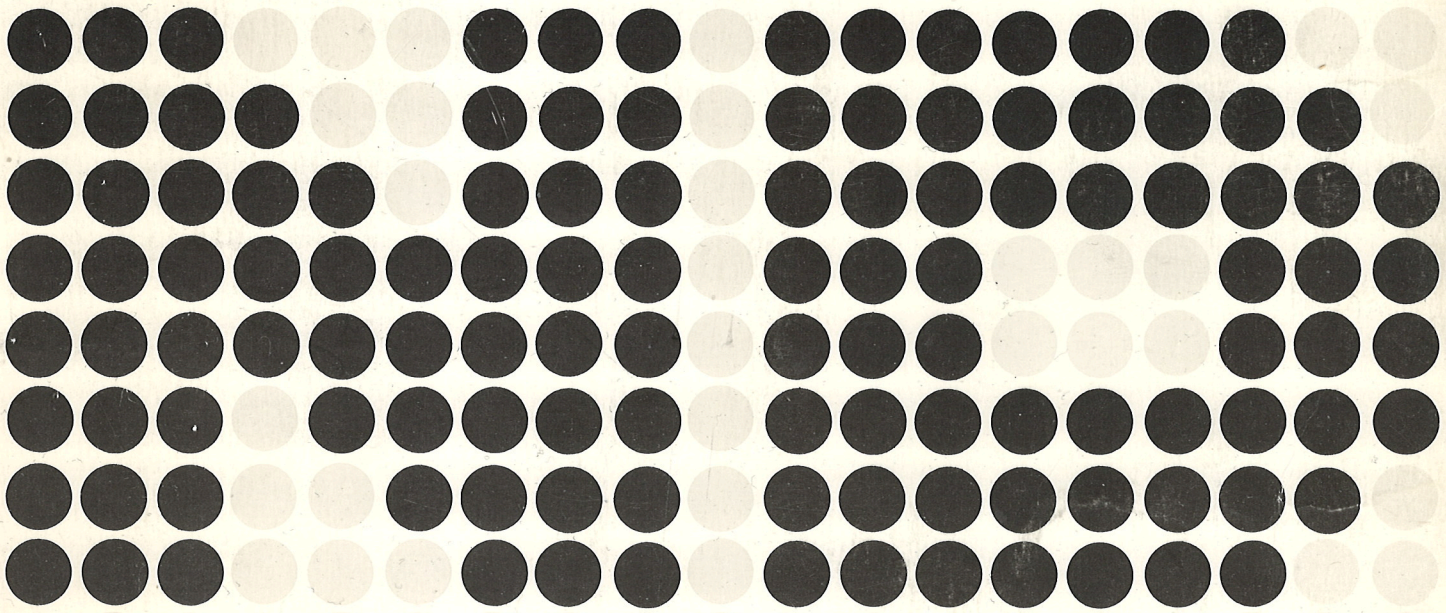
THE DATABASE SYSTEM
SIBAS II/D
ND User's Manual

ND-60.127.03

NORSK DATA A.S



THE DATABASE SYSTEM
SIBAS II/D
ND User's Manual
ND-60.127.03

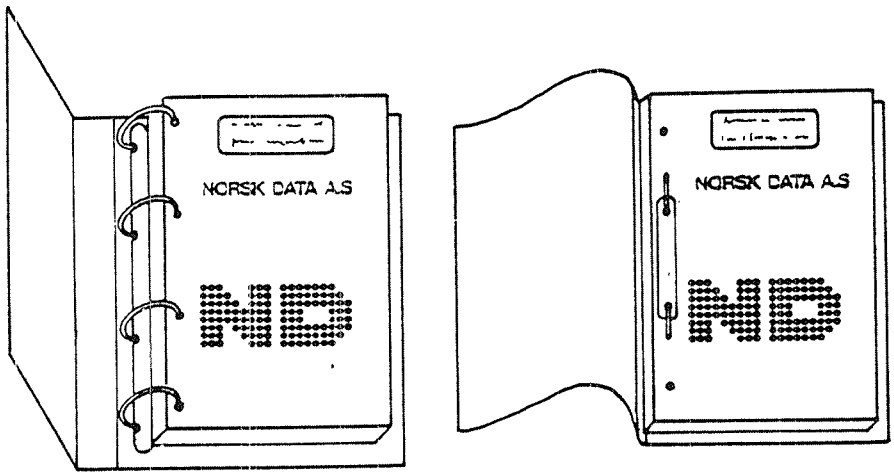
NOTICE

The information in this document is subject to change without notice. Norsk Data A.S assumes no responsibility for any errors that may appear in this document. Norsk Data A.S assumes no responsibility for the use or reliability of its software on equipment that is not furnished or supported by Norsk Data A.S.

Denne håndboken er i løsbladsystem for å forenkle oppdatering. Gamle sider kan fjernes og nye sider settes inn på en enkel måte hvis håndboken er revidert.

Løsbladsystemet gjør det også mulig å plassere håndboken i en ringperm (A) for å beskytte den og for å gjøre det lett å slå opp i den. Ringpermer med 4 ringer tilsvarende hullene i håndboken kan bestilles i to bredder, 30 mm og 40 mm. Bruk bestillingsskjema nederst på siden.

Håndboken kan også plasseres i plastomslag (B). Dette omslaget passer bedre for håndbøker på 100 sider eller mindre enn for større håndbøker. Plastomslag kan også bestilles nederst på siden.



A Ringperm B Plastomslag

Vennligst send bestillingen til det lokale ND kontoret eller (i Norge) til:

Dokumentasjonsavdelingen
Norsk Data A.S
Postboks 4, Lindeberg gård
Oslo 10

BESTILLING

Jeg ønsker å bestille:

..... Ringpermer, 30 mm, nkr 20,- pr. stk.

..... Ringpermer, 40 mm, nkr 25,- pr. stk.

..... Plastomslag, nkr 10,- pr. stk.

Navn .....
Firma .....
Adresse .....
.....
By .....

PRINTING RECORD

Printing	Notes
02/80	Version 01 – Replaces the previous manuals numbered ND-60.057.
07/80	Revision A
	The following pages have been revised:
	xiii to xv
	3–5
	4–11 to 4–16
	5–6 to 5–35
	6–5, 6–6, 6–9 and 6–21 to 6–22
	7–3 to 7–4a
	A–1 to A–2
	D–1
	F–1 to F–2
09/81	Version 02
10/82	Revision A
	The following pages have been revised or added:
	xi to xvi
	1–2, 1–4
	3–5 to 3–6, 3–15 to 3–16
	4–1 to 4–6, 4–15 to 4–16, 4–31 to 4–48
	5–2, 5–5 to 5–6, 5–19 to 5–20a, 5–29 to 5–32, 5–35 to 5–36
	6–17 to 6–20, 6–22
	7–3 to 7–5
	Index
02/83	Version 03

THE DATA BASE SYSTEM – SIBAS II
NORD User's Manual
Publication No. ND-60.127.03
Feb. 1983

NORSK DATA A.S
P.O. Box 4, Lindeberg gård
Oslo 10, Norway

Manuals can be updated in two ways, new versions and revisions. New versions consist of a complete new manual which replaces the old manual. New versions incorporate all revisions since the previous version. Revisions consist of one or more single pages to be merged into the manual by the user, each revised page being listed on the new printing record sent out with the revision. The old printing record should be replaced by the new one.

New versions and revisions are announced in the ND Bulletin and can be ordered as described below.

The reader's comments form at the back of this manual can be used both to report errors in the manual and to give an evaluation of the manual. Both detailed and general comments are welcome.

These forms, together with all types of inquiry and requests for documentation should be sent to the local ND office or (in Norway) to:

Documentation Department
Norsk Data A.S
P.O. Box 4, Lindeberg gård
Oslo 10

The SIBAS database system, originally developed by the Central Institute for Industrial Research (CIIR) in Oslo, Norway, is the first fully developed database system following the CODASYL DBTG recommendations for implementation on a mini computer.

The system described in this manual has also been implemented on other computer systems, such as UNIVAC 1100, IBM 360/370, CDC CYBER and DEC 10. It has been expanded and optimized in a joint development project with the company offering SIBAS on large computer systems: A/S Shipping Research Services, Oslo, together with the Central Institute for Industrial Research, Oslo, and Norsk Data A.S.

The implementation of SIBAS on the ND computers utilizes the advanced facilities of the SINTRAN III Virtual Storage Operating System, and offers multiple user programs simultaneous access to the same data base in a controlled and secure manner, thus minimizing the amount of additional routines in the user's programs.

Norsk Data wishes to thank the user's group reference committee for their contributions and kind assistance in the writing of this manual. Their comments have proved to be very helpful and we look forward to receiving all users cooperation in the future.

Norsk Data A.S.
Software Department

PREFACE

The Product

This manual describes the SIBAS II Database Management System, version D.

SIBAS II is delivered as one standard package. There are additional modules.

The standard system contains the following modules:

Modules:

SIBAS System Generation
- SIB-SYSGN: BATC
- SIB-LOAD:BATC

SIBAS Real-Time Segments
- SIB-MAIN: BPUN
- SIB-REEN: BPUN
- SIB-OPEN: BPUN
- SIB-WORK: BPUN

SIBAS Data Manipulation Libraries
- SIBLIB-2N-MH
- SIBLIB-1N-MH
- SIBLIB-1R-MH
- SIB2-DML-B-MH
- SIB2-DML-R-MH

SIBAS Background Programs
- SIB-DRL: PROG
- SIB-DBM: PROG
- SIB-SERV: PROG

Additional Modules are:

SIBAS Backend Programs

SIB-LOOK-AT-LOG:PROG

SIBAS I/O modules for delayed update feature (SIBIO)

SIBINTER

SIBAS Conversion Aids
- SIB-CONVERSION-PLANNING: SYMB
- SIB-CONVERT-SCHEMA: PROG
- SIB-CONVERT-DATA: PROG

THE READER

SIBAS II User's manual is written for a wide variety of users, but the different chapters are oriented towards different classes of readers:

Programmers, who write application programs which make use of SIBAS.

Database administrators who are concerned with secure and efficient operations of the overall system.

Any one else who is generally interested in database management systems.

The database administrator should read the whole manual.

The application programmer will be more concerned with Chapters 4 and 7; Data Manipulation and Error Reporting.

The "generally interested" reader may limit him/herself to the first two chapters.

PREREQUISITE KNOWLEDGE

The first two chapters do not need any prerequisite knowledge, but it is assumed that application programmers are familiar with the SINTRAN III operating system and at least one programming language. More specifically, they should be familiar with the concept of calling subroutines since SIBAS is accessed via subroutine calls.

The database administrator must be familiar with the real-time features of SINTRAN III since SIBAS makes extensive use of them.

THE MANUAL

Chapters 1 and 2 are an introduction to SIBAS and should give the necessary background to go on to the following chapters.

Chapter 3 gives a detailed description of how one can define or redefine a database — it is of special interest for a database administrator, but may also be of interest to a programmer.

Chapter 4 gives a detailed description of how to call SIBAS data manipulation functions. This chapter is oriented towards application programmers.

Chapter 5 describes how to administrate and operate a SIBAS database. This chapter is written for database administrators.

Chapter 6 is a description of some utility programs provided with SIBAS. This chapter is also written for database administrators.

Chapter 7 is a list of errors and how to handle them.

The appendicies give reference information in a compressed form.

Related Manuals:

SINTRAN III User's Guide
SIBINTER User's Guide
NORD RELOCATING LOADER User's Manual

TABLE OF CONTENTS

+ + +

Section:		*Page:*
1	INTRODUCTION	1—1
1.1	The SIBAS Database System	1—1
1.1.1	SIBAS Backend	1—2
1.2	ND SIBAS Implementation	1—3
1.3	SIBAS Using the ND-500 System	1—4
1.4	SIBAS Modules	1—6
2	SIBAS PRINCIPLES	2—1
2.1	The Database Concept	2—1
2.2	Data Structure	2—9
2.2.1	Items	2—12
2.2.2	Group Items	2—12
2.2.3	Record Types	2—13
2.2.4	Search Keys and Indexes	2—15
2.2.5	Realm	2—17
2.2.6	Database	2—18
2.3	Data Relations	2—20
2.3.1	Search Regions	2—20
2.3.2	Sets	2—22
2.3.2.1	Set Items	2—24
2.3.2.2	Set Occurrences	2—24
2.3.2.3	Chain Representation of Set Types	2—25
2.3.2.4	Storage Class	2—30
2.4	Data Manipulation	2—33
2.4.1	Access Principles	2—33
2.4.1.1	General	2—33
2.4.1.2	Currency Indicators	2—35
2.4.2	Connecting and Disconnecting, Inserting and Removing	2—38
2.4.2.1	Connecting and Disconnecting	2—38
2.4.2.2	Inserting and Removing from an Index	2—42

Section:		*Page:*
2.4.3	Concurrent Processing	2—42
2.4.3.1	Database Reservation	2—43
2.4.3.2	Realm Usage Modes and Realm Protection Modes	2—43
2.4.3.3	Record Level Lock Out	2—44
2.4.3.4	Notification of Change	2—44
2.4.4	Privacy System	2—45
2.4.4.1	Privacy on Database Level	2—46
2.4.4.2	Privacy on Record Occurrence Level	2—46
2.4.4.3	Summary of the Setting of Current Password	2—47
3	DEFINITION/REDEFINITION LANGUAGE (DRL)	3—1
3.1	Introduction	3—1
3.2	How the Definition/Redefinition Module Works	3—3
3.3	Global Rules	3—5
3.4	Start Initiation	3—7
3.5	Start Redefinition	3—8
3.6	End/Exit	3—9
3.7	New OS File	3—10
3.8	New System Realm	3—11
3.9	New Serial Realm	3—12
3.10	New Calc Realm	3—13
3.11	New Item	3—15
3.12	New Group	3—17
3.13	New Set	3—19
3.14	New Index	3—21
3.15	Delete Set	3—22
3.16	Delete Index	3—23
3.17	Delete Item	3—24
3.18	Delete Group	3—25
3.19	Change System Realm	3—26
3.20	Change Serial Realm	3—27
3.21	Change Calc Realm	3—28
3.22	Change Set	3—30
3.23	Dimension Database Parameters	3—32
3.24	How to Run DRL on the Computer	3—34
3.25	Examples	3—35
4	DATA MANIPULATION LANGUAGE (DML)	4—1
4.1	General	4—1
4.2	Parameter Descriptions	4—2

Section:		*Page:*
4.2.1	Open Database	4—5
4.2.2	Close Database	4—6
4.2.3	Ready Realm	4—7
4.2.4	Finish Realm	4—10
4.2.5	Direct Find	4—11
4.2.6	Relative Find	4—14
4.2.7	Find Set Owner	4—17
4.2.8	Get, Getn, Get Indexes	4—18
4.2.9	Modify	4—20
4.2.10	Store	4—22
4.2.11	Erase	4—24
4.2.12	Connect	4—25
4.2.13	Disconnect	4—27
4.2.14	Insert	4—28
4.2.15	Remove	4—29
4.2.16	Remember	4—30
4.2.17	Forget	4—31
4.2.18	Lock	4—32
4.2.19	Unlock	4—33
4.2.20	Change-Password	4—33
4.2.21	Accept	4—34
4.2.22	Erase Element	4—35
4.2.23	Accumulate	4—36
4.2.24	Fetch-and-Get	4—37
4.2.25	Get Schemas Information	4—38
4.2.26	Transaction Unit	4—41
4.3	Host Language Considerations	4—43
4.3.1	FORTRAN	4—44
4.3.1.1	General Rules for Fortran on The SIBAS-500	4—44
4.3.1.2	Standard 'Cookbook' for Programming Fortran Applications	4—45
4.3.2	COBOL	4—50
4.3.2.1	General Rules for Cobol on The SIBAS-500	4—51
4.3.2.2	Standard 'Cookbook' for Programming Cobol Applications	4—52
4.3.3	PLANC	4—59
4.4	How to Load Application Programs	4—60
4.4.1	Description	4—60
4.4.2	Different types of simulators	4—60
4.4.3	Choosing simulators	4—61
4.4.4	Loading Nonreentrant Programs with SIBAS	4—61

Section:		*Page:*
4.4.5	Loading 2BANK Programs with SIBAS	4—62
4.4.6	Loading Reentrant Programs with SIBAS	4—63
4.4.7	Loading Real Time Programs with SIBAS	4—64
4.4.7.1	Cooperating RT Programs Working as one "SIBAS USER"	4—66
4.4.8	Table of SIBAS Simulators (Libraries)	4—66
4.4.9	Applications on ND-500 Systems	4—67
4.4.9.1	Applications running on the 500 CPU	4—67
4.4.9.2	Applications running on the 100 CPU	4—67
5	DATABASE ADMINISTRATION	5—1
5.1	Real-time Organization of SIBAS	5—2
5.1.1	How is SIBAS organized?	5—2
5.1.2	Organization of SIBAS on an ND-500 System	5—6
5.2	SIBAS States	5—8
5.3	Logging and Recovery Facilities	5—10
5.3.1	General	5—10
5.3.2	Checkpoint	5—11
5.3.3	Routine Logging	5—12
5.3.3.1	Critical Sequence	5—13
5.3.3.2	Reprocessing	5—14
5.3.4	Delayed Updating	5—15
5.3.4.1	SIBAS I/O System (SIBIO)	5—15
5.3.4.2	The Update File	5—16
5.3.4.3	Taking Checkpoints	5—17
5.3.4.4	Update-In-Place	5—18
5.3.4.5	Roll Back	5—19
5.3.5	Before Image Logging	5—19
5.3.6	Backup	5—20
5.3.7	System Failure/Restart	5—20
5.3.7.1	Restart from a Backup Copy and a Routine Log	5—21
5.3.7.2	Restart from a Database with Update File/ Before Image and Routine Log	5—21
5.4	Detailed Description of the Calls	5—22

Section: *Page:*

5.4.1	Start/Stop SIBAS/Get State	5—24
5.4.2	Run/Pause/Recover/Finish/Set Passive/ Repro-Status	5—25
5.4.3	Initiate-Log	5—27
5.4.4	Begin/End Sequence	5—28
5.4.5	Set Routine Logging On/Off	5—29
5.4.6	Log Message	5—30
5.4.7	Write-Log-Buffer-Onto-Routine-Log	5—30
5.4.8	Checkpoint	5—31
5.4.9	Roll-Back	5—32
5.4.10	Set-Conditions-For-Reprocessing	5—32
5.4.11	Reprocess-Routine-Log	5—34
5.4.12	Update-Data-Base-In-Place	5—35
5.4.13	Set SIBAS System Number	5—36
5.4.14	Reserve/Release SIBAS	5—36
5.4.15	Execute-Macro	5—37
5.4.16	DBA Calls	5—38
5.4.17	Force-Close Database	5—38
5.5	Special SIBAS-500 Features	5—39
5.5.1	Calls with Different Functions	5—39
5.5.2	Calls Not Available	5—39
5.5.3	Exceeding the Size of a Direct Routine Log	5—40
5.5.4	SIBAS-500 MACROS	5—40
5.6	How to Install SIBAS	5—40
6	UTILITIES	6—1
6.1	Database Maintenance Module	6—1
6.1.1	Introduction	6—1
6.1.2	Start	6—3
6.1.3	Exit, Stop the DBM Module	6—3
6.1.4	Ready Realms	6—4
6.1.5	Finish Realms	6—4
6.1.6	Print	6—5
6.1.7	Patch	6—6
6.1.8	Reset-Error-Flags	6—7
6.1.9	Privacy	6—8
6.1.9.1	General	6—8
6.1.9.2	Define Password	6—13
6.1.9.3	Remove Password	6—14
6.1.9.4	Display Password/Privacy	6—14
6.1.10	Index Compression	6—15
6.1.11	Consistency Checking	6—16

Section:		*Page:*
6.1.11.1	General	6—16
6.1.11.2	Calc Key Verification	6—18
6.1.11.3	Index Key Verification	6—19
6.1.11.4	Set Verification	6—20
6.1.11.5	Page-Link Verification	6—22
6.1.12	Free-Space-Statistics	6—22
6.1.13	Example	6—23
6.1.14	Unload/Load	6—24
6.1.15	Make-modefile for Unload/Load Program	6—25
6.1.16	Clear System-Realm	6—26
6.2	SIBAS Service Program	6—27
6.2.1	SIBAS-Service Extensions SIBAS-500	6—29
7	ERROR AND EXCEPTION CONDITIONS	7—1
7.1	Fatal Errors	7—2
7.2	Interface and Simulator Errors	7—3
7.3	DML Diagnostics, Database Exception Conditions (DBECS)	7—6
7.4	Run-time Message — Messages from SIBIO	7—16
7.5	Run-time Message — from SIBAS	7—17

Appendix:		*Page:*
A	SUMMARY OF THE DML STATEMENTS	A—1
B	SUMMARY OF THE SIB-DRL STATEMENTS	B—1
C	SUMMARY OF THE SIB-DBM STATEMENTS	C—1
D	SUMMARY OF THE SIB-SERVICE STATEMENTS	D—1
E	SUMMARY OF THE DATABASE EXCEPTION CONDITIONS	E—1
F	SUMMARY OF THE DML ROUTINE LOG NUMBERS	F—1
G	CONSTANTS AND LIMITATIONS	G—1
	INDEX	

1 INTRODUCTION

1.1 THE SIBAS DATABASE SYSTEM

SIBAS is a Database Management System, originally designed by the Central Institute for Industrial Research in Oslo, and presently implemented on IBM, UNIVAC, CDC, DEC 10, SEL and ND computers. The data management capabilities correspond to the recommendations contained in the CODASYL report.

The implementation of SIBAS for ND computers contains some extensions as compared to the CODASYL report. Another important characteristic of this implementation is the strict orientation towards a multiprogrammed, terminal oriented computing environment. This means that many users may access one database simultaneously, and also that it is possible to have several databases in a system at the same time. Great efforts have been made to provide safe and effective tools for control of data integrety and security.

The ND SIBAS system includes a data definition and redefinition facility, a run-time database manipulating package and a comprehensive set of interactive utility programs.

In general terms, a Database Management System (DBMS) is a software concept or environment which allows a database to be structured and accessed in a standarized way. It includes a set of program functions which the application programmer uses when operating in the DBMS environment. In this way, his own work will be reduced, since he does not need to solve the corresponding design problems and program the general service routines himself. The ND SIBAS system has been extensively used in a number of installations since 1975 and is today a well-proven and reliable system.

Using a DBMS is a way of adding intelligence to the computer system. Data items may be connected to each other depending on defined relational patterns. Those connections could well be done in the logic of a program using ordinary data files, but such a solution would be expensive both in development and maintenance costs. The DBMS allows application programs to be reduced in size and complexity. However, the relations between data will be described as pointers and tables within the database. This means that the overhead in program complexity is changed into an overhead in storage space.

By "overhead in space" we here mean the difference between the total size of the database and the size of the "pure data". The overhead depends on the access facilit es desired and deserves careful consideration by the database designer.

1.1.1 SIBAS BACKEND

The SIBAS DBMS may now be accessed from a remote ND-500, ND-100 or ND-10 computer through a transparent, safe and efficient communication package.

The new product may be used to increase the capacity of database oriented applications because it makes it possible and easy to implement a number of configurations.

As an example, one can imagine a system based on one machine with sizable SIBAS activity. To increase the capacity of the system, one machine may be added, and the total load split in such a way that one machine runs only applications (Application machine), and the other runs both database(s) and some applications (Database machine). Such a change may be made with small modification in the application programs. The system may be upgraded later on with more machines to further increase the capacity.

Another example is the case where a database is held at a central site, but may be accessed from (an) other computer(s) through telephone links.

The software features automatic checks and retransmission on both sides. Intermittent power failures are also taken care of.

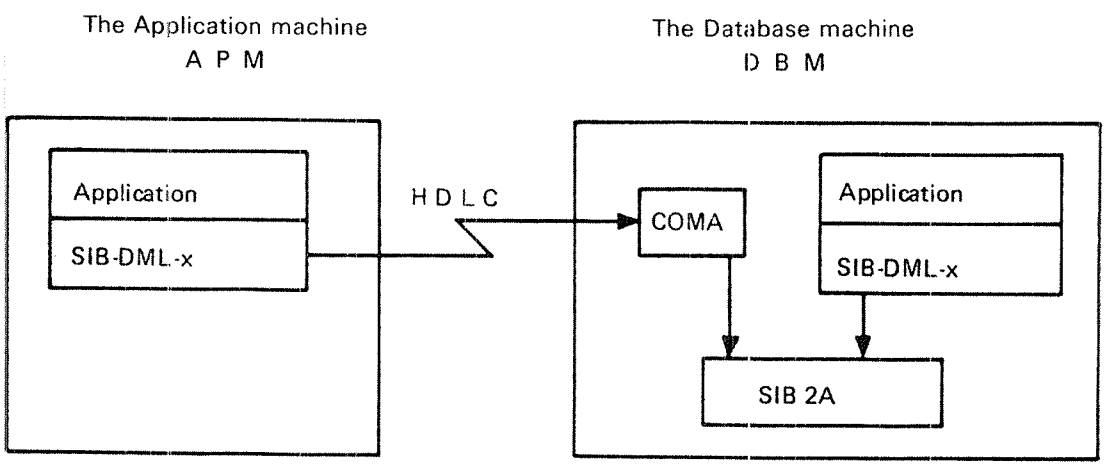


Figure 1.1: In this case, one database is accessed by several applications divided between two computers.

1.2 ND SIBAS IMPLEMENTATION

In the ND SIBAS implementation, the real-time processing facilities of SINTRAN have been used to a great extent. The run-time Database Control System (DBCS) is loaded and operated as a real-time program. Since SIBAS code is reentrant, several databases may be handled concurrently with only a limited increase in memory requirements.

The application programs may be run in time sharing, batch or real-time mode, and several users may call one database simultaneously. Only one call is processed by one SIBAS program at a time, however, and SINTRAN facilities are used to queue the calls.

Application programs communicate with the SIBAS system by means of a set of subroutine calls. The subroutines execute at the priority level of the calling program, and cause a call to be made to the separate SIBAS process by means of the internal device mechanism. The calling program is then halted until the answer arrives from the higher priority SIBAS system.

The Norsk Data versions of COBOL, FORTRAN, BASIC, PLANC and MAC all contain a CALL facility enabling application programs to communicate with the SIBAS subroutine package.

1.3 SIBAS USING THE ND-500 SYSTEM

SIBAS is implemented on the ND-500 computers and uses their huge address space and fast CPU. Running a database by means of a SIBAS-500 process will keep the whole database in virtual memory by using the «file-as-segment» concept. No explicit disk transfers are executed and consequently a reduced I/O overhead is achieved. If enough (physical) memory is available, the whole database may actually reside in physical memory at run-time.

All the SIBAS-II C-version DML-calls (with a few minor exceptions, see section 6.1) are implemented on the ND-500. Their functions are exactly the same as in a SIBAS-100 system and *the database format is also identical.* This means that the same applications (and databases) may use both SIBAS-100 and SIBAS-500, and that applications (and databases) are easily moved between an ND-100 and an ND-500 system.

Today, the definition/redifinition module (DRL) runs on the 100 CPU (of the ND-500 system) together with the database maintenance module (DBM). These modules are the same as on a SIBAS-100 system, but later on the DBM will be implemented on the 500 CPU. The SIBAS-service program can be used to supervise and control both SIBAS-100 and SIBAS-500 processes even when they are running simultaneously on an ND-500 system. We recommend starting and controlling SIBAS-500 processes by using this standard service program. Control can also be obtained by including SIBAS-calls in applications. For on-line interaction with a SIBAS-500 process, without having to write application programs, the standard SIBINTER may be used directly. SIBINTER will run on the 100 CPU.

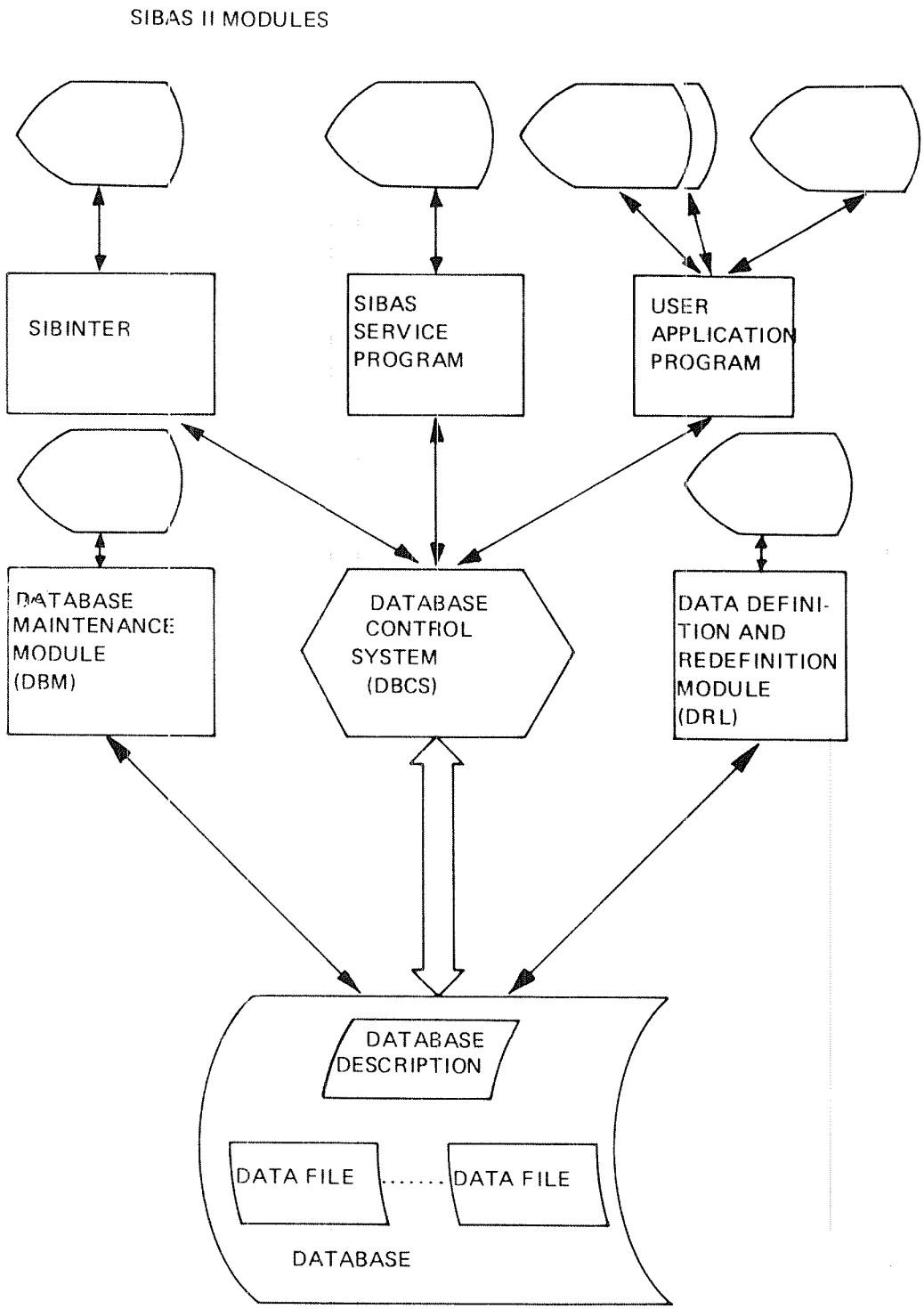


Figure 1.2: SIBAS Modules

1.4 SIBAS MODULES

The SIBAS Database Control System (DBCS) is the module called from the application program for storing, reading, modifying and deleting the information in the database. It is run as a real-time task.

The SIBAS data definition and redefinition module (DRL) is used for defining a database, i.e., defining the structure, the access keys, the size of the database and for changing such parameters.

SIBINTER is a module that enables users to access a SIBAS data base interactively without having to write application programs. It is particularly well suited for educational purposes.

The SIBAS service program is background utility to start/stop and manage the DBCS module.

The SIBAS Database Maintenance (DBM) is a background utility used mainly to check a large amount of data. It has some repair facilities and does not make use of the DBCS.

2 SIBAS PRINCIPLES

2.1 THE DATABASE CONCEPT

What is a database? Well, this is a rather complicated question to answer. Since a good understanding of some basic principles is essential for reading the rest of this manual, this chapter will discuss an example of information storage and retrieval. The description in this chapter should answer the question to such a degree that the reader will be able to understand the detailed description of the SIBAS database system in the following chapters.

Let us assume that we run a railway company. It has been growing for some years, and we are having trouble in planning and maintaining time tables, scheduling the utilization of engines and cars, planning the work of our personel, etc. What do we do? We design a database and make the computer help us keep track of our business.

First, we think of a file describing all our engines. The following information ITEMS need to be stored for each one of our 159 engines:

— serial number
— supplier name
— type
— latest service inspection
— capacity
— allocated to train number

Then we decide to have a similar file for all our cars. It must contains the following information for each car:

— serial number
— supplier name
— type
— number of passengers/tons load
— allocated to train number

We also want a personel file containing the following information for each person:

— name
  family name
  surname
— home address
— position (engine driver, conductor, etc.)
— allocated to train number

For convenience, we have grouped the two basic items Family Name and Surname in a group called Name. We call this construct a GROUP ITEM and use it in all cases when we want the entire name of the person.

Now we have three files, and we find that in the engine file there will be 159 RECORDS because we have 159 engines, each record giving us a limited description of one engine. Similarly, there will be one record for each car or person in the other files. We have also specified the records with respect to the information contained in them, and we notice that all records in one file quite naturally will have the same length.

Having our three basic files, we now want to make up a time table. In our data-base we illustrate this with a fourth file, the time table file, containing the following information:

— train number
— line number
— time of departure
— average speed
— time of arrival

But now we want to describe trains containing a variable number of cars.

What we do to assemble a train is to select the desired number of cars and put them together into a SET. In this set, we also include the engine, engine driver and conductor. Finally, we assign the train (set) to a specific entry in the time table, which is identified by means of a train number.

In the database, we create the set simply by storing the train number in the engine record, the car records, the engine driver record and the conductor record. The time table record is said to be the SET OWNER and all the others are called SET MEMBERS.

In our example, it should be apparent that a file is a collection of similar but unrelated records. With the concept of sets, we have included relations between records in the discussion. We consider a set to be a collection of records having some common characteristic, in this case the train number. While the records are restricted to fixed length, depending on the data elements contained in them, a set may contain a variable number of member records.

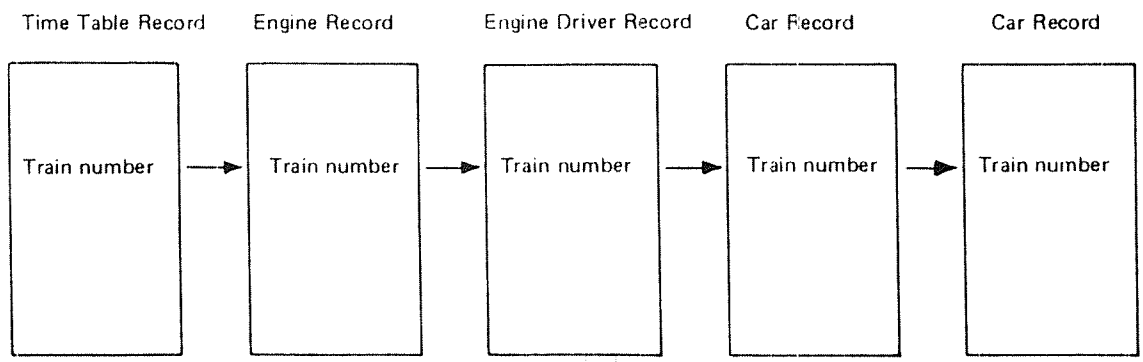


Figure 2.1: The Set

Records in one file have the same length and organization, but contain various data. Engine number 11 is not the same as engine number 137, but they are described in the same way in the file. The way the description appears is called the RECORD TYPE and each individual description is called RECORD OCCURRENCE. Those expressions are frequently used in this manual.

Similarly, we use the expression SET TYPE and SET OCCURRENCE. The set type in this example gives a description of a variable train length. The set occurrence describes a specific train connected to a specific line and departure time. Obviously, we have one set occurrence for each record occurrence in the time table file, i.e., for each set owner record.

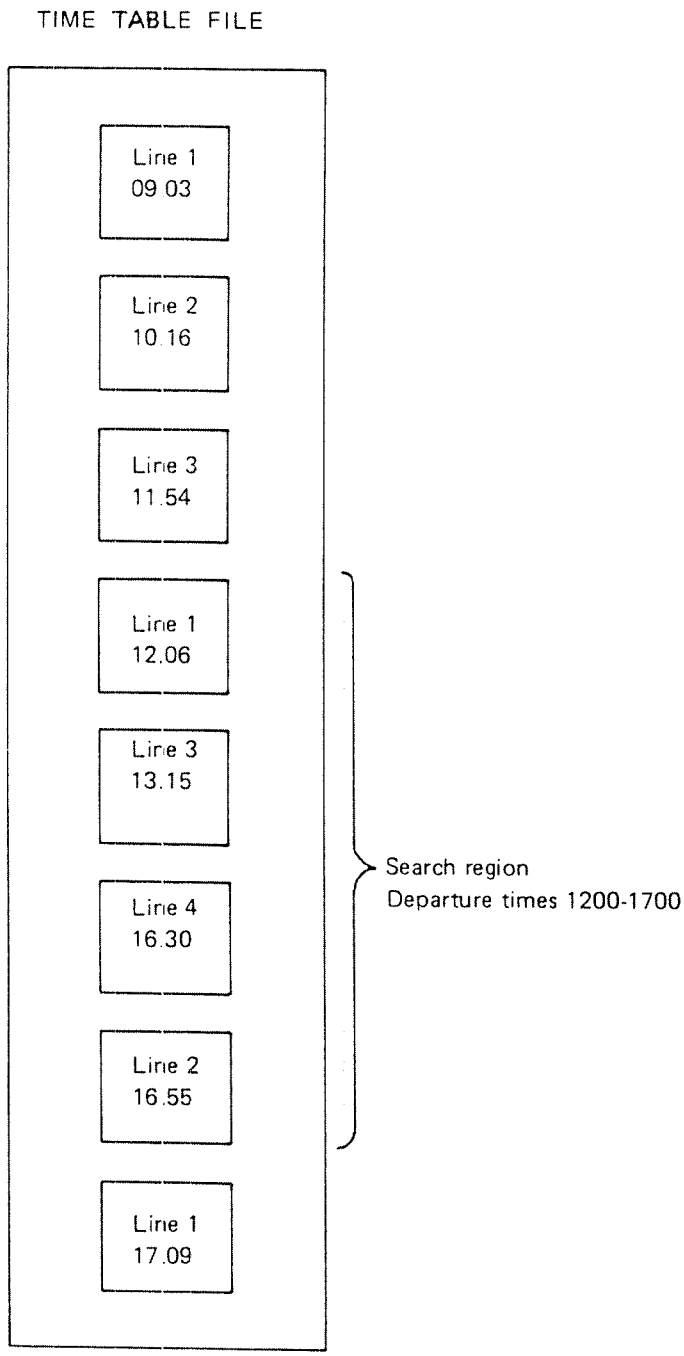


Figure 2.2: The Search Region

Now that we have our railroad database, we want to print out time tables for each line. This means that we want to scan the time table file and select all records for line 1, line 2, etc. This can also be thought of as a division of the time table into classes, one for each line.

For other purposes, we may also need other classes within the time table register, for example all departures between 12:00 and 17:00 hours, or even all records in the time table file. In SIBAS, such classes of records within a file are called SEARCH REGIONS.

Our railway network branches out from a central station in a treelike structure without connections between the different branches.

