Database Administration*
- **On including symbolic parameters in the corresponding physical AREA statements**, see "Physical Database DDL Statements" in Volume I, Part Two of *CA-IDMS Database Administration*

1.2.3 Indexing options

Unlinked indexes: An unlinked index is a system-owned index in which there are no index pointers in the member records.

Advantages of unlinked indexes:

- You can load and rebuild unlinked indexes faster.
- You can add or remove an unlinked index without restructuring the database, provided the control length of a compressed or variable length member record is not changed.
- Orphan management, which eliminates a source of gradual performance degradation in extremely active indexes, is no longer necessary.

You use the OMITTED parameter of the INDEX DBKEY POSITION clause to specify an unlinked index. You must also specify an unlinked index as MANDATORY AUTOMATIC.

Example This is an example of an index defined as unlinked.:

```
add set name is empl-ix
  mode is index
  owner is system
  member is employee
    index dbkey position is omitted
    mandatory automatic.
```

For further information:

- **On the INDEX DBKEY POSITION clause of the SET statement,** see "Schema Statements" in Volume I, Part Four, of *CA-IDMS Database Administration*

Duplicates stored in db-key sequence: You can order index entries with duplicate index key values in member db-key sequence. This option speeds retrieval by reducing I/O.

Example: This example shows the DUPLICATES BY DBKEY clause used to specify ordering by db-key.

```
.
.
.
add set name is office-employee
  order is sorted
  mode is index
  block contains 30 keys
.
.
.
member is employee
  key is emp-last-name
  duplicates by dbkey.
```

For further information:

- **On the DUPLICATES clause of the SET statement,** see "Schema Statements" in Volume I, Part Four, of *CA-IDMS Database Administration*

1.2.4 Collating options

Mixed sort sequence: You can specify both ascending and descending sort sequences for an index or sorted set with multiple sort fields.

This feature provides ordering flexibility. Where previously only one choice per sorted set existed, you can now choose ascending or descending for each sort field.

Example: This example shows the KEY IS clause used to order the SKILL-ID field in an ascending manner, and the EVALUATION-DATE field in a descending manner.

```
add set name is emp-expertise
  order is sorted
  mode is chain linked to prior
  owner is employee
    next dbkey position is 10
    prior dbkey position is 11
  member is expertise
    next dbkey position is auto
    key is (skill-id ascending evaluation-date
           descending)
    mandatory automatic;
```

For further information:

- **On the ASCENDING and DESCENDING sort options of the KEY IS clause in the SET statement**, see "Schema Statements" in Volume I, Part Four, of *CA-IDMS Database Administration*

Natural sort sequence: This option for sorted sets and indexes allows you to specify that key values are to be ordered such that negative numeric values collate lower than positive numeric values.

If natural collating sequence is inappropriate based on the data type of group and subordinate elements, then standard collating sequence prevails, and a warning message is generated.

Example: Assuming that the values below are packed or zoned decimal numbers, they are ordered first using the natural collating sequence and then according to EBCDIC collating sequence.

Natural	EBCDIC
-4268.50	15.26
-351.78	144.83
-258.00	-258.00
15.26	-351.78
144.83	2594.38
2594.38	-4268.50

In the following example, the NATURAL SEQUENCE clause will cause ACCOUNT-BALANCE values to be ordered with credit amounts in an account (negative values) preceding the debit amounts (positive values).

```

add set name is cust-account
  order is sorted
  mode is chain linked to prior
  owner is customer
  next dbkey position is auto
  prior dbkey position is auto
  member is account
    next dbkey position is auto
    prior dbkey position is auto
    key is (account-balance ascending)
    natural sequence
    mandatory automatic.

```

For further information:

- **On the NATURAL SEQUENCE parameter of the SET statement,** see "Schema Statements" in Volume I, Part Four, of *CA-IDMS Database Administration*

1.2.5 Schema and subschema compiler enhancements

Availability of user exits: Support for user exits is now available for the schema and subschema compilers. This support is the same as the support previously implemented for the IDD compiler.

The following table shows exits and what you can do in the exits.

Program an exit to occur here	To do this
After SIGNON/SIGNOFF/COMMIT	Perform security checking.
After these commands— ADD MODIFY DELETE DISPLAY/PUNCH INCLUDE/EXCLUDE	Verify naming conventions and perform security checking.
After the processing of an input statement	Pass input statements to the user-exit control block to build an audit trail of updates to the schema.
At end of converse, signaled by: <ul style="list-style-type: none"> ▪ Pressing [Enter] (online) ▪ SIGNOFF or the detection of an end-of-file condition (batch) 	Perform termination activities, such as a write to the log.

Non-quoted letters converted to upper case: All syntax components submitted to the schema and subschema compilers that are not in quotes are converted to upper case. Users can submit statement components in lower case and non-quoted letters are automatically converted to upper case.

User-defined comments: User-defined comments can be used with schema and subschemas. You define the comment using the DDDL compiler MODIFY ENTITY statement and then associate the comment with a schema or subschema on the COMMENT subclause of the ADD/MODIFY SCHEMA or SUBSCHEMA statement.

Example

```
modify subschema empss01
'subschema purpose' 'this subschema is used to process
'quarterly benefit statements.'
```

Output formatting: You can specify the number of lines per page (ranging from 10 to 60) as a SET OPTIONS for SESSION option.

Example

```
set options for session
  lines per page 60.
```

Simplified commenting: You can use *+ or anywhere on an input line to indicate that the remainder of the line is a user comment. If the *+ or is coded in positions 1 and 2, the line is not echoed.

Additionally, you can use '* ' in positions 1 and 2 to indicate that the remainder of the line is a user comment. Lines containing '* ' in positions 1 and 2 are echoed.

Semicolon as end-of-statement character: If you use the SEMICOLON ON clause of the SET OPTIONS statement, semicolons are recognized (in addition to the period) as an alternate end-of-statement character.

VALIDATE bypassed when there is no error: A subschema VALIDATE is bypassed during a GENERATE (or REGENERATE) request if the subschema is not flagged 'in error.' This reduces processing overhead.

1.3 Data dictionary enhancements

Data dictionary enhancements covered in this section are:

- New dictionary structure
- Statement enhancements and options
- Compiler options

1.3.1 New dictionary structure

Overview of the new dictionary structure: The components of the data dictionary have changed in the following ways:

- There is now a distinction between the dictionary that contains system definition information (called the **system** dictionary) and dictionaries that contain application information (called **application** dictionaries).
- There are new dictionary areas.
- User information (e.g., profiles, passwords, group information) for centralized security checking and system signon is now stored in a user catalog.

Where is everything stored now?: The table below shows what is stored in the **system** dictionary, what can be stored in an **application** dictionary, and what is now stored in a **user catalog** (separate from the dictionaries).

System dictionary	Application dictionary	User catalog
System definitions	Schema definitions	User information
<ul style="list-style-type: none"> ▪ Terminals ▪ Tasks ▪ Programs 	Subschema definitions Application definitions, such as: <ul style="list-style-type: none"> ▪ Records ▪ Maps ▪ Dialogs ▪ Programs 	Passwords Group definitions
Physical database definitions		
<ul style="list-style-type: none"> ▪ Segments ▪ DMCL ▪ Database name table 		
	User information	

New dictionary areas: These dictionary areas are new:

- **DDL**CAT — This area contains physical database entities, such as segments, database name tables, and DMCLs.
- **DDL**CATX — This area contains indexes associated with DDL
CAT entities.
- **DDL**CATLOD — This area contains load modules for DMCLs and database name tables.

Benefits of the new dictionary structure: The redesign of the data dictionary has made it possible to separate the physical database definitions from the logical database definitions.

The redesign also provides for a separate repository (the user catalog) for system-wide security definitions.

Note: Separate user definitions are still retained in the application dictionary.

For further information:

- **On the areas in the system and application dictionaries, their assignment to segments, and their relationship to the runtime environment,** see *CA-IDMS Database Administration*
- **On the user catalog and security,** see *CA-IDMS Security Administration*

1.3.2 Dictionary statement enhancements and options

Enhanced processing of COBOL PICTURE: Differences in the entry formats of COBOL record element statements are resolved. CA-IDMS/DB now analyzes a COBOL PICTURE clause and determines if any element with an equivalent definition is already present. If such an element exists, its definition is used rather than creating a new element.

This feature reduces the number of generated duplicate element definitions in the data dictionary.

Example: CA-IDMS/DB recognizes these PICTURE clauses as equivalent.

```
emp-state pic x(2).  
emp-state pic xx.
```

Option to limit DISPLAY ALL: You can set a maximum number of records to be read by a DISPLAY ALL statement. When the maximum (INTERRUPT COUNT) is reached, no more records are retrieved, regardless of whether or not the records meet the search criteria.

To set a maximum number of records, you must specify two clauses in the SET OPTIONS statement for the dictionary or session:

- DISPLAY ALL LIMIT IS ON
- INTERRUPT COUNT IS *maximum-record-count*

You can use this option to restrict access to the DISPLAY ALL command. If you specify *maximum-record-count* as 0, no records can be read.

Example: This is an example of a SET OPTIONS statement used to set display limits to 25 records.

```
set options for dictionary
  display all limit is on
  interrupt count is 25;
```

DISPLAY ALL support for DAY/MONTH/YEAR: You can specify day, month, and year as separate fields in the DISPLAY ALL WHERE subclause of:

- DATE CREATED
- DATE LAST UPDATED
- DATE COMPILED

This feature provides more flexibility in defining a search condition for an entity.

Example: This example shows a DISPLAY ALL statement in which the search condition is based on month and year.

```
display all records where prepared by eq 'jkd' and
  month created eq '06' and
  year created eq '90';
```

MODULE-to-MAP cross-referencing: You can cross-reference edit and code tables, maps, and IDD modules that have a language of HELP. In Release 12.0, cross-referencing is automatically built by the mapping facility; you can add cross-referencing to maps created in previous releases.

Cross-referencing protects modules that are actively being used by maps. If it exists, modules can be modified, but not deleted. All cross-referencing must be removed before you can delete the module.

Examples: The following example shows inclusion of MAPs as part of the MODULE and TABLE syntax.

```
modify module help01 language is help
  include map map01 version 1
  text 'help text for map01'.

modify table edit01
  include map map01 version 1
  text 'edit table for map01'.
```

The following example shows inclusion of MODULEs and TABLEs as part of the MAP syntax.

```
modify map map01 version 2
  include module help01 language is help
  text is 'help text for map01'
  include table edit01
  text 'edit table for map01'.
```

Enhanced DISPLAY MODULE/QFILE statement: The MODULE SOURCE ONLY option is a new parameter on the DISPLAY MODULE/QFILE statement. If the MODULE SOURCE ONLY option is specified, only the text of the module or qfile is displayed without any syntax.

This allows module or qfile source modules stored in the dictionary to be displayed and subsequently executed by just deleting the DISPLAY MODULE statement.

Example: Here is a sample module defined to set options for a dictionary session:

```
add module setopt
  module source follows
  set options for session
    input 1 72
    output 80
    semicolon alternate end of sentence is on.
  msend.
```

To display the setopt module with module source only, simply submit a DISPLAY MODULE statement with the MODULE SOURCE ONLY option. The setopt module is displayed as module source. Here is an example:

```
DIS MODULE SETOPT WITH MODULE SOURCE ONLY AS SYNTAX.
SET OPTIONS FOR SESSION
  INPUT 1 72
  OUTPUT 80
  SEMICOLON ALTERNATE END OF SENTENCE IS ON.
```

To execute this module, simply delete the DIS MODULE statement and submit the remaining source to the DDDL compiler.

DC OPTION IS MENU: A MENU option has been added to the DC OPTION parameter of the IDD PROGRAM statement. As in the corresponding system generation statement, this option allows you to specify whether a mainline dialog is displayed on the CA-ADS menu screen.

For further information:

- **On statement enhancements and options:** see, *IDD DDDL Reference*

1.3.3 Dictionary compiler options

User signon override: A USER SIGNON OVERRIDE ALLOWED/NOT ALLOWED clause has been added to the SET OPTIONS statement to eliminate the need to replicate passwords in dictionaries.

USER SIGNON OVERRIDE NOT ALLOWED: If you specify that signon override is NOT ALLOWED, it results in the following:

- Users who are already identified to the environment in which a CA-IDMS compiler is executing (either through signon processing or batch job submission facilities) cannot sign on as a different user within a compiler
- The PREPARED BY/REVISED BY clauses are ignored

USER SIGNON OVERRIDE ALLOWED: If you specify `USER SIGNON OVERRIDE IS ALLOWED`, you can use a different user ID when signing on to the dictionary compiler and information specified in the `PREPARED BY/REVISED BY` clauses is processed as in prior releases.

Signon processing: CA-IDMS compilers will perform password verification only if the user ID specified on a `SIGNON` statement is different from the one known to the environment in which the compiler is executing.

If the user is unknown to the environment, the compilers will forward the signon request to the CA-IDMS security facility for processing. If successful, the user will then be known to the execution environment as if the user had invoked `DC/UCF SIGNON`. If the signon is unsuccessful, compiler processing will be disallowed.

Non-quoted letters converted to upper case: All syntax components submitted to the DDDL compiler that are not in quotes are converted to upper case. Users can submit statement components in lower case and non-quoted letters are automatically converted to upper case.

Output formatting: You can specify the number of lines per page (ranging from 10 to 60) as a `SET OPTIONS` for `SESSION` option. For example:

```
set options for session
    lines per page 60.
```

Simplified commenting: You can use `*+` or anywhere on an input line to indicate that the remainder of the line is a user comment. If the `*+` or is coded in positions 1 and 2, the line is not echoed.

Additionally, you can use `* '` in positions 1 and 2 to indicate that the remainder of the line is a user comment. Lines containing `* '` in positions 1 and 2 are echoed.

Semicolon as end-of-statement character: If you use the `SEMICOLON ON` clause of the `SET OPTIONS` statement, semicolons are recognized (in addition to the period) as an alternate end-of-statement character.

1.4 Exploitation of ESA dataspace

Exploitation of ESA dataspace: A dataspace is a linear address space that is used for storage and retrieval of data. In ESA systems, you can assign a BDAM file to a dataspace as a file override specification in the DMCL.

CA-IDMS/DB automatically manages these two dataspace modes when you assign a file to a dataspace:

- **Virtual** for scratch

CA-IDMS/DB uses virtual mode for scratch storage, since recovery is not necessary. No physical file access is necessary, therefore no I/O is required.

- **Covered** for database files

CA-IDMS/DB uses covered mode for database files, since recovery is necessary. In this mode, an update of the dataspace forces a corresponding write to disk.

Potentially, multiple database files can reside in a single dataspace. Each dataspace provides up to 2G of addressable virtual storage.

The page frames for a dataspace are contained in high speed main memory, so they provide the fastest access to large quantities of data outside your own address space. Using the dataspace alternative, you reduce the I/O necessary to access the data.

Candidates for dataspace assignment: Good choices for assignment to dataspace are files that contain frequently accessed data, such as index structures.

Example: This example shows assignment of the file EMPSEG.EMPFILE to a dataspace. Assignment is through a file override specification in the DMCL.

```
alter dmcl cvdmcl
  file empseg.empfile
  dataspace yes;
```

For further information:

- **On assigning files to dataspace,** see "Physical Database DDL Statements" in Volume I, Part Two of *CA-IDMS Database Administration*

1.5 24-hour processing

You can now accomplish the following tasks without recycling the DC/UCF system:

- Allocate and deallocate database files
- Change the DMCL
- Extend areas
- Change database name tables
- Acquire buffers
- Reconfigure buffer pools

Descriptions of each of these dynamic features follows.

►► For detailed information on the new DCMT VARY commands you can use to implement dynamic functions, see *CA-IDMS System Tasks and Operator Commands*.

1.5.1 Dynamic database file management

Dynamic file management: CA-IDMS/DB uses dynamic database file allocation to:

- Deallocate and reallocate files in an active runtime environment
- Access files without identifying them in JCL
- Access new files without recycling the DC/UCF system

Allocating a database file dynamically: To allocate a database file dynamically (without JCL statements), you specify the dataset name (*dsname*) of the file in the file definition. You may **optionally** specify an external name (*ddname*) for the file:

- If an external name is specified, CA-IDMS/DB uses the external name as a key to look for a JCL specification.
 - If a JCL specification is found, it is used.
 - If no JCL specification is found, CA-IDMS/DB allocates the file dynamically using information in the file definition.
- If an external name is not specified, CA-IDMS/DB allocates the file dynamically using information in the file definition.

Special considerations for VSE: Here are some considerations for VSE sites using dynamic file allocation:

- A DDNAME must be defined in the DMCL for every database file.
- Corresponding DLBL and EXTENT information must be available for all database files.

- To bring new files online or change the location of existing files without recycling the system, you must use standard labels to specify DLBL and EXTENT information.
- The DSNAMES is optional. However, if there is at least one file defined in the DMCL with a DSNAMES, the real DSNAMES will be extracted for display purposes.

Special considerations for CMS: At sites using CMS, you must specify the following for OS-format files:

- DSNAMES — Data set name
- VIRTUAL ADDRESS — The virtual address of the minidisk to be used for a dynamically allocated file
- USERID — The owner of the minidisk to be used for a dynamically allocated file

For CMS-format files, you must specify a VIRTUAL ADDRESS.

Benefits of dynamic file management: The benefits of dynamic database file management:

- You can take files offline for repair, then put them back online without recycling the DC/UCF system.
- If you include dynamic allocation information in the file definition you do not need to include file statements in JCL. This allows the dataset name or other identifier information to be changed without having an impact on the execution JCL.
- If you do not specify an external name as part of a file definition, the dataset name of the file cannot be overridden through JCL. This ensures access to the correct file at runtime.
- You can bring new files online without recycling the DC/UCF system.

Examples: This is an example of a CREATE FILE statement (part of the segment definition) that facilitates dynamic allocation of the file EMPFILE at CA-IDMS/DB startup. EMPDD is the external file name, and PROD.EMPDB is the data set name.

```
alter segment empseg;  
create file empseg.empfile  
  assign to empdd  
  dsname "prod.empdb";
```

In this example, a DCMT command allocates the file EMPFILE.

```
dcmt vary file empseg.empfile allocate
```

For further information:

- **On the coding considerations for dynamically allocating database files**, see "Allocating and Formatting Files" in Volume I, Part Two of *CA-IDMS Database Administration*

- On DCMT VARY FILE, see *CA-IDMS System Tasks and Operator Commands*

1.5.2 Dynamic DMCL

Dynamic DMCL: You can use a DCMT command to load a new copy of the DMCL load module without recycling the DC/UCF system.

You can dynamically:

- Add or remove a segment, area, or file
- Add or remove a buffer pool
- Expand an area by:
 - Increasing the page range
 - Increasing the page size
 - Adding new files

To safeguard the dynamic process, CA-IDMS/DB quiesces affected areas, as appropriate, and updates the journal header with the timestamp of the new DMCL.

By using the dynamic DMCL feature, you make a new or changed DMCL available under central version without having to bring down and restart the DC/UCF system.

Example: In this example, a DCMT command instructs CA-IDMS/DB to reload the DMCL IDMSDMCL after the DMCL has been altered to add a new file, regenerated, punched, and linked to a load library.

```
alter dmcl idmsdmcl
  create file custseg.cust_data
  buffer customer_buffer;

generate dmcl idmsdmcl;

punch dmcl load module idmsdmcl;
```

Note: After punching the new DMCL load module, you must link edit it.

The DCMT VARY DMCL VALIDATE NEW COPY command will provide a summary of all differences between the current and new DMCL. This allows you to assess the impact of varying the DMCL on current work load.

```
dcmt vary dmcl validate;
```

The DCMT VARY DMCL NEW COPY command also provides a summary of changes and displays a prompt asking the user whether to continue processing or not. If the user requests processing to continue, the new DMCL will become the new runtime DMCL.

```
dcmt vary dmcl new copy;
```

For further information:

- **On managing DMCLs dynamically**, see *CA-IDMS Database Administration*
- **On using DCMT VARY DMCL**, see *CA-IDMS System Tasks and Operator Commands*

1.5.3 Dynamic area extension

Dynamic area extension: You can increase the page range of an existing area without unloading and reloading data.

You specify this extension in the `EXTEND SPACE` clause of the `ALTER AREA` statement, after new files have been defined in the DMCL. To use `EXTEND SPACE`, you must have previously reserved the space by specifying `MAXIMUM SPACE` in the initial page range specification of `CREATE/ALTER AREA` with a sufficient number of pages to accommodate the extension.

Note: If there are `CALC` records in the area, the records will continue to target to the original page range and use normal overflow processing, if necessary.

Example: Assume you initially allocated 200 pages for `CUST_AREA`, then determined later that you needed more space. The following example shows how you would extend the area by increasing the number of pages and mapping them to the file `EXPAND_FILE`.

```
create file custseg.expand_file
  assign to xpndfile;
alter area custseg.cust_area
  extend space 100
  within file xpndfile;
```

After extending the area you must regenerate the DMCL, punch it to a load library, link edit it, and initialize the new file.

For further information:

- **On using the `EXTEND SPACE` clause**, see "Physical Database DDL Statements" in Volume I, Part Two of *CA-IDMS Database Administration*
- **On procedures for expanding an area's page range**, see "Modifying Physical Database Definitions" in Volume II, Part Eight of *CA-IDMS Database Administration*

1.5.4 Dynamic database name tables

Dynamic database name tables: You can use a DCMT command to add, change, or remove entities in a database name table, then load the new table, without recycling the DC/UCF system.

Example: In this example, CA-IDMS/DB reloads the ALLDBS database name table. After you update the table, regenerate it.

```
alter dhtable allpbs;  
  
create dbname allpbs.benefits  
    include segment empseg  
    include segment benseg;  
  
generate dhtable allpbs;  
  
punch dhtable load module allpbs;
```

Note: After you punch the database name table, you must link edit it.

```
dcmt vary dhtable new copy;
```

For further information:

- **On using DCMT VARY DBTABLE,** see *CA-IDMS System Tasks and Operator Commands*
- **On modifying a database name table,** see "Modifying Database Name Tables" in Volume II, Part Eight of *CA-IDMS Database Administration*

1.5.5 Dynamic buffer acquisition

Dynamic buffer acquisition (CV and local mode): CA-IDMS/DB acquires buffers as it opens files, in both central version and local mode. Storage is reduced to the minimum required to meet current processing needs.

This feature has an especially significant impact on local mode processing, as previously, all buffers defined in the DMCL were acquired at the start of a session.

1.5.6 Dynamic buffer pool configuration

Dynamic buffer pool configuration: You can use a DCMT command to temporarily alter characteristics of database buffers. This function is not restricted by the DMCL and the settings you specify at system generation.

You can change the following buffer characteristics dynamically:

- The number of buffer pages currently in use
- The maximum number of pages
- Type of storage (from the IDMS storage pool or from the operating system)

Note: If the operating system supports extended addressing, CA-IDMS/DB will acquire storage above the 16Mb line.

- The amount of storage to acquire with each storage request (ADDITIONAL PAGE parameter)

Changes last for the session: Dynamic changes to buffers last for the remainder of the CA-IDMS/DB session.