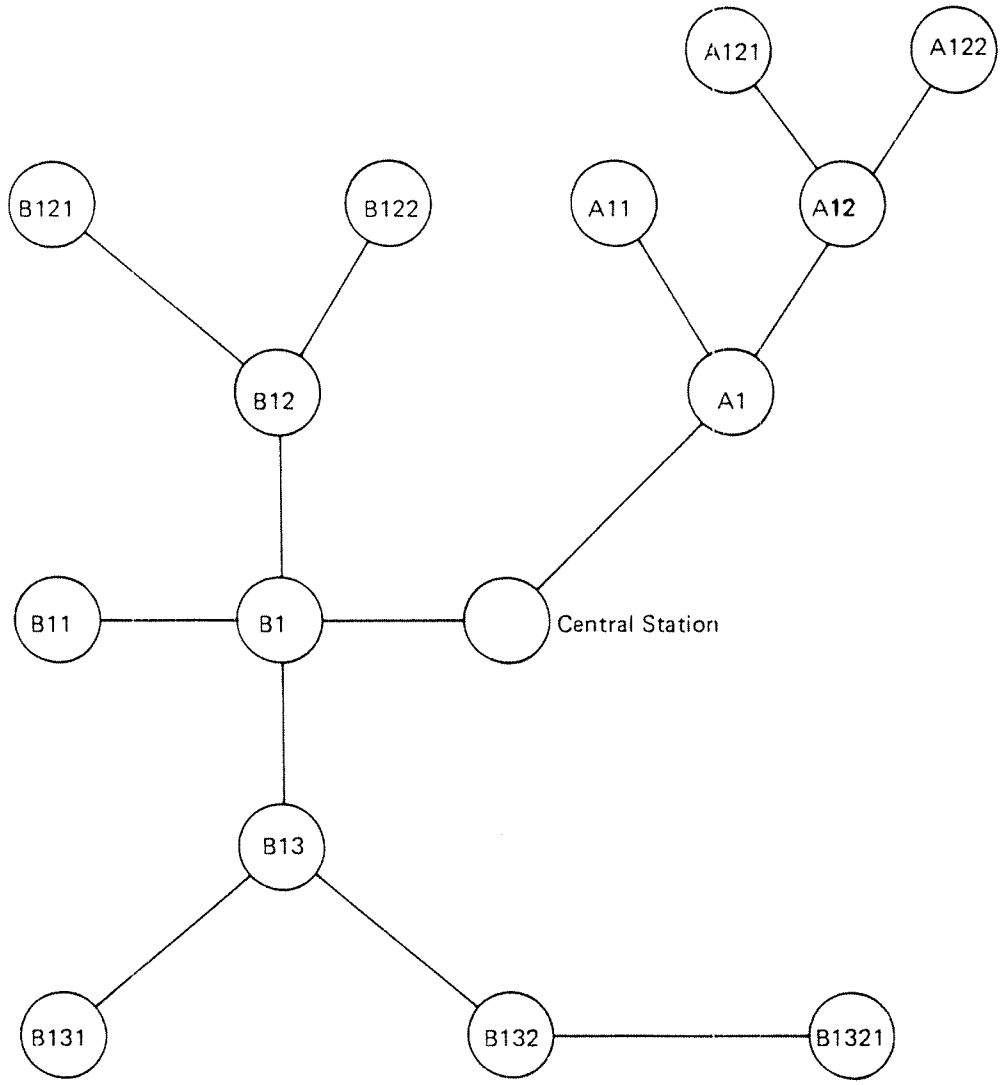


Figure 2.3: The Railroad Network

We want to add a description of this network to our database. The following items shall be included:

— station name
— name of inner station
— distance to inner station

By inner station, we mean the one which is the next station when travelling towards the central station. The station file will have the following layout:

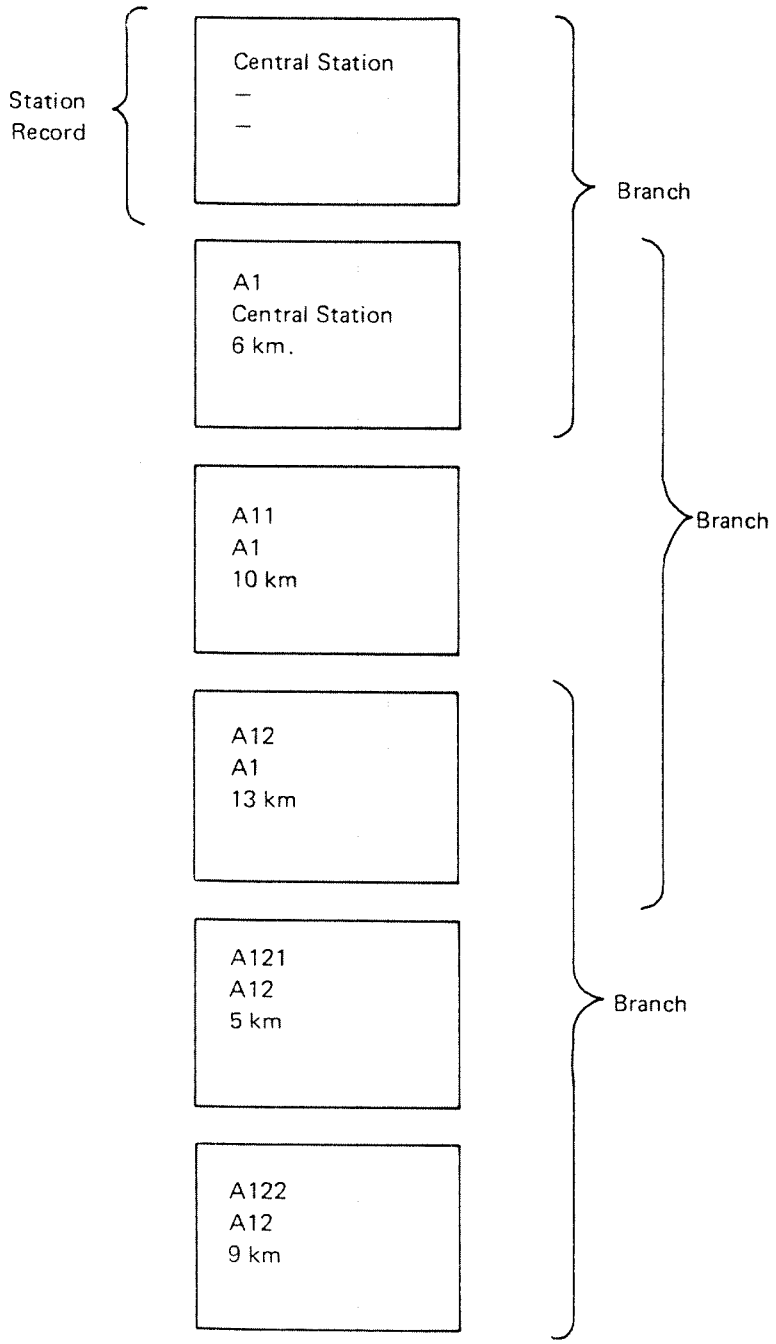


Figure 2.4: The Station File

In this file, all stations directly connected to each other are gathered in groups or classes in a similar way as were the records in the train set earlier. But in the train set, one record type was the set owner and other record types were set members. Here in the station register, the set owner record type and the set member record type are the same. However, the set owner item (station name) is different from the set member item (inner station name). This grouping of equal records is called an INVOLUTED SET.

Now our database is almost ready. We only need one extra device to make it useful. Consider the personnel file containing information on all the persons working in our company.

Quite often, we want to select the record for one specific person because we, for example, want to increase his salary. We know his name and want the rest of the information. We would like to supply the name to an access mechanism in the database system and to get the corresponding record back.

The device that facilitates this is the definition of a RECORD KEY. In this case, we use the name of the person, which happens to be a group item. Any item or group item can be defined to be a key to the file.

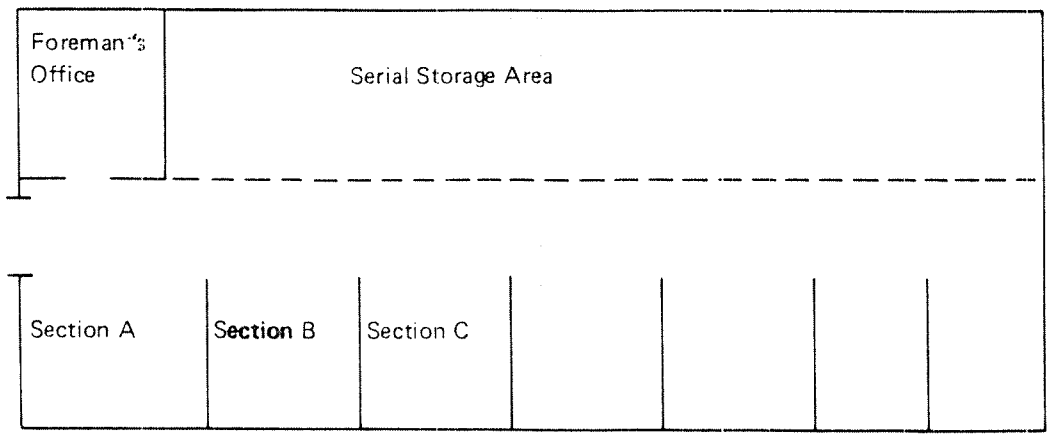


Figure 2.5: Storehouse Layout

In the storehouse the railway company has at the central station, some kinds of goods are stored in a shared area such that arriving articles are just put into the first available place in the area. Other articles, like dynamite, animals, etc., are always put in the same places or sections in the store house. Each of these two methods of allocating space in the storehouse has certain advantages and disadvantages.

The utilization of space is probably better in the first case, but it may be necessary to search for the different articles there. In order to reduce the search time, the foreman saves all the delivery notes in alphabetical order and makes a little note on them as to the location of the goods.

In the other area, on the other hand, the staff always knows exactly where to find a specific article. But the utilization of space is a bit uneconomical. For convenience, the foreman saves the deliverly notes in a bunch for each section.

We use quite similar methods for storing records in the data base. The first case in the storehouse corresponds to the SERIAL LOCATION MODE in the data base. Arriving records are stored in the first available space, and the values of the record key and location are stored in an INDEX TABLE, in sorted order. When the record is to be found again, the index table is searched until the key value is found and the location code is used to find the record.

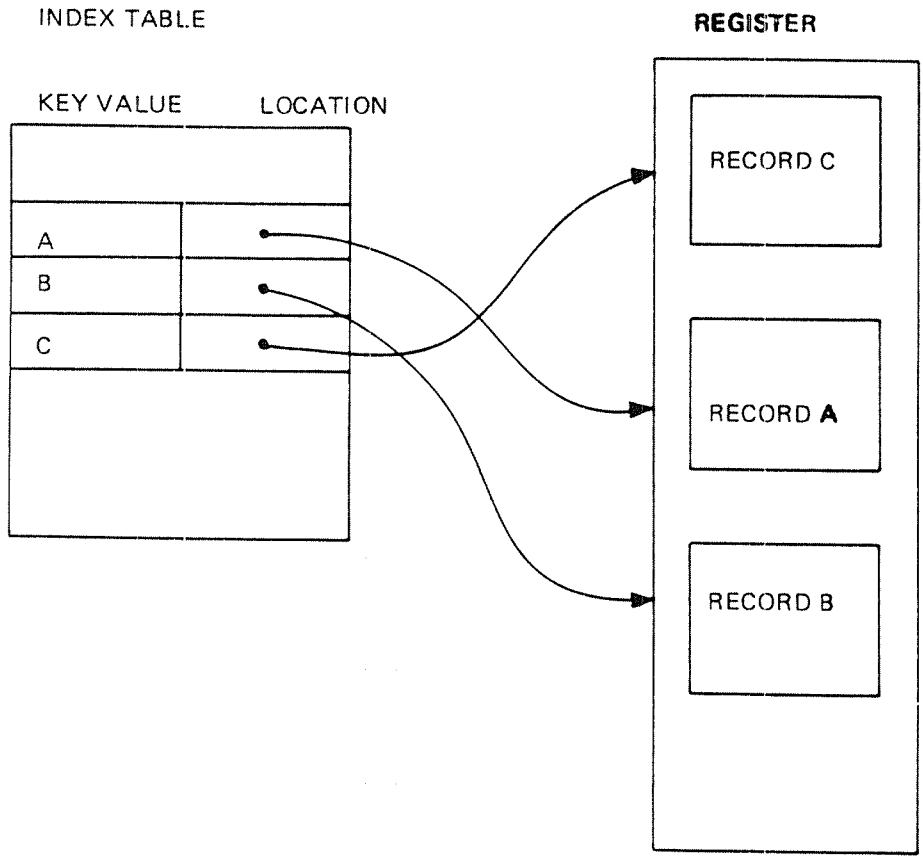


Figure 2.6: Indexed Register

The other case in the storehouse corresponds to the CALCULATED LOCATION MODE in the database. Here we divide the available storage space into a number of boxes or BUCKETS. When a record is to be stored in this part of the database, we take the key value of the record and calculate a bucket number from it. We have chosen the calculation method such that an approximately equal number of records will be stored in each bucket. When a record is to be retrieved from the database, we calculate its bucket number from the key value, go into the bucket and search it through until we find the record.

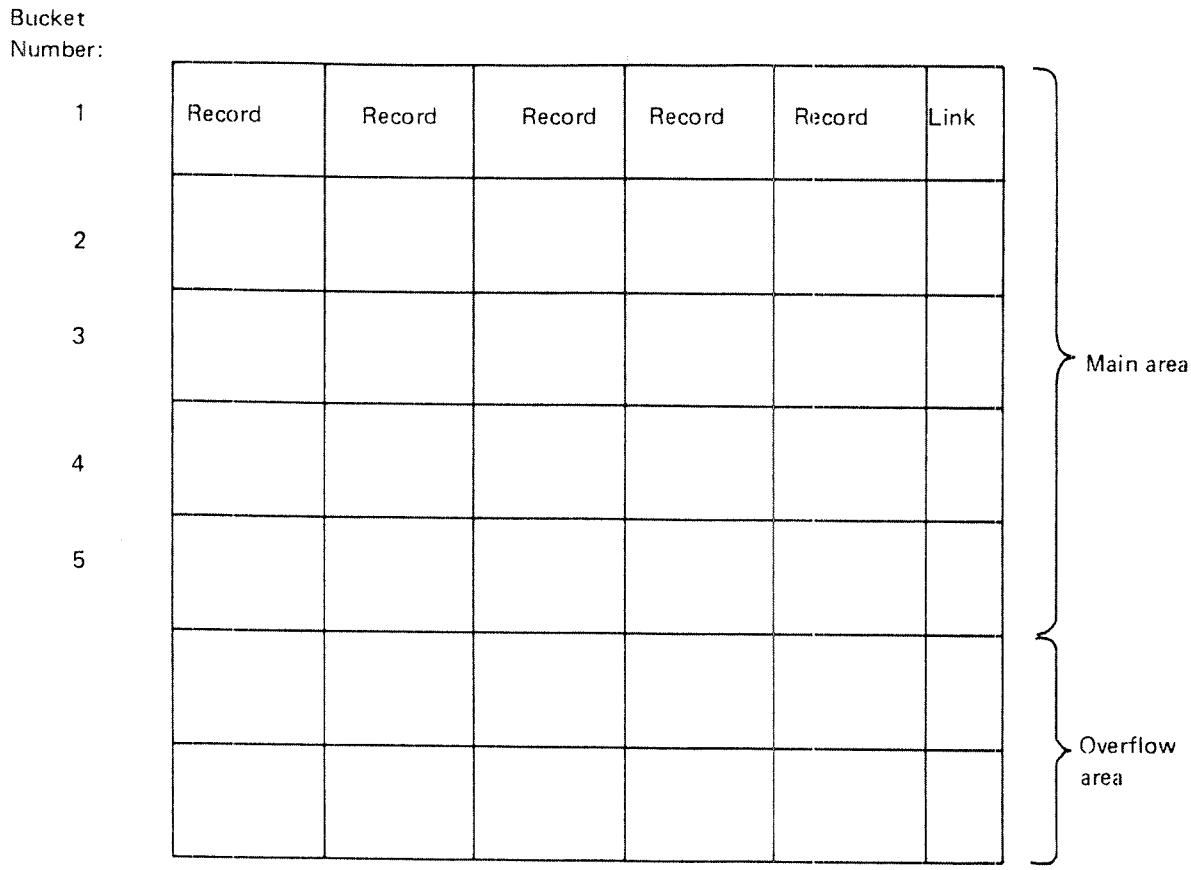


Figure 2.7: Buckets

Unfortunately, we cannot rely on the assumption that records will be equally distributed over the buckets. We must prepare ourselves for the case when a bucket overflows. We do it by reserving a number of buckets to serve as OVERFLOW BUCKETS. When we want to add a new record in a bucket that is already filled up, we make a little note in it and place the record in the overflow bucket.

Now that this little discussion reaches its end, you may feel you still don't know what a database *really* is. But is is really quite simple. A database is nothing but well defined data and relations between data, just like our little railway company example. A real database tends to be somewhat larger, but is nevertheless built up with the same building blocks.

2.2 DATA STRUCTURE

Data that are stored in the database have a certain structure, defined in the database definition. This structure defines the name, length, and role of each single data element. Data definition will be described in this section.

Data elements may also be related to each other due to common characteristics, etc. Such aspects of the database will be discussed in the next section.

The structuring principles used in SIBAS are outlined in the following figures.

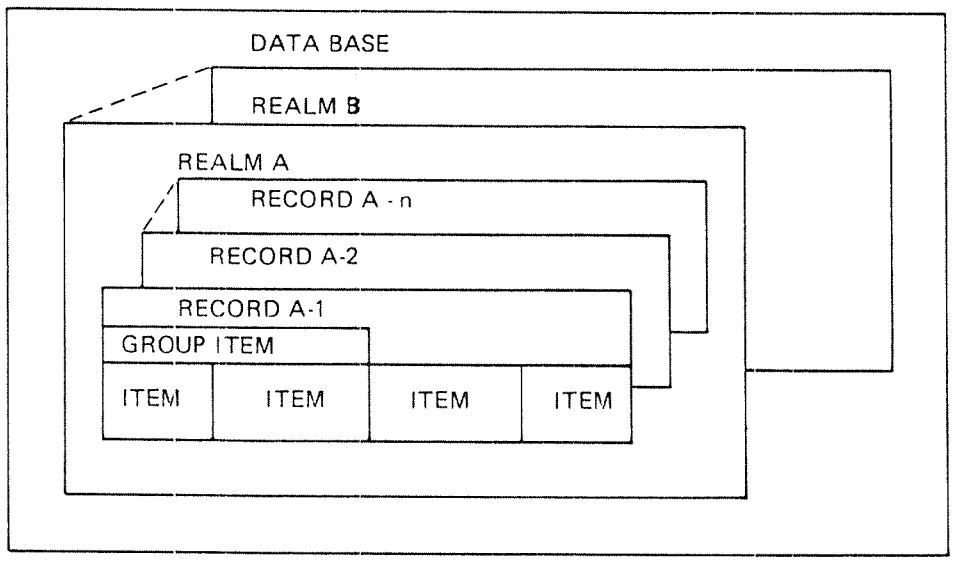


Figure 2.8.

An Example might look like this:

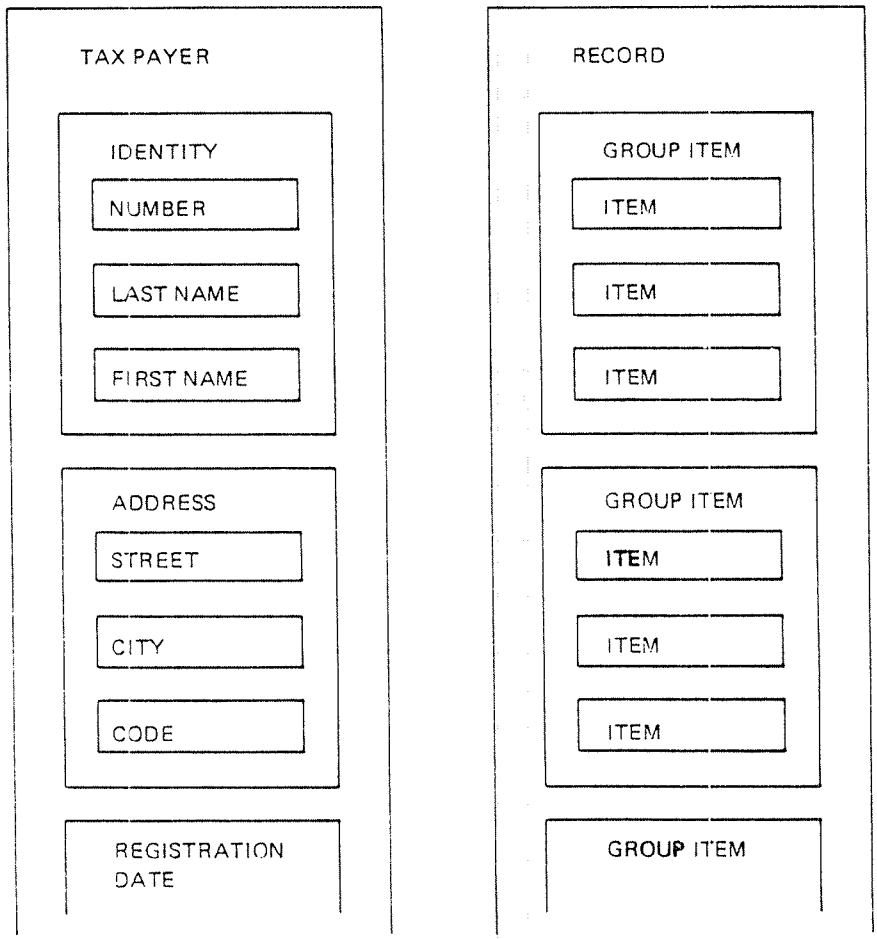


Figure 2.9.

From the figures, it will be seen that SIBAS, which follows the CODASYL terminology here, uses five different structure levels: ITEM, GROUP ITEM, RECORD TYPE, REALM, DATABASE.

As an example, consider data concerning an employee in a company:

EMPLOYEE
- NAME
- NUMBER
- BIRTH DATE
- SALARY
- JOB TITLE

The name EMPLOYEE is used to identify a record type in the data base. The database will normally contain several such record types. Furthermore, there will be several *occurrences* of each record type. If there are 1000 employees in the company, then there will be 1000 record occurrences of the record type EMPLOYEE in the database. A "record occurrence" can usually be referred to simply as a "record" with the full term "record occurrence" being used occasionally for the purpose of extra clarification. Experience with this class of DBMS has indicated that it is very important for the user to distinguish clearly between "record type" and "record occurrence".

Each record type contains a number of items. In the above example there are five items as listed. An occurrence of this record type would consist of one value for each of the five items. For instance, a record occurence might be as follows:

SMITH
74890
420531
43000
PROGRAMMER

The above concepts are fairly commonplace to any user versed in the practices of commercial data processing. It must be mentioned that in SIBAS all records of a given type are of the same length.

We will now give a fuller explanation of the terms we have used in the preceding examples.

2.2.1 Items

The item in SIBAS has the same role as the elementary item in COBOL or a variable in FORTRAN. An item declared in the schema DRL (definition/redefinition language) must be designated as either INTEGER, FLOATING or CHARACTER.

The following table indicates the correspondence between SIBAS item types and COBOL and FORTRAN item types.

SIBAS	COBOL	FORTRAN
INTEGER	COMPUTATIONAL	INTEGER
FLOATING	NOT AVAILABLE	FLOATING
CHARACTER	DISPLAY	INTEGER ARRAY

2.2.2 Group Items

It may be useful to assign a name to a collection of items in a record type. In this case, the collection is referred to as a group item. The items need not be contiguous items in the record. The sequence of the items in the group may also be different from the sequence in the record type. Only one level of naming is allowed. In other words, it is not possible to define a group item which includes another group item, and the constituents in a group item must all be elementary items. However, an item may participate in more than one group item. This could be used to implement multilevel groups by including all items from one or more group items in a new group item.

As a special case, a group could consist of only one item. This enables the user to define multiple names on items.

The group item provides a shorthand representation for identifying a collection of elementary items.

2.2.3 Record Types

Several items together are collectively referred to as a record type. Each SIBAS item is associated with a single record type in the database.

Each record type must be assigned a name which is different from other names in the schema. Furthermore, a location mode must be assigned to each record type. The location mode is essentially a mechanism which controls where the record is to be stored in the database.

SIBAS supports two location modes which are referred to as CALCULATION MODE and SERIAL MODE. In the first case, the user must designate either an item or a group item to serve as the primary record key to be used when calculating the location.

Records with serial location mode will be stored in the first available location in the realm.

CALC LOCATION MODE

For CALC records, a standard system supplied hashing or randomizing algorithm is used to distribute the record occurrences equally over a space on direct access storage (see Figure 2.10). The space assigned to a record type is called a realm. The data administrator must divide the realm into two areas called the main area and the overflow area. Each of these two areas is further subdivided into a number of buckets. This number must be a prime number.

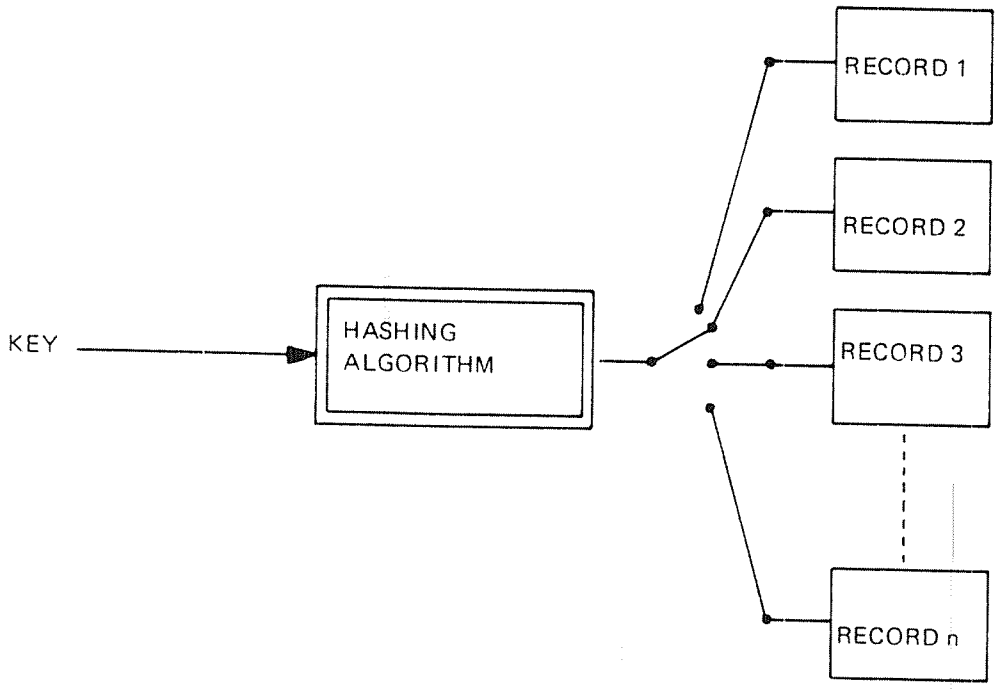


Figure 2.10: Calc Keys

Each occurrence of a CALC record type is then stored in a bucket in the main area or possibly in the overflow area. The bucket number in the main area is computed from the value of the key and the number of buckets as follows:

$$\frac{\text{Key Value}}{\text{Number of buckets}} = I + \text{Remainder}$$

where I is the integral part of the quotient. The remainder is directly used as the bucket number, and the record occurrence is stored in that bucket if there is space available. If not, then a bucket in the overflow area is used (refer to Figure 2.11).

Such overflow buckets are accessible from the main area bucket through a pointer. Records are stored in the first available location of the bucket. When the CALC key is used as a basis for finding the record, the same hashing algorithm is used and a sequential search is made through the main area bucket and if necessary also the relevant overflow bucket(s).

The data administrator must decide, when defining the CALC key item, whether or not duplicate values of the key are allowed. If not, then attempt to store a record which has a primary key value equivalent to that in a record of the same type already in the realm will be unsuccessful.

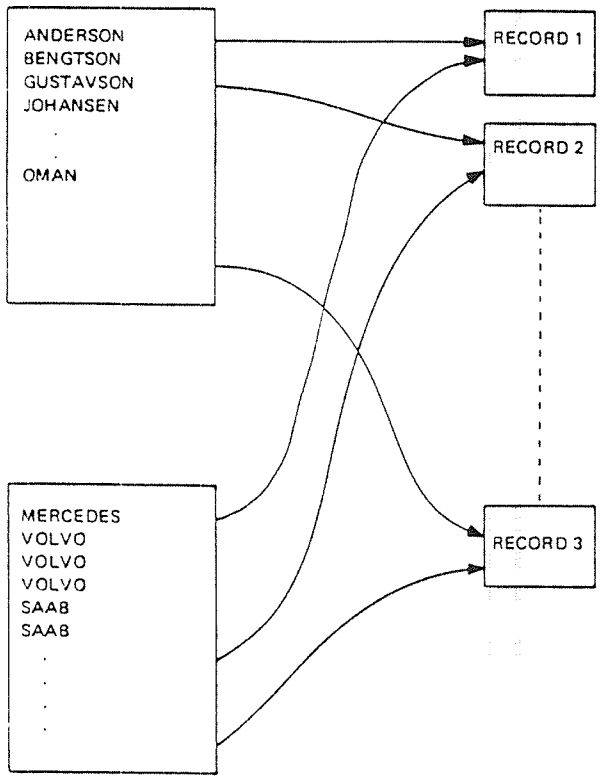


Figure 2.11: CALC Records

SERIAL LOCATION MODE:

Records for which no CALC key is designated will have location mode of SERIAL. Records of this type will be stored in the first available free location in the realm. If a record is deleted, then the next time a new record of the same type is stored in the realm, it automatically takes the space vacated by the deleted record.

2.2.4 Search Keys and Indexes

It is possible to assign one or more search keys (index keys) to a record type independent of whether its location mode is CALC or SERIAL. As in the case of CALC, a decision must be taken on whether more than one record with the same key value is allowed or not. Normally, at time of initial load, the user would be advised to ensure that records are in ascending value of a primary key value, especially if he wishes to make frequent sequential scans through these records using the primary index as the basis for his accesses.

In fact, in some cases where the record type has a location mode of SERIAL and there are search keys defined, it may be rather arbitrary which of the keys is regarded as the primary key and which are the secondary keys. In practice, if one index is more likely to be used than the others for serial processing of the records, then that index should be regarded as the primary key, and the records should preferably be loaded initially into the database in ascending order of the values of this key.

Indexes are maintained in ascending order of the key values, and the records in the realm may be processed in this sequence if required.

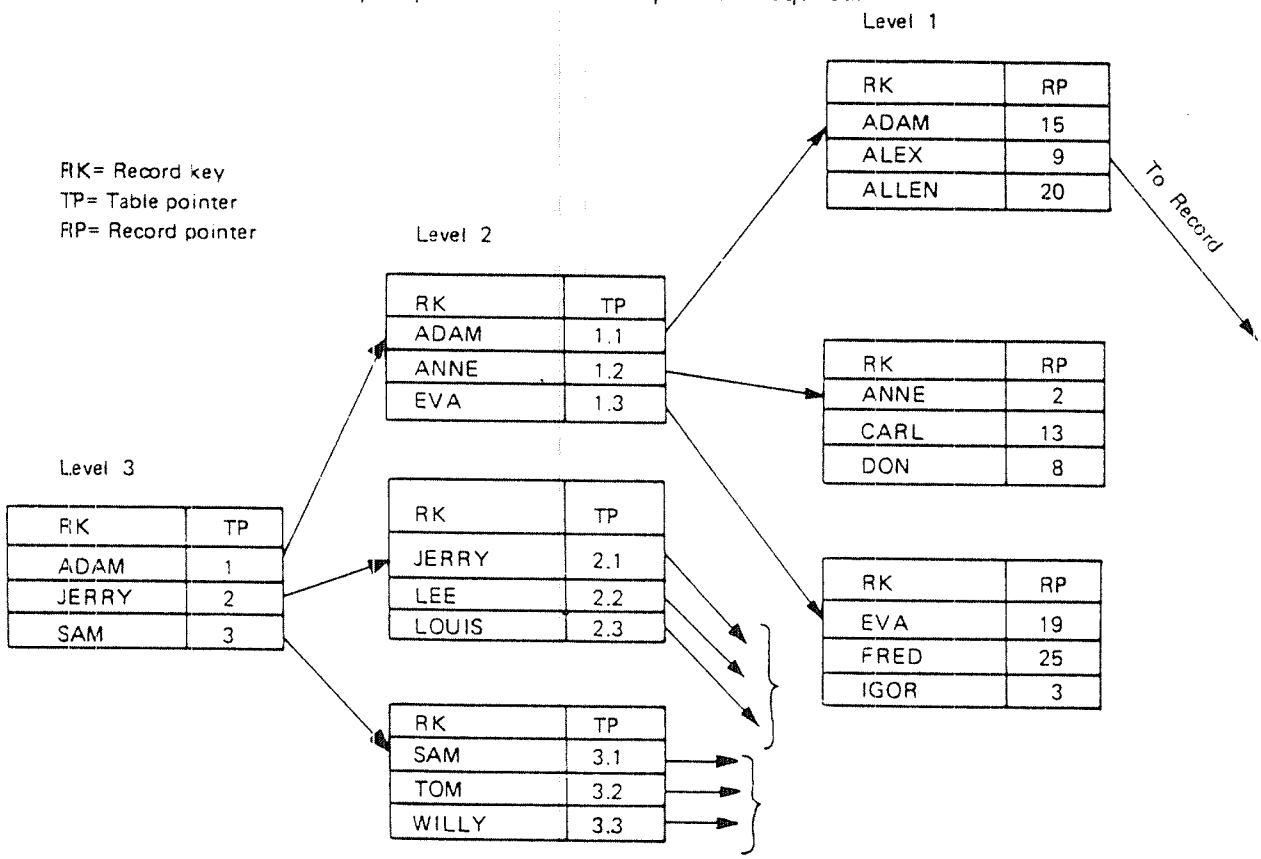


Figure 2.12: Index Keys

Since both CALC keys and search keys may be group items, it is possible for an elementary item to be used as part of several keys.

An index must be designated as either automatically maintained or manually maintained.

If the index is automatically maintained, then at the time a new record is stored in the realm, the index is automatically updated by the DBCS (database control system). If the index is manually maintained, then the programmer must include an extra statement in his program if and when he wishes to cause the index to be updated.

SIBAS COLLATING SEQUENCE:

There is no restriction on the composition of group items or single items which may serve as index keys. The values of index items are treated as bit strings and the index table is sorted in ascending order of the item values.

NULL VALUES OF KEYS:

Null values are represented by zeros or blanks depending on the item type, and any item not being a key is allowed to take that value.

Key items, however, must not take a completely null value. This applies to the calc keys, index keys, search keys, owner set items and member set items. Any of these may be a group item, in which case it may be partially null but not entirely null.

Any attempt to store a record which has a completely null value for a key or set item will be unsuccessful. Any attempt to modify an item in a record already in the database which would result in such a condition will also be unsuccessful.

INDEX TABLES — Representation of Indexes

When a record type has primary or secondary index keys, then for each key an index is built up during initial load and maintained, where necessary, during subsequent processing. Each index consists of a number of levels, and each level contains a number of index tables.

The index tables must be assigned to a system realm.

2.2.5 Realm

A realm is a storage space assigned to one record type. Often it corresponds to a SINTRAN file, but it is also possible to store more than one realm in a file. The realms are of two types, user realms containing data records and system realms containing index tables, etc.

In SIBAS, all occurrences of one record type must be assigned to one user realm. The user realm name will also be the name of the record type. As mentioned, system realms are used for storing levels of an index table when either a primary index or secondary indexes are defined.

The data administrator must estimate the number of record occurrences to be stored in each realm. Since records of the same type are of equal length, this facilitates an estimate of the maximum size of the realm.

In the case of indexed records, the data administrator must also estimate the space required for the index tables.

In the case of CALC records, it is necessary to regard the realm as being divided into a primary area and an overflow area. Each of these areas is further divided into equal size buckets. A bucket occupies one page.

2.2.6 Database

For completeness, the database is identified as the collection of all records, indexes, set types and realms which are defined in one single use of the schema DRL.

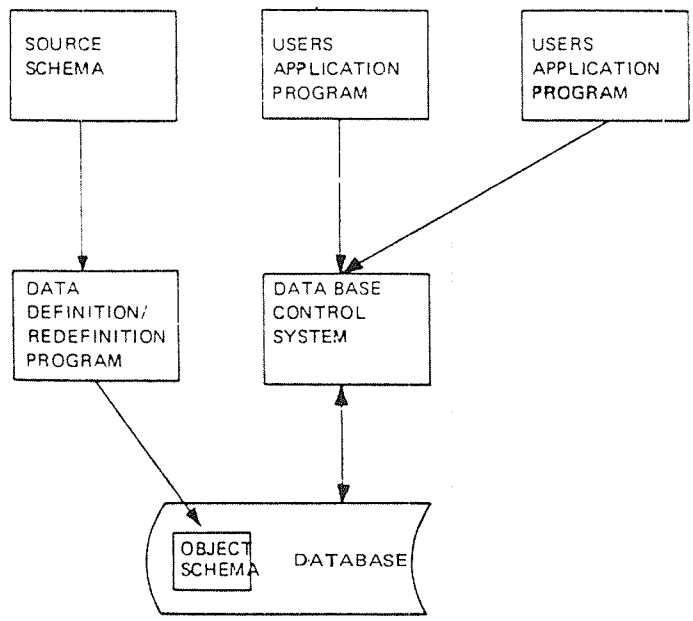


Figure 2.13: The Main Components of the Database

Each database has corresponding to it a source schema In addition, there exists an object schema which is the set of internal tables generated when a source schema is translated using the schema translator (see Figure 2.13).

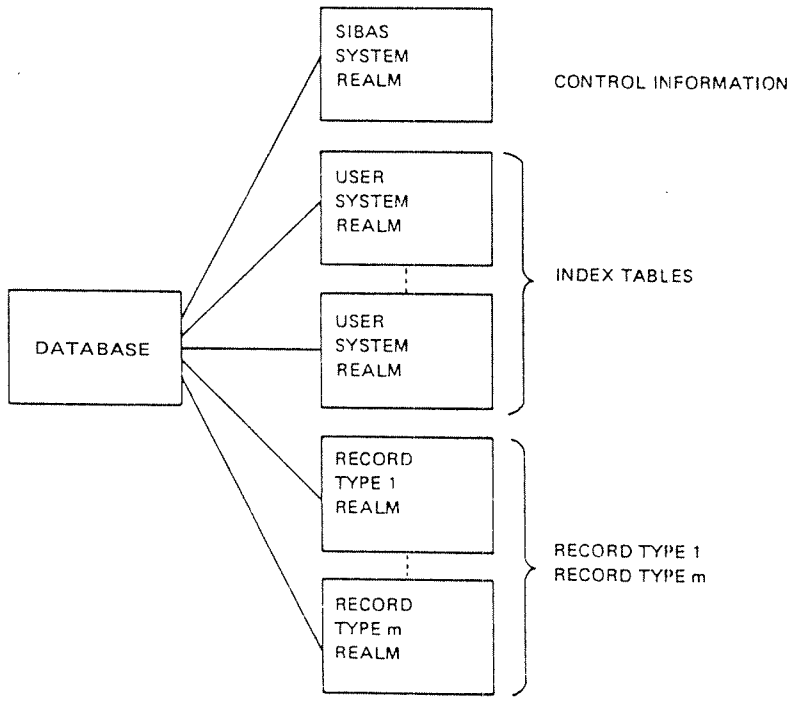


Figure 2.14: The Database Concept

A program is normally written to process the data in a single database. However, several users may access one database concurrently.