Benefits of dynamic configuration: Benefits of dynamic configuration:

- You can tailor buffer pools to suit the current processing load.
- You can evaluate the impact of temporary changes before you make them permanent.

Example: This example shows a DCMT command used to view the characteristics of the buffer EMP_BUFF:

```
dcmt display buffer emp_buff
```

And a DCMT used to increase the maximum number of pages in the buffer (the change is not effective until the buffer is closed and reopened):

```
dcmt vary buffer emp_buff maximum pages 1000
```

And a DCMT used to release storage for the buffer:

```
dcmt vary buffer emp_buff close
```

And a DCMT used to reallocate storage for the buffer and make it available to CA-IDMS/DB:

```
dcmt vary buffer emp_buff open
```

For further information:

- **On managing buffers dynamically**, see "Buffer Management" in Volume II, Part Six of *CA-IDMS Database Administration*
- **On using DCMT VARY BUFFER**, see *CA-IDMS System Tasks and Operator Commands*

1.6 Batch processing

1.6.1 SYSIDMS parameter file

SYSIDMS parameter file: A SYSIDMS file is a parameter file added to the JCL stream of batch jobs 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

Physical environment parameters:

Parameter	Description
<i>CVMACH=cms-machine-name</i>	Specifies the VM virtual machine in which the DC/UCF system is executing.
<i>CVNUM=nnn</i>	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.
<i>DBNAME=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.
<i>DICTNAME=dictionary-name</i>	Specifies the dictionary to use when loading a subschema from a load area. For dictionary-related tools like CA-IDMS compilers and precompilers, IDMSBCF, etc., specifies the dictionary to access at runtime. For SQL applications, specifies the name of the dictionary to connect to at runtime.

Parameter	Description
DICTNODE= <i>dictionary-node-name</i>	For SQL applications and dictionary-related tools running under the central version, specifies the name of the DC/UCF system that controls the dictionary to access at runtime. For applications running in local mode, this parameter is not applicable.
DMCL= <i>dmcl-name</i>	Specifies the name of the DMCL load module to use in local mode. IDMSDMCL 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 module is link-edited with the program. OFF is the default.
NODENAME= <i>node-name</i>	For non-SQL applications running under the central version, identifies the DC/UCF system to bind to at runtime.

Note: Some of the parameters in the table above were specified in previous releases as parameters in a batch job stream.

Runtime directive parameters:

Parameter	Description
CVRETRY=ON/OFF (MVS only)	ON indicates that this message is displayed on the operator console when the CA-IDMS central version is not active: <small>CV nnn NOT ACTIVE. REPLY RETRY OR CANCEL</small> ON 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.
ECHO=ON/OFF	Indicates whether all SYSIDMS parameters are displayed on the JES log. OFF is the default.

Parameter	Description
IDMSPROG= <i>module-name</i>	<p>Specifies the name of the module that will be linked to by the IDMSMVCV facility.</p> <p>Note — BS2000 users must use this parameter when executing programs in local mode. MVS and VSE users can use either the IDMSPROG parameter or the JCL PARM statement.</p>
LENGTH_PAGE= <i>nnn</i>	<p>Specifies the maximum number of lines to be printed on a page. <i>Nnn</i> must be in the range from 10 through 32,767. The default is 60.</p>
LOADAREA=ON/OFF	<p>Specifies whether the dictionary load (DDLDCLOD) area is to be accessed when loading a module.</p> <p>If OFF is specified, the dictionary load area will not be accessed. You should specify OFF only when all load modules are linked into the OPSYS load library.</p> <p>ON is the default.</p>
MSGDICT=ON/OFF	<p>Specifies whether the dictionary message (SYSMSG) area is to be accessed.</p> <p>If OFF is specified, the dictionary message area is not accessed. You should only specify OFF when running installation steps that use a DMCL not containing the SYSMSG segment.</p> <p>ON is the default.</p>
USERCAT=ON/OFF	<p>Specifies whether the user catalog is to be accessed.</p> <p>You must specify OFF when formatting the user catalog.</p> <p>ON is the default.</p>
SQLTRACE=ON/OFF	<p>ON activates a trace facility that traces all SQL database requests made by an application. OFF is the default.</p>
IDMSQSAM=ON/OFF	<p>ON activates the IDMSQSAM facility (sequential access for look-ahead database reads). OFF is the default.</p>

Parameter	Description
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 run-unit.
QSAMTRACE=ON/OFF	ON activates a trace of all the IDMSQSAM look-ahead I/O reads. The trace shows the name of the file(s) being accessed by IDMSQSAM, each RBN that was read using QSAM OR BDAM (DAM/EXCP), and a summary of the number of RBNs read using 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.
JOURNAL=ON/OFF	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.
XA_SCRATCH=ON/OFF	Specifies whether scratch space will be allocated in XA storage or not. OFF, the default, indicates that a scratch file will be used.
WIDTH_PAGE= <i>nnn</i>	Specifies a maximum number of characters to be printed on a line. <i>Nnn</i> must be an integer in the range from 71 to 132. The default is 132.
UPPER=INPUT/OUTPUT/BOTH/OFF	Specifies whether input and/or output files will be converted to uppercase: <ul style="list-style-type: none"> ■ INPUT — Converts SYSIPT input files to uppercase ■ OUTPUT — Converts SYSLST output files to uppercase ■ BOTH — Converts both SYSIPT input files and SYSLST output files to uppercase ■ OFF (the default) — Does not convert SYSIPT input files or SYSLST output files to uppercase

Parameter	Description
DLBLMOD=ON/OFF (VSE 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.
MULTIDSN=ON/OFF (VSE 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.
PARM='parameter-string'	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. <i>Parameter-string</i> can contain any 1 through 256 character parameter and it can be specified on multiple lines.

VSE file information parameters:

Parameter	Description
FILENAME= <i>file-name</i>	Specifies a file name before assigning specific overriding file attributes (file type, block size, blocking factor, device address, file label).
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 <i>SYSlogical-unit-number</i>).

Parameter	Description
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.

BS2000 file information parameters: These SYSIDMS parameters are for the BS2000 operating environment only.

Parameter	Description
QSAM#BUF= <i>nnn</i>	Specifies the number of buffers to use for IDMSQSAM simulation.
OVERPRINT=YES/NO	Specifies whether the overprint facility is used when writing to SYSLST. The default is YES.
LIST=SYSLST/SYSOUT/BOTH	Specifies whether output is written to SYSLST, SYSOUT, or both. The default is SYSLST.

Examples: In the following example, the SYSIDMS parameters included in the MVS 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 in an MVS environment.

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

For further information:

- **On including SYSIDMS parameters in a job stream,** see "Dictionaries and Runtime Environments" in Volume II, Part Seven of *CA-IDMS Database Administration*

1.6.2 Local mode security

Security enforced in local mode: Security is now enforced in local mode. All security measures you implement will apply to local mode as well as to operation under the central version.

Utility functions will also be subject to security checking.

For further information:

- **On security,** see *CA-IDMS Security Administration*

1.6.3 Loading from a load area in local mode

Loading from a load area: CA-IDMS/DB can now load modules directly from a load area in local mode, rather than requiring that you load only from load libraries.

The process is similar to the process you use when you load in DC/UCF. A load list dictates the order in which dictionaries will be searched. You can use the SYSIDMS parameter DICTNAME (described in this section under SYSIDMS parameter file) to specify the name of the first dictionary to be searched.

1.7 Lock management

New lock management software reduces CPU overhead: The architecture of the lock management software has been completely redesigned to reduce CPU overhead. This redesign is transparent to user applications.

Storage for locks: If your database supports extended addressing all lock storage will be acquired above the 16Mb line.

DCMT for notify lock management: A new DCMT command (DCMT DISPLAY LOCK) is available to display locks on areas or longterm and notify locks held by a logical terminal. This information is useful in determining which terminal sessions hold locks on areas.

Preservation of external area status: CA-IDMS/DB automatically preserves the status of an area when the DC/UCF system abnormally terminates. For instance, if you vary an area offline, and the DC/UCF system goes down, the area status will remain offline when the system is restarted.

You can override the default warmstart status using the DMCL ADD/INCLUDE SEGMENT statement ON WARMSTART status clause.

Example: This example shows the segment clause specification in the DMCL that you use to override the warmstart status of an area.

```
alter dmcl idmsdmcl
    area empseg.emparea
        on startup set status to update
        on warmstart maintain current status;
```

Transient retrieval for areas: A DCMT command or an area override specification in the DMCL allows you to set the status of an area to transient retrieval. Transactions accessing an area whose status is transient retrieval can do so only with a retrieval ready mode.

Record locks are not maintained for areas whose status is transient retrieval. This allows a database transaction access to a non-updateable area without locking overhead.

Before an area whose status is transient retrieval can be varied to update, it must first be varied offline.

Examples: The following is an example of a DCMT VARY command that instructs CA-IDMS/DB to alter an area for transient retrieval.

```
dcmt vary area empseg.emparea transient retrieval
```

The following is an example of an area override specification in the DMCL that assigns a transient retrieval status to the area EMPAREA.

```
alter dmcl idmsdmc1
  area empseg.emparea
  on startup set status to transient retrieval;
```

For further information:

- **On specifying transient retrieval as a startup status for an area**, see "Physical Database DDL Statements" in Volume I, Part Two of *CA-IDMS Database Administration*

1.8 Deadlock management

Deadlock detection control: CA-IDMS/DB now checks for deadlocks at user-specified intervals, rather than every time a task enters a wait state. Performance is improved because:

- The overhead associated with deadlock detection is incurred less often.
- You can tune the deadlock detection interval to suit your processing needs.

You specify detection interval values in the DEADLOCK DETECTION INTERVAL clause of the SYSTEM statement during system definition and generation. The interval you choose is a trade-off between the overhead associated with detecting deadlocks and the amount of time tasks will wait before a deadlock is detected and resolved.

Example: This example shows a MODIFY SYSTEM statement that sets a deadlock detection interval. Intervals use whole second values.

```
modify system system74
    deadlock detection interval is 10
```

DCMT for deadlock management: A new DCMT command is available for displaying or dynamically changing the deadlock detection interval.

►► For a list of new DCMT commands, see DCMT and DCUF Commands in Chapter 2, “Data Communications” on page 2-1. For more detailed information on DCMT commands, see *CA-IDMS System Tasks and Operator Commands*.

Deadlock resolution control: In the event of a deadlock, CA-IDMS/DB automatically terminates the most recent transaction with the lowest dispatching priority.

Should you want more control, a new user exit (user exit 30) allows you to implement your own algorithm to determine which transaction to terminate.

►► For a brief description of user exit 30, see New User Exits in Chapter 2, “Data Communications” on page 2-1.

For further information:

- On **DEADLOCK DETECTION INTERVAL**, see *CA-IDMS System Generation*
- On **deadlock resolution**, see *CA-IDMS System Generation*
- On **user exits**, see *CA-IDMS System Operations*

1.9 Utilities

Information about utilities is categorized as follows:

- The new utility interface
- Utility statements that replace utility programs
- New statements
- Examples of statements
- Status of existing programs
- Enhancements to existing programs
- Removal of IDMSRNWK
- Security for utilities

1.9.1 New interface

New utility interface: There is now a new user interface for utilities. Many of the utilities are now invoked using command statements with a consistent syntax format. You enter these command statements in a job stream using the new CA-IDMS Command Facility.

►► For information about the new command facility, see *CA-IDMS Command Facility*.

Replacing programs with command statements allows you to execute several utility functions in the same job step.

The consistent syntax of command statements makes the utilities easier to use, and the more descriptive names of the statements provide more information about what the utilities do.

1.9.2 Utility statements

Utility statements that replace programs: The following table shows the new utility statements that replace existing utility programs.

Program	New utility statement
IDMSAJNL	ARCHIVE JOURNAL
IDMSDBLU	FASTLOAD RELOAD
IDMSDUMP	BACKUP PRINT SPACE

Program	New utility statement
IDMSINIT	FORMAT
IDMSJFIX	PRINT JOURNAL FIX ARCHIVE
IDMSLDEL	CLEANUP
IDMSPCON	RESTRUCTURE CONNECT
IDMSPFIX	FIX PAGE PRINT PAGE UNLOCK
IDMSPTRE	PRINT INDEX
IDMSRBCK	ROLLBACK
IDMSRFWD	ROLLFORWARD
IDMSRSTR	RESTORE
IDMSRSTU	RESTRUCTURE SEGMENT
IDMSTBLU	MAINTAIN INDEX MAINTAIN ASF TABLE
IDMSUNLD	UNLOAD
IDMSXPAG	EXPAND PAGE
RHDCPRLG	ARCHIVE LOG PRINT LOG

PUNCH statement: In addition to the other statements that replace programs, there is a new utility statement called PUNCH. PUNCH:

1. Retrieves DMCL load modules or database name table load modules from the catalog load area of the data dictionary
2. Writes the load modules in object form to the SYSPCH file

Examples of statements: The following is an example of the utility statement UNLOAD. This syntax could be used to change the page ranges in an existing area defined in a DMCL that contains both the OLDSEG and NEWSEG segments.

```
unload segment oldseg using subschema empss01
      reload into newseg;
```

The following is an example of the statement FORMAT.

```
format file dbseg.file1;
```

The following is an example of the statement ROLLFORWARD. The rollforward goes to the end of a job or to the abort checkpoint that has a date/time stamp of October 30, 1991 at 23:20:00.

```
rollforward area empseg.emparea all
      stop at '1991-10-30-23.20.00';
```

The following is an example of the statement RELOAD. The reload process starts at step IDMSDBL2 and continues through to the end of the reload process.

```
reload from idmsdbl2;
```

1.9.3 Utility programs

Status of other existing utility programs: These utilities exist in their previous format:

```
IDMSCALC
IDMSDBAN (with enhancements)
IDMSDIRL (with enhancements)
IDMSLOOK (with enhancements)
IDMSRADM
IDMSRPTS (with enhancements)
IDMSRSTC
```

Changes in existing programs: The following table describes enhancements to utilities that continue to exist in their previous format.

Utility	Enhancements
IDMSDBAN	New syntax Support for SQL-defined databases Audits indexes more efficiently
IDMSDIRL	Loads IDMSNTWK schema and two associated subschemas (IDMSNWKA and IDMSNWKG) Loads schemas IDMSSECS and IDMSSECU and subschemas IDMSSECS and IDMSSECU used for security processing Added option (SCHEMA-DELETE) to remove all CA-defined schema definitions (those listed above) from a dictionary without loading new definitions

Utility	Enhancements
IDMSLOOK	<p>Revised parameters provide this output:</p> <ul style="list-style-type: none"> ▪ Set information (SUBSCHEMA) ▪ Buffer information (DMCL) ▪ Information reflecting the separation of logical and physical definitions (DBTABLE) ▪ Information on segments (DBTABLE) ▪ Data set name of the library that the load module was loaded from (DATES and PROGRAM) <p>New parameters provide:</p> <ul style="list-style-type: none"> ▪ Information about IDMSLOOK parameters (HELP) ▪ Information about subschema load modules bound to a database (BIND SUBSCHEMA) ▪ Value of a date/time stamp (DATETIME STAMP) ▪ Additional parameters for SQL-related objects
IDMSRPTS	<ul style="list-style-type: none"> ▪ Removal of physical information from schema and subschema reports ▪ Enhancements to schema and subschema reports — Reporting on new compiler functions ▪ New reports for physical database definitions

Removal of IDMSRNWK: IDMSRNWK no longer exists. Because of the separation of logical and physical definitions, there is no longer a need for dictionary subschema reformatting.

1.9.4 Security for utilities

The execution of utility statements is subject to security checking. The specific authority necessary to execute a utility statement depends on the function of the utility.

1.9.5 For further information

- **On utilities in Release 12.0, and the authority required to use them,** see *CA-IDMS Utilities*
- **On using the batch command facility to enter utility statements in a job stream,** see *CA-IDMS Command Facility*

1.10 Command facility

Command facility: The CA-IDMS Command Facility is a new tool you use to:

- Enter and display physical DDL statements used to define a database
- Enter utility statements used to maintain a database
- Enter DDDL statements
- Enter interactive SQL statements (SQL option only)

The command facility is available for online (OCF) and batch (IDMSBCF) processing.

OCF uses the same text editor that is used with the dictionary-related compilers.

The command facility offers format control for output. For example:

```
set option headings off;
```

The command facility also provides optional session control options. For example, the session control option below instructs CA-IDMS/DB to rollback all database changes in the event of an error.

```
set option on error rollback;
```

Benefit: The benefit of the command facility is that it offers a consistent format for statement input, both in batch and online processing.

For further information:

- **On the command facility,** see *CA-IDMS Command Facility*

Chapter 2. Data Communications

2.1 About this chapter	2-3
2.2 CA-IDMS database communications architecture	2-4
2.2.1 New design	2-4
2.2.2 Components of the architecture	2-5
2.2.3 Setting up your environment	2-6
2.2.4 Multiple DC/UCF region communications	2-7
2.2.5 Enhanced CICS interface	2-9
2.2.6 For further information	2-9
2.3 Operating system features	2-10
2.4 Profile support	2-12
2.4.1 Defining profiles	2-13
2.4.2 Associating profiles with users	2-15
2.4.3 Displaying and accessing attributes	2-15
2.4.4 Using profiles with nonterminal tasks	2-16
2.4.5 For further information	2-16
2.5 New numbered user exits	2-17
2.6 Query device support	2-18
2.7 DCMT and DCUF commands	2-19
2.7.1 Invoking DCMT and DCUF commands from programs	2-19
2.7.2 New and modified DCMT and DCUF commands	2-19
2.8 System generation	2-22
2.8.1 Changes to statements and parameters	2-22
2.8.2 System generation compiler enhancements	2-27
2.9 Enhanced language support	2-29
2.9.1 Runtime support	2-29
2.9.2 General precompiler changes	2-29
2.9.3 Parameters supporting VS COBOL II	2-30
2.9.4 Parameters supporting COBOL 85 (Fujitsu and Hitachi)	2-31

2.1 About this chapter

This chapter describes enhancements to data communications functions. While most of the features apply to CA-IDMS/DC and CA-IDMS/UCF, some features have cross-product significance. The enhancements provide:

- Increased ease-of-use
- Increased processing flexibility
- Increased processing efficiency
- Support for new technology

A highlight of this release is a completely new, network-independent, database communications architecture that is integrated with the CA-IDMS environment and CAICCI, the common communications component of the CA90s Distributed Processing Services layer.

2.2 CA-IDMS database communications architecture

This section covers these topics:

- Redesign of the architecture
- Components
- Setting up your environment
- Cross DC/UCF region communications

2.2.1 New design

Provides increased functionality and efficiency: The CA-IDMS database communications architecture has been redesigned to provide additional functionality and services for both CA-IDMS/DB and DC/UCF processing. The new architecture uses common processing routines to satisfy all database requests regardless of where the request originates (navigational DML program, SQL application, CICS transaction).

Uses a client/server model The architecture was built using a client/server communications model that:

- Separates client processing from server processing
- Separates functional layers
- Isolates operating-system dependent code
- Provides a common environment within which to execute

Benefits: This new architecture offers these benefits:

- Allows for easy expansion of architecture to include support for future technology.
- Enhanced CA-IDMS/DB batch environment.
- Release 10.2 (or later) applications are upwardly compatible.
- Forwards user identification information to the server environment and ensures user validation is performed if necessary.
- Provides the framework for creating network-independent applications that can be distributed across different hardware and operating system environments.
- Is integrated with the Distributed Processing Services layer of CA90s and uses CAICCI, the common communications component of CA90s. CAICCI surrounds communications and network software so that applications are insulated from the specifics of the environment and still communicate the required information about the data request to the targeted platform.
- Provides for both local and remote data access without requiring additional code in programs or programmers knowing where and how to access the data in the communications network.

Cross DC/UCF region communications: A DC/UCF front-end system (or systems) can now access a DC/UCF back-end system (or systems) if both (or all) systems reside on the same mainframe. In the previous release, this type of communication required CA-IDMS/DDS. CA-IDMS/DDS is no longer required for this type of configuration. For more information, see 2.2.4, “Multiple DC/UCF region communications” on page 2-7, later in this section.

Note: Data access between CPUs requires CA-IDMS/DDS.

►►For more information on CA-IDMS/DDS see, Chapter 7, “CA-IDMS/DDS” on page 7-1.

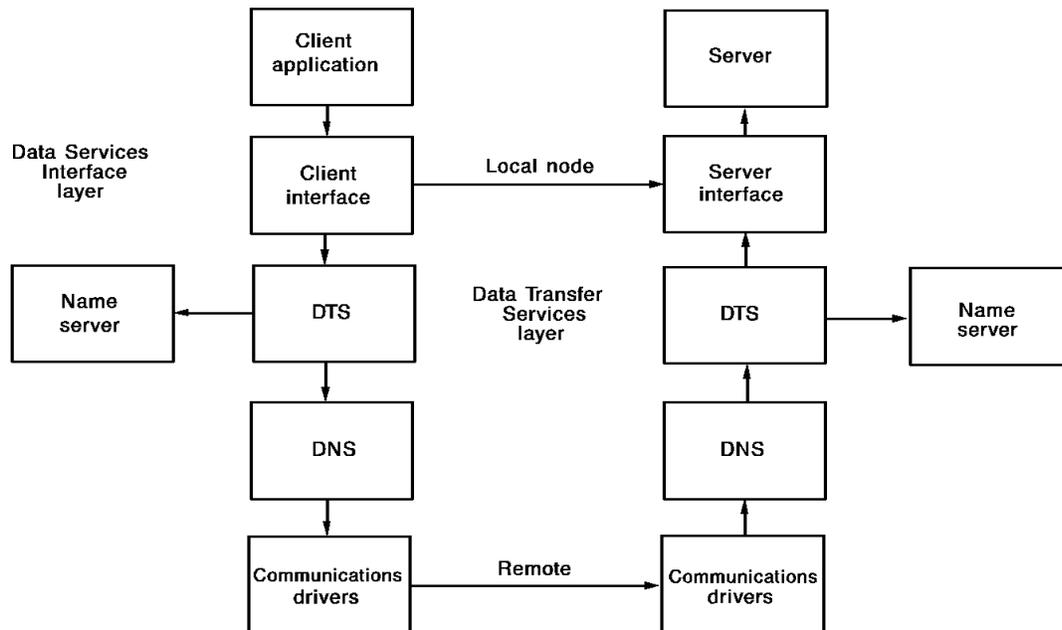
2.2.2 Components of the architecture

The CA-IDMS database communications architecture is a layered design that includes various software modules. Information about the data request, the location of data, and the type of communication to be used are obtained from these different modules.

The layers, or components, are as follows:

- Application program — Issues a request for data without regard to its location and the communications protocol required to access the data.
- Data services interface (DSI) — Accepts the request for data from the application and creates a data transfer service block (DTSB) that is sent to the data transfer services (DTS) layer. The data transfer service block contains information about the data, its format, and target resource.
- Data transfer services (DTS) — Accepts the DTS request and on a connect calls the name server table to determine the location of the target resource and which communications protocol will be used. The DTS layer passes the information from the name server to the DNS layer.
- Name server — Provides the means of determining the location of resources within the network. The name server accepts the name of the target resource from the DTS layer and looks for this name in a resource table. Each resource has an associated node. The name server looks for this node in a node table to determine which communications protocol will be used to access the data. The name server returns this information to the DTS layer.
- Distributed node services (DNS) — Manages communications for remote data access. The DNS layer establishes the communications session on the remote node using network drivers.
- Communications line drivers and access methods — Provide an application programming interface (API) based on IBM's APPC that manages the communications between environments. The drivers provide network independence by converting data formats and by mapping APPC verbs into the appropriate network protocol.

Database communications architecture



Strengths of the components: The data services interface (DSI) provides database independence while data transfer services (DTS) and distributed node services (DNS) guarantee that messages are routed in a manner transparent to the underlying network protocols. Name servers ensure that messages are delivered to the correct resource even if the resource has been moved to another platform.

Because of the flexibility of this architecture, you can implement applications that will survive changes in hardware and software environments.

2.2.3 Setting up your environment

To set up your DC/UCF environment (for local and remote data access on a single mainframe), you define:

- A resource table
- One or more nodes

You define a resource table and one or more nodes as part of system generation using two system generation statements. They are the RESOURCE TABLE and NODE statements.

NODE statement: In the NODE statement, each node on your CA-IDMS communications network is defined. In addition, the type of communication method used to reach the named node is specified.

RESOURCE TABLE statement: Using the RESOURCE TABLE statement, you identify databases available in your CA-IDMS communications network, and you associate them with the node on which they can be found. The RESOURCE TABLE

statement builds the resource table that is accessed by the Data Transfer Services (DTS) layer of the CA-IDMS communications architecture described earlier in this chapter.

Example: In this example, a resource table and two different nodes are defined to support application and database processing. Application processing will take place in one address space and the database processing for these applications will take place in another. Each node will be accessed using two different CA-IDMS central versions.

```
add node mistest
  cvnumber is 20  svc is 214;

add node misprod
  cvnumber is 30  svc is 215;

add resource table
  dbname is emptest via mistest
  dbname is empprod via misprod;
```

Displaying the resource table: You can display and vary the resource table using the DCMT DISPLAY/VARY RESOURCE TABLE DCMT command. For example:

```
dcmt display resource table
```

Displaying nodes: You can display the nodes defined to your system using this DCMT command:

```
dcmt display node
```

2.2.4 Multiple DC/UCF region communications

Using the CA-IDMS communications architecture, DC/UCF systems in the same mainframe can communicate with each other without CA-IDMS/DDS.

Note: CA-IDMS/DDS is still needed for communications across CPUs.

Setting up cross-system communications: You define multiple DC/UCF systems as part of system generation. You identify the resources and nodes involved in cross-system communications using the RESOURCE TABLE and NODE statements.

There are many ways to distribute database and application processing across DC/UCF systems. For example:

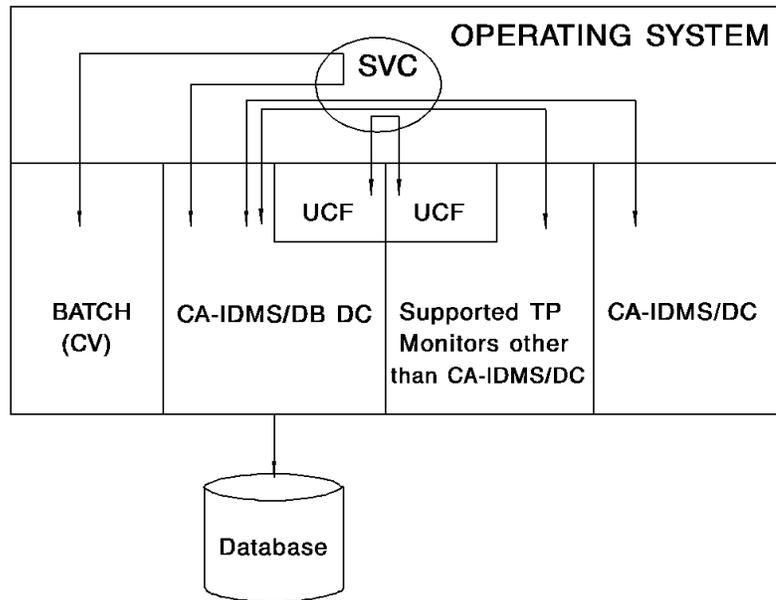
- You can define one DC/UCF system to manage the front-end processing (terminal management, etc.) of applications and another DC/UCF system to manage the CA-IDMS/DB database processing.

or

- You can distribute the front-end processing of applications across *multiple* DC/UCF systems and define other DC/UCF systems to manage the CA-IDMS/DB database processing for those applications. You might distribute applications based on geographic location, type of application, or size of application.

You decide the best way to set up multiple DC/UCF systems to meet the goals of your organization and maximize the use of your computing resources.

DC/UCF communications:



Advantages of using multiple regions: By distributing application and database processing across multiple DC/UCF systems, you minimize the constraints on system resources such as CPU and virtual storage.

Other benefits include:

- Applications access data without regard to its location or how it is accessed. No special application logic is required.
- Applications can be segregated based on geographic location.
- Database processing can be insulated from application processing.
- Applications can be insulated from each other.
- Operating system dispatching priorities can be optimized based on the types of applications being run.
- Exploitation of multi-CPU machines.

2.2.5 Enhanced CICS interface

The CA-IDMS CICS interface has been enhanced to provide the following:

- Integrated with new CA-IDMS database communications architecture (DTS/DNS)
- Command-level CICS components
- Use of standard CICS exits for tracking and recovery
- IDMSDEBUG trace facility
- Support for CICS versions 1.7 and above
- Support for SQL calls from CICS/CA-IDMS user applications
- Upward compatibility for previous releases of CA-IDMS

2.2.6 For further information

- **On defining DC/UCF systems**, see *CA-IDMS System Generation*
- **On managing DC/UCF systems**, see *CA-IDMS System Operations*
- **On CA-IDMS CICS processing**, see CA-IDMS installation manual for your operating system and *CA-IDMS System Operations*
- **On CA-IDMS/DDS**, see *CA-IDMS/DDS Design and Operations*

2.3 Operating system features

Extended addressing support (MVS, CMS, BS2000): Specific extended addressing options are now available to sites using MVS, CMS, and BS2000 operating systems that support extended addressing.

Eligible CA-IDMS/DB and CA-IDMS/DC control blocks (VIBs, dynamic LTEs and PDEs, etc.) and DC/UCF system nucleus modules now automatically reside above the 16Mb line.

Also, you can allocate storage above the 16Mb line for the following if your operating system supports extended addressing:

- Scratch storage
Specify SCRATCH IN XA STORAGE IS YES in the SYSTEM statement in system generation.
- Database buffers
- COBOL application storage

Note: You must use the most recent version of VS COBOL II to take advantage of extended addressing.

Advantages of extended addressing: When you use extended addressing, you can allocate very large buffer pools, thereby decreasing I/O.

When buffers, control blocks, or other programs reside above the 16Mb line, the result is more storage space available for use below the 16Mb line.

Examples: This example shows the system generation SYSTEM statement that instructs CA-IDMS/DB to allocate scratch storage above the 16 Mb line.

```
modify system system90
  scratch in xa storage is yes.
```

This example shows how to create an above-the-line storage pool reserved for database-related objects (excluding buffers).

```
add xa storage pool 254
  contains type (database).
```

This example shows the DMCL and system definitions required for database buffers to be acquired from IDMS storage.

```
create buffer default_buff
  ...
  cv mode buffer idms storage;
add xa storage pool 254
  contains type (system).
```

This example shows the definition required to acquire operating system storage for database buffers.

```
create buffer default_buff
...
cv mode buffer opsys storage;
```

For further information:

- **On defining the necessary system generation parameters in order to use extended addressing options**, see *CA-IDMS System Generation*

ESA dataspace: ESA operating environments provide dataspace support and buffering in the dataspace. For more information about ESA dataspace, see 'Exploitation of ESA dataspace', in Chapter 1, "Database" on page 1-1.

2.4 Profile support

What is a profile?: A profile is a set of attributes and options associated with users in a CA-IDMS environment. Profiles, intended for constructing the user's environment, are processed at signon time.

There are two levels of profiles:

- **User profiles** include attributes that apply to a user across all DC/UCF systems.
For example, you might define a user profile to include information that is system-independent and less likely to change, like an employee number, a group code, or a department code.
- **System profiles** include attributes that apply to users of a specific DC/UCF system.
For example, you might use a system profile to identify information that is specific to a DC/UCF system like the name of a dictionary and database a user can access or a command list to be executed.

You can use profiles to control a user's access to resources within and across DC/UCF systems. Profiles have been made available to replace the environment control functions of signon command lists in prior releases.

What are attributes?: Attributes are keywords and their associated values.

You can define any number and type of attributes. Additionally, CA-IDMS provides predefined attributes.

System-defined keywords: The following table lists the keywords provided by CA-IDMS.

Keyword	Definition
BREAK	Determines whether immediate-write messages will be received by a user while the user is signed on to a DC/UCF system
CASE	Specifies whether lower case letters are preserved on input: <ul style="list-style-type: none"> ▪ UPPER— On input, all alphabetic characters are translated to uppercase ▪ UPLOW— On input, no translation is performed
CLIST	Identifies a CLIST to be invoked when a user signs on to a DC/UCF system
DBNAME	Identifies the name of a database for a user's session

Keyword	Definition
DBNODE	Identifies the DC/UCF system that controls the database for a user's session
DICTNAME	Identifies the name of a dictionary for a user's session
DICTNODE	Identifies the DC/UCF system that controls the dictionary for a user's session
INSTCODE	Specifies an installation code for a user
LOADLIST	Identifies a load list for a user's session
MAPTYPE	Identifies the map type for a user's session
PRIORITY	Specifies the dispatching priority for a user
PRTDEST	Identifies the printer destination for a user's session
PRTCLASS	Identifies the print class for a user's session
SCHEMA	Identifies the name of an SQL schema for a user's session
TEST	Identifies the test version number for a user's session

User-defined attributes: You can define any number of attributes. For example, you might include the following attributes and keywords within a user profile:

- DEPT='MIS'
- TITLE='PROGRAMMER'
- LOCATION='HOUSTON'
- SHIFT='DAY'

2.4.1 Defining profiles

You define and maintain profiles with the CA-IDMS Command Facility using the CREATE PROFILE statement:

```
create system/user profile profile-name
  attribute attribute-keyword=attribute value
  override=yes/no
```

The OVERRIDE parameter specifies whether or not attributes can be changed. OVERRIDE=YES, the default, specifies that keywords and values can be changed by the user.

The special keyword INCLUDE can be used to chain profiles together.

Examples This example shows the definition of a **user profile** called JPDPROF.:

```
create user profile jpdprof
attributes groupcode='c0600'
           empid='1236'
           title='analyst' override=no
           location='boston';
```

CA-IDMS stores user profiles in the user catalog.

This example shows the definition of a **system profile** for an MIS production system.

```
create system profile misprod
attributes dictname='misdict'
           dbname='benefits'
           prtclass='47'
           prtdest='westwood';
```

CA-IDMS stores system profiles in the system dictionary.

This example shows how you can include one profile within another profile of the same type. The user profile HRGROUP is included in user profile HEAD_OFFICE.

```
create user profile head_office
attributes office='10'
           phone='617-329-7700';

create user profile hrgroup
attributes dept='5400'      override=no
           jobcode='1200'  override=no
           include 'head_office';
```

Substitution parameters: Three substitution parameters are available to facilitate the definition of generic profiles that can be shared by multiple users. The following substitution parameters are replaced with the values for the user and environment:

- **&USER** — replaced with the current user ID.
- **&SYSTEM** — replaced with the DC/UCF system name or 'BATCH' if running in local mode.
- **&GROUP** — replaced with the default group of the current user ID.

Example: This example shows the use of substitution parameters within a system profile:

- **&USER** will be substituted for the current user ID making this profile usable by all users in the HR group of MIS.
- The **INCLUDE** attribute will cause a profile specific to each user to be included.
- If the user-specific profile exists, it can change attribute values (if the attribute can be overridden) or add attributes to those specified in MIS-HR.
- If the user-specific profile doesn't exist, the **INCLUDE** is ignored.

```

create system profile mis_hr
attributes
  dictname='misdict'
  dbname='testhr'
  prtclass='16'
  test='16'
  schema=&user
  include=&user_prof;

create system profile jpd_prof
attributes
  schema='jpd1'
  case='upper';

```

2.4.2 Associating profiles with users

User profile: You use the CREATE/ALTER USER statement to associate a **user** profile with a specific user:

```

alter user jpd
  profile is jpdprof;

```

User profiles are available in local mode.

System profiles: You use the GRANT SIGNON statement to associate a **system** profile with a user:

```

grant signon
  on system system74
  to jpd
  profile is mis_hr;

```

System profiles are used with DC/UCF systems and *are not* available in local mode. If neither a user nor a system profile is associated with a user, CA-IDMS will automatically look for a system profile named DEFAULT.

What happens when a user signs on: When a user signs on to a DC/UCF system or is implicitly signed on in local mode, any user profile that exists for the user is retrieved from the user catalog. If signing on to a DC/UCF system and a system profile exists, it is retrieved from the system dictionary and merged with the user profile.

2.4.3 Displaying and accessing attributes

Interactively: The current profile for a user signed on to a DC/UCF system is displayed using the DCUF SHOW PROFILE command:

```

dcuf show profile

```

An individual attribute can be displayed using the same type of DCUF command in the form DCUF SHOW *keyword*:

```

dcuf show dictname

```

Attributes of the current session profile can be modified using the DCUF SET command:

```
dcuf set dictname mistest
```

You can change multiple attributes and add new attributes by invoking a system profile using the following DCUF command:

```
dcuf set profile mis_hr
```

Programmatically: You can access a profile programmatically by:

- Linking to the DCUF SHOW PROFILE command
- Calling the module IDMSIN01 and using function 2

You can add or change attributes programmatically by:

- Linking to the DCUF SET PROFILE command
- Calling the module IDMSIN01 and using function 3

2.4.4 Using profiles with nonterminal tasks

CA-IDMS/DC and CA-IDMS/UCF use profiles in the processing of nonterminal tasks. For example, if a user executes an application from a DC/UCF system that invokes a nonterminal task, the attributes assigned to that user are propagated to the nonterminal task.

2.4.5 For further information

- **On defining and accessing user profiles and securing both user and system profiles**, see *CA-IDMS Security Administration*
- **On defining and accessing system profiles**, see *CA-IDMS System Tasks and Operator Commands*
- **On invoking and using the IDMSIN01 module**, see *CA-IDMS Navigational DML Programming*

2.5 New numbered user exits

Exit 28 — Security preprocessing: This user exit allows you to examine security requests **before** they are processed.

You can use the exit to screen requests before implementing site-specific security requirements. You cannot use the exit to force the security system to accept a request.

Exit 29 — Security postprocessing: This user exit allows you to examine security requests **after** they are processed.

You can use the exit to check for security violations or to enforce security based on site-specific standards. You cannot override a security violation, however.

Exit 30 — Deadlock victim selection: This user exit allows you to implement an algorithm that determines which DC/UCF task involved in a deadlock will be terminated. The user exit is called after the deadlock detector scans all waiting tasks in the DC/UCF system and determines which of the tasks are truly deadlocked.

Note: The default algorithm CA-IDMS/DB uses terminates the most recently started transaction with the lowest dispatching priority.

Exit 31 — Transaction statistics: This user exit allows you to examine collected transaction statistics.

The exit is called by the transaction manager whenever statistics are written from the transaction block. The exit takes place after statistics are written but before the block is released.

For further information:

- **On user exits,** see *CA-IDMS System Operations*

2.6 Query device support

Query device support: Terminals that support extended attributes, such as highlighting and color, are now queried to determine which attributes the terminal supports. These terminals may now be defined to a DC/UCF system as 3279 devices.

At logon, the VTAM query reply structured field determines the actual extended attributes the terminal supports.

VTAM definitions may need to be changed to define terminals that support extended attributes. **CAUTION:**

Do not change any real 3278 device types to 3279 device types.

Applications using the CA-IDMS/DC and CA-IDMS/UCF online mapping facility can take advantage of extended terminal attributes when executing from a terminal that supports them.

2.7 DCMT and DCUF commands

2.7.1 Invoking DCMT and DCUF commands from programs

Invoke all DCMT and DCUF commands from programs: All DCMT and DCUF commands can now be invoked from application programs.

The same general programming requirements apply as in invoking certain DCMT VARY commands in prior releases.

User programs will continue to link to the program RHDCMT00 for DCMT commands and program RHDCUF00 for DCUF commands. Data is returned in the form of records to the scratch area. Programs can then read the display data from the scratch area.

There are minor changes to input parameters and a return code has been added.

For further information:

- **On programming requirements,** see *CA-IDMS System Tasks and Operator Commands*

2.7.2 New and modified DCMT and DCUF commands

There are new and modified DCMT and DCUF commands and DC/UCF tasks that support new CA-IDMS features.

New commands: The table below lists new DCMT and DCUF commands.

Command	Function
DCMT DISPLAY/VARY DEADLOCK	DCMT DISPLAY DEADLOCK displays the deadlock detection interval assigned at system generation on the SYSTEM statement. DCMT VARY DEADLOCK allows you to change the deadlock detection interval assigned to deadlocked tasks.
DCMT DISPLAY LOCK AREA/LTERM	Displays longterm and notify locks held by a logical terminal or displays locks held on an area by all logical terminals.
DCMT DISPLAY NODE	Displays the nodes defined to a DC/UCF system.

Command	Function
DCMT DISPLAY/VARY RESOURCE TABLE	DCMT DISPLAY RESOURCE TABLE displays the resource name table created from the system generation RESOURCE TABLE statement. DCMT VARY RESOURCE TABLE instructs CA-IDMS to load a new copy of the altered resource name table.
DCMT DISPLAY/VARY SEGMENT	DCMT DISPLAY SEGMENT displays information about areas in segments. DCMT VARY SEGMENT allows you to vary an area status and to take areas offline.
DCMT DISPLAY TRANSACTION	DCMT DISPLAY TRANSACTION displays information about database transactions.
DCMT VARY DBNAME TABLE NEW COPY	Instructs CA-IDMS/DB to load a new copy of the database name table.
DCMT VARY DMCL NEW COPY	Instructs CA-IDMS/DB to load a new copy of the DMCL.
DCMT WRITE STATISTICS	Writes current system and line statistics and histograms to the DC/UCF log file.
DCUF SET PROFILE	Allows you to alter your session profile.
DCUF SHOW PROFILE	Allows you to display all attributes of your session profile.

Modified commands: The table below lists modified DCMT and DCUF commands and DC/UCF tasks.

Command	Change
DCMT DISPLAY ACTIVE PROGRAMS	Was previously a part of the DCMT DISPLAY PROGRAM command and is now a separate command.
DCMT DISPLAY JOURNAL	Has a new JOURNAL parameter.
DCMT DISPLAY RUN UNIT	Displays only system run units and has new parameters: <ul style="list-style-type: none"> ■ SYSTEM ■ SECURITY ■ SQL LOADER ■ SQL SECURITY

Command	Change
DCMT VARY PROGRAM	Will now prompt you to select a specific program if more than one is found with the same name.
DCMT DISPLAY/VARY ACTIVE TASKS	Was previously a part of the DCMT DISPLAY TASKS command and is now a separate command.
DCMT DISPLAY/VARY CENTRAL VERSION	Deleted. Functionality provided through DISPLAY TRANSACTION and VARY UCF.
DCMT DISPLAY/VARY DATA-BASE	PROGRAMS parameter is deleted.
DCMT VARY AREA	Has new TRANSIENT RETRIEVAL parameter.
DCMT VARY BUFFER	Has new parameters: <ul style="list-style-type: none"> ■ OPEN/CLOSE ■ OPSYS/DC STORAGE ■ INITIAL/ADDITIONAL/MAXIMUM PAGES
DCMT VARY FILE	Has new ALLOCATE, DEALLOCATE (FORCE), DSNAME and dataspace parameters.
DCMT VARY JOURNAL	DRIVER parameter is deleted. RUN UNIT is now TRANSACTION.
DCMT VARY RUN UNIT	Now has all the parameters of DISPLAY RUN UNIT.
DCPASS Task	Deleted. You can now change passwords as part of the SIGNON task.

For further information:

- **On DCMT and DCUF commands, and DC/UCF tasks,** see *CA-IDMS System Tasks and Operator Commands*

2.8 System generation

There are both new system generation statements and parameters and new system generation compiler options. These are discussed below.

2.8.1 Changes to statements and parameters

New and modified statements and parameters: There are new system generation statement parameters that support Release 12.0 features. Additionally, some existing statements and parameters are now specified through other tools and have been deleted from system generation.

Statement	Parameter	Status/information
SYSTEM	SYSTEM ID is	New parameter that allows you to specify a nodename for each DC/UCF system you generate.
	AREA ACQUISITION THRESHOLD	New parameter that replaces the PRE-EMPTION THRESHOLD parameter. Allows you to specify whether CA-IDMS/DB will accumulate area locks when attempting to ready multiple areas for a single database transaction.
	DEADLOCK DETECTION INTERVAL	New parameter that allows you to specify the frequency with which the system will check for deadlocks.
	JOURNAL FRAGMENT INTERVAL	New parameter that allows you to specify the number of journal blocks to be written to the journal file before CA-IDMS/DB writes a dummy segment record to the journal file.

Statement	Parameter	Status/information
	JOURNAL TRANSACTION INTERVAL	New parameter that allows you to direct CA-IDMS/DB to defer journal I/O based on the number of active transactions running in your DC/UCF system.
	LOOKAROUND TIME	Deleted. No longer required.
	MULTIPLE SIGNONS is	Allows one user to signon to the DC/UCF system from multiple interactive or UCF terminals concurrently.
	PREEMPTION THRESHOLD	Deleted and replaced by AREA ACQUISITION THRESHOLD.
	REGISTRATION/NOREGISTRATION	Deleted and replaced by run unit security of the new security facility.
	RULOCKS	Deleted. Not required.
	RUNUNITS FOR	Added new options that allow you to specify two new types of run units that can be predefined to be initiated at startup.
	SCRATCH in XA STORAGE	New parameter that allows you to specify that the scratch area will use a 31-bit storage pool in an MVS/XA environment.
	STACKSIZE	Changed the default to 1200.
	SYSCTL is	Added SYSCTL as the default value for DDNAME, FILENAME, or LINKNAME.