It must be emphasized that in SIBAS it is necessary for the program to declare its intention to process a database by executing an explicit OPEN statement on the database. In fact, this has the effect of opening a SIBAS system realm which contains among other things the object schema. Each realm in the database which the programmer wishes to process must also be opened, and this is done using a READY statement. A system realm containing an index table to a realm will be automatically opened when the realm is readied.

In a given installation on a given hardware configuration, there may be several databases, each known to the operating system through the name of its system realm.

2.3 DATA RELATIONS

2.3.1 Search Regions

Records stored in the same realm, i.e., records with a common type, can be grouped together in search regions. This means that all records in the realm having a specific common property constitute a defined class of records within that realm.

The following record classes may be handled as search regions:

- records having the same key value (duplicates allowed)
- records having key values within a specified range
- all records on the realm

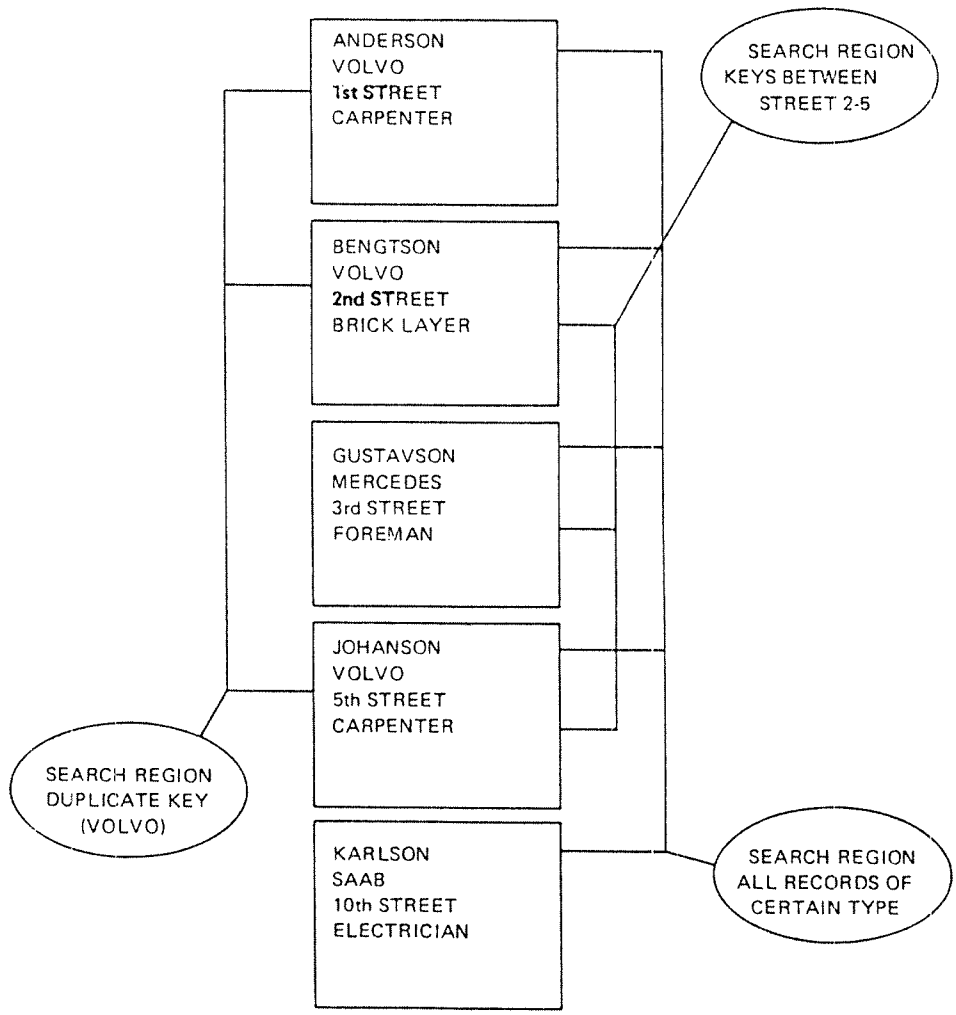


Figure 2.14B: The Search Region Concept

A search region is established as soon as a record belonging to the corresponding class is located in the database (see the description of the FIND statement). The program may then access the other records in the search region sequentially.

The search region identification is stored in a system variable called Current Search Region Indicator, which can be referenced, saved and restored by the program.

A search region is a "navigation" concept, used at run-time, and it is not necessary to declare it at the definition time. Another concept, that of a SET, must be declared at definition time.

2.3.2 Sets

A set is normally a relationship between two or more record types. In each set, one record type must be designated as the owner and each of the others is then a member. A single member set type is a set where the member records all are of the same record type, while a multi-member set type is a set where the member records are of more than one record type (see the following figures). There is also a third set type called an involuted set type which does not fall into either of these two classes and will be discussed separately.

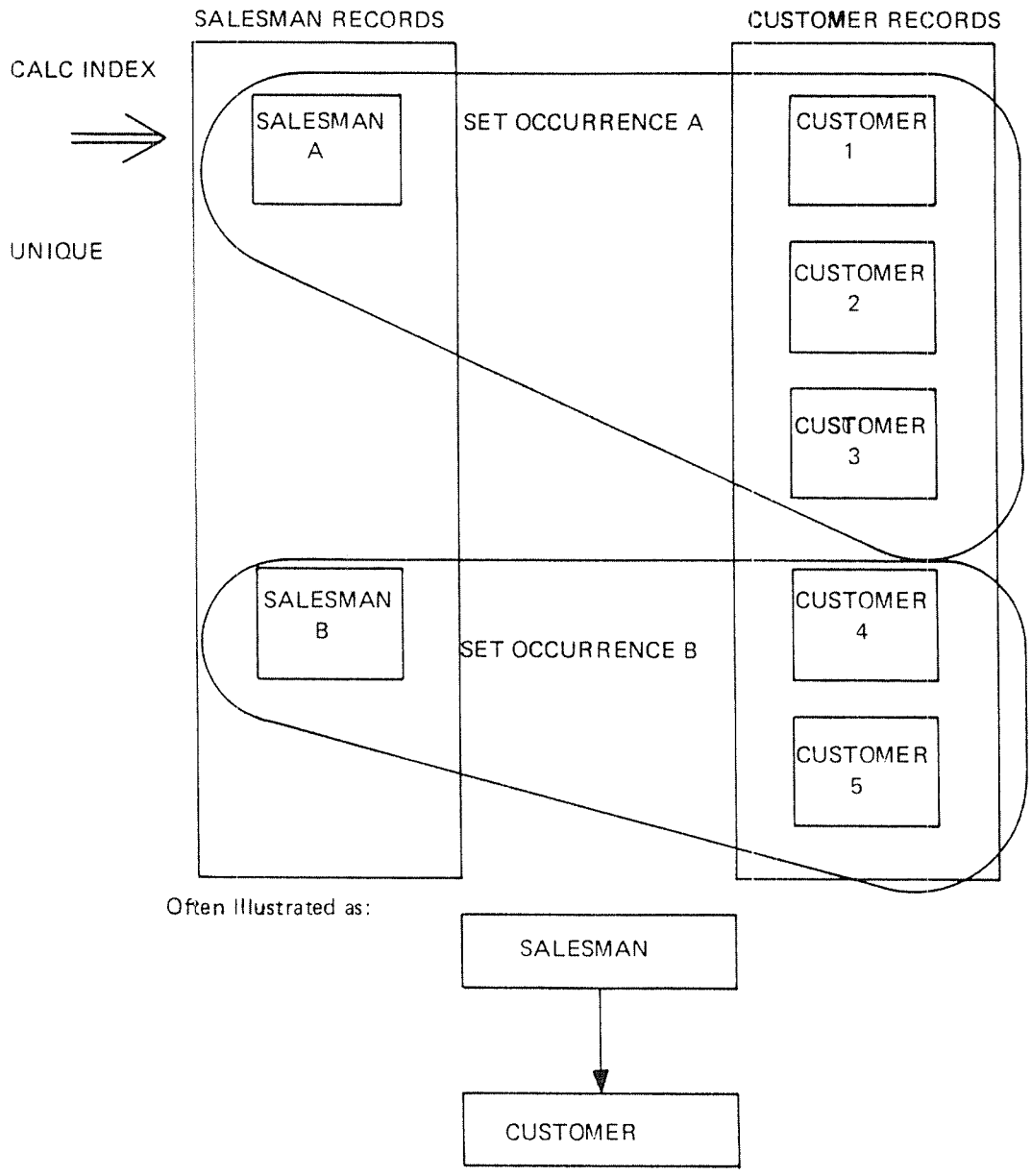
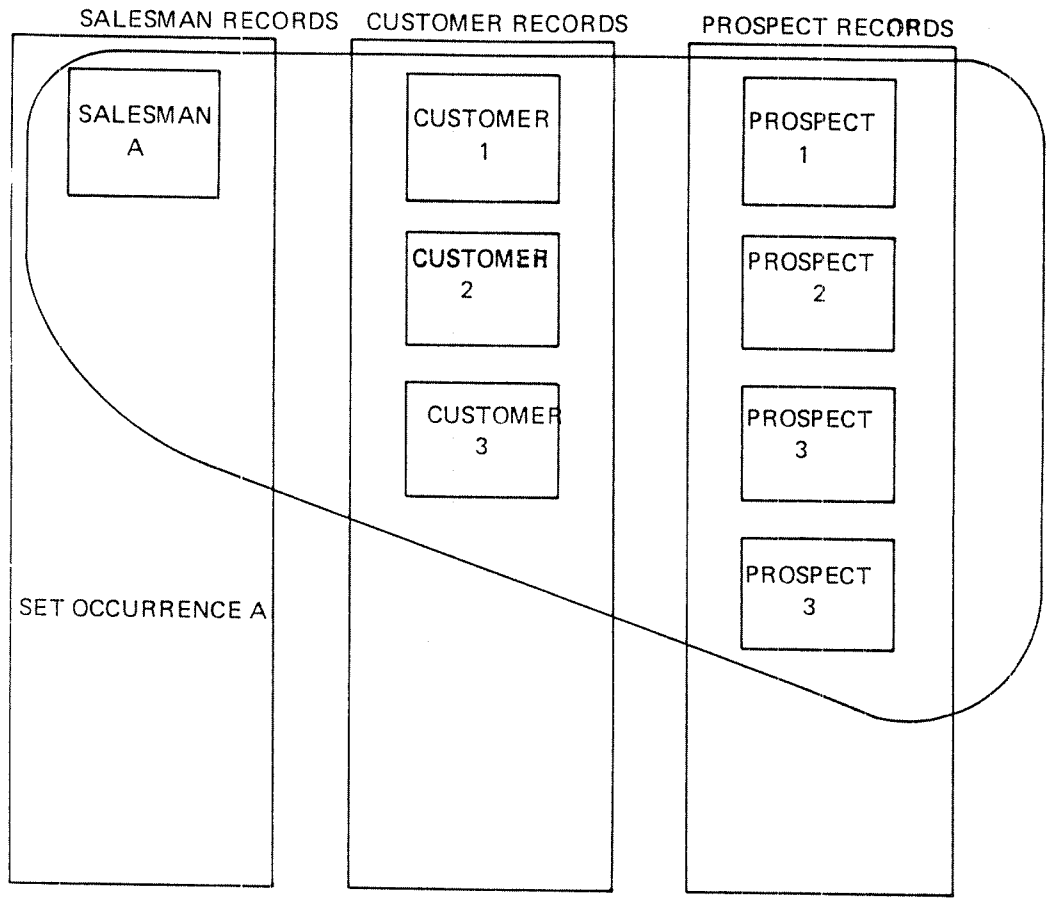


Figure 2.15: Single Member Set



Often illustrated as:

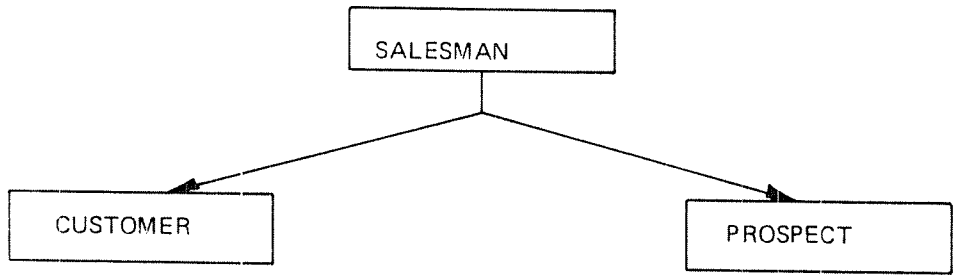


Figure 2.16: Multi-Member Set

2.3.2.1 SET ITEMS

When defining a single or multi-member set type in SIBAS, it is first necessary that a CALC or index key is defined for the owner record type. Furthermore, the key must be defined such that duplicate values of the key are *not* allowed.

To be able to define a single member set type, there must be an item (elementary or group) in both the owner record type and the member record type which "corresponds" in length and type, but *not* necessarily in name. In the case of group items, there should normally be correspondence in the constituent elementary item types, although it would be possible for an elementary character item in the owner to correspond to two or more elementary character items in the member. The item in the owner record type is referred to as the *owner set item*. The item in the member record is referred to as the *member set item*.

The owner set item must be defined as a CALC or index key for which duplicates are not allowed. The member set item may or may not be defined as CALC or index key. Duplicates will generally be allowed for member set items.

In the case of multi-member set types, there must be a member set item in each member record type which bears the relationship as described above to the owner set item. In addition, the member set item in each member must have the same name as in all the other members in the set type.

In all cases, the choice of an item to be an owner set item or a member set item imposes no restrictions on its use as primary key or search key.

2.3.2.2 SET OCCURRENCES

Each set type in the database will have a number of set occurrences (more simply referred to as sets). Each set contains one occurrence of the owner record type and zero or more occurrences of each member record type. Sets with no members are called *empty sets*.

For a given set type, there are in the database as many sets as there are occurrences of the owner record type.

It is the set item which determines how member occurrences belong to a set. If the value of the member set item for a set type has the same value as an owner set item, then the member record is "connected" to the owner's set. At what time this connection will be established depends on the "storage class" of the set type (see Section 2.3.2.4).

2.3.2.3 CHAIN REPRESENTATION OF SET TYPES

The physical representation of a set occurrence in the data base is achieved by a chaining technique. This means that the owner record in the set contains a pointer to the first member record in the set which in turn contains a pointer to the next record and so on. The last record in the set points back to the owner. The order of the member records in the set is generally determined by "time of arrival". A chain representation of this kind is essentially uni-directional. Problems can arise in long chains when a record is deleted as it is necessary for the DBCS (database control system) to circumnavigate the whole chain in order to modify the pointer in the record prior to the one deleted.

To avoid problems of time consuming deletes in long chains, it is possible and often advisable for the data administrator to designate a set type with double links, which means that each record in each occurrence of the set type contains both a "next" pointer as above and also a "prior" pointer in the opposite direction.

Defining a set type with double links does not add any extra processing capability, but it does have the effect that certain statements which depend on the set type relationship may be executed more rapidly.

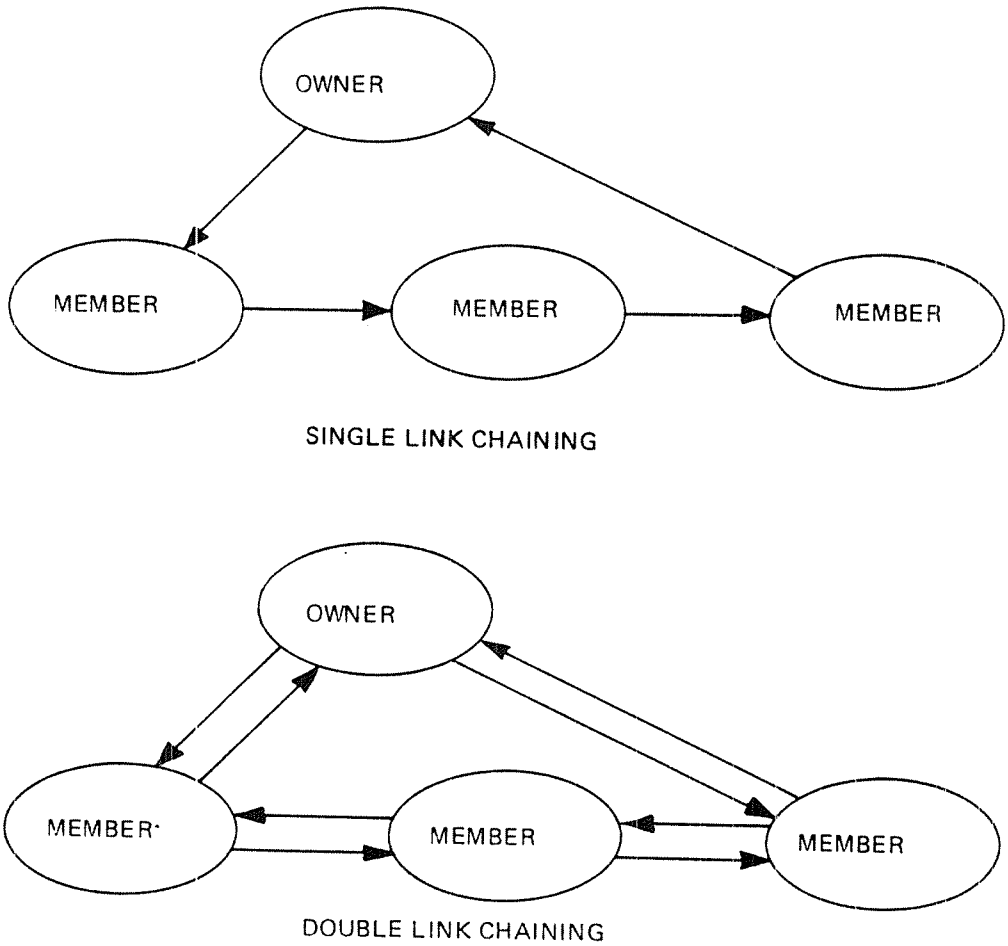


Figure 2.17: Chaining of Records

Illustration of SET TYPE and SET OCCURRENCE:

To clarify the concepts of set types and chains, Figure 2.18 illustrates a single member set type. Figure 2.19 illustrates two occurrences of this set type. Figure 2.20 illustrates how the same sets would appear if the set type in Figure 2.19 had been declared with double links. In these figures, the convention of using a rectangle to represent a record type and a circle to represent a record occurrence is followed.

In the example illustrated, the set item could be BRANCH ID which would then be found in both record types BRANCH and CUSTOMER. All occurrences of CUSTOMER having the same value of BRANCH ID would then be chained to the BRANCH record having the value for the item BRANCH ID.

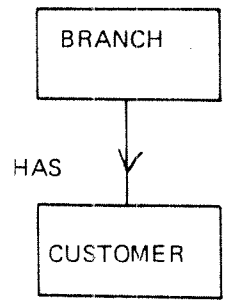


Figure 2.18: Logical Relationship

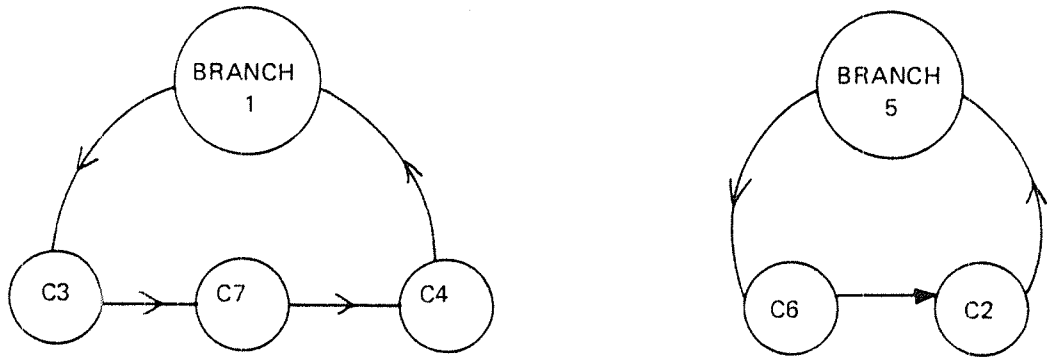


Figure 2.19: Occurrences of HAS with Link to Next Only

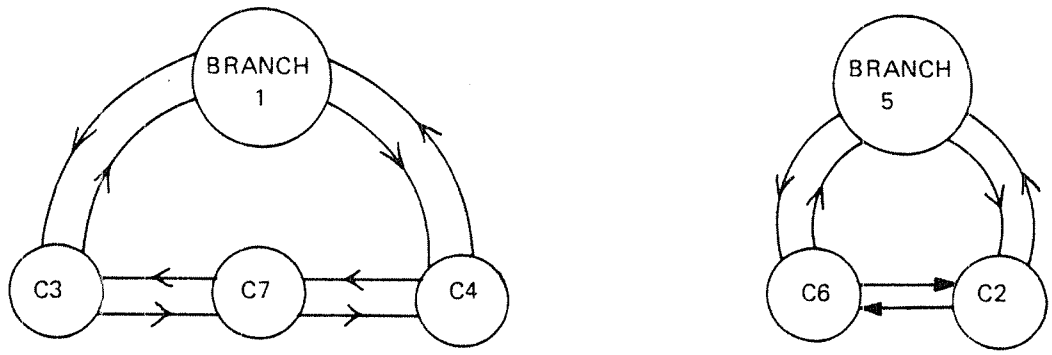


Figure 2.20: Occurrences of HAS with Link to Next and Prior

Involuted Set Types:

In SIBAS it is possible to have a special set type in which the owner record type and the member record type are the same. This special set type is referred to as an *involved set type* because the set relationship is involuted (or turns on itself).

An involuted set type may only be defined if the set item which designates ownership and the set item which designates membership are different in name and correspond in type and length. Both items are of course in the same record type.

This involuted set type (which is not supported in the CODASYL Database facility proposal) is useful for example in a Bill of Materials application. Graphically, an involuted set type is depicted as follows:

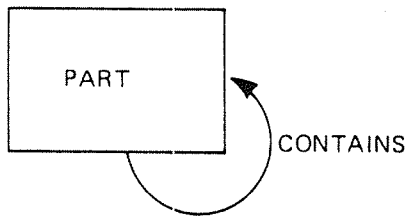


Figure 2.21.

In the example, the record type PART might contain two items, PART NO. and CONTAINED IN which should be defined with the same length and type. PART NO. will be the owner set item and CONTAINED-IN will be the member set item.

If a given assembly, X, contains three identical subassemblies Y, Z and Q then that part of the overall structure may be depicted as in Figure 2.23.

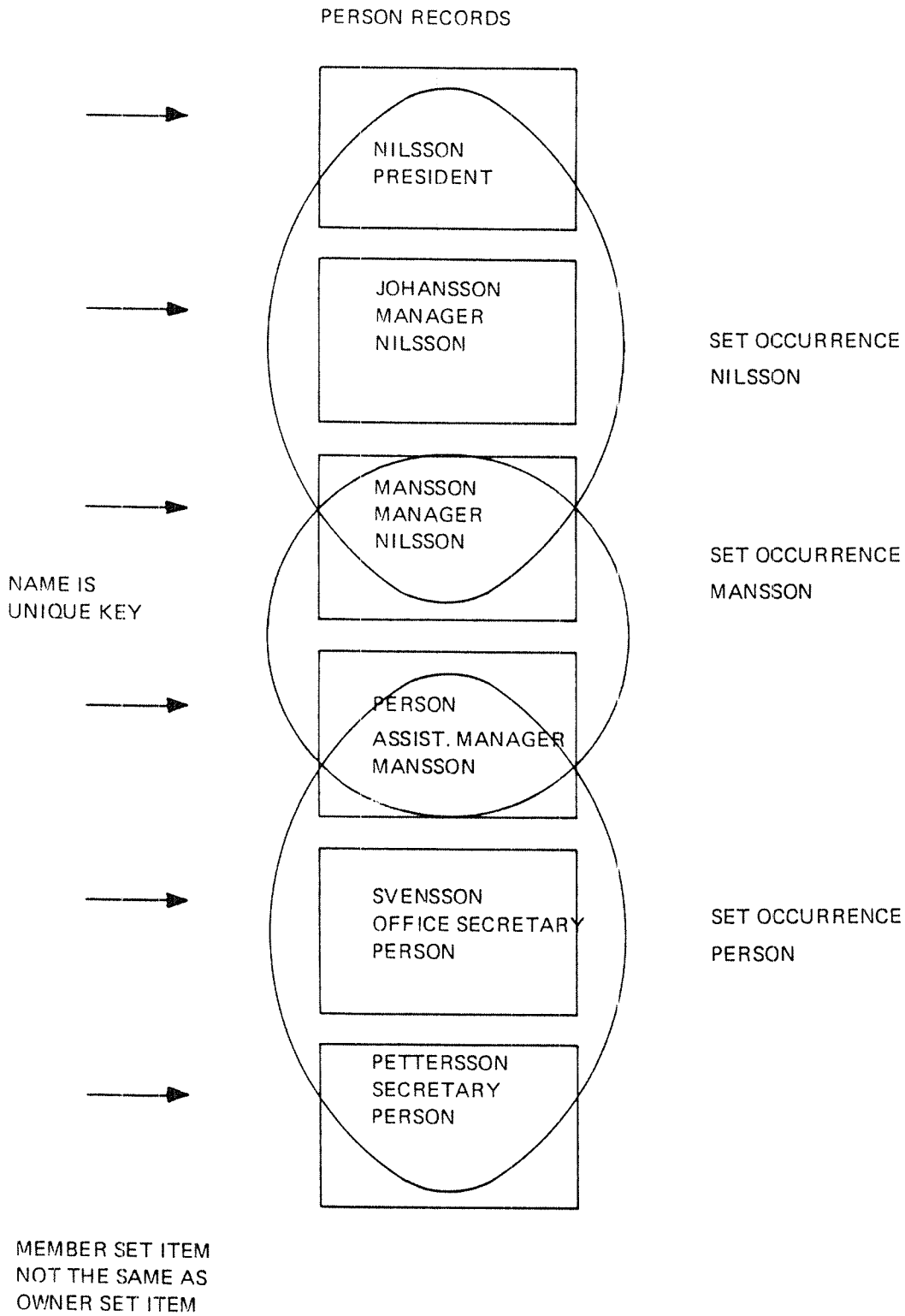


Figure 2.22: Involuted Set

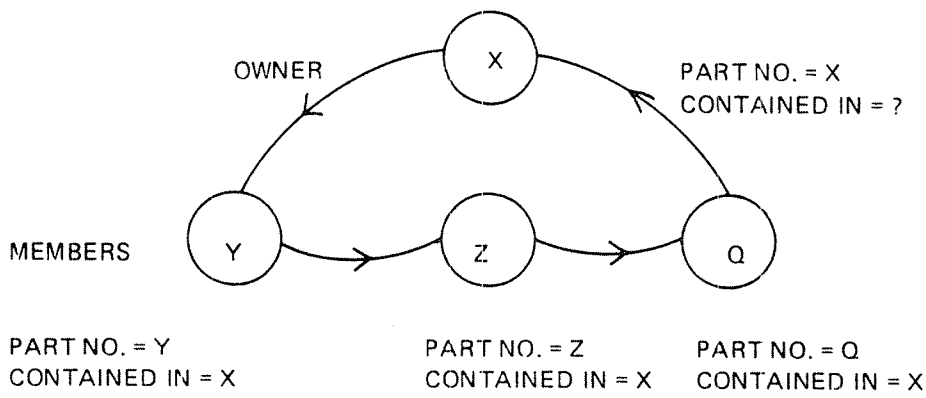


Figure 2.23: Involuted Set Type

In Figure 2.23, each of the four circles represents an occurrence of the record type PART. The owner set item (PART NO.) identifies each record occurrence uniquely. The member set item (CONTAINED ID) identifies the owner record of each set occurrence.

2.3.2.4 STORAGE CLASS

It was mentioned in Section 2.3.2.2 that the time an occurrence of a member record is connected to its associated owner occurrence depends on the storage class.

Storage class is a property of each set type. The storage class must be declared as either automatic or manual. If the storage class is automatic, then a member occurrence is automatically connected into the appropriate set occurrence at the time the record is stored in the database, using a DML STORE statement.

If the storage class is manual, then the connection is not made when the STORE is executed, but the programmer may cause the connection to be made by using a CONNECT statement. Irrespective of storage class, a record may not be connected into any occurrence of a set type into which it is already connected; furthermore, it may be connected into no more than one occurrence of any given set type.

In SIBAS, storage class is a property of a set type. This applies to a single member set type, a multi-member set type and an involuted set type. A record type may of course be defined as member of several automatic set types and, at the same time, of several manual set types.

Storage class is regarded as being of sufficient importance in the structure of a database to merit a special graphic formalism to be used when depicting the structure of the database graphically. A continuous line is used to illustrate an automatic set type relationship and a dotted line to represent a manual set type relationship. The various possibilities are indicated in Figure 2.24.

It must be noted that, in SIBAS, the storage class also has an effect on whether or not it is permissible to disconnect a record from a set. If the storage class is automatic, then this is not permitted, although the record would be moved from one set to another if the value of the member set item changes. If the storage class is manual, a record may be disconnected from a set using a DISCONNECT statement.

Finally, it should be noted that it is possible to order the members of a set type which is manually maintained. This is done by using the CONNECT BEFORE or CONNECT AFTER statement which will link the record into the set occurrence before or after an already existing record in the set occurrence.

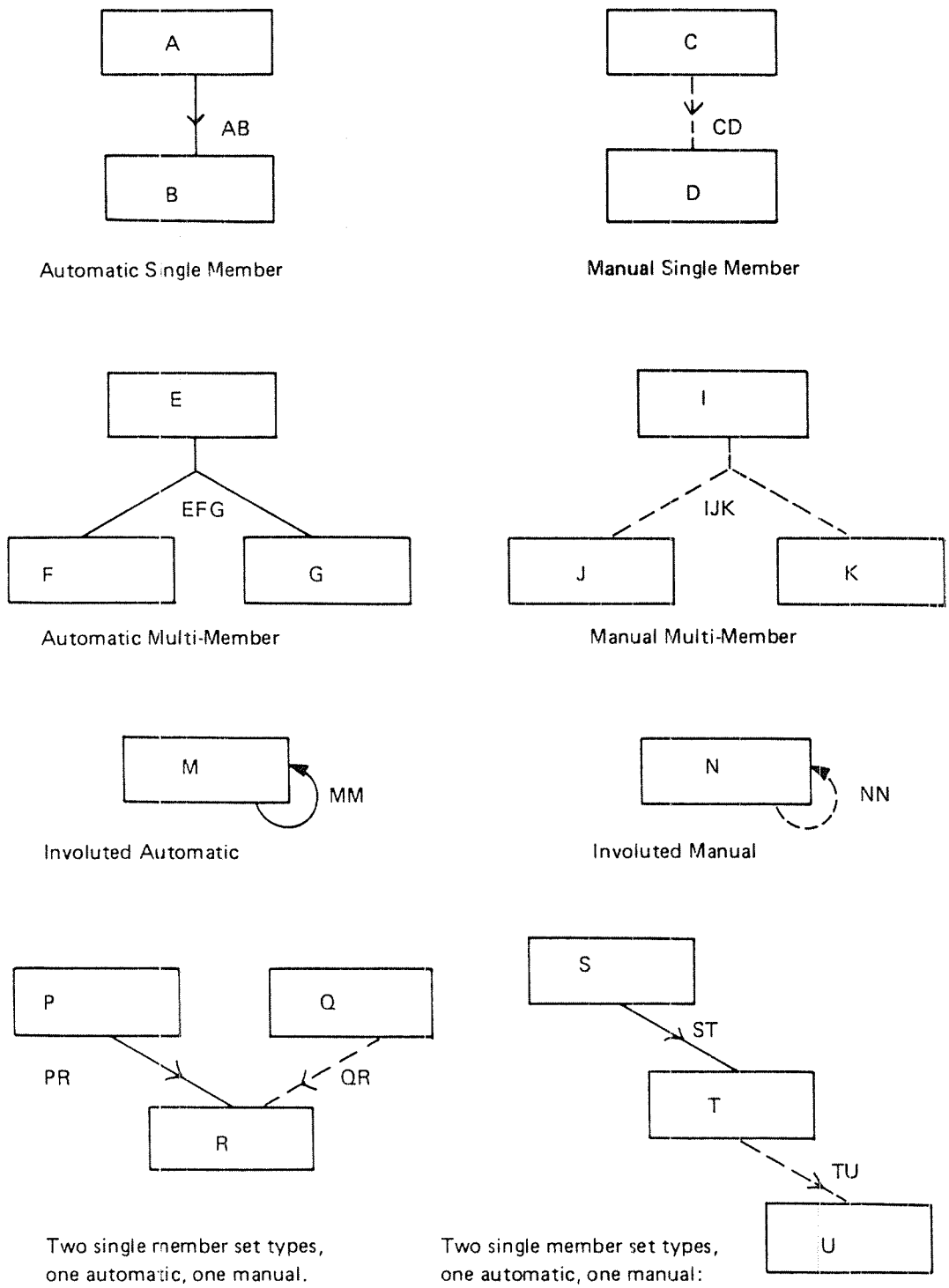


Figure 2.24: Examples of Possible Set Types

Note on Set Occurrences:

As in the CODASYL proposal there is one important property to note about the way in which a member record can be connected to a set. If the record type is a member of a given set type, then an occurrence of the record type may be connected into no more than one occurrence of that set type. That is, a member may only have one owner in one set occurrence. The record type could, however, be defined as a member of other set types (Figure 2.24).

Removal Class:

In SIBAS the removal class will depend on the option given in the ERASE statement. This is discussed in more detail under the definition of this statement.

2.4 DATA MANIPULATION

The CODASYL Database Facility approach to processing a data base calls for the programmer to be able to enter the database from outside and to navigate his way around inside. The SIBAS approach to search keys makes it possible to access all records from outside in several ways and also to conduct searches in certain regions within the database, relative to a previously found record. The fact that several users access the database concurrently necessitates some control mechanism. This is discussed in more detail in this chapter.

2.4.1 Access Principles

2.4.1.1 GENERAL

With a SIBAS database, it is possible for a program to make two kinds of accesses to the database. The first class is called an "out of the blue" access. The programmer provides the value of a key, and a single record is found in the data base whose key value corresponds to the key value specified.

The other class of access is called a relative access, and the record found always has some relationship to one found previously — normally the record most recently found.

It must be emphasized that, since the database is in direct access storage, both classes of access are essentially "direct" in the normally accepted meaning of the term. The first access to a database which is made in any program must necessarily be an "out of the blue" one. However, a program will normally contain a mix of statements from both classes.

The statement which is used to locate (that is, confirm the presence of) a record in the database is the FIND statement. Numerous options of FIND are available and may be listed as follows:

1. FIND based on calc key or index key (this could define a search region).
2. FIND first or last member record in a set occurrence.
3. FIND next or prior member record in a set relative to a record recently found.
4. FIND first record in a realm (which defines a search region).
5. FIND next record in a search region.
6. FIND owner occurrence relative to a member occurrence recently found.

The execution of a FIND statement may be successful or unsuccessful. If successful, a record is located, and an indicator is set to point to that record, called the CURRENT OF RUN-UNIT indicator. This means that further DML or host language type actions can be performed on that record. However, no host language statement such as the COBOL MOVE or a FORTRAN ASSIGN may be meaningsfully executed on the data in the record until a successful GET statement has been executed.

A FIND may be unsuccessful. In the case of an out of the blue access, for example, this may mean that there is no record of the type sought in the data base whose key values correspond to those specified in the FIND statement. The relative classes of FIND may be unsuccessful for a variety of reasons which are defined in detail in another chapter.

If the FIND, or any other statement, is unsuccessful, then a Database Exception Condition (DBEC — see Section 7.3) is set. It is the responsibility of the programmer to be fully aware of the database exception conditions which may occur in the course of execution of his program and to build in appropriate tests and courses of action in each case.

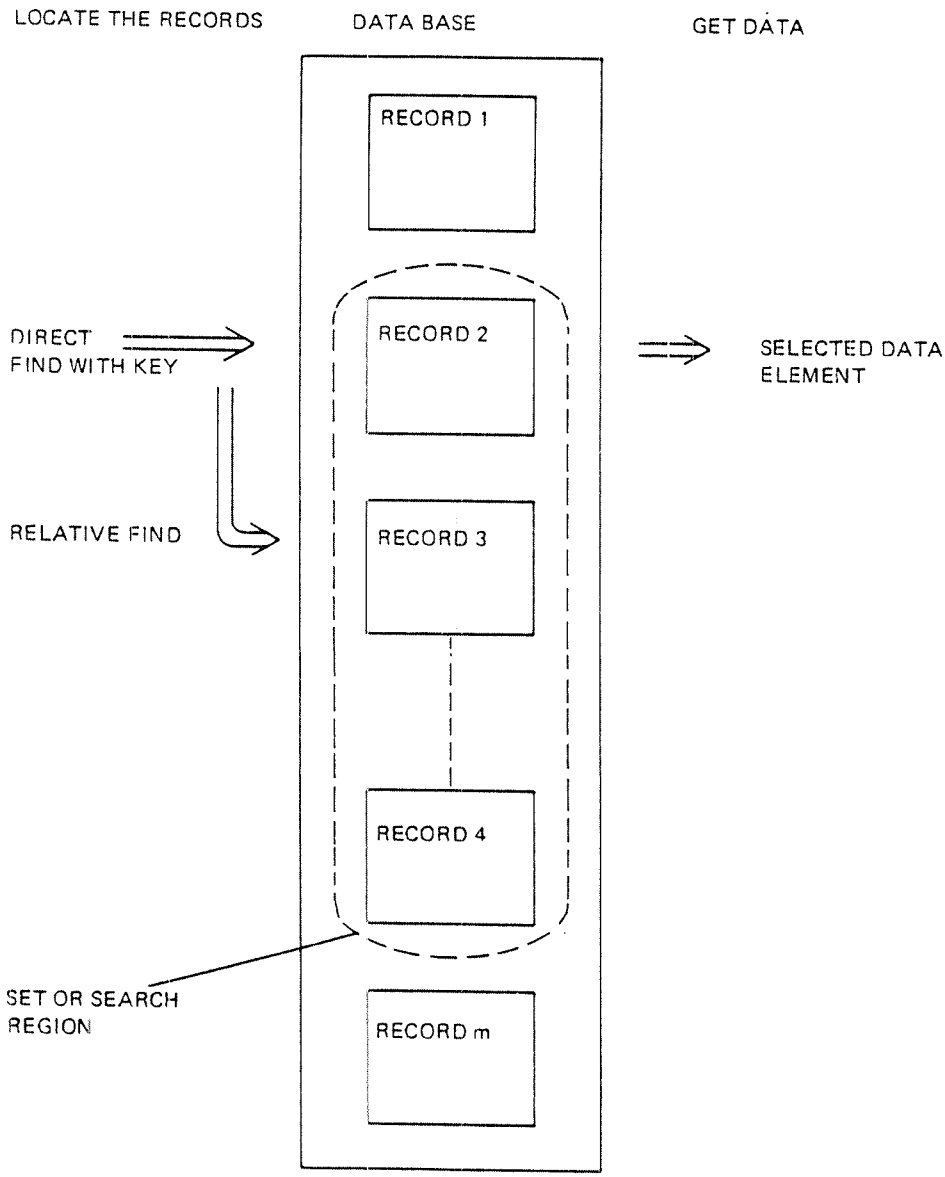


Figure 2.25: Access Ways

2.4.1.2 CURRENCY INDICATORS

Different programs accessing a SIBAS database may execute concurrently. It is also possible that the same program may be executing two or more times concurrently with different parameter values. For convenience, each executing instance of a program is referred to as a *run-unit.*

As already indicated, a run-unit in the course of its execution may need to find a record relative to some recently found record that is found in the same run-unit. The way in which both the run-unit and the DBCS keep track of where in the database processing has reached is by means of two *currency indicators.* In SIBAS, the two indicators are referred to as:

CURRENT OF RUN-UNIT INDICATOR (CRUI)
CURRENT SEARCH REGION INDICATOR (CSRI)

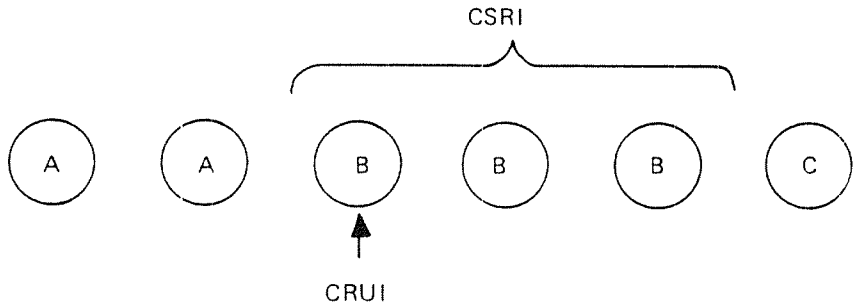


Figure 2.26: Illustration of CSRI and CRUI

Current of Run-unit Indicator (CRUI)

The CRUI is always updated after the successful execution of each FIND or STORE statement. The content of this currency indicator is always a unique identification of a *record* in the database.