Statement	Parameter	Status/information
	WARMSTRT/NOWARMSTRT	Deleted. No longer needed.
	UNDEFINED PROGRAM COUNT is	Allows you to specify one additional type of entry for which null PDEs should be allocated at system startup.
	XA STORAGE POOL is	Specifies the size of the 31-bit storage pool (storage pool 255) used for database processing.
ADSO	OPTIONAL/MANDATORY subclause to COBOL MOVE is	Allows you to specify whether application developers can override the default of NO on the COBOL MOVE IS clause during dialog generation.
DBNAME	All	Deleted and replaced by CREATE DBTABLE and CREATE DBNAME statements of the physical database definition process.
DDS	All	Deleted and replaced by the SYSTEM ID clause of the SYSTEM statement.
IDMS AREA	All	Deleted and information is now specified in the DMCL definition.
IDMS BUFFER	All	Deleted and information is now specified in the DMCL definition.
IDMS PROGRAM	All	Information is now specified using the system generation TASK statement and run unit security of the new security facility.

Statement	Parameter	Status/information
LOADLIST	LOADLIB is	You now specify the ddname or linkname of a test load library in the form <i>Vmnnn</i> . <i>Vnnnn</i> replaces the CDMSL <i>nnn</i> variable.
OLM	HELP PFKEY is	New clause that specifies the default program function key to be associated with the help function during run-time mapping sessions.
OLQ	SQL ACCESS is	A new optional clause that allows you to specify the type of SQL OLQ will use to access CA-IDMS/DB databases.
PROGRAM	Added new type of program	New program type added that is used with the SQL option of CA-IDMS/DB.
	MENU/NOMENU subclause of MAINLINE clause	Specifies whether a mainline dialog is displayed on the ADS menu screen.
	SECURITY CLASS	Security information is now specified using security classifications of the new security facility.
NODE	All	New statement that allows you to define nodes for other DC/UCF systems.
RESOURCE TABLE	All	New statement that allows you to define remote resources (databases) that will be accessed by systems.

Statement	Parameter	Status/information
RUNUNITS	SQL LOADER	Allows you to specify predefined run units for the SQL option and the new security facility.
	SQL SECURITY	
	DICTNAME IS	Deleted SYSTEM DEFAULT as the default. The default dictionary for loader run units is now specified in the RUNUNITS FOR clause of the SYSTEM statement.
TASK	AREA ACQUISITION	Specifies whether CA-IDMS/DB will accumulate area locks when attempting to ready multiple areas for a database transaction and not all areas are accessible.
	SECURITY CLASS	Security information is now defined using the new security facility.
USER	All	Deleted and users are now defined using the new security facility.
XA STORAGE POOL	DATABASE	Allows you to specify DATABASE as a type of storage that an XA storage pool can accommodate.
LTERM	INTERACTIVE BREAK/NOBREAK	Deleted. Information is now specified in a user or system profile.
	UPPER/UPLOW	Deleted. Information is now specified in a user or system profile.

Statement	Parameter	Status/information
DDS LINE	All	Deleted. Information is now specified using the teleprocessing network CCI LINE statement and the system generation NODE statement.
CCI LINE	All	New statement to define CAICCI CA90s component that CA-IDMS/DDS will use for cross CPU communications.
UCFLINE LINE PTERM	TYPE IS BULK	Added as a PTERM in addition to UCFTERM. Specifies a batch external request unit. You must define a BULK PTERM for each batch external request unit you will allow in your UCF system simultaneously within the same mainframe.

2.8.2 System generation compiler enhancements

Simplified commenting: You can use `*+ or` anywhere on an input line to indicate that the remainder of the line is a user comment. If the `*+ or` is coded in positions 1 and 2, the line is not echoed.

Additionally, you can use an `'* '` in positions 1 and 2 of an input line to indicate that the remainder of the line is a user comment. Lines containing `'* '` in positions 1 and 2 are echoed.

Non-quoted letters converted to upper case: All syntax components submitted to the system generation compiler that are not in quotes are converted to upper case. Users can submit statement components in lower case and non-quoted letters are automatically converted to upper case.

Availability of user exits: Support for user exits is now available for the system generation compiler. This support is the same as the support previously implemented for the IDD compiler.

For a list of the exits, see "Schema and subschema compiler enhancements" in Chapter 1, "Database" on page 1-1.

For further information:

- **On system generation parameters**, see *CA-IDMS System Generation*

2.9 Enhanced language support

2.9.1 Runtime support

The following new runtime support is available in Release 12.0:

- PL/I versions 2.1 and 2.2
- VS COBOL II (CMS and VSE in addition to the existing MVS support)
- COBOL 85 support (Fujitsu)
- COBOL 85 support (Hitachi)

2.9.2 General precompiler changes

New precompiler-directive statement: COPY IDMS

SUBSCHEMA-RECORD-BINDS allows you to copy a standard BIND RECORD for each CA-IDMS record in the subschema. The difference between this statement and the existing COPY IDMS SUBSCHEMA-BINDS is that the new statement does not initialize PROGRAM-NAME and issue a BIND RUN-UNIT.

New parameters: New sublanguage parameters are described in the table below.

Statement	Parameters	Comments
Task level: COMMIT FINISH ROLLBACK	All	Now available to COBOL, PL/I, and Assembler batch and CICS applications.
GET STORAGE	LOCATION IS ANY/BELOW	BELOW specifies that storage will be allocated below the 16Mb line. ANY specifies that storage is eligible for allocation above the 16Mb line. Supported by COBOL and PL/I precompilers only.
DC RETURN	IMMEDIATE	IMMEDIATE returns control to the DC/UCF system by bypassing intervening link levels. Supported by COBOL and PL/I precompilers only.

Statement	Parameters	Comments
LOAD TABLE and DELETE TABLE	DICTNODE/DICTNAME/LOADTABLE	These parameters allow you to specify which table is to be loaded or deleted. Supported by COBOL and PL/I precompilers only.
WRITE LOG	MESSAGE PREFIX IS 'message-prefix'	This parameter allows you to specify your own message prefix, so you can separate your messages from DC/UCF system messages. Supported by COBOL and PL/I precompilers only.

Examples: In the following VS COBOL II example, LOCATION IS BELOW forces CA-IDMS to acquire storage for WORK-RECORD below the 16Mb line. LOCATION IS ANY, the default, would make the storage eligible for acquisition above the 16Mb line.

```
GET STORAGE FOR WORK-RECORD
  LOCATION IS BELOW.
```

In the following VS COBOL II example, IMMEDIATE returns control to the DC/UCF system, bypassing intervening link levels.

```
DC RETURN NEXT TASK CODE IS 'EMP2'
  IMMEDIATE.
```

In the following VS COBOL II example, the DICTNAME parameter specifies that CA-IDMS/DB is to load the table ZIPCODE from the dictionary PRODDICT.

```
LOAD TABLE 'ZIPCODE' INTO ZIPCODE-AREA
  DICTNAME 'PRODDICT'.
```

In this example, the customized message prefix of 'EM' would result in the writing of the message EM999001, rather than the default DC999001.

```
WRITE LOG MESSAGE ID 999001
  MESSAGE PREFIX IS 'EM'.
```

2.9.3 Parameters supporting VS COBOL II

Optional parameter: In Release 12.0, the TO parameter of the GET STORAGE statement is optional.

DC RETURN issues a GOBACK: When the default precompiler option of 'COBOL=2' is set, a DC RETURN parameter with no operands will issue a GOBACK statement. This results in more efficient management of resources by CA-IDMS/DC.

2.9.4 Parameters supporting COBOL 85 (Fujitsu and Hitachi)

New parameters: The following table shows new COBOL 85 parameters for Fujitsu and Hitachi.

Statement	New parameter(s)
GET STORAGE	LENGTH <i>storage-data-length</i> POINTER <i>storage-data-location-pointer</i>
LOAD TABLE	POINTER <i>program-location-pointer</i>

The POINTER variables address acquired storage and loaded programs. POINTER and LENGTH parameters are recognized when COBOL 85 is specified as a precompiler option (JCL parameter 'COBOL=85').

Examples This is an example of a COBOL 85 (Fujitsu) program containing the LENGTH and POINTER parameters of the GET STORAGE and LOAD TABLE statements.:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PTRPRG.
ENVIRONMENT DIVISION.
DATA DIVISION.
BASED-STORAGE SECTION.
    01  A          BASED ON A-PTR.
        02  A1     PIC X(20).
        02  A2     PIC X(100).
WORKING-STORAGE SECTION.
    01  A-PTR      USAGE IS POINTER.
    01  A-LENGTH  PIC S9(8) COMP.
PROCEDURE DIVISION.
    MOVE FUNCTION LENG(A) TO A-LENGTH.
    GET STORAGE FOR A LENGTH A-LENGTH POINTER A-PTR.
    LOAD TABLE 'T' INTO A POINTER A-PTR.
```

This is an example of a COBOL 85 (Hitachi) program containing the LENGTH and POINTER parameters of the GET STORAGE and LOAD TABLE statements.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PTRPRG.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
    01  P-PTR      USAGE IS ADDRESS.
    01  A-LENGTH  PIC S9(8) COMP.
    01  A          ADDRESSED BY S-PTR.
        02  A1     PIC X(20).
        02  A2     PIC X(100).
PROCEDURE DIVISION.
    MOVE 120 TO A-LENGTH.
    GET STORAGE FOR A LENGTH A-LENGTH POINTER P-PTR.
    COMPUTE S-PTR = P-PTR.
    LOAD TABLE 'T' INTO A POINTER P-PTR.
```


Chapter 3. Security

- 3.1 About this chapter 3-3
- 3.2 Security facility features 3-4
- 3.3 Administering security 3-6
 - 3.3.1 Privileges 3-6
 - 3.3.2 Resources 3-7
 - 3.3.3 Authorization identifiers 3-8
- 3.4 Granting and revoking privileges 3-9
 - 3.4.1 Granting privileges 3-9
 - 3.4.2 Revoking privileges 3-10
- 3.5 Security display facility 3-11
- 3.6 For further information 3-15

3.1 About this chapter

This chapter describes the new security facility. Highlights of the security facility are:

- Centralization of security (both online and batch in local and central version)
- A new security interface
- An integrated interface to external security packages

3.2 Security facility features

What is the new security facility?: CA-IDMS now provides a new, centralized facility to control access to the resources in your CA-IDMS environment.

You can use the new facility to secure all CA-IDMS runtime components, when running under central version or when running in local mode. Definition-time protection is provided for system generation, physical database and SQL-defined entities.

You will continue to use the data dictionary to implement security for schemas, subschemas, and other dictionary entities.

Features of the security facility: The new security facility supports:

- Central storage of user definitions — You define all users (online and batch) in a single user catalog. The user catalog contains user IDs, passwords, group structures, and profiles. The user catalog includes the information previously stored in the USER-047 record and some of the information stored in the ACCESS-045 record in prior releases of the data dictionary.
- User groups — You can assign users to groups. You can establish security at the group level and it applies to all users in the group.
- Profiles — Profiles can be secured with the new security facility.
- Security classification and access lists — CA-IDMS provides two mechanisms for protecting resources:
 - Security classifications — are numbers associated with a group of resources that you want to secure. Security classifications are efficient for resources that require a single mode of access and are used to protect most DC/UCF system resources.
 - Access lists — are lists of operations that can be performed and are typically used for definition-time checking.
- Generic resource names (wildcards) — You can use wildcards to specify similar entities. For example, PROGRAM MRP * can be used to refer to all programs that begin with MRP.
- Central security administration — You administer security using a single tool. In addition, you use the consistent and simple SQL Data Control Language commands to secure resources.
- Security standards — CA-IDMS internal security follows the ANSI SQL database security standards and the Department of Defense (DOD) security model.

Security architecture: In addition to the above features, the security facility architecture provides:

- A standard security interface — All security requests are funneled through a central security interface. This permits uniform validation of security requests.

- An external security interface — CA-IDMS security enforcement can be replaced with external security packages such as CA-ACF2®, CA-TOP SECRET®, and RACF®.

When using an external security package to control access to an SQL-defined database, the DOD security model is used to enforce security.

External security interface features: Some features of the external security interface are:

- The external security interface uses CAISSF, the standard security facility of the CA90s Integration Services layer. Because CAISSF is a common component of the CA90s architecture it is integrated with all CA solutions.
- The CA-IDMS security facility is designed to support the existing rule base of external security packages so that you can continue using your existing rule base. The rules have been extended to cover new CA-IDMS resources.
- Any centrally secured CA-IDMS resource can alternatively be secured using an external security package.

3.3 Administering security

You control access to database and system resources by granting and revoking privileges to and from users. You grant users privileges that allow them to perform certain operations on resources. Conversely, you can revoke these privileges at any time.

You can use the CA-IDMS Command Facility to submit two types of statements to grant and revoke privileges:

- **GRANT** — You use the GRANT statement to grant a resource privilege to a particular user or group. Here is the syntax for the GRANT statement:

```
grant privilege
    on resource
    to authorization-identifier;
```

- **REVOKE** — You use the REVOKE statement to revoke a resource privilege that was previously granted to a particular user or group. Here is the syntax for the REVOKE statement:

```
revoke privilege
    on resource
    from authorization-identifier;
```

Privileges, resources, and authorization-identifiers are explained in the following sections.

Note: For examples of GRANT and REVOKE, see "Granting and revoking privileges", later in this chapter.

3.3.1 Privileges

What are privileges?:

A privilege is an explicitly-granted right to access a particular resource and perform a particular operation on that resource. You can grant three types of privileges to users:

- Definition privileges
- Access privileges
- Administration privileges

Definition privileges: Definition privileges can be granted for these operations:

- CREATE allows a user to add resource entity definitions.
- ALTER allows a user to modify or replace resource entity definitions.
- DROP allows a user to delete resource entity definitions.
- DISPLAY allows a user to display or punch resource entity definitions.
- USE allows a user to use the entity definition.

Note: Definition privileges apply only to entities on which you can perform one of the above operations.

Access privileges: There are three categories of access:

- **Execute access** allows a user to run an application, load module, or activity.
- **Table access** allows users to perform data manipulation operations on data tables (SQL option only).
- **Special access** allows users to perform specific operations within a database area or DC/UCF system:
 - DBAREAD allows a user to run read-only utilities against an area.
 - DBAWRITE allows a user to run update utilities against an area.
 - SIGNON allows a user to sign on to a specific DC/UCF system.

Administration privileges: There are three types of administrative privileges:

- **SYSADMIN** allows users to administer security on all resources within the security domain.
- **DBADMIN** allows users to administer security on database resources.
- **DCADMIN** allows users to administer security on DC/UCF system resources.

3.3.2 Resources

What are resources?: CA-IDMS/DB and CA-IDMS/DC resources are the entities in your environment to which you control access.

You must explicitly define some resources, like users and groups, using the CREATE resource statement. Other resources are implicitly defined simply by granting privileges.

Resources and entities: The table below lists resources and *some* of their associated entities that can be secured centrally.

Categories of resources	Types of resources
Global	Users Groups User profiles
Database	Areas Databases DBtables DMCLs Run units Schemas Tables
System	Activities (application functions) Applications DC/UCF systems Load modules Programs System profiles Tasks Queues

3.3.3 Authorization identifiers

What is an authorization identifier?: An authorization identifier (*authorization-id*) represents the **user** or a **group** you grant privileges to. An authorization identifier can be:

- **User identification codes (user IDs)** represent users of CA-IDMS/DB and CA-IDMS/DC resources. You define and maintain users and user IDs with the CREATE/ALTER/DROP USER statements.
- **Groups** represent a collection of users. You define and maintain a group with the CREATE/ALTER/DROP GROUP statements. Rules for groups are:
 - Users can belong to one or more groups.
 - Groups cannot belong to other groups.
 - Security granted to a group applies to all users in that group.
 - Every user belongs to the group PUBLIC.

Examples: This is an example of USER creation.

```
create user sas
  name 'samuel a simpson'
  status active;
```

This is an example of GROUP creation.

```
create group misgroup
  description 'contains the MIS application
  developers'
  add user jpd, sas, jks;
```

3.4 Granting and revoking privileges

3.4.1 Granting privileges

GRANT statement: You grant privileges using the GRANT statement.

Example: In this example, a system signon privilege is granted to user MDK.

```
grant signon on system74 to mdk;
```

WITH GRANT OPTION: You can optionally include the WITH GRANT OPTION on a GRANT statement. This allows the named user to grant the specified privilege (without the WITH GRANT OPTION) to other users or groups.

Example: In this example, user SAS is granted definition privileges on the resource TESTDMCL. User SAS is authorized to grant definition privileges on TESTDMCL to other users and groups.

```
grant define
  on testdmcl
  to sas
  with grant option;
```

Group privileges: You can grant privileges to groups as well as individual users.

Users you add to a group indirectly hold privileges granted to the group; users removed from the group lose all privileges held indirectly through the group.

Example: In this example, the group MISGROUP is granted DEFINE privileges on the resource TESTDMCL.

```
grant define on testdmcl to misgroup;
```

Using wildcards: In most cases, you can use wildcards (*) when granting and revoking privileges.

If your site uses naming standards, the use of wildcards allows you to secure groups of resources at a time. Fewer security definitions also take less storage space.

Example: In this example, EXECUTE privileges are granted to user SAS on all resources where the resource name is qualified by QA.

```
grant execute on qa.* to sas;
```

3.4.2 Revoking privileges

REVOKE statement: You revoke privileges by using the REVOKE statement. For each type of GRANT statement, there is a corresponding REVOKE statement.

Example: In this example, ALTER privileges on the resource QA.BENEFITS are revoked from user SAS.

```
revoke alter on qa.benefits from sas;
```

3.5 Security display facility

Use DISPLAY statements: You submit DISPLAY statements to the CA-IDMS Command Facility to report on security information.

There are several forms of the DISPLAY statement that allow you to report on various aspects of your security environment. In almost all cases, there is a corresponding DISPLAY statement for each CREATE statement in the security facility.

Format of the DISPLAY statement: Security display statements take this form:

```
DISPLAY resource-type resource-name  
WITH display-options
```

Display options: Here are some of the display options:

- All
- DETAILS
- GROUPS
- HISTORY
- PRIVILEGES
- USERS
- VERBS

Examples: You might issue the following DISPLAY statements to report on security information in your environment:

```
DISPLAY USER
```

```
DIS USER RAA WITH ALL AS SYNTAX;
  CREATE USER RAA
**    USER IS ACTIVE
      DESCRIPTION 'CHIEF ANALYST'
      NAME 'RICHARD A. ANALYST'
**    PASSWORD ASSIGNED
      PROFILE PAYROLL
**    CREATED 1991-07-18-12.51.17.187373 BY KKS
**    LAST UPDATED 1991-07-18-12.51.23.000517 BY KKS
**    WITHIN GROUP DEVTEAM
**    HOLDS DEFINE PRIVILEGES ON USER TAM
**    HOLDS DEFINE PRIVILEGES ON SYSTEM SYSTEM71
**    HOLDS SIGNON PRIVILEGES ON SYSTEM SYSTEM71
**    HOLDS DBADMIN PRIVILEGES ON DB DBA
**    HOLDS DBAWRITE PRIVILEGES ON AREA THESEGMENT.CORPTSP
**    HOLDS USE PRIVILEGES ON DMCL THEDMCL
**    HOLDS USE PRIVILEGES ON DBTABLE THEDBTABLE
**    HOLDS USE PRIVILEGES ON DB THEDB
**    HOLDS EXECUTE PRIVILEGES ON CATEGORY CATE3
**    HOLDS EXECUTE PRIVILEGES ON ACTIVITY DCMT.N002
**    HOLDS REFERENCES PRIVILEGES ON ACCESS MODULE THESCHEMA.THEAM
**    HOLDS EXECUTE PRIVILEGES ON ACCESS MODULE THESCHEMA.THEAM
**    HOLDS ALL PRIVILEGES ON TABLE THESCHEMA.THE*
**    HOLDS DISPLAY PRIVILEGES ON ACCESS MODULE THESCHEMA.*
**    HOLDS DISPLAY PRIVILEGES ON ACCESS MODULE THESCHEMA.DBA*
**    HOLDS REFERENCES PRIVILEGES ON TABLE THESCHEMA.THETABLE
**    HOLDS SELECT, INSERT, UPDATE PRIVILEGES ON TABLE THESCHEMA.THETABLE
**    HOLDS DISPLAY PRIVILEGES ON TABLE THESCHEMA.*
;

```

DISPLAY GROUP

```
DIS GROUP DEVTEAM WITH ALL AS SYNTAX;
  CREATE GROUP DEVTEAM
**    GROUP IS ACTIVE
      DESCRIPTION 'DEVELOPMENT TEAM'
**    CREATED 1991-07-18-12.51.23.762405 BY KKS
**    LAST UPDATED 1991-07-18-12.51.25.246924 BY KKS
      ADD USER RAA
      ADD USER HAR
;

```

DISPLAY RESOURCE DMCL

```
DISPLAY RESOURCE DMCL THEDMCL WITH ALL AS SYNTAX;
**    RESOURCE DMCL THEDMCL
**    CREATED 1991-07-18-12.52.22.592819 BY KKS
**    LAST UPDATED 1991-07-18-12.52.22.592819 BY KKS

```

DISPLAY PRIVILEGES ON RESOURCE DMCL

```
DISPLAY PRIVILEGES ON RESOURCE DMCL THEDMCL WITH ALL AS SYNTAX;
GRANT USE ON DMCL THEDMCL
**      CREATED 1991-07-18-12.52.23.660055 BY KKS
**      LAST UPDATED 1991-07-18-12.52.23.660055 BY KKS
      TO RAA
      ;
GRANT CREATE, ALTER, DROP, DISPLAY ON DMCL THEDMCL
**      CREATED 1991-07-18-12.52.23.234993 BY KKS
**      LAST UPDATED 1991-07-18-12.52.23.234993 BY KKS
      TO HAR
      ;
GRANT DEFINE ON DMCL THEDMCL
**      CREATED 1991-07-18-12.52.24.501862 BY KKS
**      LAST UPDATED 1991-07-18-12.52.24.501862 BY KKS
      TO JPD
      WITH GRANT OPTION
      ;
GRANT DEFINE ON DMCL THEDMCL
**      CREATED 1991-07-18-12.52.22.592819 BY KKS
**      LAST UPDATED 1991-07-18-12.52.22.592819 BY KKS
      TO TAM
      ;
```

DISPLAY RESOURCE SYSTEM

```
DISPLAY RESOURCE SYSTEM SYSTEM71 WITH ALL AS SYNTAX;
CREATE RESOURCE SYSTEM SYSTEM71
**      CREATED 1991-07-18-12.51.31.073777 BY KKS
**      LAST UPDATED 1991-07-18-12.51.31.073777 BY KKS
      ;
```

DISPLAY PRIVILEGES ON RESOURCE SYSTEM

```
DISPLAY PRIVILEGES ON RESOURCE SYSTEM SYSTEM71 WITH ALL AS SYNTAX;
GRANT DEFINE ON SYSTEM SYSTEM71
**      CREATED 1991-07-18-12.52.08.554659 BY KKS
**      LAST UPDATED 1991-07-18-12.52.10.229425 BY KKS
      TO RAA
      ;
GRANT SIGNON ON SYSTEM SYSTEM71
**      CREATED 1991-07-18-12.52.08.554659 BY KKS
**      LAST UPDATED 1991-07-18-12.52.10.229425 BY KKS
      TO RAA
      ;
GRANT DEFINE ON SYSTEM SYSTEM71
**      CREATED 1991-07-18-12.51.31.073777 BY KKS
**      LAST UPDATED 1991-07-18-12.52.08.554659 BY KKS
      TO HAR
      ;
GRANT SIGNON ON SYSTEM SYSTEM71
**      CREATED 1991-07-18-12.51.31.073777 BY KKS
**      LAST UPDATED 1991-07-18-12.52.08.554659 BY KKS
      TO HAR
      ;
GRANT DEFINE ON SYSTEM SYSTEM71
**      CREATED 1991-07-18-12.51.33.233208 BY KKS
**      LAST UPDATED 1991-07-18-12.52.10.690789 BY KKS
      TO JPD
      WITH GRANT OPTION
      ;
GRANT SIGNON ON SYSTEM SYSTEM71
**      CREATED 1991-07-18-12.52.08.554659 BY KKS
**      LAST UPDATED 1991-07-18-12.52.08.554659 BY KKS
      TO TAM
      ;
```

3.6 For further information

On defining and administering security and on using the security display facility,
see *CA-IDMS Security Administration*.

Chapter 4. CA-IDMS Performance Monitor

- 4.1 About this chapter 4-3
- 4.2 General enhancements 4-4
- 4.3 Real-Time monitor 4-5
 - 4.3.1 New screens 4-5
 - 4.3.2 New fields 4-6
- 4.4 Application monitor 4-7
 - 4.4.1 New screen 4-7
- 4.5 Interval monitor 4-8
 - 4.5.1 New screens 4-8
 - 4.5.2 New fields 4-9

4.1 About this chapter

This chapter describes enhancements to the CA-IDMS Performance Monitor. The enhancements provide:

- New installation options
- New data fields
- New screens

4.2 General enhancements

Dynamically turn logging on and off: With DC/UCF logging turned on (i.e., the #PMOPT macro AMDCLOG and/or IMDCLOG options specified as YES), you can dynamically turn on and off the writing of CA-IDMS Performance Monitor records to the DC log using fields on the Application and Interval Monitors.

Use the PMIM Status/Options screen in the Interval Monitor and the PMAM Status/Options screen in the Application Monitor to specify YES or NO on the Write to DClog field to turn logging on or off.

SMF type 30 (MVS only): You can specify SMF record type 30 as a parameter in #PMOPT, the macro that specifies run-time options.

SMFTY30=YES/NO specifies whether Type 30 SMF records (subtype 3 — step termination records) are created and written to the SMF file.

SMF type 30 is a field in the Application monitor and Interval monitor Status/Options screens.

Run unit becomes transaction: A database **run unit** is now called a **transaction** in all applicable fields. A transaction is a recoverable unit of work.

In fields that hold non-SQL database statistics, a database transaction is the equivalent of one run unit.

In fields that hold SQL database statistics, a database transaction can be equivalent to a run unit, but it may involve multiple run units managed as a single recovery unit.

4.3 Real-Time monitor

4.3.1 New screens

SQL Overview screen: This screen provides summary SQL information for the entire system since startup.

```
PM-R12.0 SYSTEM72      Computer Associates Intl. V72    91.198 13:15:57.04
CMD-->                Window :02
                        Refresh:10
```

02 SQL Overview

_ Row Level Activity

Fetch	Insert	Update	Delete
12	15	6	3

Sort Activity

Total #	Hi-Row	Low-Row	# Rows
500	165	10	824

Access Module	Number of SQL
Recompiles	Statements
10	65

Rows Current	Pages Requested
30	5

SQL Detail screen: This screen provides one line of information about each active SQL transaction.

```
PM-R12.0 SYSTEM72      Computer Associates Intl. V72    91.198 13:15:57.04
CMD-->                Window :02
                        Refresh:10
```

02 SQL Detail

Trans	Rows	Rows	Number	Hi-Row	Lo-Row	Rows	Pages	Pages	Pages
Number	Updated	Deleted	Sorts	Sorts	Sorts	Sorted	Written	Read	Requested
25	8	5	48	25	15	377	5	8	10
32	15	8	3	70	25	115	10	16	22

Note: More fields are available by paging the screen to the right.

4.3.2 New fields

Screen	New field	Description
Transaction Detail	Last Verb	Shows the number (in hexadecimal representation) of the last verb issued for a transaction
Buffer I/O	Page Size	Shows the page sizes of allocated buffers

4.5 Interval monitor

4.5.1 New screens

PMIM Status/Options screen: This screen displays Interval monitor options specified by the system administrator.

```
PM-R12.0 SYSTEM72      Computer Associates Intl. V72   91.198 13:15:57.04
CMD-->                               Window :02
```

```
02 05:35 OPT Interval Monitor Options in Effect
```

```
#PMOPT Assembly Date/Time          900907   09:13
```

```
* Online Options *
```

```
* Statistics Destinations *
```

PMIM Active	YES	Write DC Stats	YES
Online Active	YES	Write to DClog	YES
Max # Intervals	100	Write to SMF	NO
Size of Interval	5	SMF Buffer Size	8180
# of CDMSLIB Recs	10	SMF Record ID	230
# of DBkey Recs	5		
Site Save Allowed	YES		
User Save Allowed	YES		

SQL Information screen: This screen displays an overview of SQL activity by interval.

```
PM-R12.0 SYSTEM72      Computer Associates Intl. V72   91.198 13:15:57.04
CMD-->                               Window :02
```

```
02 05:35 SQL SQL Information
```

	Start Time	Rows Fetched	Rows Inserted	Rows Updated	Rows Deleted	Total Sorts	Hi-Row Sorts	Lo-Row Sorts	i> Rows Sorted
_	15:35	250	150	201	75	800	210	10	953
_	15:40	293	161	212	89	820	224	22	961
_	15:45	288	175	230	111	835	240	36	978
_	15:50	267	151	211	95	811	242	15	953
_	15:55	247	168	228	113	827	222	37	968
_	16:00	323	179	239	123	840	236	49	981
_	16:05	264	152	226	109	827	230	33	960
_	16:10	273	164	240	111	831	241	31	953

Note: More fields are available by paging the screen to the right. More intervals are available by paging the screen down.

Specific SQL Information screen: This screen displays SQL statistics for a specific interval.

```
PM-R12.0 SYSTEM72      Computer Associates Intl. V72    91.198 13:15:57.045
CMD-->                Window :02
```

```
02 13:55 SSQ Specific SQL Information
```

Row Level Information		Statistic Information	
Fetched	175	Select Locks	18
Inserted	230	Update Locks	31
Updated	111	Pages Read	42
Deleted	76	Pages Written	39
		Pages Requested	43
Sort Information		CALC With Overflow	22
# of Sorts	835	CALC No Overflow	230
High Rows	240	VIA With Overflow	31
Low Rows	36	VIA No Overflow	176
# Rows Sorted	977	Rows Requested	225
		Rows Current of Trans	42
Access Module Information		Total # of DBMS Calls	900
Recompiles	8	# of Fragments Stored	53
SQL Statement Information			
# Processed	598		

4.5.2 New fields

Screen	New field	Description
Interval Information Specific Interval Information	# Runaway	Shows the number of times the system encountered a task that exceeded the system limits between I/O activities
Interval Information Specific Interval Information	# Times SOS	Shows the number of times the system has run short on storage

Chapter 5. CA-ADS and Mapping Facility

- 5.1 About this chapter 5-3
- 5.2 Integration with centralized security 5-4
- 5.3 CUA-style user interface 5-5
- 5.4 Enhanced compiler support 5-7
- 5.5 Mapping enhancements 5-9
- 5.6 Support for century date variables and built-in functions 5-15
- 5.7 Numeric test 5-16
- 5.8 Trailing sign BIF support 5-17
- 5.9 READY NOREADY 5-18

5.1 About this chapter

These new CA-ADS and mapping facility features are discussed in this chapter:

- Integration with centralized security
- New CUA-style user interface
- Enhanced compiler support
- Mapping enhancements
- Support for century date variables and built-in functions
- Numeric test
- Trailing sign support
- NOREADY option for subschema areas

5.2 Integration with centralized security

Access to the CA-ADS runtime system is controlled through the centralized security facility. Entity definitions, such as dialog and record, continue to be under dictionary control.

►► See Chapter 3, “Security” on page 3-1 for further information on the security facility.

For further information

- **On the security facility**, see *CA-IDMS Security Administration*

- The second screen shows the half-screen help text overlaying the screen from which you requested the help. The page, scroll, and return options are listed below the help text.

Note: Help can be either half-screen or full-screen.

Placing the cursor in a field:

Options and Directives

Type and select each option and directive. Then Enter.

```

Dialog . . . . . : ADDEMP
Version . . . . . : 1

Message prefix . . . . . DC
Autostatus record . . . . . ADSO-STAT-DEF-REC
Version . . . . . 1

Options and directives . . . . .
- Mainline dialog
- Symbol table is enabled
/ Diagnostic table is enabled
/ Entry point is premap
- COBOL moves are enabled
/ Activity logging
/ Retrieval locks are kept
/ Autostatus is enabled
    
```

Enter F1=Help F3=Exit F4=PrevStep F5=NextStep

Viewing help in a half-screen format:

Specify MAINLINE if the dialog will be invoked from the CA-IDMS/DC prompt or by an APPC (send-receive option) request.

Mainline dialogs are potentially eligible to appear on the ADS MENU screen.

```

Return F3 _____ Page F7/F8 _____ Scroll: 010 _____
Options and directives . . . . .
- Mainline dialog
- Symbol table is enabled
/ Diagnostic table is enabled
/ Entry point is premap
- COBOL moves are enabled
/ Activity logging
/ Retrieval locks are kept
/ Autostatus is enabled
    
```

Enter F1=Help F3=Exit F4=PrevStep F5=NextStep

5.4 Enhanced compiler support

Copy function: A copy function, similar to that currently available for maps, is provided for dialogs and applications. You can now create a new dialog or application based on a previously-created dialog or application by using the copy option and naming existing and new dialogs or applications.

This option is available on the dialog compiler and application compiler Main Menu screens.

Copy function of the Add pull-down menu:

Add	Modify	Delete	Compile	Display	Switch
copy from					
Name _____			CA-ADS Online Dialog Compiler		
Version 1			puter Associates International, Inc.		
-----			nter or select an action.		
F3=Exit					
Dialog name			ADDEMP		
Dialog version			1		
Dictionary name			_____		
Dictionary node			_____		
Screen 1			1. General options		
			2. Assign maps		
			3. Assign database		
			4. Assign records and tables		
			5. Assign process modules		
Command ==>					
Enter F1=Help F3=Exit F10=Action					

Browse capability: You can now browse maps, similar to current dialog and application browsing capability. You can specify Browse from the MAPC (map compiler) Main Menu screen.

Browse function of the Display pull-down menu:

5.4 Enhanced compiler support

Add	Compile	Delete	Display	Switch
			1	1. Browse
				2. Summary
				3. Image
	Comp		-----	p Compiler
			F3=Exit	rnational, Inc.

Map Name: ADDEPLM Version 1 Dictionary Name TSTDICT Node _____

- Screen _
1. General Options
 2. Map-Level Help Text Definition
 3. Associated Records
 4. Layout
 5. Field Definition

Command ==> _____
Enter F1=Help F3=Exit F10=Action

Improved compiler error management: The application programmer can exit to edit text using full-screen presentation with search capabilities.

5.5 Mapping enhancements

Automatic screen painter: A screen painter is available to automatically create screens. This means that you can create a standard screen quickly and easily.

To automatically create screens:

1. On the Main Menu screen, specify a name for the map you are creating.
2. On the Associated Records screen, specify the records to use in creating the map.
3. On the Automatic Screen Painter screen, select fields you want to appear on the map.
4. Press PF5 to create and display the screen.

Specifying records used to create the screen:

Map name:	Associated Records	Page 1 of 1
Record name	Ver	Role name
1 EMPLOYEE	1	Drop (/)
2		—
3		—
4		—
5		—
6		—
7		—

DC350201 Map options processed successfully.

F1=Help F3=Exit F4=PrevStep F5=NextStep F6=Preview F7=PrevPage
F8=NextPage F9=Autopaint

Selecting fields to appear on the screen:

Map name: ADDEMLM	Automatic Screen Painter	Page 1 of 2
Select (/)	Element Level and Name	Occurs
01	EMPLOYEE VERSION 0001	
/	02 EMP-ID	
	02 EMP-NAME	
7	03 EMP-FIRST-NAME	
/	03 EMP-LAST-NAME	
	02 EMP-ADDRESS	
7	03 EMP-STREET	
/	03 EMP-CITY	
/	03 EMP-STATE	
	03 EMP-ZIP	
7	04 EMP-ZIP-FIRST-FIVE	
	04 EMP-ZIP-LAST-FOUR	
7	02 EMP-PHONE	
/	02 STATUS	
/	02 SS-NUMBER	

DC351901 Select the fields which are to appear on the screen.

F1=Help F3=Exit F4=PrevStep F5=NextStep F7=PrevPage F8=NextPage

Displaying the new screen:

```

;EMP-ID ;____*
;EMP-FIRST-NAME ;_____ *
;EMP-LAST-NAME ;_____ *
;EMP-STREET ;_____ *
;EMP-CITY ;_____ *
;EMP-STATE ;__ *
;EMP-ZIP-FIRST-FIVE ;____ *
;EMP-PHONE ;_____ *
;STATUS ;__ *
;SS-NUMBER ;_____ *
...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+

```

Enter F1=Help F2=Select F3=Exit F4=PrevStep F5=NextStep F6=Preview
F8=Bottom F9=SetCursor F10=Deselect F11=AltKeys

Function keys for editing text: You can use function keys to move, copy, or delete fields, lines or blocks of text on the Format screen of the mapping compiler. This makes it easier to format the map.

Sample Layout screen with alternate set of PF keys displayed:

```

;DEPT-ID-0410      ;____*
;DEPT-NAME-0410   ;_____*
```

```

;DEPT-HEAD-ID-0410 ;____*
%FUNCTION ;_____*
```

```

...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
```

```

F1=Help F2=Mark F3=Copy F4=Move F5=Delete F6=Preview F8=Bottom
F9=SetCursor F10=ClrMark F11=MainKeys
```

Help facility: A help facility is built into mapping and transparent to the user program. The user specifies text in a data dictionary module and associates the module with either the map or a map field. At runtime, the end user requests help by pressing a PF key. If the cursor is currently on a field for which help has been defined, **field** help will be presented. If the cursor is not on a field, or if there is no field help defined, **map** help will be presented.

Note: The default PF help key for the mapping facility is PF1. If you use PF1 for another function in any of your applications, change the default PF help key for the mapping facility on the system generation OLM statement.

When the user exits from help, the map is restored to its previous state; all attribute bytes are set to what they were before the user went to the help screen; all data entered remains on the screen.

Example: This is an example of the creation and use of help:

1. First screen — You add field-level help text (in the form of a help module called EMP-ID-HELP) to the dictionary using the IDD editor.
2. Second screen — You associate the help module EMP-ID-HELP with the map ADDEMPLM.
3. Third screen — The field-level help as it would appear when accessed by the user.

Creating the help text:

5.5 Mapping enhancements

IDD 12.0 ONLINE NO ERRORS DICT=TSTDICT 1/6
ADD MOD EMP-ID-HELP LANGUAGE IS HELP MODULE SOURCE FOLLOWS

Enter the 4 character numeric employee identifier.

To browse employees by name, press F4.
MSEND.

Associating the help module with a map:

Map name: ADDEMLM	Map-Level Version: 1	Help Text Definition	Page 1 of 1
Help name:	Help key: PF01		Drop Help (/) _
Element name In record	EMP-ID EMPLOYEE		Subscript Version

Window format 1 1. Half 2. Full

Origin of help text . . 1 1. No text
2. Module **EMP-ID-HELP** _____
Version _____

DC366306 Select help text options

Enter F1=Help F3=Exit F4=Prev F5=Next F6=Preview

Accessing the new help text:

EMP-ID
EMP-FIRST-NAME
EMP-LAST-NAME
EMP-STREET
EMP-CITY
EMP-STATE

Enter the 4 character numeric employee identifier.
To browse employees by name, press F4.

Return F3 _____ Page F7/F8 _____ Scroll: 010 _____

5.5 Mapping enhancements

		Map Read/Write Options	Page 2 of 6
Map name: ADDEMLM			
	Element name	EMP-ID	Subscript
	In record	EMPLOYEE	Version 1
Map Read options	Transmit data entry (/) /	
	Zero when null (/) /	
	Translate to upper case (/) /	
	Justify data. 1	1. Left 2. Right
	Pad character format . Display _	
	Hexidecimal _	
Map Write options	Blank when zero (/) _	
	Underscore blank fields (/) 7	
	Display without trailing blanks _	
	Set modified data tag (/) _	
	Transmit. . . 1	1. Data and attribute byte	3. Erase field
		2. Attribute byte only	4. Nothing
DC353401 Select input/output edit options.			
F1=Help F3=Exit F4=PrevStep F5=NextStep F6=Preview F7=PrevPage			
F8=NextPage			

5.6 Support for century date variables and built-in functions

The current DATE and JULIAN field names return the current date (in Gregorian and Julian) without the century portion of the year. New system-supplied variables will return the current date with the century included:

- **DATEX variable** — returns an 8-byte displayable Gregorian date of the form `yyyymmdd`
- **JULIANX variable** — returns a 4-byte packed number/date of the form `yyyddd+`

Corresponding built-in functions allow you to replace the six-byte no-century dates with eight-byte dates including century. In the BIF, the value 8 replaces the value 6 (where appropriate), and an X (for extended) is added to the keyword. For example:

`TODAY(ft)` becomes `TODAYX(ft)`

`CGDATE(cdate-6)` becomes `CGDATEX(cdate-8)`

`WEEKDAY(date-6,ft)` becomes `WEEKDAYX(date-8,ft)`

`DATECHG(date-6,ft,ft)` becomes `DATECHGX(date-8,ft,ft)`

5.7 Numeric test

You can test to see if a field contains numeric data prior to moving the contents of that field. This protects automatic data conversion.

Example

The example below shows the numeric test of an alphanumeric field prior to moving the contents of that field to a numeric variable.

```
if numeric (emp-id)
    move emp-id to (emp-numeric-id).
else
    display msg text is 'emp-id is not numeric'.
```

5.8 Trailing sign BIF support

Through three new built-in functions, ADS now supports trailing sign characters for display format fields. Specifically, the BIFs allow you to:

- Validate the format of COBOL-created fields before conversion
- Convert fields from the COBOL format to the ADS format
- Convert fields from the ADS format to the COBOL format

Validating COBOL fields and converting to ADS: The name you use to invoke validation is GOODTRAILING. The name you use to invoke conversion from COBOL to ADS is TRAILING-TO-ZONED.

Example:

```
if (goodtrailing(xyz)) then
    trailing-to-zoned(xyz).
else
    call flderror.
```

5.9 READY NOREADY

You can specify that an area of the subschema named in the dialog not be readied. As a result, the area is flagged NOREADY in the ready area table (RAT). Previously, if you did not explicitly ready an area of the subschema, the area would automatically be readied in the default usage mode for the subschema.

READY NOREADY eliminates unnecessary READY AREA overhead. It allows ADS programmers to avoid readying an area that will not be accessed by the ADS dialog.

Example: In the following example, area EMP-DEMO-REGION is readied with a SHARED UPDATE status. Area INS-DEMO-REGION is not readied, as specified by NOREADY.

```
ready emp-demo-region usage-mode update.  
ready ins-demo-region usage-mode noready.
```

Chapter 6. CA-Culprit, CA-OLQ, and CA-ICMS

- 6.1 About this chapter 6-3
- 6.2 Overview 6-4
- 6.3 CA-Culprit 6-5
 - 6.3.1 Double word binary support 6-5
- 6.4 For further information 6-6
- 6.5 CA-OLQ 6-7
 - 6.5.1 Extended selection criteria 6-7
- 6.6 For further information 6-9
- 6.7 CA-ICMS 6-10

6.1 About this chapter

This chapter describes enhancements to:

- CA-Culprit
- CA-OLQ
- CA-ICMS

Discussion of enhancements includes new features and upward compatibility. Support for the CA-IDMS/SQL Option is discussed in the second half of the document, which starts at Chapter 8.

6.2 Overview

CA-Culprit, CA-OLQ, and CA-ICMS are upward compatible with the CA-IDMS Release 12.0 product line.

Also, these products support the CA-IDMS/SQL Option. The SQL option provides statements to define relational databases and access both SQL-defined and non SQL-defined databases.

The SQL option meets ANSI standards and provides full SQL functionality. The SQL option also offers SQL extended features that provide additional options for data definition, data manipulation, and data control.

►► For more information about the CA-IDMS/SQL Option, see the second half of the document, which starts at Chapter 8.

6.3 CA-Culprit

6.3.1 Double word binary support

The maximum length for a binary field is now 8 bytes to support long integers or double word binary integers.

You use the REC card to define the field length.

Example: The example below defines the DATE field as 8 characters long with a data type of 1 which indicates a binary field. The 20 indicates the start position.

```
REC date 20 8 1
```

6.4 For further information

- **On defining double word binary fields**, see *CA-CULPRIT User Guide*

6.5 CA-OLQ

6.5.1 Extended selection criteria

You can define additional selection criteria for a column using a new screen: the Selection Criteria screen.

The Selection Criteria screen works like the Selection criteria field of the Column Select screen, enabling you to enter simple and compound expressions. In addition to specifying simple and compound expressions on the Selection Criteria screen, you can also define:

- Logical record keywords
- Criteria expressions for subscripted fields

Column select screen: The Column select screen below begins to define selection criteria to list department heads located in certain states earning within a specified salary range and who have a start date of 1988. Because the selection criteria is lengthy, it does not completely fit on the Column Select screen. The Selection Criteria screen is selected to continue specifying the criteria.

```

CA-OLQ Release 12.0                                     *** Column Select ***
→                                                    Page    1 of    1
124000 Select columns, specify selection criteria and press the ENTER key

Columns currently selected:    0          Selection criteria
DEPARTMENT
- 02 DEPT-ID                  *
- 02 DEPT-NAME
- 02 DEPT-HEAD-ID
EMPLOYEE
- 02 EMP-ID                  *
- 02 EMP-NAME
- 03 EMP-FIRST-NAME          *
- 03 EMP-LAST-NAME           *
- 02 EMP-ADDRESS
- 03 EMP-STREET
- 03 EMP-CITY
- 03 EMP-STATE                *
- 03 EMP-ZIP
- 02 SALARY-AMOUNT           *

```

Additional Selection Criteria: dept-head-id eq emp-id
and (emp-last-name matches 's' or emp-last-name matches 'c')
and (emp-state eq 'ny' or emp-state eq 'nj') and (salary-amount
Proceed to Selection Criteria Screen? Y Y/N

```

1=HELP          3=QUIT          4=MESSAGE          6=MENU          PA2=REFRESH

```

Continuation of selection criteria: On the Selection Criteria screen, continue specifying selection criteria.