This record identification is a quantity which distinguishes one record occurrence in the database from all others. It is not based on the data values in the record but rather on the physical address of the record in the data base. The physical address of a record may of course change during the life of a run-unit, but the CRUI will then be updated accordingly.

The CURRENT OF RUN-UNIT INDICATOR is maintained by the exeuction of the FIND and STORE statements. Several other DML statements actually operate on the record designated by the CRUI, but only *successful* execution of FIND or STORE will update CRUI.

Temporary-Database-Key

It is possible for a program to "remember" a CRUI in a temporary-database-key. The CRUI could then be referred to directly from the same run-unit by use of the temporary-database-key, even if another record is current. If the user remembers more than one CRUI, the system will build up a *remembered list* where the temporary-database-keys are used to identify the entries in the list. Each time a REMEMBER statement is executed a new entry is added to the list and the entries are removed from the list by executing the FORGET statement.

Any statement which operates on a record identified by the CRUI can equally well operate on a record which is identified by a temporary database key. For example, it is possible to MODIFY a record identified by a temporary-database-key without making it CURRENT OF RUN-UNIT first.

If a record which is identified by a temporary-database-key is moved physically in the realm, the address in the temporary-database-key, and all other entries in the currency and temporary-database-key lists for all concurrent run-units referring to this unique record will be updated accordingly.

Note that a temporary-database-key may only be used during the "life of a run-unit".

Current Search Region Indicator (CSRI)

An "out of the blue" access to the database may have the effect of setting the CSRI to a new search region. A search region can be defined as a collection of records which have something in common. It can be any of the following:

1. All records with same value of CALC KEY (duplicates allowed).
2. All records with same value of an INDEX KEY (duplicates allowed).
3. All records in a realm (i.e., of same type).
4. All records whose index key values are between defined limits.

The setting of the CSRI depends partly on the form of the FIND statement and partly on the key specified in the FIND. The setting of the CSRI to the four types of search regions given above is done in the following way:

1. FIND using a CALC key for which duplicate values are allowed.
2. FIND using an INDEX key for which duplicate values are allowed.
3. FIND first in realm using the name of the realm.
4. FIND between limits giving the upper and lower limit of an index key item.

These four forms of the FIND statement are the *only* possible ways of changing the value of the CSRI.

As with the CRUI, it is possible to "remember" the contents of the CSRI in a *temporary search region indicator.* The system builds up a remembered list for temporary search region indicators in the same way as for temporary database keys.

Also, either the CSRI or a remembered temporary search region indicator may be used in accesses to the database which are in the class: "relative to some previously found record".

The Use of CRUI and CSRI

At the beginning of the execution of any run-unit, both the CRUI and the CSRI are regarded as undefined. Hence, the first FIND statement to be executed must be one which does not use these indicators, but which does in fact set them.

When a FIND NEXT in search region relative to some previously found record is executed and if CSRI is used to identify the search region, the search region will be the one defined in the latest executed FIND of one of the different forms, i.e., the current search region.

Furthermore, it should be noted that if the current record has been ERASED, CRUI will be undefined. If the current record has been MODIFIED, CRUI will still be defined, but the record it is identifying may have been moved out of the current search region. This situation will be illustrated by an example.

In the example above, a FIND using a key (INDEX or CALC) for which duplicates are allowed has been executed. The current search region will be defined as all records with the same value (B) of the key, and the current record will be the first of these records. If a FIND NEXT in search region using CSRI and CRUI is executed, the next record with value B on the key will be found and made the current record, and CSRI will remain unchanged. If the key is then MODIFIED in this record, the record will be moved out of the current search region, but it will remain the current record.

A FIND NEXT using CSRI and CRUI in this situation will have no meaning. If the user wants to FIND the third record with value B on the key, he should execute REMEMBER for the first record using a temporary-database-key, and then perform a FIND relative to this record. It should be noted that this situation only occurs if the key used to *define* the search region has been MODIFIED.

2.4.2 Connecting and Disconnecting, Inserting and Removing

2.4.2.1 CONNECTING AND DISCONNECTING

Connecting and disconnecting records to sets is normally done automatically by SIBAS through execution of STORE, MODIFY or ERASE statements.

Manually, however, it is possible under certain circumstances to connect a record into a set and disconnect it from a set. In SIBAS, it is possible to use similiar facilities to update an index. Each is described separately.

Connecting To and Disconnecting from a Manually Maintained Set

If a record type participates in a set type as a member, then its occurrences may (at any time during the life of the database) be either connected or not connected into a set of that set type. When the connection actually takes place depends on the storage class of the set type.

If storage class is automatic, it means that the record will be connected at the time the STORE is executed. This means that there must be an occurrence of the owner record type in the database whose owner set item values correspond to the member set item values in the record being stored. If this is not the case, then the record cannot be stored, and hence not connected. However, if the attempt to store the record does not include an attempt to store the member set item (it may be a group item), then the store may be successful, if all other restrictions are satisfied, but the connection into the set item is not made. The member set item value will then be undefined. A subsequent modification of such a record which provides a value or values for the complete member set item would cause the connection to be made. Considerable care is called for in a multi-user environment when allowing this situation to occur.

If the storage class of the set type is manual, then no connection is made when the record is stored. However, the CONNECT statement may be used to connect a record into the set of the set type in which it is a member. Again there must be an owner in the database with an equal valued set item for the connection to be successful. Exactly where in the set the record is connected depends on the option used. It is possible to connect it at the end of the set (i.e., last in order of the link to next) or else adjacent to some previously found record in the set. In this case, it can be connected before or after the previously found record. If the storage class is manual, then it is also possible to DISCONNECT a record from a set into which it previously had been connected.

The various alternative actions which can take place when a STORE, CONNECT or DISCONNECT is executed are summarized in the following table. The storage class is taken into account, as is also, for each storage class, the value of the member set item (MSI) with respect to owner set item values (OSI) already in the database.

It must be noted that the STORE statement operates on a record occurrence built up in a record area in core by the programmer. The programmer must designate which of the items in the record type he intends to provide values for. The CONNECT and DISCONNECT act on a record which is already stored in the database, and it is the value of the member set item there which may influence the success or failure of the statement.

A DISCONNECT or a CONNECT or both may take place implicity during the course of execution of a MODIFY if the member set item values are changed. What exactly happens depends also on the storage class of the set type and also on whether or not the member record was already connected into some set. The complete picture is summarized in the following table, which examines 12 situations depending on storage class of the set type, whether the member record was previously connected or not and the relationship of the new member set item values to owner set item values already in the data base. In the cases where the member was in fact connected, it is only the set item values of other owners which are of interest.

Storage Class	Situation	STORE	CONNECT	DISCONNECT
Automatic	MSI = OSI (some)	Y (connect)	Not applicable	Not allowed with automatic
	MSI ≠ OSI (some)	N		
	MSI not completely given in record area	Y (no connect)		
	MSI null in member record in database	N null member set item value not allowed		
Manual	MSI = OSI (some)	Always successful. MSI not examined.	Y	Y
	MSI ≠ OSI (some)		N	Not applicable
	MSI not completely given in record area		Not applicable	Not applicable
	MSI null in member record in database		N	Not applicable

Explanation:

MSI means member set item value
OSI means owner set item value
Y means execution should be successful if no other conditions prevent it
N means execution will not be successful

Table 2.1: Using the STORE, CONNECT, DISCONNECT Commands

Storage Class	Previous State	Situation	Modify member set item values: DISCONNECT from old	CONNECT to new	Net result of MODIFY
Automatic	Connected	new MSI = NULL	Possible but not done	Not possible	Fail
		new MSI ≠ OSI (other than previous owner)	Possible but not done	Not possible	Fail
		new MSI = OSI (other than previous owner)	Y	Y	Success
	Not connected	new MSI = NULL	Not applicable	Not possible	Fail
		new MSI ≠ OSI (any in database)	Not applicable	N	Fail
		new MSI = OSI (any in database)	Not applicable	Y	Success
Manual	Connected	new MSI = NULL	Y	N	Success including DISCONNECT
		new MSI ≠ OSI (other than previous owner)	Y	N	
		new MIS = OSI (other than previous owner)	Y	N	
	Not connected	new MSI = NULL	Not applicable	N	Success MSI not examined
		new MSI ≠ OSI (any other)	Not applicable	N	
		new MSI = OSI (some)	Not applicable	N	

Explanation:

MSI means member set item values
OSI means owner set item values
Y means action performed unless MODIFY fails for other reason
N means action not performed

Table 2.2: Using the MODIFY Command

2.4.2.2 INSERTING INTO AND REMOVING FROM AN INDEX

If there are one or more index keys (search keys) defined for a record type, then the data administrator must decide when defining the schema whether the indexes are automatically maintained or manually maintained. For completeness and consistency it must be emphasized that when a record type has a location mode of CALC, the calc access mechanism is of necessity "automatically maintained", but the data administrator must not define this for CALC key items.

Returning to indexes, the concept of an automatically maintained index is almost completely analogous to an automatic set type. The "insertion" is normally made when the STORE is executed, but it depends on the value of the index key item or search key item. It also depends on whether the key item is named in the list of items to be stored. If, because of the omission of these items from the list, the index is not automatically updated at time of STORE, it will be automatically updated if the index key item in this record occurrence is given a value later (using MODIFY).

A manually maintained index is also analogous to a manual set type. It is possible to insert and subsequently to remove a record from an index by using the INSERT or REMOVE statements. The value of the key item is important in a similar way to the importance of the member set item of the manual set type.

In the case of both automatically and manually maintained indexes, the data administrator must decide whether or not to allow duplicate values of the key item in the index. If duplicates are allowed, there is never any problem about inserting a record with non-null key values into an index. If duplicates are not allowed, then whether an INSERT or, in the case of automatically maintained index, a STORE, is successful or not depends on the absence or presence of an entry in the index with the same key value as the new record.

2.4.3 Concurrent Processing

In SIBAS considerable attention has been given to concurrency problems. The philosophy has been to avoid deadlocks and associated costly logic at the expense of some few restrictions. There are four levels of protection between concurrently executing run-units:

1. Database reservation
2. Realm protection mode
3. Record lock
4. Notification of change

2.4.3.1 DATABASE RESERVATION

A run-unit may reserve/release SIBAS, preventing any other run-unit from accessing SIBAS during the duration of the sequence enclosed by reserve and release. This is a very effective method of preventing interferences between concurrent run-units, but it has its drawbacks. The sequences *must* be short and cannot contain terminal input/output.

The ACCUMULATE calls are examples of this method. The possibility given to the user of writing so-called "MACRO"s which are executed uninterrupted is another example.

2.4.3.2 REALM USAGE MODES AND REALM PROTECTION MODES

At the time a run-unit executes a READY statement, the programmer is required to declare the way in which he intends to use the realm and at the same time how he wishes his run-unit to co-exist with other run-units using the realm. These two factors are called the usage mode and the protection mode respectively.

SIBAS supports three realm usage modes as follows:

RETRIEVAL (FIND, GET)
LOAD (STORE, CONNECT, FIND, GET)
UPDATE (ALL)

and two realm protection modes:

NON-PROTECTED (other run-units may update the realm concurrently)
EXCLUSIVE UPDATE (no other run-units may perform update or connect in realm, but may retrieve records in the realm)

When a run-unit readies a realm in usage mode RETRIEVAL, the realm will be available to the run-unit for execution of FIND and GET statements only. Usage mode LOAD allows the user to perform STORE and CONNECT in addition to FIND and GET. Usage mode UPDATE includes use of all SIBAS statements on records in the realm.

When protection mode EXCLUSIVE UPDATE is given for a realm, concurrent run-units will be restricted to perform FIND and GET statements on the realm (i.e., retrieval only).

When a realm is readied for "NON-PROTECTED" use, concurrent run-units may update, load and retrieve in the realm.

2.4.3.3 RECORD LEVEL LOCK OUT

In the case when a realm is readied for "NON-PROTECTED" use, it is possible for the programmer to lock individual records. This is necessary if the programmer will ensure that a record or a group of records are not updated while he is using them. (Protection mode of EXCLUSIVE UPDATE avoids this problem by locking out the whole realm for other run-units which intend to update it.)

The record level lock out is imposed using a LOCK statement. The LOCK statement can be used to lock a single specified record or a group of specified records. In the latter case the LOCK statement will only be successfully executed provided that *all* the desired records are simultaneously available. The criterion for a record to be available is that it is not concurrently locked by any other run-unit. This restriction is necessary to prevent deadlock situations.

When a run-unit has successfully executed a LOCK statement, all the locked records must be released by performing an UNLOCK statement before another LOCK statement can be executed. This restriction is necessary if deadlock is to be avoided.

2.4.3.4 NOTIFICATION OF CHANGE

The record level lock out enables a programmer to ensure that a record or a group of records are protected against concurrent run-units. But a programmer might find it too restrictive to lock records, or records might be modified, erased, etc. during the time it takes to locate all the records the programmer intends to lock simultaneously in a LOCK statement. To solve this problem, the current record of a run-unit and all the records a run-unit has remembered (i.e., all records on the remembered list), are always in what is called extended monitor mode. If a record has been modified, erased, connected or disconnected by another run-unit while it is in extended monitor mode, a warning will be issued to the run-units which have the record on their remember list. The warning will have the form of a DBEC (Database Exception Condition), which will have a specific value depending on what other run-units have done to the record, and what the present run-unit is trying to do. The programmer will then have to take action according to the DBEC. The DBEC could be:

1. Record has been connected or disconnected
2. Record has been modified
3. Record has been erased
4. Record is locked for exclusive update by concurrent run-unit
5. Record has been inserted in or removed from an index
6. Record's physical location on the database has changed

2.4.4 Privacy System

SIBAS supports two levels of privacy.

1. Privacy on database level
2. Privacy on record occurrence level

The privacy checks performed on all levels use a password supplied by the run-unit to check if the run-unit has authority to carry out the intended operation. All privacy checking in SIBAS is performed at run-time and it is therefore possible to redefine the passwords as often as desired.

A run-unit supplies the run-unit's password when the database is opened. This password remains the run-unit's "current password" until modified using the CHANGE CURRENT PASSWORD statement. This special statement may be used to change the run-unit's current password whenever necessary.

The table below shows how privacy restrictions on a database are defined, how and when passwords may be defined and modified, and when the privacy checks are performed by the SIBAS run-time control system (DBCS).

Privacy Level	How Privacy Restr. s Defined	How Valid Passwords are:		When Passwords are Checked
		Defined	Changed	
Database	using DBM module	using DBM module		at database open
Record occurrence	using schema re-definition language	when a record occurrence is stored	when a record occurrence is modified	when run-unit wants to modify, delete or get items

The password is of the same length and type as used for definition of data item names for the installation.

2.4.4.1 PRIVACY ON DATABASE LEVEL

As indicated above, database privacy restrictions and passwords are defined by use of the Database Maintenance Module (collection of utility programs).

The password is given as a parameter in the OPEN DATABASE STATEMENT.

There is a limit to the number of times a run-unit unsuccessfully may try to open the database.

2.4.4.2 PRIVACY ON RECORD OCCURRENCE LEVEL

It is possible with SIBAS to define privacy items on the record occurrence level.

This privacy item is stored together with the record. For this reason, the definition of the privacy item which will contain the value of the record occurrence password has to be part of the record type description. Privacy restrictions on the record occurrence level must therefore be defined using the Definition/Redefinition Language. Record occurrence passwords are considered as a special data item type just as other items may be of type INTEGER or CHARACTER.

The privacy item is given a value in the same way as other items in the record, when the record is stored or modified (see Figure 2.27).

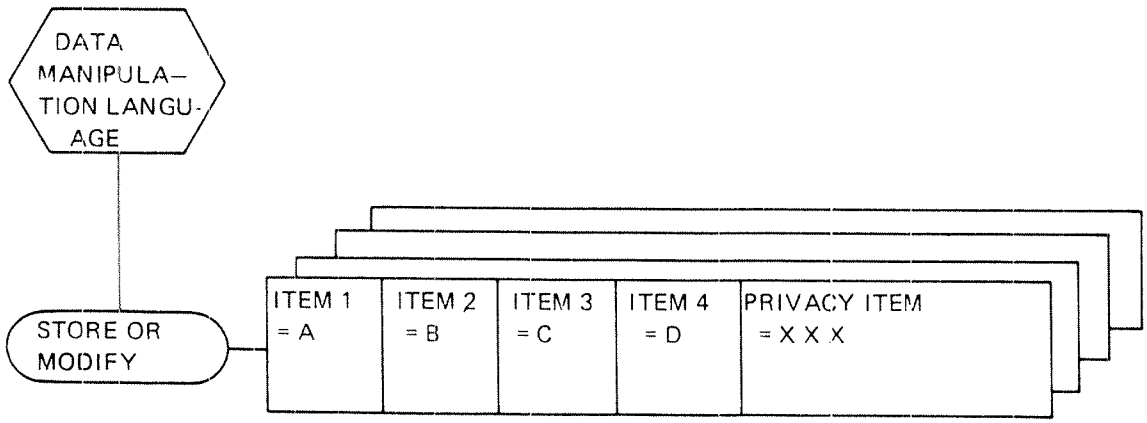


Figure 2.27: Giving Value to Privacy Item

Like other items, the privacy item need not be given a value when the record is stored. The privacy item will then be set to a null value by the DBCS. A record for which privacy on record occurrence level is defined, but with null value on the privacy item, may be manipulated as if no privacy item was defined for that record type.

The privacy check is performed when a run-unit tries to retrieve information from the record (the GET statement) and when a run-unit tries to modify or delete the record or its set membership. Note that no restriction is put on the use of FIND statements.

2.4.4.3 SUMMARY OF THE SETTING OF CURRENT PASSWORD

Initially the current password is set for a run-unit when the database is opened (see Figure 2.23). Unless a CHANGE PASSWORD is performed, the value of current password will remain unchanged. When a READY REALM is performed, current password must match a password which is defined for the desired mode of operation on the realm. If the run-unit performs a record manipulation statement on records where the value of the record lock is different from the realm password, current password for the run-unit must be changed before the manipulation statement is successfully executed.

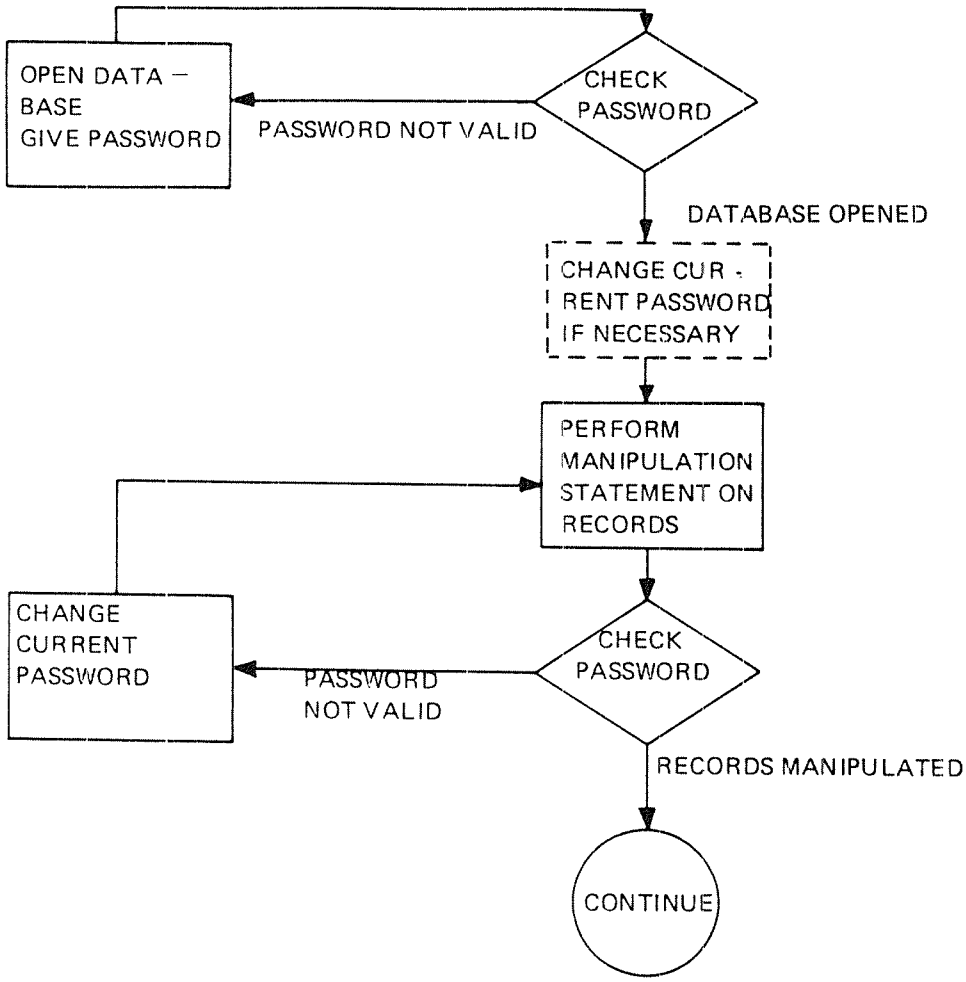


Figure 2.28: Use of Current Password

3 DEFINITION/REDEFINITION LANGUAGE (DRL)

3.1 INTRODUCTION

A SIBAS database must be defined before any data may be loaded in it. A definition is the process of producing an internal representation of the schema, the object schema, from the source schema written in a COBOL like syntax. A redefinition is the process of amending the object schema, and making the changes on the database.

Experience with all DBMS to date has indicated the importance of being able to redefine the database when new requirements are identified. It is a widely recognized objective that this redefinition should be possible without causing *unnecessary* modification to the programs, which have been written to process the database as initially structured. The degree to which a DBMS can meet this objective is essentially a measure of the degree of *data independence* offered by the DBMS.

With SIBAS, the same language is used to define or redefine a database. The statements (directives) provided may be classified in 3 categories:

1. creations — the NEW ... statements
2. deletions — the DELETE ... statements
3. changes — the CHANGE ... statements

Each of these statements will be described in detail later in this chapter.

The DRL statements available are:

START INITIATION	first statement of an initiation (definition) run
START REDEFINITION	first statement of a redefinition run
END	last statement of a run
NEW OS-FILE	adds a SINTRAN file to the database
NEW SYSTEM REALM	defines a new system realm
NEW SERIAL REALM	defines a new user realm with location mode serial
NEW CALC-REALM	defines a new user realm with location mode CALC and the corresponding CALC key
NEW ITEM	defines a new item in an existing or new record type
NEW GROUP	defines a new group item in an existing or new record type

NEW SET	defines a new set type in the database
NEW INDEX	adds the index key property to an existing or new item, and defines the storage of the index table
DELETE SET	removes a set type from the database
DELETE INDEX	removes the index property from an existing item and deletes the corresponding index table
DELETE ITEM	deletes an item from an existing record type
DELETE GROUP	removes a group item definition from an existing record type
CHANGE SYSTEM-REALM	changes the definition of an existing system realm
CHANGE SERIAL-REALM	changes the definition of an existing user realm with location mode serial, or changes the location mode from CALC to serial
CHANGE CALC-REALM	changes the definition of an existing user realm with location mode CALC, or changes the location mode from serial to CALC and defines the corresponding CALC key.
CHANGE SET	changes the definition of an existing set type

3.2 HOW THE DEFINITION/REDEFINITION MODULE WORKS

The DRL module requires exclusive use of the whole database and accesses the realms directly without using a SIBAS process at all.

The functions of the statements are to create and update the object schema and perform the corresponding actions on the database. A documentation of the database may also be produced as shown in Figure 3.1.

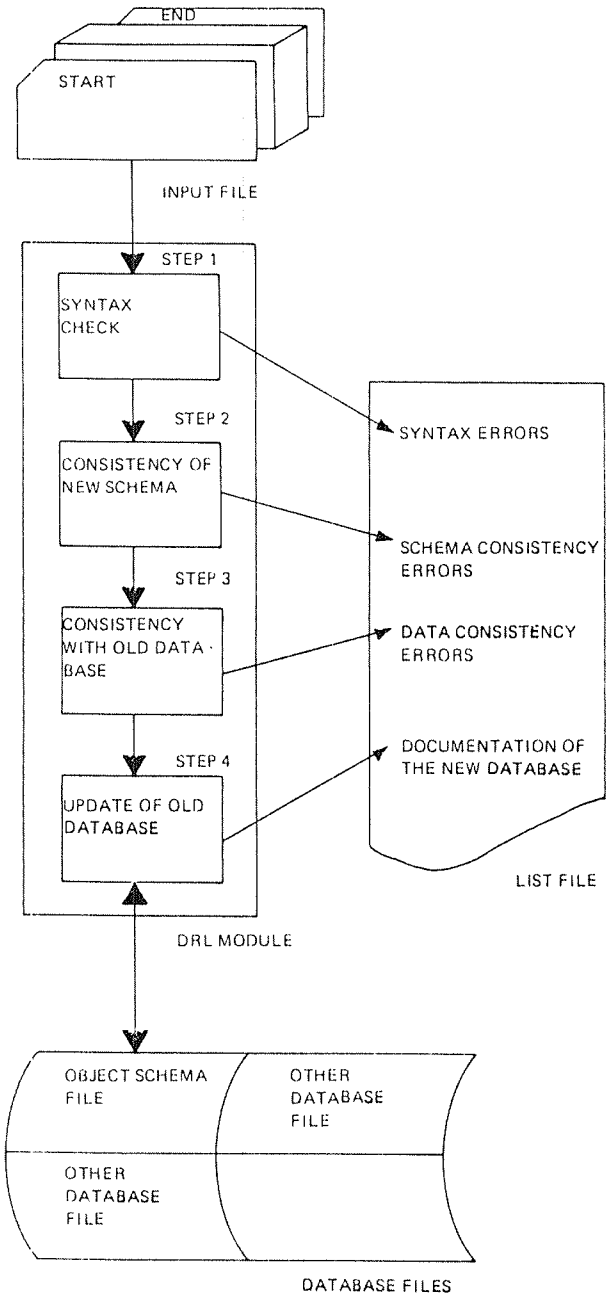


Figure 3.1: The Data Definition and Redefinition Module

When the DRL module is used, it requires the exclusive use of the database files.

Definition:

The SINTRAN files on which the database will be initiated must have been created before the DRL module may be used.

A complete example of a DRL run is shown at the end of this chapter. It must be noted that if there are CALC realms, the DRL module must preformat them (this operation may take time).

Redefinition:

When the DRL module is used for redefining a database, two modes of operations are available, the test mode and the production mode. When running the test mode, only the three first steps will be executed (see Figure 3.1). In production mode, all the steps will be executed. It must be noted that some apparently minor amendments might result in large computer resource usage. As a good practice, take a full back-up copy of the database before you run a redefinition. An example of a DRL run for redefinition is shown at the end of this chapter.

3.3 GLOBAL RULES

Syntax

Statements must be between columns 1 to 72 otherwise the DEF/REDEF module truncates.

The syntax of the definition and redefinition language is sentence oriented, just like COBOL. It means that all statements consist of a series of one or more words terminated by a period ".", the period indicating the end of a statement. A statement may begin anywhere in a line and may continue on any number of lines. However, a word cannot cross a line boundary. Words in a sentence may be key words or parameters. Key words may be abbreviated, parameters cannot be abbreviated. The parameters may be names or numbers.

A line starting with asterix "*" in column one will be treated as a comment line and ignored.

The syntax is described with the following conventions:

ANY-STRING	ANY-STRING is a key word which must be present
ANY-STRING	is merely a noise word which helps document the input, but may be omitted.
<any-name-or-value>	"any-name-or-value" is a parameter.
! <realm-name> ! ! KEY !	one of the two alternatives*must* be given: either the parameter "realm-name" or the key word "KEY".
(NOT)	the key word NOT is optional

Statement Sequence

A DRL input sequence must start with the START statement and end with the END statement or EXIT.

The sequence of the other statements is generally free, but when a statement refers to an existing name, the name must have been previously defined. For example, the statement

NEW SYSTEM-REALM <realm-name> OS-FILE <file name> ....

defines a new "realm-name" but refers to the "file-name".

Names

SIBAS recognizes a number of names; such as database name, set name, item name, etc. Each name in SIBAS must contain between 1 and 8 alphanumeric characters. No embedded blanks are permitted, but terminal blanks are. The first character of a SIBAS name must be alphabetic.

Abbreviation Lookup

All key words (not parameters) can be abbreviated. However, ambiguity is not handled. The first match is always used.

Numbers

In some of the statements, the length of a record expressed as a number of computer words must be given. On the ND-10 or ND-100, a computer word is taken as 2 bytes (16 bits). On the ND-500 a computer word is 32 bits, but in format descriptions in *this* manual «word» means 2 bytes (16 bits) also for the ND-500, to make f.i. the same schema run on all ND-machines. See also 4.3.

Additions to a Schema

The most common kind of definition or redefinition which would be performed is the addition of new structural components.

Changes to a Schema

Many changes can be performed on an existing schema. In the CHANGE statements, most of the possible changes are given as options and the default value will *always* be that the definition is unchanged.

Deletions from a Schema

In the case of deletions, any program which uses any of the properties deleted must be carefully modified. Normally, however, deletions would only be made if the programs which process the database using these properties are themselves obsolete.

3.4 START INITIATION

Function

A new database is defined using the DRL module. The statement START INITIATION will define a new database with the name given in the START statement.

Format

```
START  INITIATION  DATABASE  <db-name>
  ( SUPPRESS (REALM) (RECORD-TYPE) (ITEM) (SET) (INDEX-TABLE) )
  SIZE  <no-of-64w-pages> .
```

Rules:

1. In the START INITIATION statement, the name of the database which is to be defined is given. It is the name of a SINTRAN Operating System (OS) file of type DATA which must have been previously created. This OS file is used as the SIBAS system realm and cannot be shared by any other realm. It should not be declared with a NEW OS FILE statement. The SIBAS system realm is where the object schema is stored. Additional user system realms may be defined with the NEW SYSTEM REALM statement.

2. SUPPRESS. The SUPPRESS clause can be used to suppress the *documentation* of realms, record types, items, sets or index tables. This will have no influence on the resulting database definition. If the SUPPRESS clause is omitted, a full documentation of the database will be printed.

3. SIZE. In the SIZE clause the expected size of the object schema is given in number of 64 word blocks. The object schema is stored in the SIBAS System Realm. To avoid problems, give a large number, for example 1000, and create the SINTRAN file as an "INDEXED FILE". This may be done by @CREATE-FILE <db name>,,, before the DRL module is used.

3.5 START REDEFINITION

Function

This statement will start the schema DRL for the database identified in the statement. The size of the file provided for the object schema (i.e., the SIBAS system realm) may be changed. The statement is also used to select the mode of operation.

Format:

```
START  REDEFINITION  DATABASE  <db-name> ( DBA-PASSWORD <password> )
       ( SUPPRESS (REALM) (RECORD-TYPE) (ITEM) (SET) (INDEX-TABLE) )
       SIZE  <no-of-64w-pages>
             ! TEST       !
       MODE  !            !  COPY OF SYSTEM-REALM   <file-name-1>
             ! PRODUCTION !
       SCRATCH-FILE  <file-name-2> .
```

Rules:

1. In the START REDEFINITION statement the name of the database which is to be redefined is given, and if privacy is defined for the database through the DBM modules (see 6.1.7) the DBA PASSWORD is given.

2. All realms in the database will automatically be readied with protection mode EXCLUSIVE when the START statement is given.

3. SUPPRESS. The SUPPRESS clause can be used to suppress the *documentation* of realms, record types, items, and sets of index tables. This will have no influence on the resulting database definition. If the SUPPRESS clause is omitted, a full documentation of the new database will be printed.

4. MODE. The two modes of operation are Test Mode and Production Mode. One should take a backup copy of the database before running the redefinition program in production mode.

5. SCRATCH FILE. If the execution of the DRL incluces a CHANGE REALM or NEW INDEX the "file-name-2" must be the name of a file with maximum 8 characters, which is big enough to hold *any* of the realms in the database which are to be changed. If the scratch file is not big enough, the execution of the redefinition may stop in the middle of step 4, leaving a destroyed database.

6. COPY OF SYSTEM REALM. "File-name-1" must hold the name of a file with maximum 8 characters which holds a copy of the object schema (SIBAS system realm). The schema DRL will give an error message if this is not the case. After an unsuccessful execution of the DRL or after an execution in Test Mode, the content in "file-name-1" must be copied back to SIBAS system realm for the database.

3.6 END/EXIT

Function:

The statements indicate the end of the database definition or redefinition.

Format:

END REDEF.

or

EXIT

Rules:

1. Any statement following END or EXIT will be ignored.

3.7 NEW OS FILE

Function:

The statement will define a new OS file for the database.

Format:

```
NEW  OS-FILE  <file-name>  ( PAGESIZE <no-of-words> )
        ( DIRECTORY <abbreviated-dir-name> ) .
```

Rules:

1. FILE NAME. The parameter "file name" must not be the same as the name of any existing SINTRAN file for this database. The type of the file is automatically DATA. The "file name" is treated as any other SIBAS name. The file must have been previously created in SINTRAN.

2. PAGE SIZE. The "page size" is the number of words that will be read into the SIBAS buffer area when a realm located on this OS file is accessed. The default value is 512 words. Guidelines on how to estimate the "page size" are given at the end of this chapter. It must be between 64 and 2048.

3. DIRECTORY. The "abbreviated-directory-name" is a four character abbreviation of the directory where the file is placed. If the substatement is omitted, the default directory will be used.

Hints:
In a test phase it is recommended that a SINTRAN "INDEXED FILE" is used. When the database is in operation, response time will be improved by changing the file to CONTINOUS. This is done by

```
@CREATE—FILE    <new>:DATA      <size>
@COPY—FILE      <new>:DATA      <file-name
@RENAME—FILE     <file-name>     <new>:data
```

<size> can be found in the SIB—DRL documentation.

3.8 NEW SYSTEM REALM

Function:

The statement will define a new user system realm for the data base.

Format:

```
NEW  SYSTEM-REALM  <realm-name>  OS-FILE <file-name>
          REALMSIZE <no-of-pages> .
```

Rules:

1. REALM NAME. The parameter "realm-name" must not be the same as the name of any existing realm.

2. FILE NAME. The parameter "file-name" must be the name of an OS file previously defined using NEW OS-FILE.

3. REALM SIZE. The parameter "no.-of-pages" gives the size of the system realm in terms of OS file pages. Guidelines on how to estimate the size of system realms are given at the end of this chapter.

4. STATEMENT SEQUENCE. The OS file referred to must be defined prior to this statement using NEW OS-FILE.

3.9 NEW SERIAL REALM

Function:

The function of this statement is to define a new serial realm for the database, which implies adding a new record type to the database.

Format:

```
NEW  SERIAL-REALM <realm-name>  OS-FILE  <file-name>
       REALMSIZE  <no-of-pages>
       RECORD LENGTH  <no-of-words>
       ( MAIN <system-realm>  ( ADDITIONAL <system-realm>
                                (<system-realm>) (<system-realm>))) .
```

Rules:

1. REALM NAME. The parameter "realm-name" must not be the same as the name of an existing realm.

2. FILE NAME. The parameter "file-name" must be the name of an OS file previously defined using NEW OS FILE.
   (Remember there is an upper total limit of 65533 pages that can be assigned to this file).

3. REALM SIZE. The parameter "no.-of-pages" gives the size of the realm in number OS file pages.

4. RECORD LENGTH. The record "length" must be given for all user realms. The record length must include all pointers in number of words in the record. The length of a pointer is 2 words. Space must be allowed for two or one pointers for each set type the record type is defined with link to, depending on whether the set type is defined with link to prior or not.

5. SYSTEM REALMS. The parameters "main-system-realm" and "system-realm-1" to "system-realm-3" must contain the names of system realms defined by using NEW SYSTEM-REALM prior to this statement. The system realms will be used for storing index tables. The *same* system realm may be used for more than one user realm. if MAIN option not used, the first system realm defined by NEW SYSTEM REALM will be used.

6. MINIMUM RECORD CONTENT. For all user realms, there must be at least one elementary item defined by using NEW ITEM.

7. STATEMENT SEQUENCE. The OS file and any system realms must be defined prior to this statement using NEW OS-FILE and NEW SYSTEM-REALM.

3.10 NEW CALC REALM

Function:

The function of this statement is to define a new calc realm for the database, which implies that a new record type will be added to the data base.

Format:

```
NEW ,CALC-REALM  <realm-name>  OS-FILE  <file-name>
        REALMSIZE  <no-of-pages>
MAIN-AREA  <no-of-pages>
RECORD LENGTH <no-of-words>
CALC-KEY    <key-name>    DUPLICATES  ARE ( NOT )  ALLOWED
( MAIN <system-realm>  ( ADDITIONAL <system-realm>
                         (<system-realm>) (<system-realm>))) .
```

Rules:

1. REALM NAME. The parameter "realm-name" must not be the same as the name of any existing realm.

2. FILE NAME. The parameter "file-name" must be the name of an OS file previously defined using NEW OS-FILE.

3. REALM SIZE. The parameter "no.-of-pages" gives the total size of the realm in number of OS file pages.

4. MAIN AREA/OVERFLOW AREA. The space in which the records are to be stored must be divided into a main area and an overflow area. Each of these areas must be further divided into a number of buckets. All buckets both in the main area and in the overflow area are of equal size. The bucket size is always one SIBAS page. The number of buckets in MAIN AREA, "no.-of-pages" should be a prime number. The number of buckets in OVERFLOW AREA will be the difference between the total number of pages given for REALM SIZE and the number of pages given for MAIN AREA.

5. RECORD LENGTH. The "record-length" must be given for all user realms in the number of words. The length of a pointer is 2 words. Space must be allowed for one or two pointers for each set type the record type is defined as member or owner of, depending on whether the set type is defined with link to prior or not.

6. CALC KEY. The "key-name" must refer to an item or a group item which must be defined for the record type using NEW ITEM or NEW GROUP in a later statement. The item/group item will automatically be assigned the CALC KEY property. Duplicates will be allowed on the key, unless the NOT option is given.

7. SYSTEM REALMS. The parameters "main-system-realm" and "system-realm-1" to "system-realm-3" must contain the names of system realms defined by using NEW SYSTEM-REALM prior to this statement. The system realms will be used for storing index tables. The same system realm may be used for more than one realm.

8. MINIMUM RECORD CONTENT. For all CALC realms there must be defined an elementary item serving as CALC key defined by using NEW ITEM.