CA-OLQ Release 12.0 ***Selection Criteria***
→ Line 1 of 1

146000 Type in selection criteria, and press the ENTER key.
Please Enter Additional Selection Criteria:
DEPT-HEAD-ID EQ EMP-ID AND (EMP-LAST-NAME MATCHES 'S'
OR EMP-LAST-NAME MATCHES 'C')
AND (EMP-STATE EQ 'NY' OR EMP-STATE EQ 'NJ') AND (SALARY-AMOUNT
gt 40000 and salary-amount lt 100000) and (start-year gt 88)

1=HELP 3=QUIT 4=MESSAGE 6=MENU PA2=REFRESH

6.6 For further information

- On using the Selection Criteria screen, see *CA-OLQ Reference*

6.7 CA-ICMS

In addition to providing support for the CA-IDMS/SQL Option, CA-ICMS is upwardly compatible with Release 12.0.

►►For more information on CA-ICMS and the CA-IDMS/SQL Option, see the second half of the document, which starts at chapter 8.

Chapter 7. CA-IDMS/DDS

7.1 About this chapter	7-3
7.2 CA-IDMS/DDS enhancements	7-4
7.3 For further information	7-6

7.1 About this chapter

This chapter summarizes the enhancements to the CA-IDMS/DDS environment for Release 12.0 as follows:

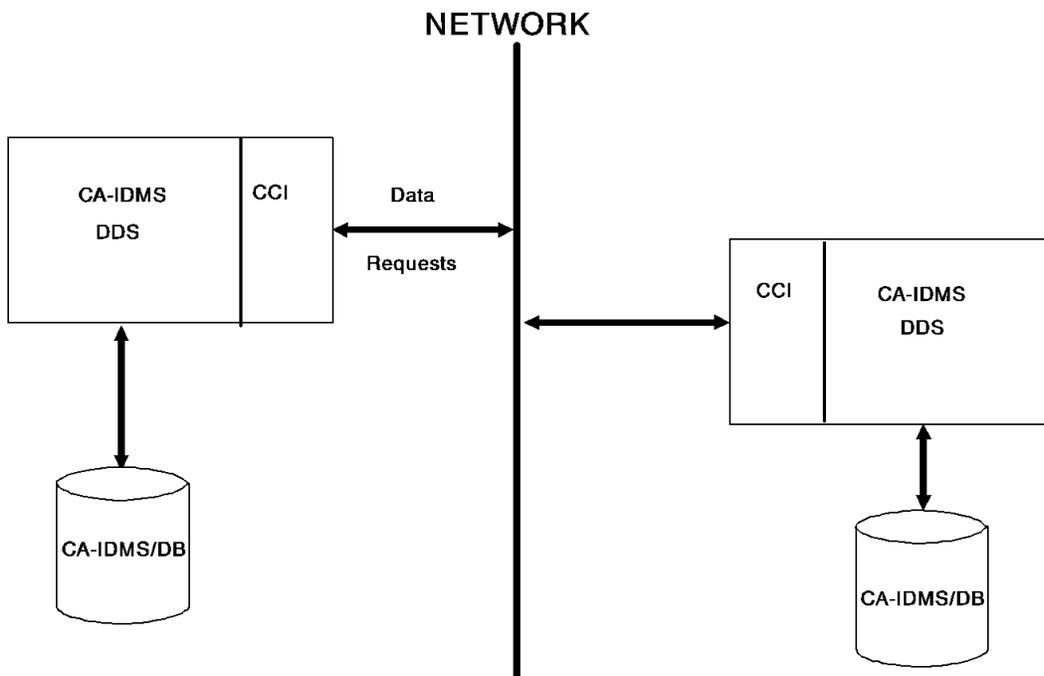
- CA-IDMS/DDS enhancements
- Defining CA-IDMS/DDS
- Accessing data using CA-IDMS/DDS
- Maintaining CA-IDMS/DDS

7.2 CA-IDMS/DDS enhancements

General enhancements: With cross DC/UCF region communications an integral part of the CA-IDMS database communications architecture, CA-IDMS/DDS is needed primarily to allow DC/UCF systems that are located on different CPUs to communicate with each other. In addition to being part of the new CA-IDMS database communications architecture, CA-IDMS/DDS has been enhanced to provide the following:

- Integration with the CA90s Common Communications Interface (CAICCI)
- More efficient communications
- Establish the foundation for cross-platform communication

Integrated with CAICCI



Defining CA-IDMS/DDS: Once CA-IDMS/DDS is installed, to define remote DC/UCF systems, (that will be accessed by a specific DC/UCF system) take these steps:

1. Define the remote resources to be accessed using the system generation RESOURCE TABLE and NODE statements
2. Define a CCI line using the system generation CCI LINE statement

Accessing data using CA-IDMS/DDS: A CA-IDMS program requiring access to a database located on another CPU needs no additional or special programming logic to access the database. The programmer does not need to know where the database is located or how it will be accessed.

Depending upon how you configure your CA-IDMS/DDS environment, the program need only identify the name of the database it will access. A node name, while still supported, is not required.

The CA-IDMS communications architecture determines where and how to access the target system.

Maintaining CA-IDMS/DDS: To modify the location of a database accessed by CA-IDMS/DDS, modify the system generation RESOURCE TABLE statement and regenerate the system definition. Then, use the DCMT VARY RESOURCE TABLE statement to access the modified resource table.

7.3 For further information

- **On defining and maintaining CA-IDMS/DDS**, see *CA-IDMS/DDS Design and Operations*

Chapter 8. Introduction to SQL-Defined Databases

- 8.1 What is an SQL-defined database? 8-3
- 8.2 ANSI and FIPS support 8-4
- 8.3 Benefits and features 8-5
- 8.4 Components 8-7
- 8.5 SQL as a language 8-8
- 8.6 Tables, rows, columns 8-9
- 8.7 Schemas and views 8-10
- 8.8 Table operations 8-12
- 8.9 Integrity and constraints 8-13
- 8.10 Storing SQL definitions 8-15
- 8.11 For further information 8-16

8.1 What is an SQL-defined database?

A CA-IDMS/DB SQL-defined database is a relational database that supports Structured Query Language (SQL) for database definition and data manipulation.

The database is made up of tables containing rows and columns. Tables are combined into schemas. Values in a table can be limited through constraints. A portion of a table (or multiple tables) can be defined as a view.

An SQL-defined database is defined using logical SQL data definition language statements to create tables, indexes, calc keys, schemas, referential constraints, and views.

The physical database for an SQL-defined database is defined using the same physical database statements as a non-SQL defined database.

8.2 ANSI and FIPS support

CA-IDMS/DB supports both ANSI (as defined in ANSI X3.135-1989 (Rev) and X3.168-1989) and FIPS SQL implementation. It also provides a number of extensions to basic SQL for added functionality.

►► For more information about extensions, see Chapter 11, “SQL Extended Features” on page 11-1.

8.3 Benefits and features

An SQL-defined database gives you many advantages.

Tabular representation: Data in tabular form is easily understood and can be accessed quickly and easily through SQL syntax.

Structural flexibility: The SQL data manipulation language isolates the user from knowledge of the underlying database structure. Existing database structure is easy to change through SQL statements without affecting existing application programs.

Program insulation: When applications are designed, programmers do not have to be aware of the details underlying the physical database structure. Program coding and database design and implementation can therefore proceed simultaneously.

Because of the inherent physical database independence of SQL, any query can be coded through the SQL DML language with no knowledge of the physical database storage method.

Because programs do not contain explicit data access logic, they are insulated from database changes.

Access to SQL-defined and non-SQL defined databases: Programs can use SQL data manipulation statements to access either an SQL-defined or an existing non-SQL database. In addition, a single SQL data manipulation statement can access data from both an SQL defined database and a non-SQL defined database.

In this way, investments in existing applications can be preserved while taking advantage of SQL technology.

Data integrity and security: Both data integrity and security are enforced at the DBMS level. This ensures that such rules cannot be inadvertently or deliberately violated by the application program or an interactive user. This also means that application code does not have to be written to ensure the integrity of data values.

High performance: CA-IDMS/SQL Option uses the high-performance CA-IDMS/DB database engine. This engine, with its physical tuning options, has proved itself effective in supporting high-volume production applications.

Recovery: Automatic recovery capabilities are available should a program or system terminate abnormally.

Fault tolerance: Fault-tolerance is built into the product to help insulate the system from hardware and software errors

Dynamic database maintenance: You can modify databases while they are online. For example, you can add columns to a table and create or drop indexes on a table.

Utilities: All utilities necessary for a production-power DBMS are provided.

SQL extensions for performance: A number of CA-IDMS/DB SQL extensions are provided:

- Bulk data access
- Physical tuning options
 - Hashing
 - Clustering
 - Indexes
- Pseudoconversational programming

Active dictionary: The components of the dictionary are active participants in the run-time environment.

Performance monitor: Performance can be monitored using the CA-IDMS Performance Monitor.

8.4 Components

The main components of the CA-IDMS/DB SQL environment are:

- The precompiler — Prepares application programs for execution in the CA-IDMS/DB environment
- The Command Facility — Supports interactive or batch submission of SQL and utility statements to define, access, secure, and maintain the database
- The parser — Checks for syntax errors and converts SQL statements into the format used by the optimizer
- The optimizer — Determines and generates the optimal access strategy for SQL DML statements
- The logical database engine — Coordinates the processing of SQL statements, performing some of the execution itself, and passing other elements of the statement execution to components such as the physical database engine, centralized security, and the optimizer
- The physical database engine — Performs physical data access and integrity management
- Data dictionary manager — Retrieves and maintains table and schema definitions in the dictionary

8.5 SQL as a language

SQL serves as a standard relational processing language that:

- Can be used either for ad hoc queries and updates or in application programs
- Eliminates the need for the user to know how the database is structured

Using SQL, you can:

- Define a database
- Manipulate data in the database

Data description: You use SQL data description language (DDL) statements to create the logical definition of a database.

There are three basic SQL DDL statements:

- **CREATE** — Adds a new entity to the database definition
- **ALTER** — Changes an existing database entity
- **DROP** — Deletes an existing database entity

Data manipulation: You use SQL data manipulation (DML) statements to manipulate the data in tables.

There are four basic SQL DML statements:

- **SELECT** — Retrieves row(s) of data from the database into a result table
- **INSERT** — Adds new rows of data to the database
- **UPDATE** — Changes data in the database table
- **DELETE** — Deletes a row or rows of data from the database

Data control: You use SQL-like statements to control access to data. There are two basic statements for this purpose:

- **GRANT** — Allows another user to access data
- **REVOKE** — Removes access from a user

►► For information on the GRANT and REVOKE statements, see Chapter 3, “Security” on page 3-1.

8.6 Tables, rows, columns

Tables: SQL-defined databases present information as a collection of tables. Each table is defined to contain related data.

A table is made up of columns and rows.

The diagram shows a table with three columns and three rows. The columns are labeled EMP_ID, FIRST_NAME, and LAST_NAME. The rows contain the following data: (0120, Jean, Lane), (3594, Mary, Smith), and (4492, John, Marcotte). Arrows labeled 'Columns' point to the column headers, and arrows labeled 'Rows' point to the data rows.

EMP_ID	FIRST_NAME	LAST_NAME
0120	Jean	Lane
3594	Mary	Smith
4492	John	Marcotte

Columns: A table has one or more columns. Each column:

- Has entries containing a single type of data
- Is displayed vertically
- Is identified by a name

Rows: A table has zero or more rows. Each row:

- Contains one value in each column
- Is displayed horizontally
- Is not named

8.7 Schemas and views

Schemas: A schema is a named collection of tables or views.

You create a schema to:

- Logically group tables (often tables used for an application or a group of applications)
- Add security to a group of tables using GRANT

Tables are associated with schemas through a qualification of the table name:

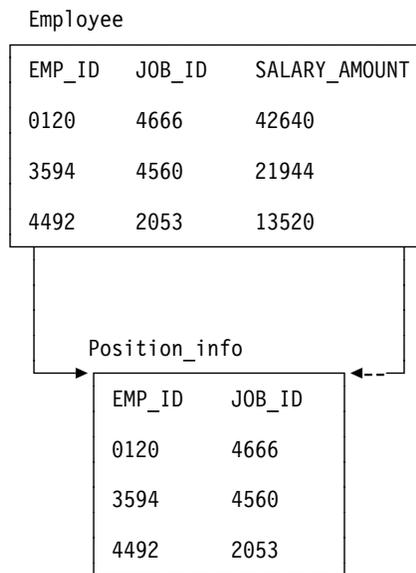
empprod.department
 ↑
 schema name

A default schema can be established for an SQL session to qualify table names not explicitly qualified by a schema. You can create a schema that is a logical reference to a non-SQL defined schema. In this situation, every record in the non-SQL defined schema can be referenced in SQL as a table.

►► For more information on referencing non-SQL defined records as tables, see Chapter 10, “Accessing a Database Using SQL” on page 10-1.

Views: A view is an alternate way of accessing data in a table or tables through a different description of that table. A view can be a subset of columns and rows in one or more tables.

Example: In this example, the view POSITION_INFO is a subset of columns from the EMPLOYEE table.



A view is represented internally by a stored statement, not stored data.

You use a view to:

- Limit the data that can be seen or updated when accessing a table using the view
- Predefine a complex select statement
- Assign alternate names for base tables

8.8 Table operations

The three types of operations used most often in SQL involve accessing specified rows, particular columns, and more than one table.

SELECT: You can use the WHERE clause of the SELECT statement to limit access to specified rows of a table or tables. This is called a **select** operation.

Example

```
select *  
from prod.employee  
where emp_id = 4437;
```

PROJECT: You can identify a subset of columns to be retrieved.

This type of operation is called a **project** operation.

Example

```
select emp_id, emp_lname  
from prod.employee;
```

JOIN: You can retrieve data from more than one table at the same time, typically by specifying criteria (in the WHERE clause) for matching rows from the different tables.

This type of operation is called a **join** operation.

Example

```
select emp_lname, emp_fname, department.dept_id, dept_name  
from employee, department  
where department.dept_id = employee.dept_id;
```

In addition to these basic operations, you may also append one table to another (the **union** operation).

You use one or more of these basic operations to retrieve data from the database.

8.9 Integrity and constraints

Data integrity is enforced at the database level rather than programmatically.

Uniqueness: Uniqueness is enforced through the use of `CALC` keys and/or indexes defined with the `unique` option. Duplicate rows cannot be entered if either of these has been defined on a column or group of columns within a table.

Domain constraints: The data value of a column can be limited through the use of:

- **Null option** — You can specify that nulls are allowed or not allowed for a column.
- **Data type** — You must specify a data type for each column. All data entered into that column must conform to that data type.
- **Check constraint** — In a check constraint, you can further limit the actual values for a column by specifying boolean expressions which must be satisfied by a row before an update or insert operation can occur.

Primary keys: To ensure that duplicate rows of data are not stored, a column or combination of columns is identified as the primary key of the table when the table is defined. Consequently, each entry in the primary key column or columns must be unique; there can be no duplicates. As a result, the primary key uniquely identifies each row in the table.

Example: In this example, the primary key is `EMP_ID`.

Primary key



EMP_ID	FIRST_NAME	LAST_NAME
0120	Jean	Lane
3594	Mary	Smith
4492	John	Marcotte

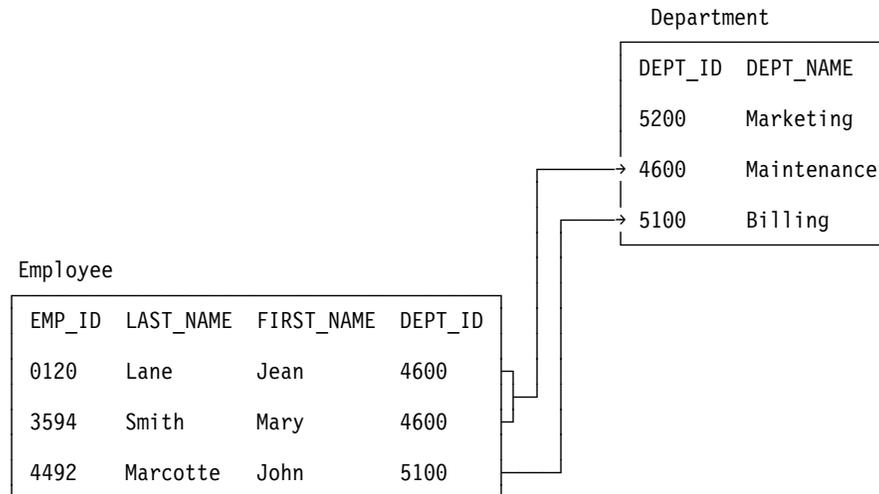
When you request data from a table and specify a value for the primary key, you see only one row returned.

Referential integrity: Referential integrity is enforced through referential constraints. A referential constraint limits the data in a column of a table (the **referencing** table) to that data found in a similar column in another table (the **referenced** table).

A referential constraint restricts the data in the referenced table, so that the data in the referenced table cannot be erased as long as there remains matching data in the referencing table. Also, the referenced column values of a row in the referenced table cannot be changed when rows exist that match rows in the referencing table.

Referencing columns are called **foreign keys**.

Example: In this example, values inserted into the DEPT_ID column in the EMPLOYEE table must match existing values in the DEPT_ID column in the DEPARTMENT table. Rows in the DEPARTMENT table cannot be deleted from the database so long as there is a matching value in the EMPLOYEE table. Thus, a department cannot be deleted so long as there remains an employee belonging to that department. Also, the DEPT_ID column of the DEPARTMENT row cannot be changed as long as there is an employee belonging to that department.



8.10 Storing SQL definitions

SQL definitions are stored in new dictionary areas as shown below:

Name of area	This area holds
DDLCAT	SQL entities including: <ul style="list-style-type: none">▪ Schemas▪ Tables▪ Views
DDLCATX	Indexes associated with DDLCAT items
DDLCATLOD	Access modules for SQL applications

8.11 For further information

- **On SQL as a language**, see *CA-IDMS SQL Reference* and *CA-IDMS SQL Programming*
- **On database definition**, see *CA-IDMS Database Administration*

Chapter 9. Defining a Database Using SQL

- 9.1 SQL data definition language 9-3
- 9.2 The definition process 9-4
- 9.3 For further information 9-7

9.1 SQL data definition language

To define a database with SQL, you use the following basic operations:

- CREATE
- ALTER
- DROP

You use CREATE, ALTER, and DROP to define or modify:

- Schemas
- Tables
- CALC keys
- Indexes
- Referential constraints
- Views

►► The process of creating the *physical* database definition is similar to the process used when creating a non-SQL defined database. For more information, see Chapter 1, “Database” on page 1-1.

9.2 The definition process

To create an SQL-defined database once the database design is complete, follow these steps:

1. **Create a schema** — A schema contains tables that are related to one another through common application usage.
2. **Create tables** — Create the tables that are a part of the database. For each table, define the columns in that table and include:
 - Data type
 - Null attribute, if any
 - Check constraint, if any
3. **Define indexes and CALC keys** — You create indexes and CALC keys based on the database design. Some indexes and CALC keys are required for implementation of referential constraints while others are necessary for efficient database access.
4. **Define referential constraints** — You create referential constraints based on the database design. A referential constraint can be linked or unlinked. Referential constraints reflect the logical relationships that exist between database tables.

▶▶ For more information on linked and unlinked referential constraints, see Chapter 13, “Administration of an SQL-Defined Database” on page 13-1.
5. **Define views** — You create views based on the application's needs.

Creating a schema: When you create a schema:

- You assign a name to the schema.
- The schema is added to the dictionary.

Example: This is an example of a schema definition.

```
create schema prod;
```

Creating a table: When you create a table:

- You assign a name to the table.
- You assign names to the columns in the table.
- You specify the kind of data each column can contain.

Categories of valid data types are:

- Approximate and exact numeric
- Binary
- Character string

- Datetime
- Graphics character string
- You specify whether a value is required in each column (null attribute).
- You optionally specify a list or range of acceptable values for a column (check constraint).
- You specify whether a default value should be used for a column.
- You options specify CA-IDMS Presspack compression for the table.
- You assign a location for the table rows (area name).

The table and column definitions are stored in the dictionary.

Example: This example shows the definition of a table. There are five columns and a check constraint that restricts the values in the CODE and AUTHOR_ID columns.

```
create table prod.document
  (title          char(50)    not null,
   author_id     integer(6)  not null,
   publisher     char(40)    not null,
   code          numeric(5)  not null,
   description   varchar(50) ,
   check ( code > 10000 and code < 55555 ) and
         ( author_id <= 999999 ) )
in area prodseg.docarea;
```

Creating a CALC key: When you create a CALC key, you:

- Specify the table on which the CALC key is defined
- Specify the column or columns which make up the CALC key
- Indicate if the key value must be unique for all rows in the table

Example: This is an example of creating a unique CALC key for the DOCUMENT table. The CALC key is the CODE column.

```
create unique calc key on prod.document (code);
```

Creating an index: When you create an index, you:

- Assign a name to the index
- Specify the table on which the index is defined
- Specify the columns which make up the index key
- Specify if the index value must be unique for all rows in the table
- Specify if the table rows are to be physically stored in index sequence
- Specify physical tuning options such as key compression, location (area name of the index structure), etc.

Example: This is an example of creating an index on the PUBLISHER column in the DOCUMENT table.

```
create index doc_pub on prod.document (publisher);
```

Creating a referential constraint: When you create a referential constraint, you:

- Assign a name to the constraint
- Specify the referenced and referencing tables
- Specify the columns within each table which must be verified for data integrity
- Specify physical tuning options such as clustered, indexed, linked, etc.

Example: This is an example of a referential constraint definition. When a new document is inserted into the DOCUMENT table, the author's ID that was entered will be checked to see that it is a valid ID in the AUTHOR table.

```
create constraint author_doc
  prod.document (author_id)
  references prod.author (author_id);
```

Creating a view: When you create a view, you:

- Assign a name to the view
- Optionally assign names to columns in the view
- Identify the column(s) and table(s) to be accessed using a SELECT statement

Example: This is an example of creating a view. The view is made up of two columns from the DOCUMENT table.

```
create view prod.doc_view (title, doc_description)
  as select title, description
  from prod.document;
```

9.3 For further information

- **On database definition**, see *CA-IDMS Database Administration*
- **On SQL syntax and data types**, see *CA-IDMS SQL Reference*

Chapter 10. Accessing a Database Using SQL

- 10.1 Data manipulation with SQL 10-3
- 10.2 Interactive and embedded SQL 10-4
 - 10.2.1 Interactive SQL 10-4
 - 10.2.2 Embedded SQL 10-4
 - 10.2.3 Dynamic SQL 10-6
- 10.3 CA-IDMS tools support for SQL 10-7
 - 10.3.1 CA-ADS support 10-7
 - 10.3.2 CA-OLQ support 10-7
 - 10.3.3 CA-ICMS support 10-8
 - 10.3.4 CA-Culprit support 10-8
- 10.4 SQL access to a non-SQL defined database 10-9
 - 10.4.1 How to do it 10-9
 - 10.4.2 Database requirements 10-10
- 10.5 For further information 10-11

10.1 Data manipulation with SQL

Structured Query Language (SQL) is a standardized non-procedural language used to retrieve and update information in a database.