9. STATEMENT SEQUENCE. The OS file and any system realms must be defined prior to this statement using NEW OS-FILE and NEW SYSTEM REALM. The CALC key item must be defined later using NEW ITEM or NEW GROUP.

10. All the main buckets are preformated at initiation time. The user must ensure there is enough disk space and be aware that the preformating takes some time.

3.11 NEW ITEM

Function:

The function of this statement is to define a new item for a record type previously defined using NEW CALC REALM or NEW SERIAL REALM. Items defined must be given type and length, and the position within the record type may be specified.

Format

```
NEW  ITEM  <realm-name>  <item-name>

          ! INTEGER        !
     TYPE ! FLOATING       !   START  <word-no>
          ! CHARACTER      !
          ! PRIVACY-ITEM   !

                    ! BIT  POSITION <first-bit>  !
     LENGTH  <no>   ! WORD                       ! .
                    ! BYTE POSITION <first-byte> !
```

Rules:

1. REALM NAME. The "realm-name" must refer to a realm defined using NEW CALC-REALM or NEW SERIAL-REALM prior to this statement.

2. ITEM NAME. The "item-name" must be different from all other items or group items in the same record type.

3. ITEM TYPE. A type must be specified for the item. If an item is defined as PRIVACY-ITEM, the length and definition of the item must be the same as for item names, realm names, etc. (i.e., four words). When a record of this type is retrieved, the person retrieving the record must provide the value of the privacy item in order for the retrieval to be successful.

4. START POSITION. The "word-number" must contain an integer greater than or equal to 1 to indicate in which computer word in the record the start of the item value is to be stored.

5. LENGTH. If the item occupies one word or more, the length must be given in "no.-of-words". If the item occupies less than one word the length is given in number of bits or number of bytes. The postion in the word must be completed with an integer greater than or equal to zero to indicate the first bit or the first byte in the word which the item value occupies. If length is given in bytes and the position is not given the item will start in the second byte (number 1) in the word. Bit counting starts with bit number 0.

6. SIZE OF INTEGERS. If the item is defined as integer, then its minimum length is 1 bit, and its maximum length can be freely chosen by the user.

7. SIZE OF FLOATING. If the item is defined as floating, it will normally occupy an integral number of words which may be freely chosen by the user.

8. SIZE OF CHARACTER. If the item is defined as character, then it may occupy less than one word. The item may occupy more than one word as long as it is allocated in integral number of words.

9. CALC KEY ITEM. The NEW CALC-REALM statement is used to define the item as CALC KEY.

10. OWNER SET ITEM. The NEW SET statement is used to define the item as owner set item.

11. MEMBER SET ITEM. The NEW SET statement is used to define the item as member set item.

12. INDEX KEY. The NEW INDEX statement is used to define the item as an index key.

3.12 NEW GROUP

Function:

The function of this statement is to give a name to a group of elmentary items within a record type. The items need not be contiguous in the record type. The sequence of the items in the group may be different from the sequence in the record type, and an item may also participate in more than one group item. Properties as calc key, index key, member set item and owner set item may be assigned to a group item in the same way as they are assigned to an elementary item. If the group item is going to be assigned the calc key property the best performance will be achieved if the group consists of contiguous items.

Format:

```
NEW  GROUP  <realm-name>  <group-name>
        <item-name>  (<item-name> ....)  .
```

Rules:

1. REALM NAME. The "realm-name" must refer to realm defined using NEW CALC-REALM or NEW SERIAL-REALM prior to this statement.

2. GROUP NAME. The "group-name" must be different from all item or group item names in the record type.

3. ITEM NAMES. The "item-name-1", "item-name-2", etc. must refer to elementary items in the record type defined by using NEW ITEM prior to this statement.

4. ORDER OF ITEMS. The order in which the elementary items are defined may be quite independent of the order in which they are defined using NEW ITEM. However, the order, once defined, is significant and must be preserved when values of the group are given in DML statements.

5. ITEMS IN MORE THAN ONE GROUP ITEM. Any elementary item may be a constituent item in one or more groups of the same record type.

6. NUMBER OF ITEMS. The maximum number of elementary items in a group item is approximately 50.

7. CALC KEY ITEM. The NEW CALC-REALM statement is used to designate the group item as CALC KEY.

8. OWNER SET ITEM. The NEW SET statement is used to define the group item as owner set item.

9. MEMBER SET ITEM. The NEW SET statement is used to define the groups item as member set item.

10. INDEX KEY. The NEW INDEX statement is used to define the group item as an index key.

11. EXTRA NAMES OF ITEMS. If the group item consists of only one elementary item, the effect of the group item will be to give an extra name to the elementary item. This can be useful if an elementary item is used as a member set item in two different set types.

3.13 NEW SET

Function:

The statement defines a new set type for the database. The owner and member record types must be record types defined prior to this statement. The statement will also assign the properties of member set item and owner set item to the item/group item in the member and owner record type.

Format:

```
NEW  SET  <set-name>

     LINK IS   ! SINGLE !
               ! DOUBLE !

     STORAGE-CLASS IS   ! AUTOMATIC !
                        ! MANUAL    !

     OWNER  <owner-set-item>  <realm-name>

     MEMBER <member-set-item> <realm-name> ( <realm-name> .... ) .
```

Rules:

1. SET NAME. The "set name" must be different from the name of any other set type in the same database.

2. LINK. If the SINGLE option is given, the set will have a link to next member only. If the DOUBLE option is given, each member will have link to next and prior member.

3. STORAGE CLASS. If the AUTOMATIC option is given, the set type will have a storage class of automatic. When a STORE or MODIFY is executed on a member record, it will be automatically connected into a set occurrence. If the MANUAL option is given, the set type has a storage class of manual and records will not be connected into a set of this type when a STORE is executed. In SIBAS the storage class is the same as the removal class. Whether a member record is automatically erased when the owner is erased, depends on the option given in the ERASE statement.

4. OWNER. The "owner-set-item" must be defined as an item or group item for the owner record type. The name of the owner record type is given in "realm-name". Furthermore, the "owner-set-item" must either be defined as calc key or as index key with NO DUPLICATES allowed. The key must be defined prior to this statement. The item/group item given will be assigned the property of owner set item, unless it already has this property.

5. MEMBER. The "member-set-item" must be defined as an item or group item for the member record type(s). The member record type(s) are given in "realm-name-1", "realm-name-2", etc. (maximum 46 member record types). The item/group item given will be assigned the property of member set item, unless it is already used for another set type. In this case the item must be given an extra name by defining it as a group item. Note that the member set items must have the same name in all realms.

6. INVOLUTED SET TYPE. If the member record type and the owner record type is the same for a set type, the set type is involuted. Then the "member-set-item" and the "owner-set-item" must both be defined as items or group items for the record type, but they must have different names.

7. CORRESPONDENCE BETWEEN MEMBER SET ITEM AND OWNER SET ITEM. The "owner-set-item" and the "member-set-item" must correspond in length and item type. The two items may also have the same name unless the set type is involuted. Correspondence in the case of group items means that it must be possible for the concatenated values of the constituent elementary items to be exactly equal.

3.14 NEW INDEX

Function:

The function of this statement is to define an item or group item as index key and define the storage of the corresponding index table.

Format:

```
NEW INDEX  <realm-name>  <key-name>

    UPDATE IS  ! MANUAL    !   DUPLICATES ARE ( NOT )  ALLOWED
               ! AUTOMATIC !

  (SYSTEM-REALM   <system-realm-name>)

  ( MIN-VALUE  <value>   MAX-VALUE  <value> )  .
```

Rules:

1. REALM NAME. The "realm-name" must refer to a realm defined prior to this statement.

2. KEY NAME. The "key-name" must refer to an item or group item defined for the record type named in "realm-name". The item/group item will be assigned the index key property.

3. UPDATE If the MANUAL option is given, the index will be manually maintained, and the index table will not be updated when a STORE or MODIFY is executed. If the AUTOMATIC option is given, the index table will be automatically maintained when a STORE or MODIFY is executed.

4. DUPLICATES. If the NOT option is given, an attempt to store a record of this record type will fail if there is already an entry in the index table with this key value. If the NOT option is omitted, it means that duplicate values of this index key are permitted.

5. SYSTEM REALM. The "system-realm" in which all tables are stored must either be the main system realm for the record or an additional system realm, already defined by using NEW SYSTEM REALM.

6. MIN VALUE/MAX VALUE. If the actual minimum and maximum values of the index key are known at the time when the database is defined, these values should be given to achieve better performance when using the index key. Note that it is enough that the key *usually* is between the limits, exceptions are allowed. The parameter "value" must be a positive integer. If a key consists of more than one word, the value of the first word is given. If the key is alphanumeric, "value" should be the corresponding integer value (if any) of the first word of the key.

3.15 DELETE SET

Function:

The function of this statement is to delete a set type from the database schema. When a set type is deleted, the record types which serve as its owners and members remain in the database. All these record types are adjusted so that there is no space assigned for pointers, but the record length will remain unchanged unless it is changed by use of CHANGE REALM. The member set item of all member record types will cease to have this role. The owner set item of the owner record types will cease to have this role if it was owner set item only for the deleted set type.

Format:

DELETE SET - <set-name> .

Rules:

1. Name of SET TYPE. The "set name" must be the name of a set type which is defined in the old database schema.

2. OWNER SET ITEM. If the owner set item of the owner record type does not serve as owner set item of any other set type, the item will automatically be redefined such that it no longer is an owner set item.

3. MEMBER SET ITEM. The member set item of all member record types will automatically be redefined such that they no longer are member set items.

3.16 DELETE INDEX

Function:

The function of this statement is to remove the index property from an item or group item, and to delete the corresponding index table.

Format:

```
DELETE  INDEX     <realm-name>  <key-name>  .
```

Rules:

1. REALM NAME. The "realm-name" must be the name of an existing realm.

2. KEY NAME. The "key-name" must identify an item or group item defined as index key for this record type.

3. INDEX KEY PROPERTY. The index key property will automatically be removed from the item identified by "key-name".

4. SET OWNER. If the item given in "key-name" is defined as owner set item, the set must be deleted prior to this statement.

3.17 DELETE ITEM

Function:

The function of this statement is to remove an item from an existing record type. The record length will remain unchanged unless it is changed by use of CHANGE REALM.

Format:

DELETE ITEM <realm-name> <item-name> .

Rules:

1. NAME OF REALM. "Realm-name" must be the name of the realm where records of this type are stored.

2. NAME OF ITEM. "Item-name" must be the name of an item which is defined for the record type, identified by "realm-name". The item may play different roles in the record type and the consequences of the DELETE ITEM are given in the following rules.

3. INDEX KEY ITEM. If the item given is defined as an index key or is part of an index key, the index table must be deleted prior to this statement using DELETE INDEX.

4. MEMBER OF GROUP ITEM. If "item-name" identifies an item which is defined as member of a group item, the item will automatically be deleted from the group description, unless the group is defined as index key, calc key or set item (see rules 3, 5 and 6). It is not necessary to change the group composition using a CHANGE GROUP statement, which is therefore not provided.

5. CALC KEY. If the item given is defined as a calc key or is a part of a calc key, a new calc key must be defined for the record type (using CHANGE CALC-REALM) or the location mode of the realm must be changed from calc to serial (using CHANGE SERIAL-REALM). The CHANGE CALC-REALM or CHANGE SERIAL-REALM must be given prior to DELETE ITEM.

6. MEMBER SET ITEM/OWNER SET ITEM. If the item given in "item-name" is defined as member set item or owner set item for a set type, or if it is part of a set item, the set type must be deleted using DELETE SET or changed using CHANGE SET prior to this statement.

7. STATEMENT SEQUENCE. The following statements may have to be given prior to DELETE ITEM (see rules 3, 5 and 6).

   CHANGE CALC-REALM
   CHANGE SERIAL-REALM
   DELETE SET
   CHANGE SET
   DELETE INDEX

3.18 DELETE GROUP

Function:

The function of this statement is to remove a group item from an existing record type. The result of this is that the group item can no longer be referred to from the DML statements, but the items constituting the group item will remain in the record type.

Format:

DELETE GROUP <realm-name> <group-name> .

Rules:

1. NAME OF REALM. "Realm-name" must be the name of the realm where records of this type are stored.

2. NAME OF GROUP ITEM. "Group-name" must be the name of a group item which is defined for the record type identified by "realm-name". The group item may play different roles in the record type and the consequences are given in the following rules.

3. INDEX KEY ITEM. If the group item given is defined as an index key, the index table must be deleted prior to this statement using DELETE INDEX.

4. CALC KEY. If the group item given is defined as a calc key, a new calc key must be defined for the record type (using CHANGE CALC-REALM), or the location mode of the realm must be changed from calc to serial (using CHANGE SERIAL REALM). The CHANGE CALC-REALM or CHANGE SERIAL-REALM must be given prior to DELETE GROUP.

5. MEMBER SET ITEM/OWNER SET ITEM. If "group-name" is defined as member set item or owner set item for a set type, the set type must be deleted using DELETE SET or changed using CHANGE SET prior to this statement.

6. STATEMENT SEQUENCE. The following statements may have to be given prior to DELETE GROUP (see rules 3, 4 and 5).

   CHANGE CALC-REALM
   CHANGE SERIAL-REALM
   DELETE SET
   CHANGE SET
   DELETE INDEX

3.19 CHANGE SYSTEM REALM

Function:

The function of this statement is to change the realm size of an existing system realm, or the page size by moving it to another OS-file with a different page size.

Format:

CHANGE SYSTEM-REALM <Realm-name>

(REALMSIZE <no-of-pages>) .

Rules:

1. REALM NAME. The "realm-name" must identify an existing user system realm (not SIBAS system realm).

2. REALM SIZE. The parameter "no.-of-pages" gives the maximum size of the system realm in terms of OS file pages. Guidelines on how to estimate the size of system realms are given at the end of this chapter.

3.20 CHANGE SERIAL REALM

Function:

The function of this statement is to change the definition of an existing serial realm, or to change an existing calc realm to serial realm. In the latter case the calc key will automatically cease to have this role.

Format:

```
CHANGE  SERIAL-REALM  <realm-name>
        ( OS-FILE  <file-name> )  ( REALMSIZE  <no-of-pages> )
        ( RECORD LENGTH  <no-of-words> ) .
```

Rules:

1. REALM NAME. The parameter "realm-name" must be the same as the name of existing serial realm or calc realm.

2. OS FILE. The "file-name" must be the name of an existing OS FILE.

3. REALM SIZE. If this option is used, the parameter "no.-of-pages" must give the maximum number of OS file pages estimated for the realm. If the realm is changed from CALC to serial the REALM SIZE option *must* be given.

4. RECORD LENGTH. If new items have been defined for the record type, or if the record type is defined as owner or member of new set types, the record length may have to be increased. The record length must include all pointers in the record. The length of a pointer is 2 words. Space must be allowed for one or two pointers for each set type the record type is defined as member or owner of, depending on whether the set type is defined with link to prior or not.

5. CHANGE OF LOCATION MODE. If "realm-name" identifies a realm with location mode calc, the location mode will be changed to serial and the calc key will automatically cease to have this role. If no "main-system-realm" was defined for the calc realm, the MAIN option must be given.

6. STATEMENT SEQUENCE. The following statements may have to be given prior to this statement (rule 5).

   DELETE INDEX
   NEW INDEX
   NEW SYSTEM-REALM

3.21 CHANGE CALC REALM

Function:

The function of this statement is to change the definition of an existing calc realm, or to change an existing serial realm to calc realm. In the latter case, an existing item in the record type must be defined as calc key.

Format:

```
CHANGE  CALC-REALM  <realm-name>
        ( REALMSIZE  <no-of-pages> )
        ( MAIN-AREA  <no-of-pages> )
        ( RECORD LENGTH <no-of-words> )
        ( CALC-KEY   <key-name>  DUPLICATES ARE (NOT) ALLOWED ) .
```

Rules:

1. REALM NAME. The parameter "realm-name" must be the same as the name of an existing serial realm or calc realm.

2. REALM SIZE. The "no.-of-pages" gives the total length of the realm in number of OS file pages.

3. MAIN AREA/OVERFLOW AREA. If this option is given, all records in the realm will be recalculated and stored according to the new definition. "No.-of-pages" should be a prime number.

4. RECORD LENGTH. If new items have been defined for the record type, or if the record type is defined as owner or member of new set types, the record length may have to be increased. The record length must include all pointers in the record. Space must be allowed for one or two pointers for each set type the record is defined as member or owner of, depending on whether the set type is defined with link to prior or not.

5. CALC KEY. If this option is given, the "key-name" must refer to an item or a group item which is defined for the record type. The item/group item must have non null values on the database. The item/group item will automatically be assigned the CALC KEY property. No other item/group item in the record type must have been defined as CALC KEY. If the "realm-name" refers to a realm with location mode SERIAL, the location mode will be changed to CALC. In this case the CALC KEY option must be given. It must be given whether DUPLICATES are allowed for the key or not. If "key-name" already has the role of CALC KEY, this option may be used to change it from DUPLICATES NOT ALLOWED to DUPLICATES ALLOWED or vice versa.

6. CHANGE OF LOCATION MODE. If "realm name" identifies a realm with location mode serial, the location mode will be changed to calc. The CALC KEY and the MAIN AREA options must then be given.

7. STATEMENT SEQUENCE. The following statements may have to be given prior to this statement (rule 7).

   DELETE INDEX
   NEW INDEX
   NEW SYSTEM-REALM

3.22 CHANGE SET

Function:

The function of this statement is to change the properties of an existing set type. The link may be changed from single to double or vice versa. The storage class may be changed from manual to automatic or vice versa. New member record types may be added or existing member record types may be deleted.

Format:

```
CHANGE SET  <set-name>

    ( LINK IS   ! SINGLE !   )
                ! DOUBLE !

    ( STORAGE-CLASS IS  ! AUTOMATIC !   )
                        ! MANUAL    !

    ( MEMBER <member-set-item> <realm-name> ( <realm-name> .... ) ) .
```

Rules:

1. SET NAME. The "set-name" must be the name of an existing set type.

2. LINK. This option may be used to remove prior link (SINGLE) or to include prior link (DOUBLE). If a change is made to remove the prior link, then for all member record occurrences the space occupied by the link is made available. It must be noted that the record length as specified in NEW REALM for this record type is the length including set pointers, and consequently removing the prior link of a set type will leave empty space in the owner and member record type.

   If a change is made to include the prior link, then the record length may have to be increased for the owner and member record type. In production mode, the link to prior is automatically established for every set occurrence.

3. STORAGE CLASS. The storage class of the set type may be changed from manual to automatic or vice versa. If the storage class of a set type is changed from manual to automatic, then all occurrences of the member record types are examined to see whether they can be connected to a set of the set type. If so, the connection is made in the same way as if a CONNECT were executed on the record and set type. Member records for which no matching set exists in the database are listed in a report, and the production mode will not be executed. If the storage class of a set type is changed from automatic to manual, then no changes are made to occurrences of the set type.

4. MEMBER. It is possible to define new member record types in a set type and to remove old member record types. If the MEMBER clause is given, all the member record types for the changed set type must be listed. Whether the new members are connected into sets will depend on the storage class of the set type. If it is automatic, then an attempt is made to connect each new member. Cases are listed where no owner is found in the database for the values of the member set item. If the storage class is manual, no connections are made, and step 4 (see fig. 3.1) is not executed.

   In the case that the set type is old and the member record type is new (that is being defined in the same use of the restructuring facility), then there are no occurrences of the record type in the database, and the existing sets of the set type are not affected.

   The MEMBER clause may also be used to change member set item. The "member set item" must be defined as an item or a group item for all member record types, and this item will automatically be given the property of member set item. It must, of course, correspond to the owner set item (see 3.13.7).

   When member set item is changed for a set type, all existing members will be disconnected from the set, and if the set type has automatic storage class all members will be connected according to the value of the new member set item and cases are listed where no owner is in the database for the values of the member set item. In this case step 4 will not be executed.

3.23 DIMENSIONING DATABASE PARAMETERS

SIBAS realms are stored on SINTRAN files, which may be created as INDEXED or CONTINUOUS files. Such files always occupy an integral number of "SINTRAN pages", i.e., 1024 words. SIBAS default length is 512 words. The minimum page size is 64 words, the practical maximum is 2K words (SINTRAN-III default value for the device buffer size). If the file type is INDEXED, disk space is allocated when the demand arises. One is then recommended to be generous when estimating the size of the REALM. SIB-DRL prints out useful information about the size of the realms and gives estimates for the size of the index tables. The maximum number of "SIBAS pages" on an OS—FILE (SINTRAN file) is 65533. I.e. the practical maximum size of one OS—FILE is about 250 Megabytes. The maximum number of "SIBAS pages" for any realm type is 65533.

SIBAS System Realm

When a SIBAS database is defined and initialized or redefined using the SIBAS Definition/Redefinition Language, an object version of the data base definition will be generated. The object version of the schema will be stored on the SIBAS system realm. In the following we will give some rules for estimating the space requirements for the SIBAS system realm.

The page size is always 64 words, and the system realm must be on a separate SINTRAN file. It will contain the realm description table, the I/O table, the set description table, the record description tables and the index description tables — usually the record description tables are the most voluminous. As a rule of thumb, 50 + 10* (no. of realms) pages suffice. However, one is recommended to be generous when estimating the size of this realm, to allow future redefinition, change, etc.

User System Realms

A user system realm contains one page reserved for SIBAS and a number of pages for the index tables stored on it. Index tables are organized as a hierarchy of tables, each occupying one page.

One index table is divided into a table head and a number of entries. At the lowest level, by far the most common, one index table entry consists of one key and one pointer. Since the key values are stored randomly, the packing density of the index tables is on average 60%.

$$\text{Number of entries on a page} = 60\% * \frac{\text{Page size} - 11}{\text{Key size} + 2}$$

$$\text{Number of pages for one index} = \frac{\text{Number of records}}{\text{Number of entries on a page}}$$

It must be noted that index tables might be compressed by a utility statement of the SIB-DBM module.

Serial Realms

A serial realm contains one page reserved for SIBAS and a number of pages for use by the records. A page is always headed by a pointer (2 words) and contains an integer number of records. Estimating the size of a serial realm is then an easy matter.

$$\text{Number of records on a page} = \frac{\text{Page size} - 2}{\text{Record length}}$$

$$\text{Number of pages for the realm} = \frac{\text{Maximum number of records}}{\text{Number of records on a page}}$$

CALC Realms

The main parameter when dimensioning a calc realm is the number of buckets in the main area. Choosing an optimum value for this number is not a straight-forward procedure. This number will also give the number of pages in the main area and, together with the page size, it will limit the total number of records in the main area. A prime number is used to give a better distribution with the randomizing algorithm SIBAS uses.

Overflow pages are linked to the main page when overflow occurs. They (overflow pages) are not preallocated to a specific main bucket (page). An overflow page belongs to only one main page.

Main and overflow pages have the same size and the same layout: they are headed by a pointer and contain an integer number of records. Estimating the size of a calc realm may be done as follows:

$$\text{Number of records on a page} = \frac{\text{Page size} - 2}{\text{Record length}}$$

$$\text{Number of buckets in main} = \text{Numbers of pages in main}$$

$$\text{Number of pages in main} = \frac{\text{Total number of records}}{\text{Number of records on a page}}$$

$$\text{Number of pages for the realm} = \text{Number of pages in main} + \frac{1}{60\%} \cdot \frac{\text{Estimated Overflowing Records}}{\text{Number of records on a page}}$$

3.24 HOW TO RUN DRL ON THE COMPUTER

Use an editor to write the source schema (with the necessary statements from this chapter) and store it on a file with name <database name>:SYMB, for example create the necessary SINTRAN files, by commands like:

```
@CREATE-FILE <database name>:DATA
@CREATE-FILE <name of OS-FILE1>:DATA
@CREATE-FILE <name of OS-FILE2>:DATA
.
.
.
```

Make sure you were logged in under the SINTRAN-user name where you want the database to reside.

Make sure user RT has write access to this user's files, by a command like:

```
@CREATE-FRIEND RT
```

for example.

Then run the DRL by the command

```
@SIB2-DRL.
```

(See the examples in the next section.) Answering questions by just CR, will be taken to mean NO.

3.25 EXAMPLES

An initiation run:

```
@DELETE-FILE    FORDB:DATA
@DELETE-FILE    SYSFILE:DATA
@CREATE-FILE    SYSFILE:DATA   0
@CREATE-FILE    FORDB:DATA     0
@SIB2-DRL

S I B 2 - D R L          SEPT  79

EXPLANATION ? NO
INTERACTIVE ? NO
INPUT-FILE  : FORDB:SYMB
LIST-FILE   : L-P



*************************************************************************
  D A T A B A S E   FORDB      I N I T I A T E D   36   15   12   10 1979*
*************************************************************************

000754 STOP    0
```

The output from the initiation run:

```
 1*    *
 2*    *       C O U R S E   .  DATABASE          OCT.  79
 3*    *
 4*
 5*     START INITIATION  DATABASE FORDB   .
 6*               SUPPRESS  REALM RECORD  ITEM  SET  INDEX
 7*               SIZE   171 .
 8*
 9*     NEW  OS-FILE  SYSFILE  PAGESIZE   256.
10*
11*     NEW SYSTEM-REALM   SYSFILE   OS-FILE SYSFILE   REALMSIZE       196.
12*     NEW SERIAL-REALM   PERSON    OS-FILE SYSFILE   REALMSIZE        51
13*               RECORD LENGTH      51
14*               MAIN SYSFILE .
15*     NEW SERIAL-REALM   JOBB      OS-FILE SYSFILE   REALMSIZE       119
16*               RECORD LENGTH      30
17*               MAIN SYSFILE .
18*     NEW   CALC-REALM   RAPPORT   OS-FILE SYSFILE   REALMSIZE       142
19*               MAIN-AREA          122
20*               RECORD LENGTH      25
21*               CALC-KEY  JOBNR    DUPLICATES
22*               MAIN SYSFILE .
23*
24*     NEW  ITEM  PERSON    AVDELING
25*               TYPE CHARACTER
26*               START   39  LENGTH       3.
27*     NEW  ITEM  PERSON    FDATO
28*               TYPE CHARACTER
29*               START    1  LENGTH       3.
30*     NEW  ITEM  PERSON    FNR
31*               TYPE CHARACTER
32*               START    4  LENGTH       3.
```

```
33*     NEW  ITEM  PERSON     KJONN
34*               TYPE CHARACTER
35*               START   35  LENGTH      1.
36*     NEW  ITEM  PERSON     PERSADR
37*               TYPE CHARACTER
38*               START   20  LENGTH     15.
39*     NEW  ITEM  PERSON     PERSNAVN
40*               TYPE CHARACTER
41*               START    7  LENGTH     13.
42*     NEW  ITEM  PERSON     PRESERVE
43*               TYPE CHARACTER
44*               START   42  LENGTH      6.
45*     NEW  ITEM  PERSON     TIMELONN
46*               TYPE INTEGER
47*               START   36  LENGTH      3.
48*     NEW  GROUP PERSON     AVDPERS
49*               AVDELING  PERSNAVN   .
50*     NEW  GROUP PERSON     PERSNR
51*               FDATO     FNR        .
52*
53*     NEW  ITEM  JOBB       BUDGTIM
54*               TYPE INTEGER
55*               START   10  LENGTH      2.
56*     NEW  ITEM  JOBB       FDATO
57*               TYPE CHARACTER
58*               START   15  LENGTH      3.
59*     NEW  ITEM  JOBB       FNR
60*               TYPE CHARACTER
61*               START   18  LENGTH      3.
62*     NEW  ITEM  JOBB       JOBBNAVN
63*               TYPE CHARACTER
64*               START    2  LENGTH      5.
65*     NEW  ITEM  JOBB       JOBBNR
66*               TYPE INTEGER
67*               START    1  LENGTH      1.
68*     NEW  ITEM  JOBB       JOBBTYPE
69*               TYPE CHARACTER
70*               START    7  LENGTH      3.
71*     NEW  ITEM  JOBB       JRESERVE
72*               TYPE CHARACTER
73*               START   21  LENGTH      6.
74*     NEW  ITEM  JOBB       MEDGTIM
75*               TYPE INTEGER
76*               START   12  LENGTH      3.
77*     NEW  GROUP JOBB       JOTYPENR
78*               JOBBTYPE  JOBBNR     .
79*     NEW  GROUP JOBB       PERSNR
80*               FDATO     FNR        .
81*
82*     NEW  ITEM  RAPPORT    ANTTIM
83*               TYPE INTEGER
84*               START    9  LENGTH      3.
85*     NEW  ITEM  RAPPORT    FDATO
86*               TYPE CHARACTER
87*               START    1  LENGTH      3.
88*     NEW  ITEM  RAPPORT    FNR
89*               TYPE CHARACTER
90*               START    4  LENGTH      3.
91*     NEW  ITEM  RAPPORT    JOBBNR
92*               TYPE INTEGER
93*               START    7  LENGTH      1.
94*     NEW  ITEM  RAPPORT    PERIODE
95*               TYPE INTEGER
96*               START    8  LENGTH      1.
97*     NEW  ITEM  RAPPORT    RRESERVE
98*               TYPE CHARACTER
99*               START   12  LENGTH      6.
```

```
100*      NEW  GROUP RAPPORT   PERJOBB
101*               PERIODE   JOBBNR     .
102*      NEW  GROUP RAPPORT   PERSNR
103*               FDATO     FNR        .
104*
105*
106*      NEW  INDEX    PERSON       AVDPERS
107*               UPDATE  IS AUTOMATIC
108*               DUPLICATES  ARE  NOT  ALLOWED.
109*      NEW  INDEX    PERSON       PERSNAVN
110*               UPDATE  IS AUTOMATIC
111*               DUPLICATES  ARE  NOT  ALLOWED.
112*      NEW  INDEX    PERSON       PERSNR
113*               UPDATE  IS AUTOMATIC
114*               DUPLICATES  ARE  NOT  ALLOWED.
115*      NEW  INDEX    JOBB         JOBBNR
116*               UPDATE  IS AUTOMATIC
117*               DUPLICATES  ARE  NOT  ALLOWED.
118*      NEW  INDEX    JOBB         JOTYPENR
119*               UPDATE  IS AUTOMATIC
120*               DUPLICATES  ARE  NOT  ALLOWED.
121*      NEW  INDEX    RAPPORT      PERJOBB
122*               UPDATE  IS AUTOMATIC
123*               DUPLICATES  ARE  ALLOWED.
124*      NEW  INDEX    RAPPORT      PERSNR
125*               UPDATE  IS AUTOMATIC
126*               DUPLICATES  ARE  ALLOWED.
127*
128*      NEW  SET PERRAP
129*               LINK  IS  DOUBLE
130*               STORAGE-CLASS  IS  AUTOMATIC
131*               OWNER   PERSNR       PERSON
132*               MEMBER  PERSNR       RAPPORT
133*      NEW  SET JOBRAP
134*               LINK  IS  DOUBLE
135*               STORAGE-CLASS  IS  AUTOMATIC
136*               OWNER   JOBBNR       JOBB
137*               MEMBER  JOBBNR       RAPPORT
138*
139*      END REDEF.

END OF STEP  1          NUMBER OF ERRORS =     0

END OF STEP  2          NUMBER OF ERRORS =     0

END OF STEP  3          NUMBER OF ERRORS =     0
```

```
END OF DATABASE DEFINITION
--------------------------

          NUMBER OF WARNINGS =      0
          NUMBER OF ERRORS   =      0

          SIZE OF DML RESIDENT TABLES  =    832

THE DATABASE IS INITIATED

          **********


END OF STEP  4            NUMBER OF ERRORS =      0


*************************************************************************
  D A T A B A S E   FORDB      I N I T I A T E D   36   15   12   10 1979*
*************************************************************************
```

A redefinition run:

```
@COPY-FILE ''FORDBCOP:DATA'' FORDB:DATA
@CREATE-FILE SCRATCH:DATA,,,

@SIB2-DRL

S I B 2 - D R L          SEPT  79

EXPLANATION ? NO
INTERACTIVE ? NO
INPUT-FILE  : FOR-CHANGE
LIST-FILE   : TERM

    1*    *
    2*    *            C O U R S E   DATABASE  CHANGE
    3*    *
    4*    START  REDEFINITION  DATABASE  FORDB.
    5*           SUPPRESS    REALM  RECORD  ITEM  SET  INDEX
    6*           MODE PRODUCTION
    7*           COPY OF  SYSTEM-REALM  FORDBCOP
    8*           SCRATCH-FILE   SCRATCH.
    9*
   10*    *
   11*    *       EXPAND THE  JOBB REALM
   12*    *
   13*    CHANGE  SERIAL-REALM  JOBB
   14*           REALMSIZE   150.
   15*
   16*    END.


END OF STEP  1          NUMBER OF ERRORS =     0

.. WARNING ..  103  STATEMENT STARTING LINE   16
CALC KEY IS NON-CONSECUTIVE GROUP, BAD PERFORMANCE


END OF STEP  2          NUMBER OF ERRORS =     0


END OF STEP  3          NUMBER OF ERRORS =     0

END OF DATABASE DEFINITION
--------------------------

         NUMBER OF WARNINGS =     1
         NUMBER OF ERRORS   =     0


         SIZE OF DML RESIDENT TABLES  =   832

THE DATABASE IS INITIATED

         **********

END OF STEP  4          NUMBER OF ERRORS =     0


*****************************************************************
  D A T A B A S E   FORDB      I N I T I A T E D   33   15   12   1(
*****************************************************************
```

4 DATA MANIPULATION LANGUAGE (DML)

SIBAS provides a selection of DML statements. Each DML statement has 2 forms, a short form, e.g., GET, MODIFY, STORE, and an encoded CALL form. The CALL form is to be used in application programming. The short forms are used in SIBINTER.

4.1 GENERAL

For a program to be able to access a SIBAS database, some or all of the record types in the database must be defined in the host language program.

It is important to note that not all record types in the data base need to be defined, but only those required. Furthermore, the same applies to items in a record type. If a program does not need to process all the items in a given record type, then those not required may be omitted from the record description in the program. This provides a subschema facility and enables the programmer to minimize the core space required at execution time.

The DML statements in SIBAS have the general form of a CALL statement. When this form is used, SIBAS may be used from any host language which provides a CALL statement facility. The description of records and items must then follow the conventions of the host language.

The programmer may choose his own names to identify the parameters in the various DML CALL statements. In order to clarify the role of each parameter in the following sections, each parameter is identified by a lower case narrative name which does not necessarily conform to the name conventions of the host languages.

In many of the DML statements, it is necessary to use parameters which identify a FORTRAN one dimensional array or a COBOL storage area. The values to be used by the Database Control System (DBCS) when processing the DML statement must be stored in the array or table prior to the execution of the DML CALL. It is important to note that each value which is to be passed to the DBCS in this way must start on a word boundary.

The form of a DML CALL statement in Fortran is as follows:

```
CALL SDML (param-1, param-2, .... )
```

In COBOL the form is CALL 'SDML' USING param-1, param-2,.....

A full description of the DML statement is given later in this chapter, with the Fortran form of the call indicated.

4.2 PARAMETER DESCRIPTIONS

To avoid repetition in defining the statements, the syntax of the most common parameters is defined here. Other parameters are described as "special parameters" under the special statements where they are used. This section should not be read alone, but along with the special statements.

When parameter names are passed through arrays or areas, it is important to note that there must be exactly eight characters in each name, left justified and with trailing blanks.

The general description of the parameters are given below. For examples: See 4.3.

The specific usage is defined in the various DML statements.

"mode"

"Mode" is a single integer which declares whether the run-unit wants to change the database or not.

"data-base-name"

"Data-base-name" defines a field or an array in the user area containing the eight character name of the database. This name must be identical to that defined in the Database Schema.

"password", "new-password"

"Password" and "new-password" define a field or an array in the user area containing the eight character passwords to be checked by the database control system.

"realm-name"

"Realm-name" defines a field or an array in the user area containing the eight character name of the relevant realm. This name must be identical to a realm name in the database schema.

"no.-of-realms"

"No.-of-realms" defines a single integer variable in the user area containing the number of realms to be readied in one READY REALM statement.

"key-name"

"Key-name" defines a field or an array in the user area containing the eight character name of an item or a group item defined in the data base schema as an index key or calc key for the relevant record type.

"key-value"

"Key-value" defines a field or an array in the user area containing or receiving the eight character value of an index key or a calc key.

"low-limit", "high-limit"

"Low-limit" and "high-limit" define fields or arrays in the user area containing lower and upper limit values of a corresponding index key. The length and type of "low limit" and "high limit" must be the same as that of the corresponding key.

"set-name"

"Set-name" defines a field or an array in the user area containing the eight character name of a set type defined in the database schema.

"temporary-data-base-key"

"Temporary-data-base-key" defines a single integer variable in the user area.
Using the value zero in this parameter means that the call (e.g., GET or MODIFY) will work on the current record (defined by the CRUI, see 2.4.1.2). If you want the call to work on a record *not* current anymore, you must have issued a REMEMBER when the record still was current. A number identifying the record would then have been stored in your "temporary-data-base-key"-variable. Using this number instead of zero in the call, will make the call work on that record instead of the record now being current. Note that the parameter is an output parameter only in case of REMEMBER, otherwise it is an input parameter.

"temporary-search-region-indicator"

"Temporary-search-region-indicator" defines a single integer variable in the user area.
The value zero in this parameter means that the current search region is to be used — as defined by the CSRI (see 2.4.1.2). In case you want to operate on a search region *not* current any longer, you must have issued a REMEMBER for that search region when it still was current. The identifying number then stored in your "temporary-search-region-indicator"-variable, must be used instead of the zero when you want this search region.

Note that the parameter is an output parameter only for REMEMBER, otherwise it is an input parameter.

"no.-of-items", "no.-wanted", "no.-found"

"No.-of-items" defines a single integer variable in the user area containing the number of item names that have been placed in "item-list". "No.-of-items" must have a value greater than or equal to one and less than or equal to the total number of items and group items in the record type. "No.-wanted" defines an integer value giving the number of records or keys the run-unit wishes to read, "no.-found" tells the run-unit how many records or keys it received.

"item-list"

"Item-list" defines a field or an array in the user area containing eight character names of data items or group items defined in the database schema for a record type.

"item-values"

"Item-values" defines a field or an array in the user area containing or receiving the values of the items and group items named in the "item-list" in corresponding order. Space must be allocated for each item corresponding to the data format definition in the database schema.

"option-code", "usage-mode", "protection-mode"

"Option-code", "usage-mode" and "protection-mode" define single integer variables whose values are used to specify certain options to be selected in various DML statements.

"key-length", "value-length"

"Key-length" and "value-length" are single integer variables defining the length of a field to be passed to SIBAS, expressed in number of words.

"status"

"Status" is an output parameter (single integer variable) which the DBCS sets to different values. The status value +1 indicates that the statement execution has been successful. The other values indicate an unsuccessful execution, implying a Database Exeception Condition (DBEC) in most cases (see Chapter 7).

Summary:

1:	successful
0:	normal exception condition such as end of search region
—1:	abnormal exception condition, more information is to be found by calling SDBEC
—2 to —6:	after SOPDB

Other negative values indicating error conditions may be returned to the run-unit, a list of which is given in the ERROR REPORTING chapter of this manual, but in those cases no more information may be found by calling SDBEC.

4.2.1 Open Database

Function:

The function of the OPEN-DATA-BASE statement is to indicate the run-unit's intention of processing the data in the database.

Format:

```
CALL SOPDB (mode, database name, password, status)
```

Rules:

A SIBAS process for this database must be running. This might be done by the SIBAS-service program before running your application program (see 6.2), or by including calls from section 5.4 in your program. If the SIBAS process is not number zero, a SETDV-call must be included before SOPDB, see 5.4.13.

The "mode" must define a variable or an array in the user storage area containing an integer; 0 if the run-unit will not change the database, 15473 if the run-unit intends to change the database.

The first run-unit which executes the OPEN-DATA-BASE statement will ready the SIBAS system realm. The user defined system realms will be readied when the relevant user realms are readied.

The effect of opening a database is to permit execution of READY statements on the realms on the database. If a database is not open, it's realms cannot be readied.

If privacy is defined for the database through the DBM-module (see 6.1.9), the "password" will be checked by the SIBAS DBCS to decide whether or not the user is allowed to open the data base.

The function of OPEN-DATA-BASE is essentially that of "logging in" to the particular database. The first run-unit to execute an OPEN-DATA-BASE on a closed database will cause it to be "physically" opened, and this run-unit also decides which database is opened.

When the last run-unit "logs off" with the CLOSE-DATA-BASE statement the database will be physically closed.

In case of unsuccessful open database, exception conditions cannot be set and SOPDB returns one of the following negative statuses:

- 1: illegal user identification (internal error)
- 2: unconsistent database name given
- 3: security breach occurred
- 4: one realm damaged
- 5: unable to RTOPEN database (check if user RT has write access to the database files)

— 6: SIBAS work area space is insufficient.
—72: Direct R-log is full, R-logging stopped. Illegal to open the database. DBA should reset or remove the R-log. This status will be returned from SOPDB if a direct R-log is filled.
—76: SIBAS is not active.

In case SIBAS is not running, your program will try continuously to open the database, i.e., your program will "hang". It will continue only if someone makes the SIBAS process run (through SIBAS-service or through the call SRUN from another application program).

4.2.2 Close Database

Function

The function of the CLOSE-DATA-BASE statement is to indicate that the run-unit has finished accessing the database.

Format:

CALL SCLDB (data-base-name, status)

Rules:

In order for CLOSE to be successful the database identified by "data-base-name" must have previously been opened by the run-unit.

The effect of closing a database is to prevent further execution of any DML statement other than OPEN-DATA-BASE from this run-unit, and to release allocated resources.

If realms in the database are still in ready status at the time the CLOSE is executed, then the realms are automatically finished for the run-unit.

A CLOSE, in a critical sequence, will automatically cause an ESEQU. (See 5.3.3.1.)

4.2.3 Ready Realm

Function:

The function of the READY-REALM statement is to indicate to the DBCS that the run-unit wishes to process records in one or more realms, to indicate the way in which the data will be processed, and to check possible interference with concurrently executing run-units.

Format:

CALL SRRLM (no.-of-realms, realm-names, usage-modes, protection-mode, status)

Rules:

"Realm-names" contains a list of names of the realms to be readied.

"Usage-modes" defines an array or table containing an integer value for each one of the realms to be readied. The following usage mode values apply:

Usage Mode:	Value:
RETRIEVAL	0
LOAD	1
UPDATE	2

"Protection-mode" defines an array or table containing an integer value for each one of the realms to be readied. The following protection modes/values apply:

Protection Mode:	Value:
NON-PROTECTION	0
EXCLUSIVE-UPDATE	1

Each realm in the list must be a part of the database which has been opened prior to execution of the READY-REALM statement. Each realm must not already be in ready status for the run-unit.

The effect of the READY-REALM statement is to make the records in the listed realms available for processing by other DML statements within the limitation set by the usage mode and protection mode.

The different "usage modes" given for each realm restricts execution of the DML statements on the records in the realm according to the following table:

Usage Mode:	Value:	DML Statements Allowed:
RETRIEVAL	0	FIND, GET, REMEMBER and FORGET
LOAD	1	FIND, GET, STORE, CONNECT, INSERT, REMEMBER and FORGET
UPDATE	2	ALL DML statements allowed

The different "protection modes" given for each realm are checked for possible conflict with other run units concurrently processing in the same realm according to the following table:

Protection Mode:	Value	Other Run-Units:
NON-PROTECTED	0	May execute any DML statement except ERASE.
EXCLUSIVE-UPDATE	1	May execute any DML statements

If a READY-REALM statement refers to more than one realm and any of the realms cannot be readied, the READY-REALM statement will not be sucessful, and none of the realms will be readied. All the realms will then remain unchanged but the status will indicate a DBEC condition about which information may be obtained by using the ACCEPT statement.

All realms to be readied for EXCLUSIVE-UPDATE for a run-unit must be readied in the same READY-REALM statement.

A realm cannot be readied for EXCLUSIVE-UPDATE if concurrent run-units have locked records in it.

If the user wants to change the USAGE MODE or PROTECTION MODE for a realm, then the realm must first be finished and readied again with the new USAGE MODE/PROTECTION MODE.

Resolution of Ready Conflicts:

Earlier entities by other run-units: Protection Mode	Earlier entities by other run-units: Usage Mode	Subject run-unit: Protection Mode	NON-PROTECTED			EXCLUSIVE UPDATE		
		Subject run-unit: Usage Mode	Retrieval	Load	Update	Retrieval	Load	Update
Non-Protected	Retrieval		Y	Y	Y	Y	Y	Y
	Load		Y	Y	Y	N	N	N
	Update		Y	Y	Y	N	N	N
Exclusive Update	Retrieval		Y	N	N	N	N	N
	Load		Y	N	N	N	N	N
	Update		Y	N	N	N	N	N

This table indicates how conflicts are resolved when the run-unit tries to ready a realm which has previously been successfully readied by some other concurrently executing run-unit but not yet finished. Y indicates that the run-unit is succussful, N indicates that the status indicator is set.

4.2.4 Finish Realm

Function:

The function of FINISH-REALM is to prevent further processing of the data in the realm by the run-unit.

Format:

CALL SFRLM (no.-of-realms, realm-names, status)

Rules:

"Realm-names" contains a list of names of the realms to be finished.

Realms readied for the run-unit with different usage moces may all be finished in one statement.

If a realm cannot be finished, the status will indicate an error and the name of the first offending realm may be found with the ACCEPT statement. If the FINISH-REALM statement involves more than one realm, those which can be finished will be finished.

If a FINISH-REALM statement is executed on a realm previously readied for EXCLUSIVE-UPDATE by the run-unit, the realm is then available for updating by other run-units.

When a FINISH-REALM is executed all remembered or locked records of this realm are forgotten or unlocked for this run-unit.

The effect of executing the FINISH-REALM statement is that the finished realms will not be available to the run-units until a new READY-REALM statement is executed. The contents of the SIBAS system buffers belonging to the finished realms will be written back to secondary storage. If the calling run-unit is the last one using a realm, the realm can be regarded as physically closed.

4.2.5 Direct Find

Function:

The function of DIRECT FIND is to locate a specific record. The record is specified by means of a calc key or an index key.

A search region will be established, its type depending on the statement format used.

Format:

Format 1:

```
FIND-USING-KEY
CALL SFTCH (realm-name, key-name, key-value, status, key-length)
```

Format 2:

```
FIND-FIRST-BETWEEN-LIMITS-USING-KEY
CALL SFEBL (realm-name, key-name, low-limit, high-limit, status,
key-length)

FIND-LAST-BETWEEN-LIMITS-USING-KEY
CALL SFLBL (realm-name, key-name, low-limit, high-limit, status,
key-length)
```

Format 3:

```
FIND-FIRST-IN-REALM
CALL SRFIR (realm-name, status)
```

Rules:

The realm named in "realm-name" must have been previously readied by the run-unit.

The "key-name" defines the name of an item or a group item which is defined as an index key or a calc key in the database schema.

The "key-value", "low-limit" and "high-limit" must have the same type and length as the corresponding item or group item. The "key-length" is expressed in number of words.

If format 2 is used, the "key-name" must identify an item or a group item which is defined as an index key in the database schema.

After successful execution of a FIND statement, the contents of the record may be processed by means of the GET, MODIFY, and ERASE statements.

After successful execution of the FIND statement, the current of run-unit indicator is set to a unique value identifying the record found.

After successful execution of a FIND statement the setting of the current search region indicator depends on the format used. In format 1, FIND-USING-KEY, where duplicate values of the key are allowed, the indicator will be set to both the key item name and the value of the key used. If duplicate keys are not allowed the setting of CSRI remains unchanged.

In format 2, FIND-BETWEEN-LIMITS, the current search region indicator will be set to the index item name and the value range between "low-limit" and "high-limit".

In format 3, FIND-FIRST-IN-REALM, the current search region indicator is set to the realm name.

After successful execution of a FIND statement the record selected depends on the format used.

In format 1, FIND-USING-KEY, if the key item is one for which duplicate values are allowed, then the DBCS selects the "first" record where the meaning of "first" is the record with the lowest physical address (i.e., storing nearest to the beginning of the realm).

In format 2, FIND-FIRST-BETWEEN-LIMITS, the record found is either one with key value equal to or next higher to the value of "low-limit" but the value must be lower than or equal to the "high-limit" value. If duplicate values are allowed the record found is the one with the lowest physical address.

In format 2, FIND-LAST-BETWEEN-LIMITS, the record found is either one with key value equal or next lower to the value of "high-limit" but the value must be lower than or equal to the "low-limit" value. If duplicate value is allowed, the record found is the one with the highest physical address.

To obtain the next or prior record within the range specified, the FIND-NEXT-IN-SEARCH-REGION or FIND-PRIOR-IN-SEARCH-REGION statements must be used.

In format 3, FIND-FIRST-IN-REALM, the DBCS attempts to find the physically first record in the realm. If location mode is calc this will be the first record in the first non-empty bucket. If location mode is SERIAL it will be the record in the realm with the lowest physical address. To obtain the next record of the realm the FIND-NEXT-IN-SEARCH-REGION statement must be used.

The table below gives a summary of the settings of CRUI and CSRI when a FIND from outside the database is executed.

	Format of FIND	CURRENT of RUN-UNIT INDICATOR	CURRENT SEARCH REGION INDICATOR
FIND successful	Format 1 (Duplicate not allowed)	set to uniquely identify the record with the given value of the key used.	not updated
	Format 1 (Duplicates allowed)	set to uniquely identify the "first" record with the given value of the key used.	set to key item name and value of key used
	Format 2	set to uniquely identify the "first" or "last" record within the given range	set to INDEX key item name, and the value range between low limit and high limit
	Format 3	set to uniquely identify the "first" record in the given realm	set to the realm name
FIND not successful	All formats	Not updated	Not updated

4.2.6 Relative Find

Function

The function of the RELATIVE FIND is to locate a record relative to some other record, and to make it available in the SIBAS buffer area.

The record is specified by means of a set or search region and a search type (NEXT, PRIOR, etc.)

Format:

Format 1:

FIND-FIRST-IN-SET
CALL SRFSM (temporary-data-base-key, set-name. status)

Format 2:

FIND-LAST-IN-SET
CALL SRLSM (temporary-data-base-key, set-name. status)

Format 3:

FIND-PRIOR-IN-SET
CALL SRPSM (temporary-data-base-key, set-name, status)

Format 4:

FIND-NEXT-IN-SET
CALL SRNSM (temporary-data-base-key, set-name, status)

Format 5:

FIND-NEXT-IN-SEARCH-REGION
CALL SRNIS (temporary-data-base-key, temporary-search-region-indicator, status)

FIND-PRIOR-IN-SEARCH-REGION
CALL SRPIS (temporary-data-base-key, temporary-search-region-indicator, status)

Rules:

The owner and all the member record types of any set type indicated by "set-name" must be known to the program and also be in realms which have been readied for use by the run unit.

"Temporary-data-base-key" identifies the record from which the new record is searched.

In the case of FIND-FIRST or FIND-LAST, the record identified by the "temporary-data-base-key" must be an owner of the set type named in "set-name". The record found will be one which is logically contiguous to the owner in the set occurrence. If the set occurrence is empty, the FIND will be unsuccessful and the "status" parameter is set to zero.

In the case of FIND-FIRST, the record found is that which would be found earliest by following LINK-TO-NEXT, i.e., the latest connected to the set occurence.

In the case of FIND-LAST, the record found is that which would be found earliest by following the LINK-TO-PRIOR, i.e., the earliest connected to the set occurrence. If there is no LINK-TO-PRIOR for the set type, then the same record is found but the execution is normally more time consuming as one must follow the LINK-TO-NEXT round the set occurrence. In a multi-member set type, the record found may be of any member record type.

In the case of FIND-NEXT or FIND-PRIOR, the record identified by "temporary-data-base-key" must be a member of the set type named in "set-name". The record found will be one which is logically contiguous to the identified member.

If this is the owner of the set occurrence, the FIND is unsuccessful and the "status" parameter is set to zero.

In the case of FIND-PRIOR, the record found is the member record which would be found first from the identified member using a LINK-TO-PRIOR. If there is no such link, the same record is found, but the execution is normally more time consuming.

In the case of FIND-NEXT, the record found is the member record which would be found first from the identified member using LINK-TO-NEXT.

In the case of FIND-NEXT/PRIOR-IN-SEARCH-REGION, the record identified by the "temporary-data-base-key" must be located in the search region identified by "temporary-search-region-indicator".

The meaning of this is explained in the following:

— When the search region is identified by the name and the value of an index or calc key item for which duplicates are allowed, the identified record must have the same value of the key item.

— When the search region is identified by a lower and an upper limit of an index key item, the identified record must have a value for the index key item which is within the given range.

— When the search region is identified by a realm name, the identified record must be located in that realm. (Not applicable for FIND-PRIOR-IN-SEARCH-REGION.)

FIND-PRIOR-IN-SEARCH-REGION is not applicable for a search region set to the realm name (by FIND-FIRST-IN-REALM).

In the case of FIND-NEXT-IN-SEARCH-REGION, the record found will be the one which is next in the search region to the record identified by "temporary-data-base-key".

In the case of FIND-PRIOR-IN-SEARCH-REGION, the record found will be the one which is prior in the search region to the record identified by "temporary-data-base-key".

The execution of FIND-NEXT/PRIOR-IN-SEARCH-REGION, will be unsuccessful and the "status" parameter set to zero if the record identified by "temporary-data-base-key" is the last record of the identified search region.

In the case of any successful FIND, the CRUI is always updated. The CSRI will not be updated by FIND of the type "RELATIVE-TO-RECENTLY-FOUND-RECORD".

4.2.7 Find Set Owner

Function

The function of this FIND statement is to find the owner of a set occurrence from one of its members.

Format:

CALL SRSOW (temporary-database-key, set name, status)

Rules:

The owner and all the member record types of the set type named by "set name" must be known to the program and must all be in realms which have been readied for use by the run-unit.

The effect of executing FIND OWNER is to find the owner of the set occurrence of "set name" from the member record identified by "temporary database key".

If the record identified by "temporary-database-key" is not connected into an occurrence of the named set type, the FIND will be unsuccessful, and the "status" parameter set to zero.

If the execution of FIND OWNER is successful the CRUI will be updated to identify the owner record. The CSRI will remain unchanged.

4.2.8 Get, Getn, Get Indexes

Function:

The function of the GET statement is to make the relevant items or group items available in the run-unit's data area so that the items can be processed. GETN reads a number of records in a search region. GET INDEXES reads a number of index keys.

In the case of GETN or GET INDEXES, the records can be obtained in ascending or descending order in the search region.

Format:

GET
CALL SGET (temporary-database-key, no. of items, item list, item values, status)

GETN
CALL SGETN (temporary-database-key, temporary search region indicator, no. wanted, no. of items, item list, item values, no. found, status)

GET-INDEXES
CALL SGIXN (temporary-database-key, temporary search region indicator, no. wanted, item values, no. found, status)

Rules:

"Item list" must be a list of names of items and group items in the user program. The corresponding values of the items and group items will be transferred to the area named "item values". Each value in the "item values" starts on a new word boundary. "Item values" cannot be larger than 500 words.

The "item list" should contain the names of the relevant items and group items in the record identified by "temporary-database-key". Not all items and group items defined for the record type need to be given in "item list" and the sequence of the items need not be the same as defined for the record type. The same item may be repeated in the "item list" but the total number of items given must not exceed the total number of items and group items defined for the record type.

The effect of executing a GET is to cause values of the items and group items named in the "item list" to be stored in the data area of the user program. In the case of GETN, the values corresponding to "no. found" records are transferred. In the case of GET INDEX, the values corresponding to "no. found" keys are transferred.

"No. wanted" can be positive or negative. If positive, records are found in ascending order, as when FIND-NEXT-IN-SEARCH-REGION is used. If negative, records are found in descending order, as when FIND-PRIOR-IN-SEARCH-REGION is used. The maximum "no wanted" for SGETN is 50.

The values of the items will be stored in the area named "item values" in the user program in an order corresponding to the order of the "item names". The CRUI and the CSRI will remain unchanged when a GET is executed.

For a GETN or GET-INDEXES, the CRUI is updated and points to the next record within the search region, as when using FIND-NEXT-IN-SEARCH-REGION / FIND-PRIOR-IN-SEARCH-REGION. If end/begin of the search region is encountered the CRUI points to the last/first record in the search region. "Status" is set to zero.

If the run-unit attempts to get a record which has been changed by another run-unit, and is in "extended mode" (see 2.4.3.4), the GET will be unsuccessful.

4.2.9 Modify

Function:

The function of MODIFY is to give new values to one or more of the items or group items in a record already existing in the database.

Format:

CALL SMDFY (temporary-database-key, no. of items, item list, item values, status, value length)

Rules:

"Item list" must be a list of names of items and group items given in the user program.

The corresponding values of the items and group items must be given in "item values". Each value must start on a new word boundary. "Value length" is the number of words the item values occupy.

The "item list" should contain the names of the relevant items and group items in the record identified by "temporary-database-key". Not all items and group items defined for the record type need to be given in "item list", and the sequence of the items may be chosen freely.

It is the user's responsibility that the sequence of the items in "item list" corresponds to the sequence of the values in "item values".

The realm in which the identified record is stored must have been readied for update. If the value of the member set item is being modified, realms indirectly referenced via set membership must have been readied for load or updated.

The effect of executing a MODIFY is to cause the values of the items named in "item list" to be stored in the record in the database identified by "temporary-database-key". Items not named in "item list" are not affected by the MODIFY.

If the record type of the identified record is a member in an automatically maintained set type and if the value of the member set item is modified, then the record will be disconnected from the set into which it was previously connected. If an occurrence of the owner record type of the set type has an owner set item value which is equal to the new value of the member set item in the modified record, the modified record is connected to the set owned by that record. If no such owner record is in the database and the storage class is manual for the set type, then the modified record is not connected to any occurrence of the set type. If the storage class for the set type is automatic and no owner record exists, then the MODIFY is unsuccessful.

If the identified record is an owner of a non-empty set occurrence and the owner set item is named in the "item list" then the execution is unsuccessful.

If any of the items modified are index or calc keys for the record type, then the new values must not be null and must not cause prohibited duplicates. The index is updated only if the index is automatically maintained.

If any of the items modified is a calc key for the record type, then the modified record is deleted from its previous position in the realm and stored in a position based on the new value of its calc key. The new value may not be null and may not cause prohibited duplicates.

If any of the items modified is a member of a group item which is defined as an index key, calc key, owner set item or member set item, then the same rules apply as if the item was itself defined as a key or a set item.

If any elementary item is named more than one time in the "item list" either directly or indirectly in a group item, the last value given in the "item list" will be the one stored for the item.

If a privacy item is defined for the record type and the run-unit has been allowed to update the record, the privacy item may also be updated.

The CRUI and the CSRI will remain unchanged when a MODIFY is executed.

If the record type of the identified record is a member of a set, and an error has occurred when executing MODIFY, the identified record may be displaced in the chain and placed such that it will be found by executing a FIND-FIRST-IN-SET statement.

4.2.10 Store

Function:

The function of the STORE statement is to store a record or a part of a record in its designated realm in the database, taking into account the location mode of the record type. The record stored may be connected into occurrences of automatic set types. Any indexes defined for the record type, which have been defined to be automatically maintained, are updated during the course of execution of the STORE.

Format:

CALL STORE (realm name, no. of items, item list, item values, status, value length)

Rules:

"Item list" must be a list of names of items and group items given in the user programs. The corresponding values of the items and group items must be given in the "item values". Each value must start on a new word boundary. "Value length" is expressed in number of words. The total length of all the parameters cannot exceed 500 words.

The "item list" should contain the names of the relevant items and group items in the records. Not all items and group items defined for the record type need to be given in "item list", and the sequence of the items may be chosen freely.

It is the user's responsibility that the sequence of the items in "item list" corresponds to the sequence of the values in "item values".

The realm in which records of this type are stored and also the realms containing owners and members of any automatic set type in which this record type is a member must have been readied for update or load by the run-unit.

The effect of executing a STORE is to cause the values of the items and group items named in "item list" to be stored in the realm named in "realm name".

The location mode of "realm name" determines where and how the record is stored in the realm. If the location mode is calc, the calc key item must be given in the "item list" and the value must be non-null. The given calc value will be transformed into a bucket number. The record will then be stored in the first available space in the bucket or in an overflow bucket. If the location mode is serial the record will be stored in the first available space in the realm.

Not all items defined for the record type need to be given values when a STORE is executed. The items in the record type which are not named in the "item list" will be given a null value in that record occurrence. The items can be given values later by use of MODIFY.

It should be noted that a calc key item must always be given a non-null value. If location mode is serial and automatically maintained index key(s) are defined for the record type, and/or the record type is a member of an automatic set, at least one of the index keys or member set items must be given a non-null value.

When a record is stored, it will be connected into occurrences of automatic set types and inserted into indexes which are automatically maintained provided that the item or group item defined as member set item or index key is named in "item list". If an index key/member set item is a group item, at least one of the items composing the group item must be named in the "item list".

If this condition is not satisfied the record may be connected into the set(s) or inserted into the index(es) later by executing MODIFY on the relevant item(s).

The record is connected into the set occurrence(s) such that it is found by executing FIND-FIRST from the owner record.

If the record type is a member of an automatic set type, and the member set item is named in "item list", the owner record must be present when the member record is stored. If the owner record is not present, the execution of the STORE will be unsuccessful.

If a key item (calc or index key) were defined as not allowing duplicates and if storing the record in the database would violate this, then the store will be unsuccessful. When a privacy item is defined for a record type, it must be given a non-null value when a record of this type is stored.

If the store is executed successfully, then the CRUI is set to identify the stored record. CSRI is not affected.

4.2.11 Erase

Function:

The function of the ERASE statement is to remove the record and all references to it from the database.

Format:

CALL SRASE (temporary-database-key, option code, status)

Rules:

The "temporary-database-key" identifies the record that is to be erased.

The realm in which the record identified by "temporary-database-key" is stored must have been readied with usage mode of UPDATE. In the multi-user version of SIBAS, if "option code" greater than or equal to 1 is used, all indirectly and directly referenced realms must also have been readied with a protection mode of EXCLUSIVE UPDATE by the run-unit.

The "option code" can have the values 0, 1, 2 or 3 specifying the various ERASE options:

0 The record identified by the "temporary-database-key" will be erased from the database as long as it is not an owner record with connected set members. If it is, the ERASE will not be successful.

1 The record identified by the "temporary-database-key" will be erased from the database if no records are connected to the identified record as members of an automatic set type. If this is the case the ERASE will not be successful. If any records are connected to the identified record as members of a manual set type, the records will be disconnected from the identified record.

2 The record identified by the "temporary-database-key" will be erased from the database. If any records are connected to the identified record as members of a manual set type, the records will be disconnected from the identified record. If any records are connected to the identified record as members of an automatic set type, these records are also erased. If any of these records are owners of non-empty sets, then the same rules are used for these as for the record identified by the "temporary-database-key".

3 The record identified by the "temporary-database-key" will be erased from the database. If it is the owner of any non-empty sets (manual or automatic), then all member records in these set occurrences are also erased. If any of these records are themselves owners of other non-empty sets, then their connected members are also erased. This process continues down the hierarchical structure. The maximum number of levels is 16.

If an erased record has one or more index keys, the indexes will be updated whether they are defined as automatically maintained or not.

If an erased record is a member of one or more sets, the record will be removed from the set occurrences, and the links to the adjacent members will be updated.

After execution of ERASE, the erased record and it's associated records, if any, are no longer available for processing. All CRUI's and "temporary-database-keys" referring to the erased record(s) are forgotten in the single user version or marked as "erased" in a multi-user environment.

4.2.12 Connect

Function:

The function of the CONNECT statement is to link a record already stored in the database into a manually maintained set of which its record type is defined as a member.

Format:

CONNECT:
CALL SCONN (temporary-database-key-1, set name, status)

CONNECT-BEFORE:
CALL SCONB (temporary-database-key-1, temporary-database-key-2, set name, status)

CONNECT-AFTER:
CALL SCONA (temporary-database-key-1, temporary-database-key-2, set name, status)

Rules:

The set type identified by "set name" must have been defined as manually maintained in the database schema.

The owner record type and the member record type(s) of the set type "set name" must be in realms which have been readied for load or update by the run-unit.

In the case of CONNECT, the record identified by "temporary-database-key-1" will be connected into the set occurrence whose owner set item value is equal to the member set item value of the identified record. The identified record will be connected into the set occurrence such that it is found by executing FIND-FIRST from the owner of the set occurrence. The record must not already be connected to the set occurrence.

If the BEFORE or the AFTER option is used, "temporary-database-key-1" must identify a record which is not connected into a set occurrence of the set type identified by "set name" and "temporary-database-key-2" must identify a record which was previously connected into a set occurrence of the set type identified by "set name". Furthermore, the value of the member set item for the set type must be the same for the two records.

If the BEFORE option is used, the record identified by 'temporary-database-key 1" will be connected to the correct set occurrence of the set type identified by "set name". It is connected so that the record identified by "temporary-data base-key-2" is found executing FIND-NEXT relative to "temporary-database-key-1".

If the AFTER option is used, the record identified by "temporary database key 1" will be connected to the correct set occurrence of the set type identified by "set name". It is connected so that the record identified by ' temporary data base key 2" is found by executing FIND-PRIOR relative to "temporary database key 1". It should be noted that if the set type has link to next only, the CONNECT-AFTER can be very time consuming.

The CRUI and CSRI will remain unchanged when a CONNECT is executed.

4.2.13 Disconnect

Function:

The function of the DISCONNECT statement is to delink a record in the database from a manually maintained set into which it has previously been connected.

Format:

CALL SDCON (temporary-database-key, set name, status)

Rules:

The set type identified by "set name" must have been specified as a manual set type in the database schema. The realms containing the owner records and the occurrences of other member record types must be in realms which have been readied for update by the run-unit.

The effect of executing a DISCONNECT is to delink the identified record from the occurrence of the set in which it has previously been connected. The identified record remains in the database and it remains connected into sets of other set types into which it was connected before the DISCONNECT was executed. The two records which were logically contiguous to the identified record in the set before the DISCONNECT is executed are logically contiguous to each other after execution. If the record was not in fact connected into any occurrence of the set, the DISCONNECT is unsuccessful and "status" is set to zero.

The CRUI and CSRI will remain unchanged when a DISCONNECT is executed.

4.2.14 Insert

Function:

The function of the INSERT statement is to insert an index key of a record already stored in the database into a manually maintained index.

Format:

CALL SINSR (temporary-database-key, key name, status)

Rules:

"Key name" must identify the name of an item or group item defined as a manually maintained INDEX key for the record type in the database schema.

The item or group item named "key name" in the identified record must have been given a non-null value prior to the execution of INSERT. It must not previously have been inserted into the index.

The effect of executing INSERT is to update the index with the value of the item or group item named "key name", so that the record may be accessed by use of the "key name".

If duplicates are not allowed for the index key item, an attempt to INSERT a duplicate value will cause a database exception condition to occur.

The CRUI and CSRI will remain unchanged when INSERT is executed.

4.2.15 Remove

Function:

The function of the REMOVE statement is to remove an index key from a manually maintained index.

Format:

CALL SREMO (temporary-database-key, key name, status)

Rules:

The record must be in a realm which has been readied for update by the run-unit. "Key name" must identify the name of an item or group item defined as a manually maintained index key for the record type in the database schema.

The identified record must previously have been inserted into the index.

The effect of executing a REMOVE is to take out the entry from the index table, so that the "key name" cannot be used as an access key to the record identified by "temporary-database-key".

The CRUI and CSRI remain unchanged when a REMOVE is executed.

4.2.16 Remember

Function:

The function of the REMEMBER statement is to remember either the identification of the record contained in the CRUI or else to remember the search region which is contained in the CSRI. A remembered record or search region can be referenced directly in all DML statements as an alternative to the CRUI or CSRI.

Format:

CALL SREMB (temporary id, option code, status)

Rules:

"Option code" must be given one of the values 0 or 1. If "option code" is set to 0, the REMEMBER-RECORD is executed and "temporary id" will identify "temporary-database-key". If "option code" is set to 1, then REMEMBER-SEARCH-REGION is executed and "temporary id" will identify a "temporary search region indicator". All other settings of "option code" are prohibited.

The effect of executing REMEMBER-RECORD is to make the record identified by CRUI available to the run-unit after the CRUI has been updated. This record is later referenced by use of the number received by the REMEMBER statement in the variable "temporary id".

The effect of executing REMEMBER-SEARCH-REGION is to make the search region identified by CSRI available to the run-unit after the CSRI has been updated. This region is later referenced by use of the number received in the variable "temporary id".

A REMEMBER is local to the run-unit. Two concurrently processing run-units may remember the same record or the same search region without conflict.

A REMEMBER lasts only for the duration of a run-unit. After closing the data base, anything which has been remembered for a run-unit is automatically forgotten.

The number of times a REMEMBER may be executed in a run-unit without executing a FORGET depends on the variant of SIBAS in use. It is, however, recommended that each REMEMBER is matched with a FORGET as soon as what has been remembered is of no use to the run-unit. This is because the FORGET statement releases entries in the remembered list for further use by the REMEMBER statement.

The CRUI and the CSRI will remain unchanged when the REMEMBER is executed.

The maximum number of remembered record by run-unit is 30 while the maximum number of remembered search-region for one run-unit is 5.

4.2.17 Forget

Function:

The function of the FORGET statement is to nullify the effect of executing a REMEMBER.

Format:

CALL SFORG (temporary id, option code, status)

Rules:

"Option code" must be set to an integer between 0 and 3 and the meaning of the "temporary id" will depend on the setting of "option code".

The meaning of the possible values of "option code" is explained below:

"option code"	FORGET option executed	meaning of "temporary id"
0	FORGET-RECORD	"temporary-database-key"
1	FORGET-SEARCH-REGION	"temporary search region indicator"
2	FORGET-ALL-RECORDS	undefined
3	FORGET-ALL-SEARCH-REGIONS	undefined

FORGET-RECORD causes the record identified by "temporary-database-key" to be deleted from the list of the remembered records for the run-unit.

FORGET-ALL-RECORDS causes all records previously remembered by the run-unit to be deleted from the remembered list.

FORGET-SEARCH-REGION causes the search region identified by "temporary search region indicator" to be deleted from the list of remembered search regions for the run-unit.

FORGET-ALL-SEARCH-REGIONS causes all search regions previously remembered by the run-unit to be deleted from the remembered list.

The number of times a REMEMBER may be executed in a run-unit without executing a FORGET depends on the variant of SIBAS in use. It is, however, recommended that each REMEMBER is matched by a FORGET as soon as what has been remembered is of no use to the run-unit. This is because the FORGET statement releases entries in the remembered list for further use by the REMEMBER statement.

The CRUI and the CSRI will remain unchanged when a FORGET is executed.

If the records specified in FORGET were locked, they are automatically unlocked by a successful execution of the FORGET statement.

4.2.18 Lock

Function:

The function of the LOCK statement is to indicate to the DBCS that the run-unit wishes to obtain one or all of its remembered records (those in extended monitor mode) for exclusive update.

Format:

CALL SLOCK (temporary-database-key, option code, status)

Rules:

The "option code" must have one of the two values 0 or 1, where:

0: lock record identified by "temporary-database-key"
1: lock all the records in the run-units remembered list.

The value of "temporary-database-key" needs only to be defined for "option code" value 0.

The effect of the LOCK statement is to cause one or all records in one run-unit's remembered list to be set in the status of EXCLUSIVE UPDATE for the run-unit including the record identified by the CRUI.

The LOCK statement will only be successful as long as none of the required records are already locked to another run-unit, and none of the records are in realms readied for EXCLUSIVE UPDATE by another run-unit.

If the run-unit has previously executed a LOCK statement, an UNLOCK statement must be executed prior to the execution of a new LOCK. This restriction avoids the problem of deadlock between records in SIBAS.

After the successful execution of a LOCK statement the status will be set to either 0 or 1. In the case of status 0, one or more of the locked records have been affected by another run-unit. By "affected" is meant that one of the following statements has been executed on the record: ERASE, MODIFY, CONNECT, DISCONNECT, INSERT or REMOVE. In the case of status 1, none of the locked records have been affected after they were set in extended monitor mode by the run-unit.

Locked records can be released for updating by other run-units after execution of an UNLOCK, FORGET-ALL or a CLOSE-DATA-BASE statement. FORGET and FORGET-ALL do not unlock the CRUI.

The CRUI and the CSRI will remain unchanged when a LOCK statement is executed.

4.2.19 Unlock

Function:

The function of the UNLOCK statement is to make any records that are locked to the calling run-unit available for updating by concurrent run-units.

Format:

```
CALL SUNLK (status)
```

Rules:

The UNLOCK statement is always successfully executed.

4.2.20 Change-Password

Function:

The function of the CHANGE-PASSWORD statement is to change the value of the current password for the calling run-unit, to conform with the password of a record to be looked at.

Format:

```
CALL SCHPW (new password, status)
```

Rules:

The current password will be set to a value for each run-unit when OPEN-DATA-BASE is executed. The effect of executing CHANGE-PASSWORD is to change the value of the current password for the calling run-unit. The use of current password is described in Chapter 2. See also Section 6.1.9.

4.2.21 Accept

Function:

The function of the ACCEPT statement is to move to user defined areas the contents of various system registers set when a database exception condition occurs.

Format:

CALL SDBEC (set name, realm name 1, realm name 2, item name, DML statement code, dbec)

Rules:

"Set name", "realm name 1", "realm name 2" and "item name" must be defined in the host language program to correspond to a SIBAS character item which will hold eight characters.

"DML statement code" and "dbec" must both be defined in the host language program to correspond to an integer item which could hold at least four digits.

The ACCEPT statement will always be successful. The most recently executed DML statement will set the system registers to the values which are obtained by the ACCEPT statement.

Before the OPEN-DATA-BASE statement is executed by the run-unit, the system registers will have a null value.

The effect of executing the ACCEPT statement is to move the contents of various system registers into the user defined parameters. The setting of the parameters will be as follows:

- "Set name" will be set to the name of the set type referenced in the most recently executed DML statement. If no set is referred to, "set name" will be set to null value.

- "Realm name 1" will be set to the name of the realm referenced in the most recently executed DML statement. If no realm is referred to, "realm name 1" will be set to null value.

- "Realm name 2" will be set to the name of the realm which caused the DBEC if this is different from "realm name 1". If not, "realm name 2" will be set to null value.

- "Item name" will be set to the name of the item or group item which caused the most recently executed DML statement. If no item is referred to, "item name" will be set to null value.

— "DML statement code" will be set to the code for the most recently executed DML statement. The codes for all DML statements are listed in Appendix E.

— "DBEC" will be set to the code of the DBEC. If the DML statement was successfully executed, "dbec" will be set to null value.

The table containing all possible values for DBEC and DML statement codes is given in the chapter "Error Reporting".

4.2.22 Erase Element

Function:

The function of ERASE-ELEMENT is to give null values for one or more items or group items in a record already existing in the database.

Format:

CALL SEREL (temporary-database-key, no. of items, item names, status)

Rules:

"Item list" must be a list of names of items and group items given in the user program.

The "item list" should contain the names of the relevant items and group items in the record identified by "temporary-database-key". The sequence of the items may be chosen freely.

The realm in which the identified record is stored must have been readied for update. Realms indirectly referenced via set membership must have been readied for load or update.

The effect of executing an ERASE-ELEMENT is to cause the values of the items named in "item list" to be modified to null values in the record identified by "temporary-database-key". Items not named in "item list" will only be affected by the ERASE-ELEMENT if they are members of a group item and the group item is named in the "item list". If a group item is named in the "item list", all member item values of that group item will be erased for this record occurrence.

If the record type of the identified record is a member of a set and if the value of the member set item is erased, then the record will be disconnected from the set into which it was previously connected.

If the identified record is an owner of a non-empty set occurrence and the owner set item is named in the "item list" or is a group item which has been modified to null, then the execution is unsuccessful.

If any of the items modified to null are index keys for the record type, then the corresponding entry is removed from the index.

If any of the items modified to null is a calc key for the record type, then the execution is unsuccessful.

If all key items and member set items which exist for the record are modified to null, then the execution is unsuccessful.

If any item is named more than once in the "item list", this has no effect.

If a privacy item is defined for the record type and the run-unit has been allowed to modify the record, the privacy item may also be set to a null value.

The CRUI and the CSRI will remain unchanged when an ERASE-ELEMENT is executed.

4.2.23 Accumulate

Function:

Accumulates integer or floating or double integer data elements for one or more items in an already found record. It is a GET followed by a MODIFY in only one statement. These statements reduce the possibility of interference between concurrent run-units.

Format:

```
CALL ACCID/ACCFD/ACCDD (temporary-database-key, no. of items,
item list, increments, new values, status)
```

Rules:

The record identified by "temporary-database-key" must be in a realm which has been readied for update by the run-unit. "Increments" are the values to be added to the items named in the "item list". The "new values" are the values returned by the call.

The item names in "item list" must be elementary items, i.e. not groups.

ACCFD is not available from SIBAS-500.

4.2.24 Fetch-Get

Function:

The function of FETCH-GET is to retrieve a specific record. The record is specified by means of a calc key or an index key.

A search region will be established if the key has duplicates allowed.

Format:

CALL SFTGT (realm-name, key-name, length of key, key value, number of items, item list, item values, status)

Rules:

Same as if the following was executed:

call SFTCH
if sftch-status = 1 then call SGET endif

The realm named in "realm-name" must have been previously readied by the run-unit.

The "key-name" defines the name of an item or a group item which is defined as an index key or a calc key in the database schema.

After successful execution of the FIND statement, the current of run-unit indicator is set to a unique value identifying the record found.

After successful execution of a FIND statement, the current search region indicator will be set to both the key item name and the value of the key used. If duplicate keys are not allowed the setting of CSRI remains unchanged.

If the key item is one for which duplicate values are allowed, then the DBCS selects the "first" record where the meaning of "first" is the record with the lowest physical address (i.e., storing nearest to the beginning of the realm.

The values of the items will be stored in the area named "item values" in the user program in an order corresponding to the order of the "item names". The CRUI and the CSRI will remain unchanged when a GET is executed.

"Item list" must be a list of names of items and group items in the user program. The corresponding values of the items and group items will be transferred to the area named "item values". Each value in the "item values" starts on a new word boundary. "Items values" cannot be larger than 500 words.

The "item list" should contain the names of the relevant items and group items in the record identified by "temporary-database-key". Not all items and group items defined for the record type need to be given in "item list" and the sequence of the items need not to be the same as defined for the record type. The same item may be repeated in the "item list" but the total number of items given must not exceed the total number of items and group items defined for the record type.

The effect of executing a GET is to cause values of the items and group items named in the "item list" to be stored in the data area of the user program.

If the run-unit attempts to get a record which has been changed by another run-unit, and is in "extended mode" (see 2.4.3.4), the GET will be unsuccessful.

4.2.25 Get Schemas Information

Function:

The function of GET-SCHEMAS-INFORMATION is to get information about realms, records and items from the database schemas at run-time.

Format:

CALL SINFO (code, name1, name2, length, array, status)

Rules:

Realm(s) must be in ready mode.