CA-IDMS/DB processes interactive SQL statements submitted through the Command Facility and embedded SQL in a host language program that has been precompiled and optimized.

10.2 Interactive and embedded SQL

You can issue SQL statements either interactively or from within an application program. Data accessed through SQL statements is returned in the form of a **result table**. A result table is a subset of the rows and columns in one or more tables or views.

10.2.1 Interactive SQL

When you use interactive SQL to enter a request, you get immediate results. This is the typical way of entering ad hoc statements.

Interactive SQL can be executed either online or in batch, using the Command Facility or CA-OLQ.

Example: In this example, the SQL statement returns the last name and first name of all employees residing in Boise.

```
select emp_lname, emp_fname
       from prod.employee
       where city = 'Boise';
```

10.2.2 Embedded SQL

You can embed SQL statements in host application programs. With embedded SQL, the program receives the result of the request and typically displays or prints it.

Any SQL statement can be embedded in a program.

Embedding SQL in the program does not affect any rules that apply to using the host language.

Within the program, you can embed SQL statements to:

- Access the database
- Access the dictionary
- Define the structures needed to transfer data between the program and the DBMS
- Manage SQL sessions and transactions

Example: In this example, the embedded SQL statement returns to the program the last name and first name of an employee requested by the program.

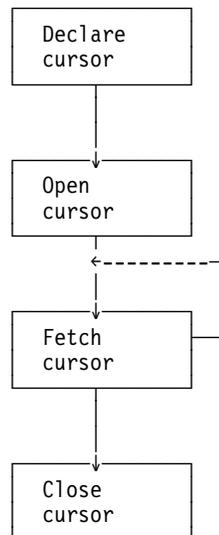
```
EXEC SQL
SELECT EMP_LNAME, EMP_FNAME
       INTO :EMP-LNAME, :EMP-FNAME
       FROM DEMOEMPL.EMPLOYEE
       WHERE EMP_ID = :EMP-ID
END-EXEC.
```

The DML to be executed is the same across languages; the begin and end indicators (EXEC SQL and END-EXEC) may vary by language.

Cursors: Usually, application programs need to access multiple rows from one or more tables. To do this, the program must use a construct called a **cursor**. The cursor defines the result table and the program retrieves the rows of the result table one at a time with a FETCH statement.

If the cursor definition meets certain requirements, the program can update or delete the current row of data.

As shown below, the cursor is first declared. Then it is opened and positioned. Repeated fetches return data to the program. When the program no longer needs the cursor, it is closed.

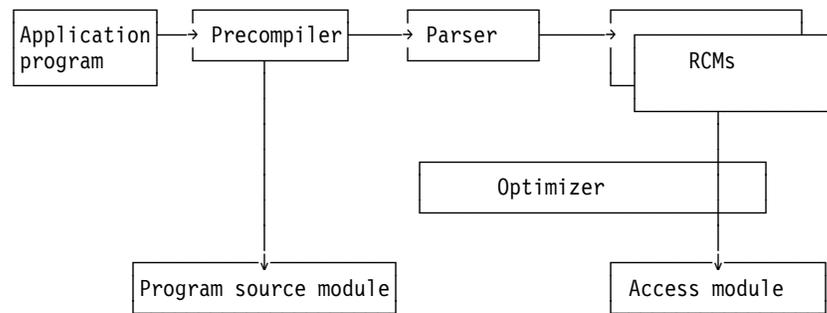


Creating executable modules: Since the host language program contains an embedded sublanguage, you precompile the program to **separate** the SQL from host language source. You then create an **access module** that contains an optimized data access strategy for the precompiled SQL statements.

As shown below, the precompiler replaces embedded SQL statements with an internal form and stores them in a module called an RCM. It replaces embedded SQL in the source module with host language calls to the DBMS. These calls, unlike the SQL statements they replace, are intelligible to the host language compiler.

After the program precompiles successfully, you compile and link the modified source program to create an executable program load module.

The access module is used at runtime to direct the processing of an SQL request. You create the access module from one or more RCMs.



You can change the schema name qualifiers of tables when you create an access module. For example, if a program contains references to tables in schema HR, all references to HR can be replaced by HRTEST in one access module and HRPROD in another access module. This capability allows you to use the same program source and corresponding RCM to access different sets of tables.

10.2.3 Dynamic SQL

Dynamic SQL refers to an SQL statement that is not known to the program at precompile time and therefore is compiled dynamically when the program executes.

CA-IDMS/DB provides dynamic SQL to allow the program to formulate, compile, and execute an SQL statement at runtime.

10.3 CA-IDMS tools support for SQL

Under Release 12.0, these CA-IDMS tools allow you to access your data with SQL:

- CA-ADS
- CA-OLQ
- CA-ICMS
- CA-Culprit

10.3.1 CA-ADS support

What you can do: Under release 12.0, you can embed SQL statements in CA-ADS dialogs. All CA-IDMS/DB SQL statements are valid in premap and response process modules except INCLUDE TABLE. You implicitly include a table by specifying the table on the Work Record screen.

Declaration module: A declaration is a new type of CA-ADS module that can issue DECLARE CURSOR statements and WHENEVER directives whose scope is the entire dialog. A declaration module is not an executable module.

Example: The following is an example of an SQL DML request embedded in a CA-ADS dialog.

```
EXEC SQL
  SELECT EMP_ID, EMP_LNAME, EMP_FNAME
  INTO :EMPID, :LNAME, :FNAME
  FROM DEMOEMPL.EMPLOYEE
  WHERE EMP_ID = :EMP-ID;
END-EXEC.
```

10.3.2 CA-OLQ support

What you can do: Under Release 12.0, CA-OLQ provides Command Mode and Menu Mode support for CA-IDMS/DB SQL. To use SQL access, you simply flip the access switch from OLQ to IDMS. In Menu Mode, a new Selection Criteria screen, accessible from the Column Select screen, provides additional space to view, add, or modify selection criteria.

Converting ASF tables: You can use CA-OLQ to convert existing ASF tables to SQL-defined tables. The steps are:

1. Set the access switch to OLQ
2. Create and save a report containing the ASF-table data
3. Set the access switch to IDMS
4. Specify the dictionary connection and the schema
5. Issue a SEND TABLE CREATE command

10.3.3 CA-ICMS support

What you can do: Under Release 12.0, you can access CA-IDMS/DB with SQL from CA-INFOGATE and CA-GOLDENGATE. A new CORP-owned folder named `_SQL` contains lower level folders representing schemas in the current CA-IDMS/DB database. Tables in each schema are recognized as DATA-TABLE objects.

How it works: Commands to access tables are automatically translated for CA-IDMS/DB processing. Some SQL data types are translated in the upload/download process. Null value is displayed as blank or zero, depending on the data type.

10.3.4 CA-Culprit support

What you can do: Under Release 12.0, you can use SQL in Culprit processing to access CA-IDMS/DB data. New subparameters provide the means to do this:

- IN card
 - DB(Q) SQL = SQL compliance specification
 - DICTIONARY = Dictionary containing table definitions
- OUT card
 - SCHEMA = Schema-name qualifier of the table
 - DICTIONARY = Dictionary containing table definitions
 - TYPE = Type of update (CREATE/ADD/REPLACE/DELETE)
 - SQLTABLE = Target table for update

Features: Features of CA-Culprit SQL support include:

- Automatic REC card generation
- Null indicator support
- Details-only or Totals-only specification

10.4 SQL access to a non-SQL defined database

You can use SQL to access a non-SQL defined database. This means that you do not have to redefine your existing databases in order to access them using SQL.

10.4.1 How to do it

To access an existing database using SQL statements, you create an SQL-defined schema referencing the non-SQL defined schema.

You do not have to create tables, indexes, referential constraints, or any other SQL component. The DBMS uses the name of the SQL-defined schema as a window to the non-SQL defined schema definitions.

Example: In this example, the SQL-defined schema being created references the non-SQL defined schema EMPSCHEM.

```
create schema emp
    for nonsql schema empschm;
```

Joining records: In a non-SQL defined database, records typically participate in a set relationship, eliminating the need for embedded foreign keys. When issuing a joining of such records in an SQL statement, you specify set name.

Example

```
select emp_id, dept_id
    from employee, department
    where "DEPT-EMPLOYEE";
```

Primary and foreign keys: Schema set syntax has been expanded to support the definition of primary and foreign key sets. A PRIMARY KEY option has been added to the OWNER clause and a FOREIGN KEY option has been added to the MEMBER clause of the SET statement.

Both SQL and DML: A single application program or CA-ADS dialog can use both embedded SQL and navigational DML. Such concurrent access is coordinated by using task level COMMIT, FINISH, and ROLLBACK statements.

Joining records and tables: You can join data in your existing non-SQL defined databases with data in SQL-defined databases.

Example: In this example, **sales** represents a non-SQL schema and **hr** is the schema-name qualifier of an SQL-defined table:

```
select
  emp_id_0415 as "Employee ID",
  emp_last_name_0415 as "Last Name",
  office_code_0450 as "Office",
  vac_accrued-vac_taken as "Hours Remaining"
from sales.office,sales.employee,hr.benefits
where "OFFICE-EMPLOYEE"
  and emp_id_0415=emp_id
order by 3, 2;
```

10.4.2 Database requirements

Restrictions: Using SQL against a non-SQL schema record requires that the rules of SQL be satisfied. Because of these rules, there are some restrictions in the following areas:

- Data structures used within the record
- Data types
- Element/column names

Data structures: The non-SQL defined database must meet the following SQL requirements:

- You can access only the lowest level element of a group structure.
- You can access only the primary definition of an element, not a redefinition.
- You cannot access variably-occurring data; elements having a fixed number of occurrences are suffixed with the occurrence count.

Data types: The data types supported for SQL access are:

- Signed and unsigned zoned decimal
- Signed and unsigned packed decimal
- Fullword, halfword, and double-word binary
- Character
- Long and short floating point
- Graphics

Other data formats, such as bit and external floating point, may be accessed through SQL. The format will be translated into an SQL-compatible format.

Names: Element names are translated automatically to valid SQL synonyms for use in an SQL statement. For example, hyphens are converted to underscores. Alternatively, you can explicitly define SQL synonyms for these elements in the dictionary.

Dictionary support for SQL synonyms: You can create an SQL record synonym for access to non-SQL defined databases. This synonym is created by associating a LANGUAGE IS SQL with the target record synonym. A given record can have only one record synonym with a language of SQL.

10.5 For further information

- On basic SQL, see *SQL Self-Training Guide*
- On SQL syntax, see *CA-IDMS SQL Reference*
- On embedding SQL in CA-ADS, COBOL, and PL/I programs, see *CA-IDMS SQL Programming*

Chapter 11. SQL Extended Features

11.1	What are SQL extended features?	11-3
11.2	Database definition extensions	11-4
11.2.1	Data types	11-4
11.2.2	32-character column names	11-4
11.2.3	Database tuning extensions	11-4
11.3	Data access and manipulation extensions	11-5
11.3.1	Bulk access to tables	11-5
11.3.2	Scalar functions	11-5
11.3.3	Special registers	11-7
11.3.4	Date/time arithmetic	11-7
11.3.5	Temporary tables	11-8
11.3.6	Modular programming	11-8
11.3.7	Dynamic SQL	11-8
11.3.8	Access to non-SQL defined databases	11-8
11.4	Precompiler directive extensions	11-9
11.5	Session management extensions	11-10
11.5.1	Specifying a dictionary	11-10
11.5.2	Pseudoconversational support	11-10
11.5.3	Establishing session characteristics	11-11
11.6	Transaction management extensions	11-12
11.6.1	CONTINUE/RELEASE parameters on the COMMIT WORK statement	11-12
11.6.2	Dynamic selection of access module	11-12
11.6.3	Overriding access module defaults	11-13

11.1 What are SQL extended features?

SQL extended features provide enhanced functionality to ANSI-standard SQL. CA-IDMS/DB provides these features to support production-oriented, high-performance environments.

The discussion in this chapter covers the major extensions provided by CA-IDMS.

11.2 Database definition extensions

CA-IDMS/DB supports these database definition extensions:

- Additional data types
- 32-character column names
- Database tuning options

11.2.1 Data types

CA-IDMS/DB supports the data types shown in the following table. An asterisk indicates a data type that is a CA-IDMS extension.

Category	Data types
Approximate numeric	DOUBLE PRECISION FLOAT REAL
Binary	BINARY (*)
Character string	CHARACTER VARCHAR (*)
Datetime	DATE (*) TIME (*) TIMESTAMP (*)
Exact numeric	DECIMAL UNSIGNED DECIMAL (*) INTEGER LONGINT (*) NUMERIC UNSIGNED NUMERIC (*) SMALLINT
Graphics character string	GRAPHIC (*) VARGRAPHIC (*)

11.2.2 32-character column names

Column names can be up to 32 characters long.

11.2.3 Database tuning extensions

CA-IDMS/DB provides a number of database tuning options as an extension to SQL:

- CALC keys
- Indexes
- Data clustering around a referenced occurrence
- Linked constraints
- Table occurrences stored in index key sequence
- Updating statistics using the UPDATE STATISTICS statement

►► SQL tuning options are discussed in Chapter 13, “Administration of an SQL-Defined Database” on page 13-1

11.3 Data access and manipulation extensions

CA-IDMS/DB supports these extensions for data access and manipulation:

- Bulk processing
- Scalar functions
- Special registers
- Date/time arithmetic
- Temporary tables
- Global cursors
- Dynamic SQL
- Using SQL to access a non-SQL defined database

11.3.1 Bulk access to tables

You can access tables in bulk from within an application program.

You can identify a host variable defined as an array (table). CA-IDMS/DB will then assign the values in the result table to this array.

This extension allows your program to process many rows with a single FETCH, SELECT, or INSERT statement.

Example: The following SELECT statement returns information on the cost of insurance plans in Iowa into an array identified by the host variable :STATE-INFO-ARRAY:

```
EXEC SQL
  SELECT STATE_CODE, NAME, CAPITOL_CITY
  BULK :STATE-INFO-ARRAY
  FROM CORP.STATE
END-EXEC.
```

11.3.2 Scalar functions

A scalar function is a function whose argument includes at least one value expression on which the function operates. The result of a scalar function is a single value. This value is derived from the expression or expressions named in the argument.

The following scalar functions are supported by CA-IDMS/DB:

Name	Function
CAST	Forces conversion of a value to a specified data type

Name	Function
CHAR	Returns a character string representation of a date/time value
DATE	Returns the date from a value
DAY	Returns the day part of a date
DAYS	Returns an integer representation of a date
DECIMAL	Returns a decimal representation of a value
DIGITS	Returns a character string representation of a value
FLOAT	Returns a floating-point representation of a value
HEX	Returns a hexadecimal representation of a value
HOUR	Returns the hour part of a time
INTEGER	Returns an integer representation of a value
LENGTH	Returns the length of a value
MICROSECOND	Returns the microsecond part of a time
MINUTE	Returns the minute part of a time
MONTH	Returns the month part of a date
PROFILE	Returns the value of a specified attribute of the user session
SECOND	Returns the second part of a time
SUBSTR	Returns a substring of a value
TIME	Returns the time from a value
TIMESTAMP	Returns a timestamp from a value or pair of values
YEAR	Return the year part of a date

Example: The following example uses the SUBSTR scalar function to extract the first five characters of the zip code for a listing of zip codes in the EMPLOYEE table:

```
select distinct substr(zip_code,1,5) as Zips
from employee;
```

11.3.3 Special registers

Special registers are system-supplied values that can be referenced with keywords. The value associated with a special register is determined at runtime.

These special registers are supported by CA-IDMS/DB SQL:

Keyword	Value at runtime
USER	Identifier of user executing the SQL session
GROUP	Default group associated with the executing user
CURRENT DATE	Current date when the SQL statement is executed
CURRENT TIME	Current time when the SQL statement is executed
CURRENT TIMESTAMP	Current date and time when the SQL statement is executed
CURRENT TIMEZONE	Difference between current time and GMT
CURRENT DATABASE	Dictionary to which the SQL session is connected
CURRENT SCHEMA and CURRENT SQLID	Schema qualifier for the current SQL session

Example: In this example, an ad hoc query lists scheduled projects expected to begin sometime after the query is issued.

```
select
  proj_id,
  est_start_date,
  proj_desc
from demoproj.project
where est_start_date > current date
order by 2;
```

11.3.4 Date/time arithmetic

You can perform date and time arithmetic.

Example: The example below calculates the age at which an employee was hired.

```
select emp_id, emp_lname, start_date - birth_date as "Age When Hired"
from demoempl.employee
where dept_id = 2010;
```

11.3.5 Temporary tables

You can create a temporary table for use within a transaction. The data in a temporary table can be accessed and manipulated in the same way as with a permanent table.

Example: The example below creates a temporary table, TEMP_EMP, to hold part of the EMPLOYEE table for the life of the transaction.

```
create temporary table temp_emp
(empid      unsigned numeric(4,0) not null,
 empfname  char(20)                not null,
 emplname  char(20)                not null,
 street    char(40)                ,
 city      char(20)                not null,
 state     char(02)                not null,
 zipcode   char(09)                not null,
 phone     char(10)                );
```

11.3.6 Modular programming

External and global cursors: Under CA-IDMS/DB, a program can use a cursor defined in another program. CA-IDMS/DB supports DECLARE EXTERNAL CURSOR which identifies an externally defined global cursor to be used by the application program.

Example: The following statement identifies ALL_EMP_CURSOR as a cursor declared in another application program module, but which can be accessed from this program.

```
EXEC SQL
  DECLARE ALL_EMP_CURSOR EXTERNAL CURSOR
END-EXEC.
```

11.3.7 Dynamic SQL

Dynamic SQL refers to an SQL statement that is not known to the program at precompile time and therefore is compiled dynamically when the program executes.

CA-IDMS/DB provides dynamic SQL to allow the program to formulate, compile, and execute an SQL statement at runtime.

11.3.8 Access to non-SQL defined databases

Using CA-IDMS/DB, you can access a non-SQL defined database using SQL statements.

►► See Chapter 9, “Defining a Database Using SQL” on page 9-1 and Chapter 10, “Accessing a Database Using SQL” on page 10-1 for further information on accessing a non-SQL database.

11.4 Precompiler directive extensions

INCLUDE: CA-IDMS/DB supports the use of INCLUDE as a precompiler directive. INCLUDE directs the precompiler to create host variable definitions for a specified table in the application program.

Example: The statement below directs the precompiler to define host variables corresponding to the named columns in the INSURANCE_PLAN table. The host variables will be defined as a 10-row array named INS-COST-BUFFER. The name of each element in the array will begin with INS-. Each element will have a level number of 02.

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.  
EXEC SQL  
  INCLUDE TABLE INSURANCE_PLAN  
  AS INS-COST-BUFFER  
  (PLAN_CODE, COMP_NAME, MAX_LIFE_COST, FAMILY_COST, DEP_COST)  
  AS (PLANCODE, COMPNAME, MAXLIFE, FAMCOST, DEPCOST)  
  NUMBER OF ROWS 10  
  PREFIX 'INS-'  
  LEVEL 02  
END-EXEC.  
EXEC SQL END DECLARE SECTION END-EXEC.
```

11.5 Session management extensions

All session management statements are extensions to the ANSI standard. These include:

- **CONNECT** — Establishes a connection to a specified dictionary
- **RELEASE** — Releases a connection and ends the SQL session
- **SUSPEND SESSION** — Suspends an SQL session and any transaction currently active within the session
- **RESUME SESSION** — Resumes a suspended SQL session
- **SET SESSION** — Establishes SQL session characteristics

11.5.1 Specifying a dictionary

You can explicitly specify the dictionary that contains the definitions of the data the program accesses, thus overriding the default value associated with your user session. You use the **CONNECT** statement to establish this connection.

Example: The following statement establishes a connection to the database name contained in the host variable :DB-NAME.

```
EXEC SQL
  CONNECT TO :DICT-NAME
END-EXEC
```

11.5.2 Pseudoconversational support

Pseudoconversational programming is an online programming technique that frees resources while the system waits for a response from the online user.

To facilitate pseudoconversational programming in an SQL application, CA-IDMS/DB supports:

- **SUSPEND SESSION** — Directs the DBMS to release all resources associated with the SQL session except those needed to resume the current session and transaction (such as cursor position)
- **RESUME SESSION** — Reestablishes the active SQL session and database transaction. All characteristics and cursor positions of the session and transaction are restored to what they were when the program issued the **SUSPEND SESSION** statement.

11.5.3 Establishing session characteristics

You can establish the following session characteristics using a `SET SESSION` statement:

- Default schema — the schema to be used for unqualified table names
- Standards enforcement — the SQL syntax standard to which your SQL statements must conform

These session characteristics apply only to SQL submitted either interactively through the Command Facility or for dynamic compilation during the execution of an application program. The `SET SESSION` statement does not cause impact precompiled SQL statements; however, similar options exist as precompiler directives.

Example: The following statement directs CA-IDMS/DB to use the SALES schema as the default schema for the remainder of the SQL session.

```
set session current schema sales;
```

11.6 Transaction management extensions

Transaction management statements are extensions to the ANSI standard. They include:

- **COMMIT CONTINUE** — Specifies that the SQL transaction continues upon committing the data
- **COMMIT RELEASE** — Specifies that the SQL transaction ends upon committing the data, and the SQL session is terminated
- **SET ACCESS MODULE** — Overrides the access module specification made at precompile time
- **SET TRANSACTION** — Overrides the access module default for isolation level or transaction state

11.6.1 CONTINUE/RELEASE parameters on the COMMIT WORK statement

CONTINUE directs CA-IDMS/DB not to end the current transaction after committing the changes to the database. CA-IDMS/DB keeps all share locks and cursors open, retains all temporary tables, and downgrades or releases all update locks. It also commits to the database all changes made either since the last COMMIT CONTINUE statement or since the beginning of the transaction.

When you specify the RELEASE parameter of the COMMIT statement, CA-IDMS/DB commits the current transaction and terminates the SQL session.

Example: The following statement commits to the database changes made during the current transaction but does not end the transaction.

```
EXEC SQL
  COMMIT CONTINUE
END-EXEC.
```

11.6.2 Dynamic selection of access module

CA-IDMS/DB provides support for dynamic selection of an access module through the SET ACCESS MODULE statement. You use this statement to identify the access module to be used by a transaction at runtime, overriding the specification made at precompile time.

Example: The following statement specifies that the current transaction is to use the access module identified by the host variable TRANS-ACC-MOD.

```
EXEC SQL
  SET ACCESS MODULE :TRANS-ACC-MOD
END-EXEC.
```

11.6.3 Overriding access module defaults

CA-IDMS/DB provides support for SET TRANSACTION to override access module defaults for transaction characteristics such as isolation levels.

Example: The following example specifies that the transaction has an isolation level of transient read.

```
EXEC SQL
  SET TRANSACTION TRANSIENT READ
END-EXEC.
```


Chapter 12. CA-IDMS Access Module Creation

- 12.1 What is the optimizer? 12-3
- 12.2 Compilation strategy 12-4
- 12.3 How does optimization work? 12-5
 - 12.3.1 Automatic reoptimization 12-6
 - 12.3.2 Describing the access strategy 12-6
- 12.4 For more information 12-7

12.1 What is the optimizer?

Access module compiler: The CA-IDMS/DB optimizer is responsible for compiling all SQL data manipulation statements into the form required for execution by the runtime system.

For embedded SQL, the optimizer creates the most efficient access strategy for pre-compiled SQL statements in one or more host language programs and compiles the access module. The access module is stored for later execution.

The optimizer also optimizes and compiles individual dynamic SQL statements for immediate execution.

Why optimize?: The objective of optimization is to provide access to data with minimum use of system resources, primarily I/O. Optimization provides the most efficient access to data and is a key factor in the ability of SQL database systems to remove data access navigation responsibility from the application programmer.

12.2 Compilation strategy

When you compile an access module, the following functions are performed on each SQL data manipulation statement:

- Validation of table and column references in the statement against the dictionary
- Selection of the most efficient database access strategy for the statement
- Generation of the code used by the runtime software to execute the SQL request

12.3 How does optimization work?

To develop an optimized access strategy for an SQL statement, the optimizer considers:

- The type of statement (SELECT, INSERT, UPDATE, DELETE)
- The WHERE clause criteria
- The physical structure of the database as defined in the dictionary, including any calc keys, indexes, linked constraints, and other tuning options
- Statistics stored in the dictionary as a result of UPDATE STATISTICS statements, or as extrapolated from the ESTIMATED ROWS clause of the CREATE or ALTER TABLE statements

Statistics considered: These statistics include:

- Number of rows in the area
- Number of occupied pages
- Storage density of the area
- Number of rows in a table
- Referential constraint statistics such as:
 - Number of referenced rows
 - Average number of referencing rows per referenced row
 - Average number of physical pages spanned by a referenced row and its related referencing rows
- CALC and index key statistics such as:
 - Number of distinct key values
 - Low and high values
 - Number of intermediate index levels
 - Average number of physical pages per distinct key value

Determining the access strategy: The optimized access strategy is determined by a combination of physical tuning options chosen by the database designer and the statistics which describe the current state of the database.

The database design gives the optimizer the raw physical structures which it uses to select efficient access strategies. The optimizer takes advantage of all indexes, clustered and linked constraints, and CALC keys.

Database statistics are important in choosing an optimal strategy when there are equivalent physical database structures.

12.3.1 Automatic reoptimization

Adaptive query management: The access module is automatically recompiled if changes are made to the underlying physical database structure, thereby providing for the best access for each SQL execution and protection for the application program and the integrity of the database

12.3.2 Describing the access strategy

The EXPLAIN statement: The EXPLAIN statement is a CA-IDMS/DB extension that allows you to describe, or explain, the access strategy determined by the optimizer for a specified SQL statement or for all SQL statements in an access module.

Description in table form: The output of the EXPLAIN statement is one or more rows per SQL statement in a table. Information in the result table includes the type of access and the number, type, and sequence of processing steps required to satisfy the request.

12.4 For more information

- On creating an access module, see *CA-IDMS SQL Reference*
- On updating statistics, see *CA-IDMS Database Administration*
- On database tuning options, see *CA-IDMS Database Administration*
- On explaining the access strategy, see *CA-IDMS SQL Reference*

Chapter 13. Administration of an SQL-Defined Database

13.1	Tuning the database	13-3
13.1.1	Indexes	13-3
13.1.2	CALC keys	13-3
13.1.3	Referential constraints	13-3
13.1.4	Clustering	13-4
13.2	Utilities in the SQL environment	13-5
13.3	Locking	13-6
13.3.1	Types of locks	13-6
13.4	Security for SQL-defined databases	13-8
13.4.1	Privileges	13-8
13.4.2	Granting privileges	13-9
13.4.3	Ownership	13-9
13.4.4	Security checking	13-9
13.5	For further information	13-11

13.1 Tuning the database

With CA-IDMS/DB Release 12.0, you can tune the database using:

- Indexes
- CALC keys
- Referential constraint options
- Clustering

These tuning options allow you to create a database that can be efficiently accessed.

13.1.1 Indexes

An index provides an alternate method of table access and a logical ordering of data stored in the database. An index can also implement entity integrity by requiring that index keys be unique.

13.1.2 CALC keys

A CALC key provides direct access to a row of a table without the overhead of maintaining and searching an index. The location of the row is calculated from the value of the columns forming the CALC key. A CALC key can implement entity integrity by requiring that CALC keys be unique.

13.1.3 Referential constraints

There are several database tuning options available for referential constraints.

Referential constraints can be **linked** or **unlinked**.

Unlinked: Unlinked referential constraints (the default) are enforced without a physical link between referencing and referenced tables. Instead, index and calc keys are used internally to enforce the integrity rules.

Unlinked constraints must have at least one index or a CALC key defined on the referenced and referencing keys of each table. Rows from both tables are retrieved directly through these keys.

Linked: A linked constraint establishes a physical structure between two tables. It is implemented using either a chained or index set. Linked referential constraints can make data retrieval between tables more efficient by reducing the number of I/Os required to retrieve referencing rows.

Linked...order by: You can use ORDER BY on a linked referential constraint to specify that the referencing rows are to be maintained in sequence based on values for the named columns (either ascending or descending order).

Using ORDER BY can eliminate the need for an internal sort when rows are retrieved in a specified order.

13.1.4 Clustering

You can cluster rows of data in a table around a row of data from another table or around an index. Clustering enables you to group rows that are likely to be accessed together.

If a referencing table is stored clustered around a referential constraint, CA-IDMS/DB stores each row as close as possible to the corresponding row in the referenced table (the row to which it is logically related). CA-IDMS/DB can then access all referencing rows with a minimum of I/Os.

Rows in a table can be physically stored in a specific sequence. If they are sequenced on an index column, sorted retrieval through the index will be more efficient.

13.2 Utilities in the SQL environment

Most utilities described in Chapter 1, “Database” on page 1-1 can be used with any CA-IDMS/DB database. In addition, these utilities are specific to SQL-defined databases:

Utility	How it's used
BUILD	Builds indexes and referential constraints
EXPLAIN statement	Describes the access strategy for a statement or access module
INSTALL STAMPS	Stores synchronization stamps in a newly-formatted area
LOAD TABLE	Loads data into tables
UPDATE STATISTICS	Updates statistics used by CA-IDMS/DB to optimize access to the database
VALIDATE	Checks referential constraints

Invoking the utilities: Utilities are invoked through the Command Facility.

►► For information on the Command Facility, see Chapter 1, “Database” on page 1-1.

13.3 Locking

SQL-defined database locking operations are basically the same as locking operations for a non-SQL defined database.

►► For additional information on lock management, see Chapter 1, “Database” on page 1-1.

Concurrency control: CA-IDMS/DB manages concurrent access to the same set of data with a system of locks. The degree of concurrent access allowed by a database transaction is determined by the isolation level of the transaction and the ready mode of the areas it accesses.

13.3.1 Types of locks

Locks: CA-IDMS/DB provides two types of lock:

- A **retrieval lock** prevents updates but allows retrieval of data by another database transaction
- An **update lock** prevents updates and retrieval of data by another database transaction

CA-IDMS/DB places locks at the row or area level.

CA-IDMS/DB always places an update lock on a row which is updated by a transaction, unless an update lock is already held at the area level.

CA-IDMS/DB usually places retrieval locks on rows as they are being processed. Retrieval row locks are not maintained if a retrieval lock is held on the area or if a transaction option indicates no locking is desired.

Locking at the row level: To cause locks to be set at the row level, you specify an isolation level of **cursor stability**.

Under cursor stability, the DBMS places a retrieval lock on the row on which an updatable cursor is positioned until the cursor position changes. It places a retrieval lock on all other rows accessed by the transaction, but releases the lock as soon as it finishes processing the row.

Cursor stability provides the greatest possible concurrency while guaranteeing the integrity of data read by the transaction. Under cursor stability:

- The current row of an updatable cursor cannot be updated by another database transaction while it remains current.
- A single row retrieved by a SELECT statement cannot be updated by another database transaction until the original transaction accesses another row of the table.

Cursor stability is the CA-IDMS/DB default.

No locking: To cause no locks to be set, you specify an isolation level of **transient read**.

Under transient read, the DBMS places no locks on rows accessed by the database transaction, and the transaction cannot update data in the database.

Transient read provides no protection from the effects of concurrent database transactions. It allows a database transaction to read data that has not been committed and allows concurrent database transactions to update the data. It is appropriate when the transaction is retrieval only and does not require the data to be consistent and entirely accurate.

Changing the default isolation level: You can change the default isolation level of cursor stability by using appropriate parameters of the CREATE ACCESS MODULE statement.

For a given SQL transaction, the program can override the default isolation level for the access module by issuing a SET TRANSACTION statement.

Locking at the area level: You can control concurrent access at the **area** level using the READY parameter of the CREATE ACCESS MODULE statement. You can specify what type of retrieval or update lock the DBMS sets on an area that the program accesses. You can also specify whether the lock is to be set when the first SQL statement accesses the **database** or when the first SQL statement accesses the **area**.

Setting appropriate locks at the area level (such as protected retrieval or protected update) will prevent concurrent update access to the specified area for the duration of an SQL transaction. This obviates the need to set locks at the row level and lessens the likelihood of deadlock. However, it also reduces the degree of concurrency supported by the database.

13.4 Security for SQL-defined databases

CA-IDMS SQL processing uses the centralized security facility of Release 12.0. The basic set of security administration statements is compliant with the ANSI SQL standard.

At runtime security is checked through calls to the central security manager issued by CA-IDMS software at strategic points in processing. Enforcement is performed by CA-IDMS internal security, an external security package, or not at all, depending on the type of resource involved and the means of enforcement you specify for that resource type.

Security features: The security features discussed in Chapter 3, “Security” on page 3-1 can be used to secure access to an SQL-defined database. These include:

- Support of users and groups defined in the user catalog
- Use of wildcards to specify similar entities when defining secured resources
- Interface to an external security package to check authorization for access to any or all types of resources associated with SQL processing

13.4.1 Privileges

Definition privilege: The definition privilege is the authority to define a particular resource entity.

The definition privilege for tables is unique to the CA-IDMS/DB SQL environment. This privilege allows a user to perform one or more of these operations on a table:

- CREATE — controls a user's ability to define a new table, index, access module, etc.
- ALTER — controls a user's ability to change an existing definition
- DROP — controls a user's ability to delete an existing definition
- REFERENCE — controls a user's ability to reference a table in a referential constraint definition

Access privilege: Access privilege is the authority to access a table using these statements:

- SELECT — controls a user's ability to access data
- INSERT — controls a user's ability to insert new data
- UPDATE — controls a user's ability to change existing data
- DELETE — controls a user's ability to delete existing data

Table access privileges are unique to the SQL option.

Access module execution privilege: The EXECUTE privilege is the authority to execute an access module. The user needs this privilege to run a compiled program.

13.4.2 Granting privileges

A user with the required administration privilege can grant or revoke definition, access, and execution privileges on resources related to SQL processing, using these statements:

- GRANT gives a privilege to a user or group
- REVOKE removes a privilege from a user or group

The owner of a schema can use the **TRANSFER** statement to transfer ownership of the schema to another user or group. The applicable privileges on resources in the schema are also transferred.

13.4.3 Ownership

The concept of ownership is also unique to the CA-IDMS/DB SQL environment.

Ownership at the schema level: Ownership is assigned only at the **schema** level. The initial owner of a schema is the user who created the schema. The owner has all applicable privileges on entities in the schema, as well as the privilege of granting those privileges to other users or groups.

The owner can be a user or a group. Users who are members of a group implicitly hold ownership authority over group-owned resources.

Transferring ownership: Ownership can be transferred to another authorization ID (either group or individual user) without changing the name of an object by using the TRANSFER OWNERSHIP statement. If you transfer ownership of a schema to another user or group, you no longer have any privileges on the entities in the schema.

Example: This example shows the transfer of ownership of the schema MYSCHM to the user GEORGE.

```
transfer ownership of myschm to george;
```

13.4.4 Security checking

Interactive or dynamic SQL statements: To use interactive or dynamic SQL against a database, a user must have the appropriate access privilege for the database.

A security check is issued as each statement is issued. The results of each security check are kept until the transaction ends so that the same check does not have to be repeated for access to the same table or view later in the transaction. This occurs whether you are using an external security package or CA-IDMS internal security.

Precompiled SQL statements: SQL statements embedded in an application program are precompiled and included in an access module prior to runtime.

If you are using an external security package, each SQL statement, precompiled or dynamic, is checked as it is executed and the results kept for the length of the transaction.

If you are using CA-IDMS internal security, the owner of the access module must hold all privileges necessary to execute every SQL statement in the module. If this condition is met, the owner of the access module can execute it and can give execution privileges on the access module to other users (if the owner holds the necessary grantable privileges), regardless of whether they explicitly possess the underlying table access privileges. The security check occurs each time the access module is physically loaded by the runtime system.

13.5 For further information

- On using tuning options, see *CA-IDMS Database Administration*
- On how locking works, see *CA-IDMS Database Administration*
- On using utilities, see *CA-IDMS Database Administration*
- On utility syntax, see *CA-IDMS Utilities*
- On using the Command Facility, see *CA-IDMS Command Facility*
- On security syntax and how security works, see *CA-IDMS Security Administration*

Index

Numerics

- 24-hour processing
 - dynamic area extension 1-24
 - dynamic buffer acquisition 1-25
 - dynamic buffer pool configuration 1-25
 - dynamic database file management 1-21
 - dynamic database name tables 1-25
 - dynamic DMCL 1-23

A

- access modules 10-5
- authorization-identifiers
 - groups 3-8
 - user identification codes 3-8

C

- CA-ADS
 - built-in functions 5-14, 5-16—5-17
 - century date variables 5-14
 - CUA-style interface 5-4
 - CUA-style main menu 5-5
 - integration with security facility 5-3
 - online help 5-5
 - READY/NOREADY 5-18
 - support for SQL 10-7
- CA-Culprit
 - double word binary support 6-4
 - support for SQL 6-3, 10-8
 - upward compatibility 6-3
- CA-ICMS
 - support for SQL 6-3, 10-8
 - upward compatibility 6-3
- CA-IDMS Command Facility
 - See* Command Facility
- CA-IDMS Performance Monitor
 - See* Performance Monitor
- CA-IDMS/DDS
 - data access 7-4
 - definition 7-4
 - general enhancements 7-3
 - maintenance 7-5
- CA-OLQ
 - extended selection criteria 6-6
 - support for SQL 6-3, 10-7
 - upward compatibility 6-3

- CALC keys (SQL) 13-3
 - creation of 9-5
- CICS interface 2-8
- COBOL 85
 - See* COBOL precompiler
- COBOL precompiler
 - COBOL 85 (Fujitsu) parameters 2-31
 - VS COBOL II parameter 2-30
- collating options
 - mixed sort sequence 1-11
 - natural sort sequence 1-12
- Command Facility
 - batch (IDMSBCF) 1-41
 - online (OCF) 1-41
- commenting
 - data dictionary 1-19
 - schema and subschema compilers 1-14
 - system generation compiler 2-27
- compiler, access module 12-3
- cursors (embedded SQL) 10-5

D

- data dictionary 1-15—1-19
 - changes in components 1-15
 - compiler options 1-18—1-19
 - DAY/MONTH/YEAR in DISPLAY ALL 1-17
 - DC OPTION IS MENU 1-18
 - DISPLAY ALL limits 1-16
 - enhanced DISPLAY MODULE/QFILE statement 1-17
 - module-map cross referencing 1-17
 - new areas 1-15
 - resolution of record elements 1-16
 - storage locations 1-15
- database communications
 - architecture 2-5
 - benefits 2-4
 - cross region communications 2-4
 - enhanced CICS interface 2-8
 - new design 2-3
 - NODE statement 2-6
 - RESOURCE TABLE statement 2-6
- database definition
 - See* logical database definition
 - See* physical database definition
- database name table
 - defining 1-5

database name table (*continued*)
 example of definition 1-7

dataspaces
 assignment to 1-20

DCMT and DCUF commands
 invoking from programs 2-19
 modified commands 2-20
 new commands 2-19

deadlock management
 deadlock detection 1-36
 deadlock resolution default 1-36
 specifying a detection interval 1-36
 user exit 30 1-36

definitions, linking logical and physical 1-7

domain constraints (SQL) 8-13

dynamic area extension 1-24

dynamic buffer acquisition 1-25

dynamic buffer pool configuration 1-25

dynamic database file management 1-21

dynamic database name tables 1-25

dynamic DMCL 1-23

dynamic processing features
 See 24-hour processing

dynamic SQL 10-6

E

embedded SQL 10-4
 cursors 10-5

ESA dataspaces
 exploitation of 1-20
 scratch and covered modes 1-20

EXPLAIN statement 12-6

G

GRANT and REVOKE statements
 authorization-identifiers 3-8
 privileges 3-6
 resources and associated entities 3-7

GRANT syntax 3-6
 REVOKE syntax 3-6

I

indexes (SQL) 13-3
 creation of 9-5

indexing options
 duplicates in db-key sequence 1-11
 unlinked indexes 1-10

integrity and constraints 8-13

interactive SQL 10-4

L

local mode processing
 loading from a load area 1-33
 security 1-33
 SYSIDMS parameter file 1-27

lock management
 above the line storage 1-34
 preserving the external area status 1-34
 transient retrieval for areas 1-34

locking
 considerations for SQL-defined databases 13-6

logical database definition
 link to physical 1-7
 schema 1-4
 steps for defining 1-4
 storage of definitions 1-6
 subschema 1-4

M

Mapping facility
 automatic screen painter 5-8
 browse capability 5-7
 compiler error management 5-8
 copy function 5-6
 editing function keys 5-10
 general enhancements 5-3
 help facility 5-11
 uppercase translation by field 5-13

multiple DC/UCF regions
 advantages 2-8
 definition 2-7
 set up 2-7

N

NODE statement
 definition 2-6
 displaying 2-7
 example

O

operating system features
 ESA dataspaces 2-11
 extended addressing 2-9

optimizer
 description 12-3

optimizer (*continued*)
 explaining the access strategy 12-6
 how it works 12-5
output formatting
 data dictionary 1-19
ownership (SQL)
 schema ownership 13-9
 transferring ownership 13-9

P

Performance Monitor
 Application monitor changes 4-6—4-7
 general changes 4-4
 Interval monitor changes 4-7—4-9
 Real-Time monitor changes 4-4—4-6
 SMF type 30 4-4
physical database definition
 database name table 1-4
 DMCL 1-4
 link to logical 1-7
 segments 1-4
 steps for defining 1-5
 storage of definitions 1-6
PL/I
 See runtime support
 IDMSDMLC
 See COBOL precompiler
precompilers
 general parameter changes 2-29
primary keys (SQL) 8-13
privileges
 for access 3-7
 for administration 3-7
 for definition 3-6
 granting 3-9
 granting with GRANT OPTION 3-9
 group privileges 3-9
 revoking 3-10
 using wildcards 3-9
profiles
 associating profiles with users 2-15
 attributes 2-12
 defining 2-13
 definition 2-12
 displaying and accessing 2-15
 substitution parameters 2-14
 system-defined keywords 2-12
 use with nonterminal tasks 2-16
 user-defined attributes 2-13

pseudoconversational support (SQL) 11-10

Q

query device support 2-17

R

RCMs 10-5
referential constraints (SQL) 13-3
 creation of 9-6
 linked 13-3
 unlinked 13-3
referential integrity (SQL) 8-13
RESOURCE TABLE statement
 definition 2-6
 displaying 2-7
 example
resources
 associated entities 3-7
 where definitions are stored 3-7
runtime support 2-29

S

schema and subschema compiler
 enhancements 1-13—1-14
 compiler options 1-14
 non-quoted letters 1-13
 output formatting 1-14
 simplified commenting 1-14
 user exits 1-13
 user-defined comments 1-13
schemas (SQL) 8-10
security
 considerations for SQL-defined databases 13-8
security facility
 architecture 3-4
 description and features 3-4
 DISPLAY facility 3-10—3-14
 GRANT and REVOKE statements 3-6
segments 1-4
 associating them with the DMCL 1-7
 defining 1-5
 sample definition 1-6
separating logical and physical database definitions 1-4
SQL
 access module 12-3
 access to a non-SQL defined database 10-9—10-10,
 11-10
 ANSI and FIPS support 8-4
 CA-ADS support 10-7

SQL (*continued*)

- CA-Culprit support 10-8
 - CA-ICMS support 10-8
 - CA-OLQ support 10-7
 - creating executable modules 10-5
 - data control (security) statements 8-8
 - data definition statements 8-8
 - data manipulation 10-3
 - data manipulation statements 8-8
 - definitions, storing 8-15
 - dynamic statements 10-6
 - embedded in host programs 10-4
 - interactive 10-4
 - pseudoconversational programming 11-10
 - synonyms 10-10
- ## SQL data definition
- description of the language 9-3
- ## SQL-defined databases
- benefits and features 8-5—8-6
 - CALC keys 9-5
 - components of the environment 8-7
 - definition 8-3
 - definition process 9-4
 - index creation 9-5
 - referential constraints 9-6
 - schema creation 9-4
 - table creation 9-4
 - view creation 9-6
- ## SQL, extended features 11-3—11-13
- access to non-SQL defined databases 11-8
 - bulk processing 11-5
 - data types 11-4
 - date/time arithmetic 11-7
 - dynamic access module selection 11-12
 - dynamic SQL 11-8
 - global cursor 11-8
 - modular programming 11-8
 - pseudoconversational programming 11-10
 - scalar functions 11-5
 - session management 11-10
 - special registers 11-7
 - temporary tables 11-8
 - transaction management 11-12
- ## symbolic parameters 1-8
- displacement for VIA sets 1-9
 - index specification 1-9
 - names for subareas 1-9
- ## SYSIDMS parameters
- for physical requirements 1-27
 - local mode security 1-33

- system generation
 - compiler options 2-27—2-28
 - new and modified statements 2-22—2-27

T

- ### table operations
- on columns (PROJECT) 8-12
 - on multiple tables (JOIN) 8-12
 - on rows (SELECT) 8-12
- ### tables (SQL) 8-9
- creation of 9-4
 - operations on 8-12
 - referencing and referenced tables 8-13
 - rows (SQL) 8-9
- ### tuning SQL-defined databases
- CALC keys 13-3
 - clustering 13-4
 - indexes 13-3
 - referential constraints 13-3

U

- ### unique values (SQL) 8-13
- ### user exits
- availability with schema/subschema compilers 1-13
 - availability with system generation compiler 2-27
 - deadlock victim selection 2-17
 - exit 28 2-17
 - exit 29 2-17
 - exit 30 1-36, 2-17
 - security postprocessing 2-17
 - security preprocessing 2-17
- ### utilities
- existing programs 1-39
 - new interface 1-37
 - new statements 1-37
 - security 1-40
 - SQL utilities 13-5

V

- ### VALIDATE, bypassing 1-14
- ### views (SQL) 8-10
- creation of 9-6
 - example 8-10
 - uses 8-10
- ### VS COBOL II
- See* COBOL precompiler