Input: code	Output: array
1. Get realm names in database.	1-4 first realm name 5-8 next realm name etc.
2. get realm description and free-space statistics	1 realm type 2 pagesize 3 record-length (type 2), pagesize (type 1) 4 pages reserved 5 pages used 6 freed records (type 2,3) freed pages (type 1)
3. Get record description for realm	4 words per item name

4. Get item or group description

word 1 item type subfield

Bit counting from right to left:

bit 0-1	00 integer, 01 floating 10 character, 11 mixed
bit 2 = 1	access via calc
bit 3 = 1	access via index
bit 4 = 1	member of set
bit 5 = 1	unique access key 0 - duplicates allowed 1 - duplicates not allowed
bit 6 = 1	automatic member of set
bit 7 = 1	automatic acces key
bit 8 = 1	access-lock
bit 9 = 1	owner of set
bit 10 = 1	group item
bit 11 = 1	member of a group
bit 12 = 1	pointer item
bit 13 = 1	pointer is defined to be double
bit 14 = 1	pointer is owner of chain
word 2	word start of item in record
word 3	length of item bit 12 = 0 → bit 0-11 = length bit 12 = 1 → bit 0-5 = bit start
word 4	offset to group description (0 = if item not group)
word 5	bit 12 of word 3 (1 = if item length is part of a word)
word 6	bit 1 of word 1 (1 = if character item)
	if group item then
word 7-10	first item name
word 11-14	next item name

5. get access path to realm

1	calc access, yes = 1, no = 0
2	number of index-accesses
3	number of set-accesses
	CALC ACCESS
4-7	item-name
8	duplicates (0) or not (1)
9	automatic (1) or not (0)
	INDEX ACCESS
10-13	first item name
14	duplicates (0) or not (1)
15	automatic (1) or not (0)
16-19	next item name etc.
	SET ACCESS
n- n+3	first set name
+1	member (0)/owner (1) of the set
+2	automatic (1) or not (0)
+3- +6	next set name etc.

code	1	2	3	4	5
name1	—	realm name	realm name	realm name	realm name
name2	—	—	—	item name	—

OUTPUT

"length" is length of array
"array" is a user defined integer array of maximum 500 elements.
"status" is set to -1 if errors occurs.

4.2.26 Transaction Units

Function:

A TRANSACTION UNIT is a sequence of database processing which brings the database from one user consistent state to another user consistent state. A transaction unit generally corresponds to the completion of a unit of work significant to the user.

SIBAS takes into account transaction units by explicit declaration from a user program which determines the "scope" of a transaction unit at run-time. SIBAS imposes severe restrictions on the "scope" of a transaction unit. However, the restrictions imposed on the user program make it possible to implement the TRANSACTION UNIT efficiently in SIBAS.

SUBEG declares to SIBAS the RUN-UNIT intention to process a unit of work which must be either completely executed or not executed at all. SIBAS will reserve the whole DATABASE for exclusive use for the duration of the transaction unit.

SUEND declares to SIBAS the RUN-UNIT completion of a transaction unit. The completion can be normal, or not. In the later case, the database will be restored to the state it was at SUBEG.

Format:

CALL SUBEG (run-id, t-unit type, status)
CALL SUEND (run-id, COMIT or ROLL-BACK, status)

Rules:

1. The BIM option must be in effect, otherwise an error status is given.

2. Normally, <run-id> should be left = 0, but a monitoring program can also execute SUBEG/SUEND for other run-units.

3. A transaction unit cannot last more than a certain elapsed time. This is to prevent a looping or waiting T.U. to hang up concurrent run-units. The maximum elapsed time a T.U. can last is a system generation parameter, normally 20 seconds.

4. <t-unit type> = 1 the database is reserved for exclusive read.
   = 2 the database is reserved for exclusive update.

5. <COMIT or ROLL> = 1 COMMIT, all changes are applied to the database.
   = -1 ROLL, all changes are discarded, and the database is left as it was when the transaction unit started.

6. If a transaction unit is not terminated within the specified time, SIBAS will automatically execute a SUEND (ROLL,) for the run unit.

   If a run-unit has been ROLLed back by either SIBAS or a monitoring program, it will get a negative status for the next call. All currency indicators are cleared as if FORGET-ALL-RECORDS and FORGET-ALL-SEARCH-REGIONS were executed.

7. OPEN-DATA-BASE, CHECKPOINT, ROLL-BACK, READY/FINISH-REALM, BSEQU/ESEQU, RESIB/RELSI, CLOSE-DATA-BASE are not allowed within the scope of a transaction unit, and will give an error status.

8. If application programs can be written so that SUBEG/SUEND brackets all database updates, concurrency problems are almost eliminated, i.e., LOCK/UNLOCK, notification of changes, BSEQU, ESEQU ... are unnecessary.

4.3 HOST LANGUAGE CONSIDERATIONS

SIBAS data manipulation services are generally accessed via calls. The reason is that calling subroutines is a fairly standard and formalized way of interfacing programs written in different programming languages. SIBAS adheres to the FORTRAN call formalism.

SIBAS-500

SIBAS-500 data manipulation language (DML) is generally accessed via calls just like SIBAS-100 DML. Programmers should write SIBAS application programs in a standardized way, independent of whether they are to be run on an ND-100 (using SIBAS-100) or an ND-500 system (using SIBAS-100 and/or SIBAS-500). In this chapter we present such a standardized way to construct SIBAS applications. Some special rules concerning SIBAS-500 are presented, but *programmers following the given standard "cookbooks" will be able to run their applications on both SIBAS-100 and SIBAS-500 without any modifications of their source code.* The rules given below may at first seem very complex, but experience shows that it is very easy to convert existing SIBAS applications from ND-10/100.

4.3.1 FORTRAN

Calling SIBAS subroutines from a FORTRAN program is just the same as calling any other FORTRAN subroutine, as shown in the example.

It is not possible to use character parameters directly. However, this restriction can be bypassed by "EQUIVALENCing" character fields to integer arrays.

FORTRAN ON THE SIBAS-500

Calling SIBAS subroutines from a Fortran-500 program is exactly the same process as calling any other Fortran subroutine, and hence the same as calling SIBAS from a Fortran-100 program. There are ,however, some restrictions as to how SIBAS *value buffers* are to be declared in a Fortran-500 application, i.e., a Fortran application running on the 500 CPU.

4.3.1.1 General Rules for Fortran on the SIBAS-500

The default integer size on the ND-500 CPU is 32 bits, as opposed to the default integer size of 16 bits on the ND-10/100 CPU. This is because on the ND-10/100 CPU one word is 16 bits, and on the ND-500 CPU one word is 32 bits. The SIBAS-500 simulator (SIBAS-LIBRARY) assumes that *applications are compiled in the default integer mode* and takes care of converting single integer parameters to/from SIBAS-mode (i.e., 16 bit integer format) before receiving/sending parameters from/to SIBAS. In addition, the database format is a 16 bit integer format, i.e., all value buffers (containing database values) sent to/from a SIBAS process must be packed in a 16 bit word format. To avoid problems concerning different integer modes it is best to simply declare all value buffers (passed to/from SIBAS) as INTEGER*2 in all Fortran applications. INTEGER*2 (which specifies a 16 bit integer format) is the default on the ND-10/100 CPU.

(Note that 500-applications *must* follow the given rule.) All other parameters are declared (default) integer (i.e., INTEGER, hence 32 bits in Fortran-500) and can be handled in the same way as in a Fortran-100 SIBAS application.

4.3.1.2 Standard 'Cookbook' for Programming Fortran Applications

In this manual the value buffers are:

"key-value"
"item-values"
"low-limit"
"high-limit"
"increments"
"new values"

They are used in the following SIBAS DML-calls:

SFTCH(P1, P2, "key-value', P4, P5)
SGET(P1, P2, P3, "item-values", P5)
STORE(P1, P2, P3, "item-values", P5, P6)
SMDFY(P1, P2, P3, "item-values', P5, P6)
SFEBL(P1, P2, "low-limit", "high-limit", P5, P6)
SFLBL(P1, P2, "low-limit", "high-limit", P5, P6)
SGETN(P1, P2, P3, P4, P5, P6, "item-values", P8, P9)
SGIXN(P1, P2, P3, "item-values", P5, P6)
ACCID/DD(P1, P2, P3, "increments", "new values", P6)

Declare all such value buffers to be INTEGER*2, all other parameters are declared INTEGER. In addition *this is also the only difference when applications are to be converted from ND-10/100.*

Example:

ND-100	ND-500
INTEGER ITEMP,NOITM,ISTAT	INTEGER ITEMP,NOITM,ISTAT
INTEGER IVBUF(10)	INTEGER*2 IVBUF(10)

CALL SGET(ITEMP,NOITM,"REALMXX ",IVBUF,ISTAT)

As we see, only the *declaration* of IVBUF is different, — the call sequence and all other declarations are identical. Note that the 'ND-500 solution' is the best way of construction all SIBAS Fortran applications since this solution also can be run on the ND-10/100 without any modifications.

SPECIAL CONSIDERATIONS:

1. Since the default single integer mode is 32 bits in Fortran-500, integer constants cannot be used as value buffers when such a buffer consists of only one single SIBAS-word (i.e.,length-of-value-buffer is 1).

   Example:

   ```
   CALL SFTCH( P1, P2,1999, P4, P5)
   ```

   This construction cannot be used to fetch a specific record (where P2 is the key item declared integer in the database definition with a length of 1 SIBAS-word). (A SIBAS-word = 16 bits.)

   Instead the solution shown below should be used. (The use of an array is not really necessary here.)

   ```
   INTEGER*2 IVBUF(n)
   IVBUF(1) = 1999
   CALL SFTCH( P1, P2, IVBUF , P4, P5
   ```

2. Names of the database, realms, items, etc., can be handled as they are handled in a Fortran-100 application.

   For example:

   (Note that *both* of the solutions shown can be used.)

   a. including names directly as the actual parameters surrounded by double quotes.

      Ex.:

      ```
      CALL SOPDB(15473,''TESTBAS '','GXZZXG  ',IST)
      ```

   b. 'equivalencing' CHARACTER variables (containing names) with the actual INTEGER parameters.
      To avoid a waste of space we recommend using DOUBLE INTEGER because double integer is 32 bits on both the ND-10/100 and the ND-500 CPU.

      Ex:

      ```
      CHARACTER DBNAM*8, PASSW*8
      DOUBLE INTEGER IBASE(2),IPASS(2)
      EQUIVALENCE (IBASE,DBNAM), (IPASS,PASSW)
      DBNAM = 'TESTBAS '
      PASSW = 'GXZZXG  '
      CALL SOPDB(15473,IBASE,IPASS,IST)
      ```

Remarks:

i) Instead of declaring IBASE and IPASS as double integer, they could just as well have been declared as:

```
INTEGER IBASE(4), IPASS(4)
```

In Fortran-500 applications it would be sufficient to use 2 as dimension, but this would make it impossible to run the application on an ND-10/100 CPU.

ii) A constant (15473) can be used as the first parameter because it is not a value buffer.

3. The application programmer is advised to be very careful when local *INTEGER* variables in the application program are 'equivalenced' with value buffers which are declared INTEGER*2.

```
            PROGRAM LOADPER
C
C  PURPOSE
C            LOAD PERSON RECORDS IN A COURSE DATABASE
C
C
C            DECLARE BUFFERS INTEGER*2 FOR THIS TO BE
C            COMPATIBLE WITH SIBAS-500
               INTEGER*2  IUBUF(45)
               INTEGER*2  ITVAL1(35),ITVAL2(9),HSALARY
            CHARACTER ITLIST(8)*8
            EQUIVALENCE(IUBUF(1),ITVAL1),(IUBUF(36),HSALARY),
     +                  (IUBUF(37),ITVAL2),(ITLIST,INAME)
            DATA ITLIST(1)/'BDATE   '/
            DATA ITLIST(2)/'BNO     '/
            DATA ITLIST(3)/'PERSNAME'/
            DATA ITLIST(4)/'PERSADDR'/
            DATA ITLIST(5)/'SEX     '/
            DATA ITLIST(6)/'HSALARY '/
            DATA ITLIST(7)/'DEPARTM '/
            DATA ITLIST(8)/'PRESERV '/


            WRITE(1,100)
100         FORMAT(1H1,5X,'PROGRAM FPERL-GB',
     +                    /,6X,'NEW PERSON-RECORDS TO DATABASE',/)
C
C OPEN-DATABASE:
C
            CALL SOPDB(15473,"GENDB-GB",IPASS,IST)
            IF (IST.LT.0)   THEN
                CALL ERROR(1)
                GO TO 180
            ENDIF
C
C READY  REALM FOR  LOAD , NOTE THE USE OF " TO DELIMIT MOLLER. CONSTANTE
C
            CALL SRRLM(1,"PERSON  ",1,0,IST)
            IF (IST.LT.1) THEN
                CALL ERROR(2)
                GO TO 180
            ENDIF
C
C  OPEN THE INPUT-FILE FOR READING:
C
            OPEN (UNIT=20,FILE='GPERREC:DATA',
     +          STATUS='OLD',ACCESS='R',RECL=45)
            IF (ERRCODE.NE.0) THEN
                CALL ERROR(3)
                GO TO 175
            ENDIF
C
C  LOOP HERE FOR EACH RECORD
C
125         READ (20,130,END=160)  ITVAL1,HSALARY,ITVAL2
130         FORMAT(3A2,2A2,A1,13A2,15A2,A1,I6,3A2,6A2)
            IF (ERRCODE.NE.0) THEN
                CALL ERROR(4)
                GO  TO 125
            ENDIF
            WRITE(1,140)ITVAL1,HSALARY,ITVAL2
```

```
140       FORMAT(1H ,3A2,1X,2A2,A1,1X,13A2,1X,15A2,1X,A1,1X,I6,1X,
     +    3A2,1X,6A2)
C
C  STORE THE PERSON-RECORD READ:
C
          CALL STORE("PERSON  ",8,INAME,IUBUF,IST,45)
          IF (IST.NE.1) CALL ERROR(5)
          GO TO 125
C
C  END OF LOOP
C
160       WRITE(1,170)
170       FORMAT(1H0,6X,*LOADING PERSON-RECORDS TERMINATED*)
          CLOSE(UNIT=20)
C
C  DATABASE IS CLOSED (FOR THIS USER)
C
175       CALL SCLDB("GENDB-GB",IST)
          IF (IST.NE.1)     CALL ERROR(6)
180       STOP
          END

         SUBROUTINE ERROR(N)
C
C  PURPOSE
C      PRINT OUT ERROR MESSAGES
C
C  INPUT PARAMETER
C      N      ERROR NUMBER
C
C
         INTEGER RNAME1(4),RNAME2(4),SNAME(4),INAME(4)
          CHARACTER ERRLIST(6)*30
          DATA ERRLIST(1)/'ERROR IN OPEN-DATABASE     '/
          DATA ERRLIST(2)/'ERROR IN READY-REALM'/
          DATA ERRLIST(3)/'ERROR IN OPEN INPUT-FILE'/
          DATA ERRLIST(4)/'ERROR WHEN READING A RECORD'/
          DATA ERRLIST(5)/'ERROR IN STORE       '/
          DATA ERRLIST(6)/'ERROR IN CLOSE-DB    '/

C
C  GET THE  DATABASE EXEPTION CONDITION CODE
C
          CALL SDBEC(SNAME,RNAME1,RNAME2,INAME,IDML,IDBEC)

          WRITE (1,900)  ERRLIST(N),IDBEC,IDML,SNAME,RNAME1,RNAME2,INAME
900       FORMAT('0',A,/,2X,'DBEC      :',I4,/,
     +                   2X,'DML-CALL  :',I4,/,
     +                   2X,'SET-NAME  : ',4A2,/,
     +                   2X,'REALM1    : ',4A2,/,
     +                   2X,'REALM2    : ',4A2,/,
     +                   2X,'ITEM      : ',4A2,)
          RETURN
          END
```

4.3.2 COBOL

Calling SIBAS subroutines from a COBOL program is just the same as calling a FORTRAN subroutine. The programmer must be aware of two things:

1. Parameters must always start on a word boundary
2. The values passed to or from SIBAS are always an integral number of SIBAS-words.

The concept of "word" is somewhat strange to a COBOL programmer, but a "word" is made of 2 bytes on ND-10 or ND-100, 4 bytes on ND-500. A SIBAS-word is always 2 bytes, which requires some precaution on the ND-500.

The COBOL compilers automatically align 01 level and 77 level on word boundaries.

As a good programming practice, define the length of the data items passed to or from SIBAS as an even number of bytes. If the last byte of a DISPLAY field is never used, fill it with a default byte, for example space.

To arrive at the number of bytes, add one to the number of 9's in the picture and, using integer division, divide the sum by 2. If the number of bytes is odd, you should insert a filler with a picture X, VALUE zero after the definition of the item. The following is an example of how this would look:

```
01 RECORD.

    05  REC-IT1     PIC 99          COMP-3.
    05  FILLER      PIC  X          VALUE 0.
    05  REC-IT2     PIC 599V999     COMP-3.
```

Otherwise, COBOL is very well suited to writing programs accessing a SIBAS database, mainly because it has a DATA DIVISION where the data areas are clearly defined.

4.3.2.1 General Rules for COBOL on the SIBAS-500

The programmer must be aware of four different aspects:

1. Parameters must always start on a word boundary. The ND Cobol compilers automatically align 01 level and 77 level on word boundaries. All parameters called "single integer" in the SIBAS User's manual should be given 77 level and *always* be defined as COMP (computational) *without* any PICTURE clause. Names (database name, etc.) and value buffers (see the previous section) are given 01 level. All names should be defined with the PICTURE clause without computational.

2. The values passed to/from SIBAS (100 and 500) are always an integral number of SIBAS-words. (One SIBAS-word is made up of 2 bytes). As a good practice, define the length of the data items passed to/from SIBAS as an even number of bytes. If the last byte of a DISPLAY field is never used, fill it with a default byte, for example space.

3. Value buffers ("item-values" etc., see section 4.1.3.2 for a more detailed description) are most easily handled when all items are given the type CHARACTER in the SIB-DRL input file (i.e., all database items are declared as CHARACTER in the database schema). If they are, all value buffers in the Cobol-500 application can be declared with the PICTURE clause according to the length of the item-value in the database schema. In such cases the computational clause should never be used in conjunction with the picture clause.

   Note, however, that declaring all items of type CHARACTER may require a great deal of unused/wasted disc-space.

   If there are items in the database schema of type INTEGER, some special rules have to be followed for 500 applications passing values to/from these items: Cobol-500 will cause variables declared with the COMPUTATIONAL option to occupy 32 bits (4 bytes) as long as the PICTURE clause is not used, or if the PICTURE clause is used (in combination with COMPUTATIONAL), *with field length equal to five or greater.* If computational is used togehter with a picture clause specifying *four characters or less* (ex: 01 DBVAL PIC 9(4) COMP.), then Cobol-500 will assign 16 bits of storage to the variable.

   Cobol storage allocation:

COMP only	: 32 bits
COMP and PIC 9(i) where i > 4	: 32 bits
COMP and PIC 9(j) where j < 5	: 16 bits

   Since integer items in the SIBAS database are 16 bits, 500-applications which pass value buffers to/from such integer items must define these as COMP + PIC 9(n), where n < 5.

4.3.2.2 Standard 'Cookbook' for Progamming COBOL Applications

The programmer of Cobol applications should follow the "cookbook" given below concerning SIBAS calls:

A. All parameters denoted "single integer" in the SIBAS-II User's manual should be given level-77 and defined as COMPUTATIONAL. The value clause can be used to initialize them. Do not use the picture clause.

The most common "single integer" parameters (in the manual's terminology):

"mode"
"no-of-realms"
"no-of-items"
"no-wanted"
"no-found"
"option-code"
"key-length"
"value-length"
"temporary-data-base-key"
"temporary-search-region-indicator"
"SIBAS-system-number"
"DML-statement-code"
"dbec"
"status"

Examples:

```
77   SIBAS-STATUS          COMP     VALUE 0.
77   FIVE-REALMS           COMP     VALUE 5.
77   SIBAS-DEVICE          COMP     VALUE 2.
77   OPEN-MODE             COMP     VALUE 15473.
77   KEY-LENGTH            COMP     VALUE 4.
```

Note that *length-of-value-parameters* (i.e., "key-length" and "value-length") specifies the number of SIBAS-words (i.e., the number of 16 bits words).

B. The two integer tables (arrays) "usage-modes" and "protection-modes" used in the SRRLM-call must be defined as COMPUTATIONAL with an additional OCCURS clause if more than one realm is readied. Use level-01.

Examples:

```
01      USAGE-LIST.
        03   USAGE-MODE      COMP    OCCURS 5.

01      PROT-LIST.
        03   PROTECT-MODE    COMP    OCCURS 5.
```

C. All names are given level-01 and sized by means of the PICTURE clause.

Names are:

"data-base-name"
"password"
"realm-name"
"set-name"
"item-list"
"key-name"

Examples:

```
01      SET-NAME              PICTURE X(8).

01      DB-NAME               PICTURE A(8) VALUE 'TESTBAS  '.

01      REALM-NAMES.
        03 REALM               PICTURE A(8) OCCURS 5.
```

D. Value buffers are given 01-level and are sized by means of the PICTURE clause. If all items in the database schema are defined to be of type CHARACTER (and this is recommended), all use of the computational clause should be avoided. Items declared as INTEGER in the SIBAS schema require value buffers (passing values to/from these items) to be defined with the combination of COMP and PICTURE 9(n) (where n<5).

Examples:

```
01  ART-VALUE.
    03  ART-NUMBER     PIC  X(6).       %  declared as
    03  ART-DESCR      PIC  A(16).      %  CHARACTER in
    03  ART-ANTALL     PIC  9(4).       %  the SIBAS
    03  ART-PRICE      PIC  X(6).       %  schema

01  KUNDE-VALUES.
    03  KUNDE-NR       PIC  9(4) COMP.  %  INTEGER in schema
    03  KUNDE-NAVN     PIC  9(24).
```

```
NORSK DATA COBOL - VER H. COB-EX            TIME 12.06  DATE 05.02.80

    1          IDENTIFICATION DIVISION.
    2         *
    3          PROGRAM-ID.
    4                CPERL-GB.
    5          AUTHOR.
    6                J F BOHMER  DES 1979.
    7         ***************************************************************
    8         *       PROGRAM READS PERSONAL RECORDS FROM A DISC FILE AND PLACES
    9         *       EACH RECORD INTO THE DATABASE. EACH RECORD IS ALSO LISTED
   10         *       ON THE TERMINAL AFTER EDITING.
   11         *       ** ERROR CONDITIONS ARE REPORTED ON TERMINAL **
   12         *       ** INPUT FILE WAS BUILT BY ANOTHER PROGRAM   **
   13         ***************************************************************
   14         *
   15          ENVIRONMENT DIVISION.
   16         *
   17          CONFIGURATION SECTION.
   18                SOURCE-COMPUTER N-10-S.
   19                OBJECT-COMPUTER N-10-S.
   20          INPUT-OUTPUT SECTION.
   21          FILE-CONTROL.
   22                SELECT TEXTOUT ASSIGN 'TERM'.
   23                SELECT PERSON1 ASSIGN 'GPERREC:DATA' STATUS ERROR-PRO.
   24          I-O-CONTROL.
   25         *
   26          DATA DIVISION.
   27         *
   28          FILE SECTION.
   29         *
   30          FD     TEXTOUT
   31                 LABEL RECORD OMITTED.
   32          01     TEXTLINE     PIC X(100).
   33          01     PERSREC.
   34                 02 OBDATE         PIC X(6).
   35                 02 FILLER         PIC X.
   36                 02 OBNO           PIC X(5).
   37                 02 FILLER         PIC X.
   38                 02 OPERSNAME      PIC A(26).
   39                 02 FILLER         PIC X.
   40                 02 OPERSADDR      PIC X(30).
   41                 02 FILLER         PIC X.
   42                 02 OSEX           PIC A(1).
   43                 02 FILLER         PIC X.
   44                 02 OHSALARY       PIC ZZ9.99.
   45                 02 FILLER         PIC X.
   46                 02 ODEPARTM       PIC X(6).
   47                 02 FILLER         PIC X.
   48                 02 OPRESERV       PIC X(12).
   49         *
   50          01     ERROR-LINE.
   51                 02 E-TEXT         PIC X(20).
   52                 02 E-GROUP.
   53                    03 E-DML       PIC 9(4).
   54                    03 FILLER      PIC X.
   55                    03 E-DBEC      PIC 9(4).
   56                    03 FILLER      PIC X.
```

```
NORSK DATA COBOL - VER H. COB-EX                TIME 12.06  DATE 05.02.80

 57                  03 E-INAME    PIC X(8).
 58                  03 FILLER     PIC X.
 59                  03 E-SNAME    PIC X(8).
 60                  03 FILLER     PIC X.
 61                  03 E-RNAME1   PIC X(8).
 62                  03 FILLER     PIC X.
 63                  03 E-RNAME2   PIC X(8).
 64         *
 65          FD     PERSON1
 66                 BLOCK CONTAINS 0 RECORDS
 67                 LABEL RECORD OMITTED.
 68          01     IPERSREC.
 69                 02 IBDATE         PIC X(6).
 70                 02 IBNO           PIC X(5).
 71                 02 IPERSNAME      PIC A(26).
 72                 02 IPERSADDR      PIC X(30).
 73                 02 ISEX           PIC A(1).
 74                 02 IHSALARY       PIC 9(4)V99.
 75                 02 IDEPARTM       PIC X(6).
 76                 02 IPRESERV       PIC X(12).
 77         *
 78         *
 79          WORKING-STORAGE SECTION.
 80         *****************************************************************
 81         *       THE 77 LEVELS ARE CONSTANTS USED MAINLY FOR THE CALLS TO
 82         *       SIBAS
 83         *
 84         *
 85         *
 86         *
 87           77 PROGNAME     PIC X(8) VALUE 'CPERL-GB'.
 88           77 HEADING      PIC X(30)
 89                                VALUE 'NEW PERSON-RECORDS TO DATABASE'.
 90           77 DATA-BASE    PIC X(8) VALUE 'GENDB-GB'.
 91           77 PASS-WORD    PIC X(8) VALUE '        '.
 92           77 RUN-ID                VALUE 15473 COMP.
 93           77 STAT                  VALUE     0 COMP.
 94           77 NO-OF-REALMS                VALUE 1  COMP.
 95           77 US                          VALUE 1  COMP.
 96           77 PROT                        VALUE 0  COMP.
 97           77 RECL                        VALUE 46 COMP.
 98           77 NO-OF-ITEMS                 VALUE 8  COMP.
 99           77 ERROR-NO                  COMP.
100           77 REALM              PIC X(8) VALUE 'PERSON  '.
101           77 ERROR-PRO          PIC X(2) VALUE '00'.
102             88 ON-ERROR                  VALUE '30' '34'.
103         *****************************************************************
104         *       THE ITEM-LIST CONTAINS THE *LITERAL NAMES* FOR THE ITEMS
105         *       DEFINED IN THE DATABASE DEFINITION. ONLY THOSE ITEMS
106         *       REQUIRED NEED BE LISTED AND THE ORDER IS NOT IMPORTANT.
107         *****************************************************************
108         *       THE LITERALS *MUST* BE 8 CHARACTERS LONG!
109         *****************************************************************
110         *
111          01     ITEM-LIST.
112                 02 LBDA          PIC A(8) VALUE 'BDATE   '.
```

```
NORSK DATA COBOL - VER H. COB-EX                TIME 12.06  DATE 05.02.80

 113                02 LBNO          PIC A(8) VALUE 'BNO     '.
 114                02 LPNA          PIC A(8) VALUE 'PERSNAME'.
 115                02 LPAD          PIC A(8) VALUE 'PERSADDR'.
 116                02 LSEX          PIC A(8) VALUE 'SEX     '.
 117                02 LSAL          PIC A(8) VALUE 'HSALARY '.
 118                02 LDEP          PIC A(8) VALUE 'DEPARTM '.
 119                02 LRES          PIC A(8) VALUE 'PRESERV '.
 120         ***************************************************************
 121         *      ITEM-VAL IS A "RECORD" STRUCTURE DEFINING THE PLACE THAT
 122         *      THE *VALUES* FROM/TO THE DATABASE ARE TO BE FOUND/PLACED.
 123         *      THE FIELDS *MUST* DESCRIBE *EXACTLY* THE SIZE AS DEFINED
 124         *      BY THE DATABASE DEFINITION, AND *MUST* APPEAR IN THE EXACT
 125         *      SEQUENCE SHOWN IN THE ITEM-LIST (DEFINED ABOVE).
 126         ***************************************************************
 127         *
 128          01     ITEM-VAL.
 129                 02 BDATE         PIC X(6).
 130                 02 BNO           PIC X(5).
 131                 02 FILLER        PIC X     VALUE SPACE.
 132                 02 PERSNAME      PIC A(26).
 133                 02 PERSADDR      PIC X(30).
 134                 02 SEX           PIC A(1).
 135                 02 FILLER        PIC X     VALUE SPACE.
 136                 02 HSALARY       PIC 9(5) COMP.
 137                 02 DEPARTM       PIC X(6).
 138                 02 PRESERV       PIC X(12).
 139           01    ERROR-CODE.
 140                 02 DML                     COMP.
 141                 02 DBEC                    COMP.
 142                 02 INAME     PIC X(8).
 143                 02 SNAME     PIC X(8).
 144                 02 RNAME1    PIC X(8).
 145                 02 RNAME2    PIC X(8).
 146         *
 147         *
 148         *
 149          PROCEDURE DIVISION.
 150         *
 151          MAIN-PROGRAM SECTION.
 152         ***************************************************************
 153         * P-START TO NEXT-REC.
 154         *       OPEN THE I/O FILES , SET-UP AND LIST THE REPORT HEADING,
 155         *       OPEN THE DATABASE AND READY THE REALM TO RECIEVE DATA.
 156         ***************************************************************
 157          P-START.
 158                 OPEN INPUT PERSON1, OUTPUT TEXTOUT.
 159         *
 160                 MOVE PROGNAME TO TEXTLINE.
 161                 WRITE TEXTLINE AFTER PAGE.
 162                 MOVE HEADING TO TEXTLINE.
 163                 WRITE TEXTLINE AFTER 1.
 164                 MOVE SPACE TO TEXTLINE.
 165                 WRITE TEXTLINE AFTER 1.
 166         *
 167                 CALL 'SOPDB' USING RUN-ID DATA-BASE PASS-WORD STAT.
 168                 IF STAT < 0 MOVE 1 TO ERROR-NO
```

```
NORSK DATA COBOL - VER H. COB-EX              TIME 12.06  DATE 05.02.80

                                PERFORM ERROR-REP
                                GO TO EU1.
           CALL 'SRRLM' USING NO-OF-REALMS REALM US PROT STAT.
           IF STAT < 0 MOVE 2 TO ERROR-NO
                                PERFORM ERROR-REP
                                GO TO EU1.
      ********************************************************************
      * NEXT-REC TO FIN.
      *      READ 1 RECORD FROM THE PERSON1 FILE, IF NOT END THEN MOVE
      *      ITEMS INTO THE DATABASE ITEM-VAL AREA AND THE REPORT LINE.
      *      OUTPUT THE REPORT LINE AND TO THE DATABASE.
      ********************************************************************
       NEXT-REC.
           READ PERSON1 INTO IPERSREC AT END GO TO FIN.
           IF ON-ERROR MOVE 3 TO ERROR-NO
                                PERFORM ERROR-REP
                                GO TO NEXT-REC.
           MOVE SPACE TO PERSREC.
           MOVE IBDATE    TO BDATE OBDATE.
           MOVE IBNO      TO BNO OBNO.
           MOVE IPERSNAME TO PERSNAME OPERSNAME.
           MOVE IPERSADDR TO PERSADDR OPERSADDR.
           MOVE ISEX      TO SEX OSEX.
           INSPECT IHSALARY REPLACING LEADING SPACE BY "0".
           MOVE IHSALARY  TO HSALARY OHSALARY.
           MOVE IDEPARTM  TO DEPARTM ODEPARTM.
           MOVE IPRESERV  TO PRESERV OPRESERV.
           WRITE PERSREC AFTER 1.
           CALL 'STORE'
           USING REALM NO-OF-ITEMS ITEM-LIST ITEM-VAL STAT RECL.
           IF STAT < 0 OR STAT = 0 MOVE 4 TO ERROR-NO
                                PERFORM ERROR-REP.
           GO TO NEXT-REC.
      ********************************************************************
      * FIN TO ERROR-REP SECTION.
      *      REPORT END FILE REACHED, CLOSE I/O FILES, CLOSE DATABASE,
      *      REPORT IF ANY ERROR ON CLOSE DATABASE, STOP RUN.
      ********************************************************************
       FIN.
      *      READING TERMINATED.
           MOVE 'LOADING TERMINATED' TO TEXTLINE.
           WRITE TEXTLINE AFTER 2.
       EU1.
           CLOSE PERSON1 TEXTOUT.
           CALL 'SCLDB' USING DATA-BASE STAT.
           IF STAT < 0 MOVE 5 TO ERROR-NO
                                PERFORM ERROR-REP.
           STOP RUN.
       ERROR-REP SECTION.
      ********************************************************************
      * E-S TO E-E.
      *      IF ERRORS OCCUR DURING PROCESSING THIS SECTION IS CALLED
      *      WITH ERROR-NO SET TO SELECT THE DESIRED ERROR MESSAGE TO
      *      BE REPORTED ON THE TERMINAL. RETURN TO CALLER AFTER OUTPUT.
      ********************************************************************
      *
```

```
NORSK DATA COBOL - VER H. COB-EX            TIME 12.06  DATE 05.02.80

 225          E-S.
 226         *       FOR EXCEPTIONAL CONDITIONS.
 227                 CALL 'SDBEC' USING SNAME RNAME1 RNAME2 INAME DML DBEC.
 228                 MOVE SPACE TO ERROR-LINE.
 229                 MOVE DML     TO E-DML.
 230                 MOVE DBEC    TO E-DBEC.
 231                 MOVE INAME   TO E-INAME.
 232                 MOVE '*********'    TO E-SNAME.
 233                 MOVE RNAME1 TO E-RNAME1.
 234                 MOVE '*********'    TO E-RNAME2.
 235
 236                 GO TO ERR1 ERR2 ERR3 ERR4 ERR5 DEPENDING ERROR-NO.
 237          ERR1.
 238                 MOVE 'ERROR IN OPEN DB' TO E-TEXT.
 239                 WRITE ERROR-LINE AFTER 1.
 240                 GO TO E-E.
 241          ERR2.
 242                 MOVE 'ERROR IN READY-REALM' TO E-TEXT.
 243                 WRITE ERROR-LINE AFTER 1.
 244                 GO TO E-E.
 245          ERR3.
 246                 MOVE SPACE TO TEXTLINE.
 247                 STRING 'ERROR IN RECORD ' IBDATE IBNO IPERSNAME
 248                 DELIMITED SIZE INTO TEXTLINE.
 249                 WRITE TEXTLINE AFTER 1.
 250                 GO TO E-E.
 251          ERR4.
 252                 MOVE 'ERROR IN STORE' TO E-TEXT.
 253                 WRITE ERROR-LINE AFTER 1.
 254                 GO TO E-E.
 255          ERR5.
 256                 MOVE 'ERROR IN CLOSE DB     ' TO E-TEXT.
 257                 WRITE ERROR-LINE AFTER 1.
 258          E-E.
 259                 EXIT.

**   NO DIAGNOSTIC MESSAGE(S) **
```

4.3.3 PLANC

Calling SIBAS subroutines from a PLANC program is the same as calling Fortran subroutines from PLANC. The programmer should declare his PLANC routines to be of type "ROUTINE STANDARD(VOID,VOID)". Names should be built and sent in BYTE ARRAYs. (NOTE: Always specify arrays with lower index = 0 when they are used as actual parameters in SIBAS). As in Fortran applications, single integer parameters must be declared INTEGER, and value buffers must follow the same rules outlined in the description of Fortran applications (i.e., declared INTEGER2). (For more detailed information — see section 4.3.1. Fortran.)

4.4 HOW TO LOAD APPLICATON PROGRAMS

4.4.1 Description

Application programs must be loaded with one of the available SIBAS libraries (called simulators). The simulators' functions are basically to collect parameters, send them to SIBAS and receive the output values.

4.4.2 Different types of simulators

There are five types of simulators, i.e., five different sets of routines one may load together with the application programs, depending upon the mode of operation of the application programs.

The simulators named SIBLIB-nx-xx are compiled with new FORTRAN-100 compiler and must be used with systems and applications compiled with FORTRAN-100 compiler, or compilers compatible to this compiler in 1-bank or 2-bank mode (i.e.,COBOL, PLANC).

The ones named SIB2-DML-x-xx are compiled with the old FTN compiler and must be used with applications compiled with that compiler or compatible compilers, e.g., BASIC.

For further explanation of the meaning of 1BANK, 2BANK, recursive and non-recursive, please consult the NORD RELOCATING LOADER User's Manual.

4.4.3 Choosing simulators

SIBLIB-1N-MH:BRF is for use by non-reentrant 1-bank programs, usually background programs.

SIB2-DML-B-MH:BRF is the same library compiled with the old FTN.

SIBLIB-1R-MH:BRF is for use for reentrant application, either real-time or dumped reentrant with the SINTRAN command @DUMP-REENTRANT.

SIB2-DML-R-MH:BRF is the same simulator compiled with the old FTN.

SIBLIB-2N-MH:BRF is for use by non-recursive 2-bank programs.

All libraries have two common blocks that need totally 532D locations.

4.4.4 Loading Nonreentrant Programs with SIBAS

This should not present special difficulties. The library acts as fortran library and FORTRAN-1BANK library must be loaded together with the application and the SIBLIB-1N-MH library. (See example 1.)

If SIB2-DML-B-MH is used, FTNLIBR must be loaded.

Example 1: Loading of Background Programs:

```
@FORT
ND-100 ANSI 77 FORTRAN COMPILER - 203053A
FTN:COM INSERT,,SCRA
- CPU TIME USED: 5.6 SECONDS.  92 LINES COMPILED.
- NO MESSAGES
- PROGRAM SIZE=812 COMMON SIZE=0
FTN:EX

@NRL
RELOCATING LOADER - AUGUST 18, 1982
*PROG-FILE INSERT-1N
*LOAD SCRA
FREE: 001454-177777
*LOAD SIBLIB-1N-MH
FREE: 011160-176753
*LOAD FORTRAN-1BANK
FREE: 041230-176753
*E-U
FREE: 041230-176753
*EXIT
```

4.4.5 Loading 2BANK Programs with SIBAS

The FORTRAN-2BANK library must be loaded together with the application and the SIBLIB-2N-MH library. (See example 2.)

Example 2: Loading of 2-bank Programs:

```
@FORT
ND-100 ANSI 77 FORTRAN COMPILER - 203053A
FTN: SEPARATE-DATA ON
FTN: COM INSERT, , SCRA
- CPU TIME USED: 5.6 SECONDS.  92 LINES COMPILED.
- NO MESSAGES
- PROGRAM SIZE=361 DATA SIZE=452 COMMON SIZE=0
FTN: EX

@NRL
RELOCATING LOADER - AUGUST 18, 1982
*PROG-FILE INSERT-2N
*LOAD SCRA
FREE: 000551-177777 .... FREE DATA AREA: 000704-177777
*LOAD SIBLIB-2N-MH
FREE: 006150-177777 .... FREE DATA AREA: 004172-177777
*LOAD FORTRAN-2BANK
FREE: 032316-177777 .... FREE DATA AREA: 010356-177777
*E-U
FREE: 032316-177777 .... FREE DATA AREA: 010356-177777
*EXIT
```

4.4.6 Loading Reentrant Programs with SIBAS

These are programs to be dumped reentrant with the SIII command @DUMP-REENTRANT. The FORTRAN-1BANK library must be loaded together with the applications and SIBLIB-1R-MH library. (See example 3.)

If SIB2-DML-R-MH is used, FTNRTLIBR must be loaded.

Example 3: Loading of Reentrant Programs:

```
@FORT
ND-100 ANSI 77 FORTRAN COMPILER - 203053A
FTN:REENTRANT
FTN:COM INSERT,,SCRA
- CPU TIME USED: 5.2 SECONDS.  92 LINES COMPILED.
- NO MESSAGES
- PROGRAM SIZE=965 COMMON SIZE=0
FTN:EX

@NRL
RELOCATING LOADER - AUGUST 18, 1982
*IM 100
*LOAD SCRA
FREE: 001705-177777
*LOAD SIBLIB-1R-MH
FREE: 007737-176753
*LOAD FORTRAN-1BANK
FREE: 036744-176753
*E-U
 5STLEN=000015 U
FREE: 036744-176753
*DEFINE 5STLEN 10000
*BPUN INSERT-REENT 0 0
*EXIT
```

4.4.7 Loading Real Time Programs with SIBA$

The FORTRAN-1BANK library must be loaded together with the actual SIBLIB library. (Reentrant or non-reentrant version, depending on the RT-program.) (See exaple 4, loading of a non-reentrant RT-program.)

If SIB2-DML lib is used, FTN-lib must be loaded.

The common blocks SIBSYS and SIBCOM must be placed on a write permitted segment. The common location SIBSYS(2) (IBRIX) must contain zero as an initial value. Then it will be replaced with the programs RT-description address when the program is started. If the SIBLIB code is shared between several RT programs, they should have their own common blocks loaded on different segments at the same logical address.

Example 4: Loading of non-reentrant Real Time Programs:

```
@FORT
ND-100 ANSI 77 FORTRAN COMPILER - 203053A
FTN:COM (B--S)INSERT,,SCRA
- CPU TIME USED: 6.2 SECONDS.  92 LINES COMPILED.
- NO MESSAGES
- PROGRAM SIZE=812 COMMON SIZE=0
FTN:EX
@RT-L
REAL-TIME LOADER,   SINTRAN III - H
*CL-SEG 276
RT-PROGRAMS ON SEGMENT:

  NINSERT

DELETING THIS RT-PROGRAM(S)?  Y
*NEW-SEGMENT 276,1,,,,,,
*Y
*Y PLACE COMMON BLOCKS SIBSYS AND SIBCOM
*Y
*PRESET-COMMON-ADDRESS 276 50000
*SET-LOAD-ADDRESS 276  0
*LOAD SCRA,,
*LOAD (B--S)SIBLIB-1N-MH,,,,
*LOAD FORTRAN-1BANK,,,
*WRITE-LOAD-ADDRESS,,,,
```

```
L. ADR.        0  U. ADR:     41227     C. LADR:     41230

*WRITE-COMMON-LABELS,,,

 SIBCOM          276    1021
 SIBSYS          276       3

*Y
*Y SET UPPER LIMIT OF SEGMENT BEHIND COMMON BLOCKS
*Y
*SET-LOAD-ADDRESS 276 52000
*Y
*Y INITIALIZE SIBSYS(2)  (IBRIX) WITH 0
*Y
*CHANGE-LOCATION,,,,

50001/         0 0
  42240 .
*
*END-LOAD,,,,
COMMON AREA ADDR. SPECIFIED IN THE PRESET-COMMON COMMAND IS LESS
THAN THE CURRENT LOAD ADDR.
SEGMENT NO.  276
*Y
*Y THIS WAS JUST A MESSAGE FROM THE LOADER (NOT AN ERROR OR WARNING)
*Y
*WRITE-SEGMENT 276,,,,,

 276        0    51777  12361    0    1    1  RFW NON DEMAND

*EX
```

4.4.7.1 Cooperating RT Programs Working as one "SIBAS USER"

Normally, the RT-description address of the application process is used as user identification. The identification can, however, be defined in another way. This is done by giving the common location SIBSYS(2) (IBRIX) an initial value equal to the wanted identification (NOT zero).

4.4.8 Table of SIBAS Simulators (Libraries)

The following chart (along with the preceding examples of loading and compiling programs) may help to make things a little clearer.

Application environment			Library to use	Comment Examples
New linkage convention as FORTRAN-77 COBOL PLANC	2BANK programs	non-recursive (usual)	SIBLIB-2N-MH	2 (4.4.5)
	1BANK programs	non-reentrant usually background application	SIBLIB-1N-MH	1 and 4 (4.4.4 and 4.4.7)
		reentrant and recursive usually dumped-reentrant and often RT-programs	SIBLIB-1R-MH	3 (4.4.6)
Old linkage conventions as	non-reentrant usually background application		SIB-DML-B-MH	
(@FTN compiler and FTN compatible languages)	reentrant and recursive usually dumped-reentrant and often RT-programs		SIB-DML-R-MH	

4.4.9 Applications on ND-500 Systems

4.4.9.1 Applications running on the 500 CPU

Applications running on the 500 CPU should link to a common SIBAS-LIBRARY segment (where all SIBAS entry points are defined). Note that the same library is used independently of the SIBAS-process used (SIBAS-100 or SIBAS-500). Execution of the SETDV-call will select the correct SIBAS. In addition to the SIBAS-LIBRARY, the application has to be linked to the SIBAS-500 message system (SIBAS-MESSAGE).

Example:

```
                                                          Notes:
N11: SET-DOMAIN <domain-name>
N11: OPEN-SEGMENT <segment-name>,,,
N11: LOAD-SEGMENT <application>
N11: FORCE-SEG-LINK (SIB-500)SIBAS-LIBRARY                  i
N11: LINK-SEG       (SIB-500)SIBAS-MESSAGE                  ii
N11: EXIT                                                   iii
```

Notes:

i) FORCE-SEG-LINK must be used since the SIBAS-LIBRARY segment also is linked to the message system. SIB-500 is the username where the library is assumed to reside.

ii) Linking to the message system is necessary to get access to global data. If linking to SIBAS-MESSAGE is forgotten/skipped, the error-message "PROTECT VIOLATION" will occur on your terminal when the program is started.

iii) If the size of RT-common is changed after SIBAS-500 applications are loaded, they all have to be re-loaded, including the SIBAS-LIBRARY segment.

4.4.9.2 Applications running on the 100 CPU

All applications running on the 100 CPU can use the standard SIBAS-100 LIBRARIES (see Section 4.4.2)regardless of which SIBAS process they are using (SIBAS-100 or SIBAS-500). The SIBAS system number (used in SETDV) will select the correct SIBAS process without regard to whether SIBAS is running on the 100 or the 500 CPU.

5 DATABASE ADMINISTRATION

This chapter contains information about the administration of the SIBAS system itself. In small systems, this may be done by the SIBAS users (i.e., programmers). In larger systems, there will probably be one person who has the responsibility for administrating the SIBAS system (i.e., the data base administrator).

Database administration, as described in this chapter, includes such tasks as starting and stopping SIBAS, determining the logging and recovery facilities required, and carrying out rollback and recovery functions when needed.

Those functions can be carried out in two ways: either as SIBAS-SERVICE commands or by calling the proper routines from an application program.

Other tasks of the database administrator include such things as defining privacy requirements, taking consistency checks of the data base, and printing and patching the database. These tasks are carried out using special utility programs and are described in Chapter 6.

The application oriented tasks of a database administrator, such as determining the contents and structure of the database, were discussed earlier in this manual.

This chapter starts out with some general information on the structure of SIBAS, its operating requirements and its running states. The logging and recovery facilities are then discussed, and finally the SIBAS calls used to carry out administrative functions are described in detail.

5.1 REAL-TIME ORGANIZATION OF SIBAS

5.1.1 How is SIBAS organized?

SIBAS is a multi-user, single thread database management system in that one SIBAS system is associated with one database and one SIBAS system executes only one SIBAS statement at a time. However, there may be several SIBAS systems (called processes) active, each accessing their respective databases.

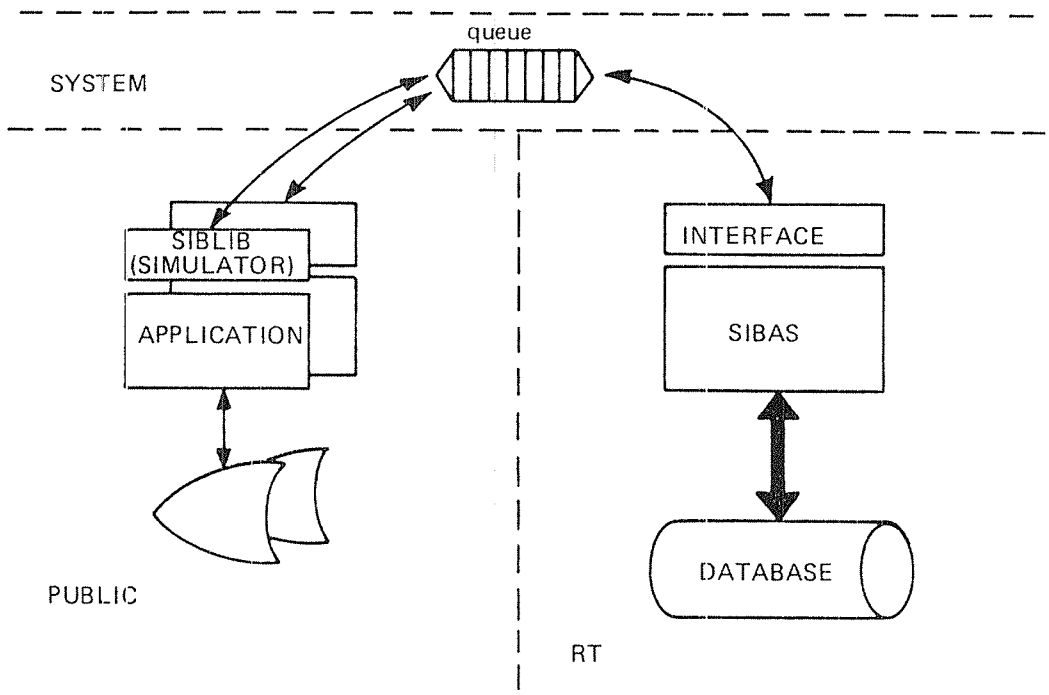
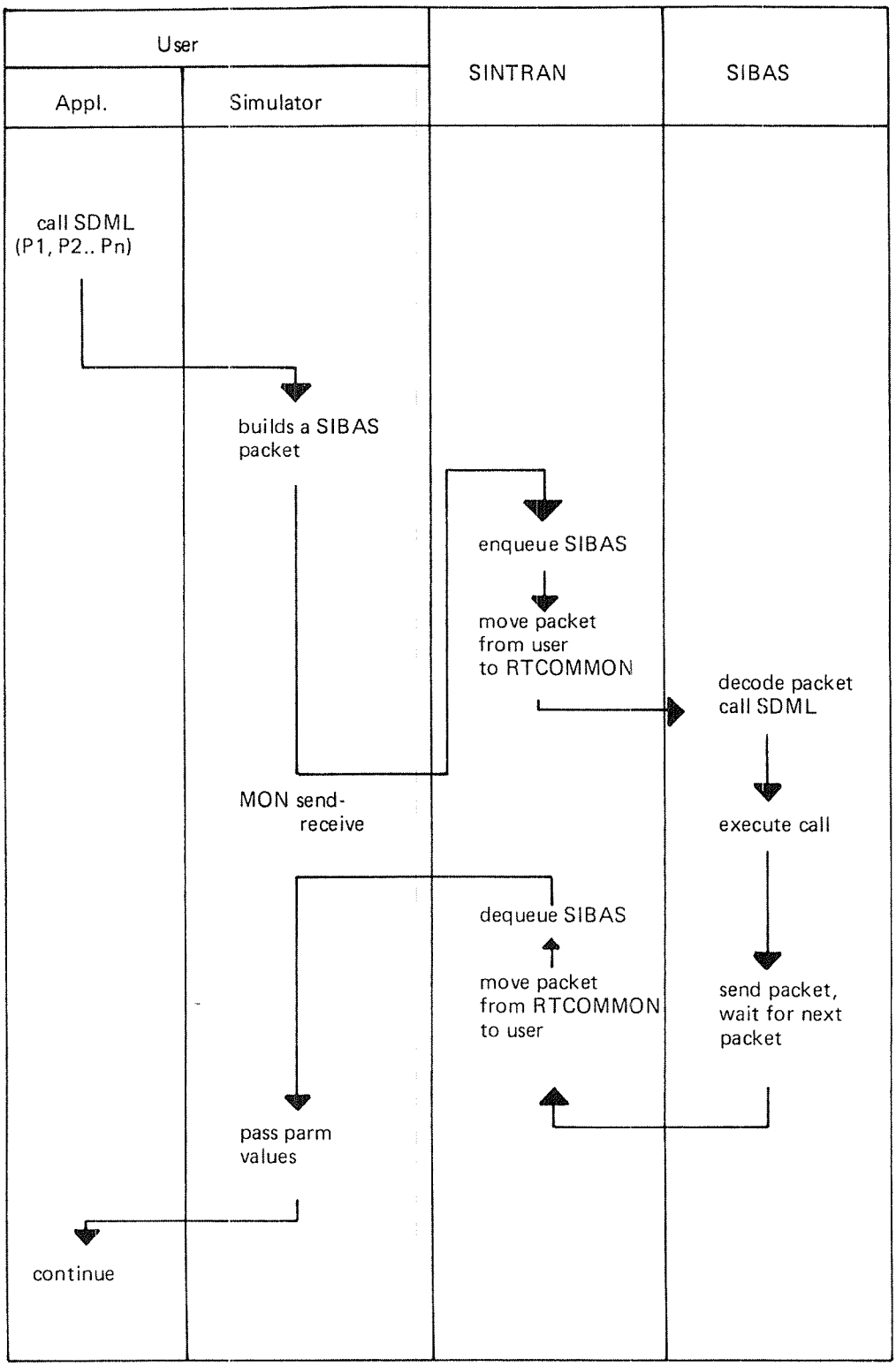


Figure 5.1: Simplified Overview of SIBAS — Application Communication on ND-10, ND-100

SINTRAN provides the two-way communication between one SIBAS process and the rest of the world. The communication path is depicted below.


User
Appl.
Simulator
SINTRAN
SIBAS
call SDML
(P1, P2.. Pn)
builds a SIBAS
packet
enqueue SIBAS
move packet
from user
to RTCOMMON
decode packet
call SDML
MON send-
receive
execute call
dequeue SIBAS
move packet
from RTCOMMON
to user
send packet,
wait for next
packet
pass parm
values
continue

Simulator:

The SIBAS simulators are a set of routines which must be loaded together with the application programs in order to access SIBAS. The simulator's functions are basically to collect parameters, send them to SIBAS and receive the output values.

As explained in the sections under 4.4 (How to load application programs) there are several versions of the simulator the use of which depend on 2 criteria:

1. The language and mode of execution in which the applications are written, eg., @FTN or FORTRAN-77 compilers.

2. Whether the SIBAS process is located in the same machine or not.

Interface:

The interface is a part of the SIBAS process and is shown here for completeness.

Backup/Recovery must make the interface more complicated than it appears in the figure, because the interface may log the packets on a logfile.

SIBAS Requirements:

This section requires a working knowledge of SINTRAN III real-time features, SIBAS requirements (real-time segments, memory) vary with the options in use and the number of processes involved.

Each SIBAS process requires 1K words of RT-COMMON area.

SIBAS Segments:

SIBAS programs are quite large. To run efficiently, up to 31K data buffer is required. This leaves about 32K for the rest of code which is actually about 42K. This code is segmented using the SINTRAN RT segmentation scheme. Most of the code is reentrant, giving the possibility of accessing simultaneously more than one database with the same code. There must be enough space on the segment files for the SIBAS segments. Necessary space is 42K words for the reentrant part and 35K words per SIBAS process.

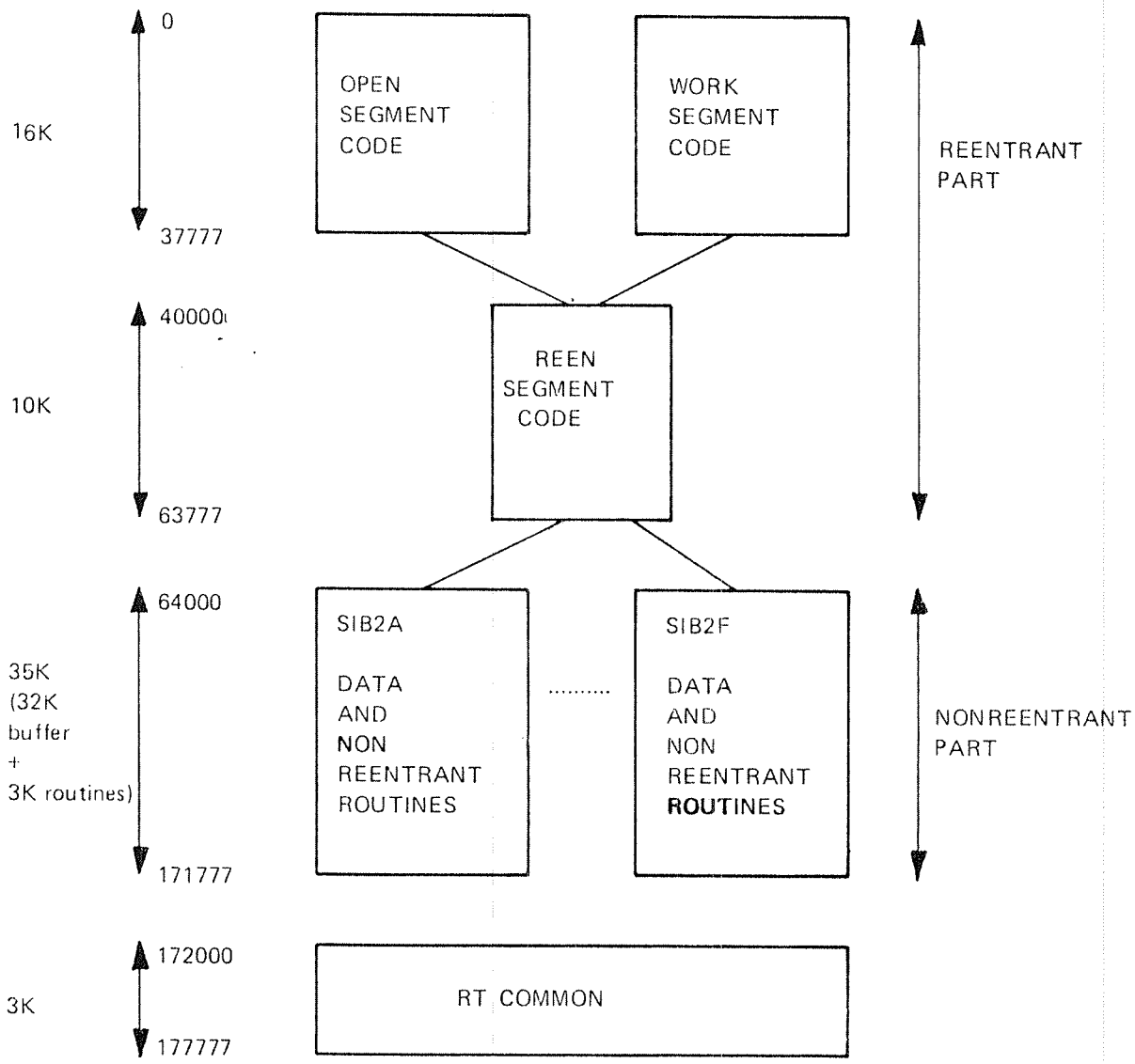


Figure 5.2: Virtual Memory Layout of the SIBAS Processes on ND-10 and ND-100.

The upper boundary of the SIBAS data segment may be changed at system generation to accomodate bigger RT-COMMON.

5.1.2 Organization of SIBAS on an ND-500 System

A SIBAS process running on the 500 CPU is called SIBAS-500, while SIBAS-100 denotes a process running on the 100 CPU.

Currently, there is an upper limit of six SIBAS processes running simultaneously on an ND-500 system (but SINTRAN may be patched to run 11 processes). It is up to the database administrator (DBA) to decide at the time of installation how many of these six processes he wants to be SIBAS-500 processes. The DBA may for instance decide to have two SIBAS-100 processes (SIB2A, SIB2B) and four SIBAS-500 processes (SIB2C, SIB2D, SIB2E, SIB2F) installed on an ND-500 system, and have them run simultaneously. (See figure 5.2A)

Application programs may access both SIBAS-100 processes and SIBAS-500 processes, i.e., they don't have to run on the same CPU as the SIBAS process.

It must, however, be emphasized that normally *the best way to increase performance is to run both the application and SIBAS on the 500 CPU.* All other solutions will introduce extra communication overhead.

Communication between applications and SIBAS

The SIBAS-LIBRARY (simulator) linked to an appplication program running on the 500 CPU builds a packet for each SIBAS call. These packets are passed through a special purpose message system common to all SIBAS-500 processes. Packets to SIBAS-500 coming from 500-applications are linked into a queue associated with the destination SIBAS-500 system number. SIBAS-500 will handle these packets in standard FIFO (first-in-first-out) manner.

After having linked its packet into the correct SIBAS-500 queue, the 500-application will enter a waiting state. It will later on be restarted by SIBAS when its answer-packet has been made ready in the message system. A SIBAS-500 process will stay active as long as there is at least one SIBAS call waiting for execution on this SIBAS-500 process. When the queue is empty, SIBAS enters a waiting state. SIBAS will be restarted as soon as there is a call in its queue.

All data areas used in the message system reside on a shared ND-500 data segment partly fixed in memory.

If the SIBAS calls address a SIBAS-100 process, the packets will be sent via RT-common to the SIBAS-100 process. It picks them up just as if the packets came from an application running on the 100 CPU (see 5.1.1).

Applications running on the 100 CPU will (by their library, see 4.4.1) send packets through RT common as described in 5.1.1. If the calls address a SIBAS-500 process, the packets will be picked up by a special 500-application program (associated with that SIBAS-500 process) called a Server. It then sends the packets in the standard way to SIBAS-500 via the message system.

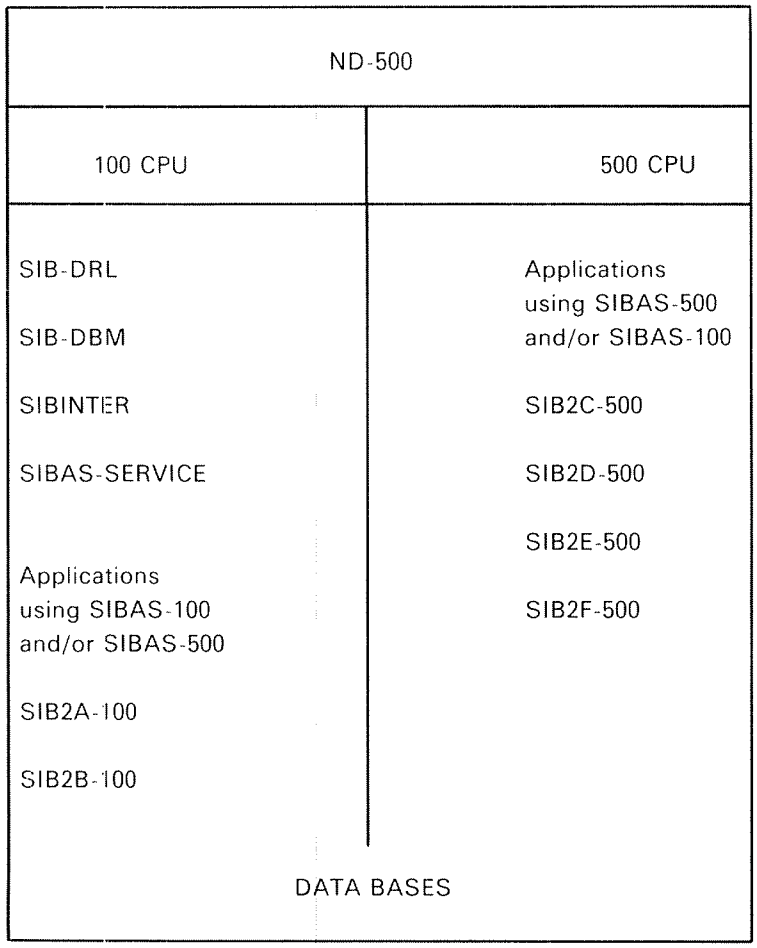


Figure 5.2A: Example of an ND-500 system running six SIBAS processes

5.2 SIBAS STATES

A SIBAS process is always in one of the five different states shown in the following diagram:

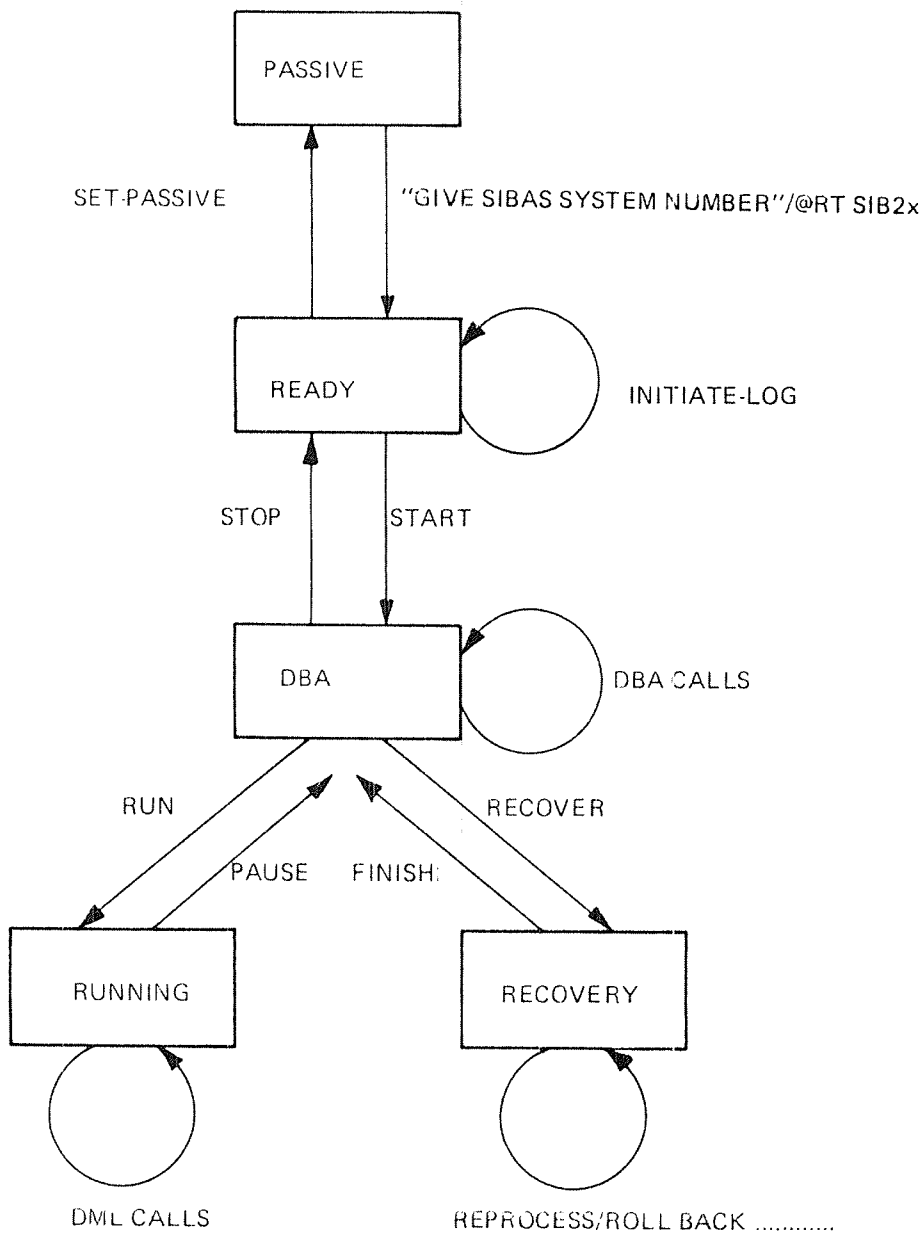


Figure 5.3.

A box represents a state
An arrow represents an action

If SIBAS is in any other state than RUNNING and a DML call is requested, the call will be placed in a waiting queue and executed when SIBAS enters RUNNING state again.

PASSIVE State:

After loading of the SIBAS processes with the RT loader, all the SIBAS processes are in PASSIVE state. The SIBAS process may also enter the PASSIVE state by executing the SPASS call.

READY State:

A SIBAS process is put in READY state by issuing the command @RT SIB2x or by giving the system number when using @SIBAS-SERVICE. One can then initiate the various types of logs using the INLOG call.

DBA State:

This is a privileged state where different maintenance calls may be executed while SIBAS process activity is suspended.

RUNNING State:

This is the most common and normal state of the SIBAS process where the different data manipulation calls are executed.

RECOVERY State:

This is an exception state where reprocessing and rollback activities are carried out.

5.3 LOGGING and RECOVERY FACILITIES

5.3.1 General

SIBAS offers three kinds of logging facilities

1. Routine logging
2. Delayed updating (not with a SIBAS-500 process).
3. Before image logging

illustrated as follows:

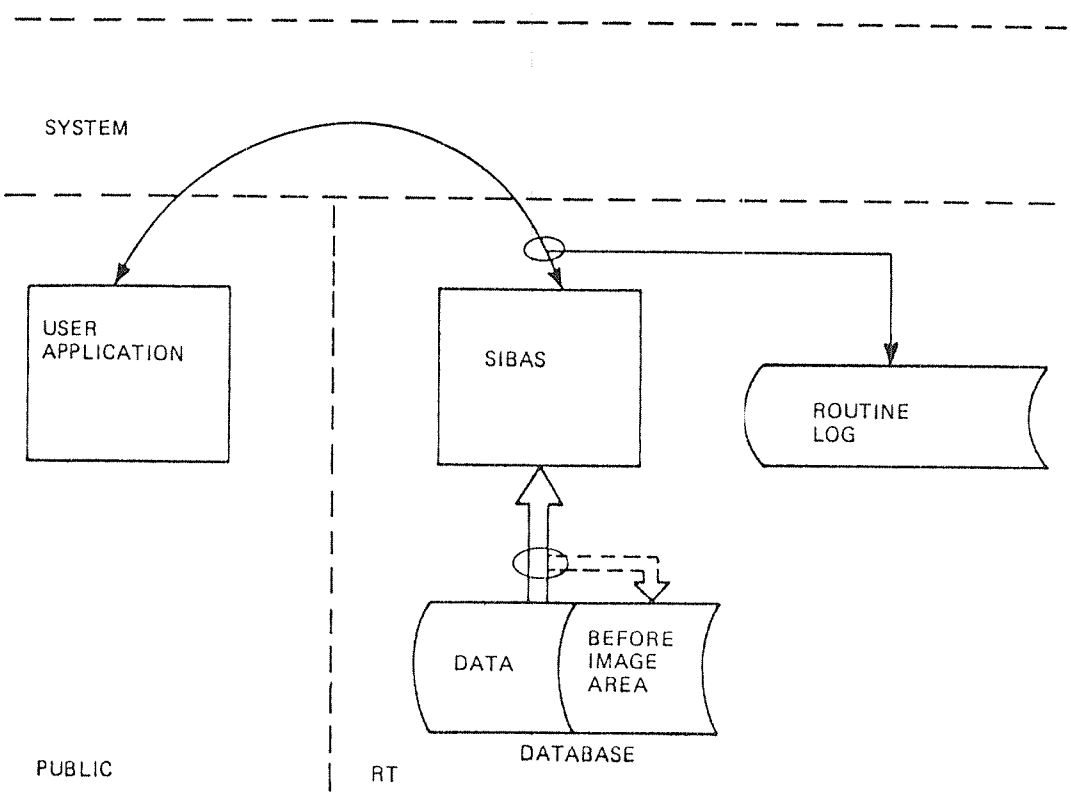


Figure 5.4.

When a run-unit calls SIBAS, a "call packet" is passed to SIBAS, and before the routine in SIBAS is activated, the "call packet" can be recorded on the ROUTINE log. If SIBAS processing results in database changes, the changes can be recorded on an UPDATE FILE and the database itself will be updated later on.

Before image and delayed updating excluded each other and are optional.

Routine logging is also optional and works independent of the other forms of logging.

5.3.2 Checkpoint

A checkpoint is a point of time where the database is consistent on the disk. All buffers are written on the disk and also all internal tables. A special checkpoint record is written to indicate that a checkpoint has been taken at this time.

Even though the database is internally consistent when a checkpoint is taken (i.e., a complete VERIFY of the database would not detect errors), it does not necessarily follow that the database is meaningful, i.e., that the data itself is correct.

A checkpoint is normally taken automatically when the database is physically closed, when an update-in-place is initiated or when the before image log approaches the end of the file. Additional checkpoints must be initiated by the user either with the GCHPO/SCHPO call or the CHECKPOINT commands.

Even if delayed update or before image logging are not in use, a checkpoint record is always written on the routine log when the database is physically closed. This gives a point of time where the database is consistent and, in case of failure, the possibility to reprocess up to that particular point.

5.3.3 Routine Logging

The routine log (R-log) is essentially a sequential file where the SIBAS input packets (calls) are recorded before they are processed. SIBAS output packets are also recorded. Optionally, returned values from SGET might be omitted to save disk space. Routine logging is specified in the starting procedure of SIBAS and provides a simple and robust reprocessing mechanism.

When routine logging is on, SIBAS input packets from updating run-units are written on the log file. In case of breakdown, this log file can be used in conjunction with a backup copy of the database or a rolled back database, to reprocess the input to SIBAS and bring the database to a state just before the breakdown occurred.

The log file is buffered, i.e., the input packets are not immediately written on the output medium. If the SIBAS process is aborted, the content of the log buffer is lost, and reprocessing brings the database back to a state corresponding to the last written log block.

Some calls force the current log block to be written: UTBLK, SOPDB, SCLDB, SCHPO, GCHPO, BSEQU, and ESEQU. The same calls (except UTBLK) also force the first block (a control block) of the routine log to be written out. It is possible to set a flag which forces the log-block to be written for each call.

Ample facilities are provided to edit, print and select calls on the routine log, providing a useful aid in recovery situations. (See 5.4.10 and 5.4.11.)

If the routine log is used in conjunction with the delayed update or before image option, the file may be used in a circular manner. In this way, one saves disk space but one also loses the possibility of reconstructing the database from a full back-up copy of the database.

The routine log is activated using @SIBAS-SERVICE which initiates/resets or removes the R-log using the INITIATE-LOG command. (The file must first be created by the "database owner".) The name of the routine log file is the same as the database name. The file type is :LOGG. The directory name is given in the INITIATE-LOG command (INLOG call).

Routine log statistics are displayed by the DATABASE-STATUS command under @SIBAS-SERVICE.

5.3.3.1 Critical Sequence

Consider a transaction which updates a price list by updating the unit price for some PARTS records and then updating the SUM record.

If the transaction aborts after updating the PARTS records but before it got a chance to update the SUM record, the price list will no longer be valid.

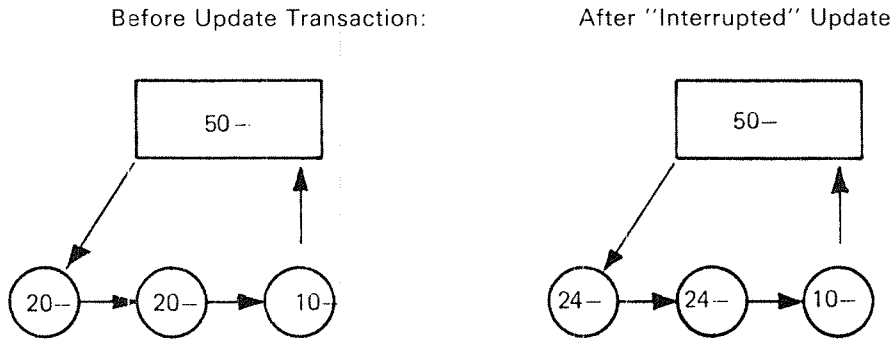


Figure 5.5.

Reprocessing of the database calls will not help in this situation. The transaction contains a "critical sequence" which must be completed once it has started. However the database could be consistent if one stripped off the incompleted sequence from the routine log before it is reprocessed.

This is done by the following mechanism:

In the programs, 2 SIBAS calls must be given:

Call BSEQU (<sequence name>, <time>, <status>) before the sequence begins
Call ESEQU (<sequence name>, <time>, <status>) after the sequence ends

In case of breakdown, this procedure must be followed:

Remove incompleted sequence from routine log file.
Reprocess the routine log.

5.3.3.2 Reprocessing

After a system failure, the routine log (R-log) is used to reprocess the calls to SIBAS, either from a backup copy or a rolled back database.

It is often desirable to avoid reprocessing of some of the calls, for example, all uncompleted critical sequences. Another example is all changes made to the database after a well defined point of time.

Facilities are supplied to list the routine log and/or set conditions for subsequent reprocessing.

When conditions are set, the calls which are not reprocessed will be marked on the routine log, i.e., the routine log is edited. Normally the editing and reprocessing are carried out in one pass. For more explicit control of the editing, one can use more than one pass, for example, one pass to list without reprocessing the ending section of the R-log and one pass to edit and reprocess the R-log. Should the reprocessing fail, it is possible to reinsert the removed calls and reprocess once more.

The editing conditions are set up by the SET-CONDITION-FOR-REPROCESSING command (SICON call); only one condition may be specified for one run-unit. The actual reprocessing/listing/editing is initiated by the REPROCESS command (SREPR call).

5.3.4 Delayed Updating (not available with a SIBAS-500 process)

A database can consist of two parts: the main part which is unchanged and the update file which records all the changes requested for the main part. There may be several subdivisions in this update file, each one corresponding to a checkpoint. The main database is regularly merged with the update file, thereby emptying it (UPDATE-IN-PLACE). An update file is a type of audit trail, but actual updates have not yet occurred. If the system crashes at any point in time, the database is automatically rolled back to the state it had at the latest checkpoint by deleting the updates after the checkpoint (ROLL BACK).

After ROLL BACK to the checkpoint, RECOVERY may be done in two ways:

1. Without ROUTINE LOGGING, by re-executing the run-units (i.e., rerunning the programs). In many situations this is not feasible because the run-unit execution may depend on terminal input which has been lost.

2. With ROUTINE LOGGING, by reprocessing the input to SIBAS from the log.

5.3.4.1 SIBAS I/O System (SIBIO) (not available for SIBAS-500)

The delayed update function uses a separate RT process named SIBIO. SIBIO handles all the disk addresses for SIBAS and manages the update file.

A roll-back can be performed automatically and immediately by SIBIO in case of system failure. Roll-back cannot cross the time of an update-in-place, but all other combinations are allowed (see Figure 5.6).

All communication between SIBAS and SIBIO is performed through a pair of internal devices and RT common.

An application program will not see any difference between a SIBAS with or without SIBIO. The difference will only be visible for the database administrator (or TPS).

The SIBIO buffer size is a system generation parameter, specified in the SIBIO-SYSGN file.

5.3.4.2 The Update File (not relevant for a SIBAS-500 process)

The delayed update option is activated by the @SIBAS-SERVICE program which initiates the update file with the INITIATE-LOG command (the file must first be created by the "database owner"). The name of the update file is the same as the database name. The file belongs to the database "owner". File type is :UPDT. The directory name is given in the INITIATE-LOG command (INLOG call).

The update file is organized as a ring and may contain an unlimited number of checkpoints and update-in-places. The file size is limited to 128 Mb and it is the user himself who sets the size when the file is initiated the first time.

All updates are written to the update file and the user can at any time start the operation called "update-in-place" to write the updates to the database itself. When this is done, a checkpoint is first generated and then all updates done after the last update-in-place and up to the checkpoint are written to the database. Normal SIBAS processing, including updates, can be done even if an update-in-place is active. These updates will be written to the database at the next update-in-place.

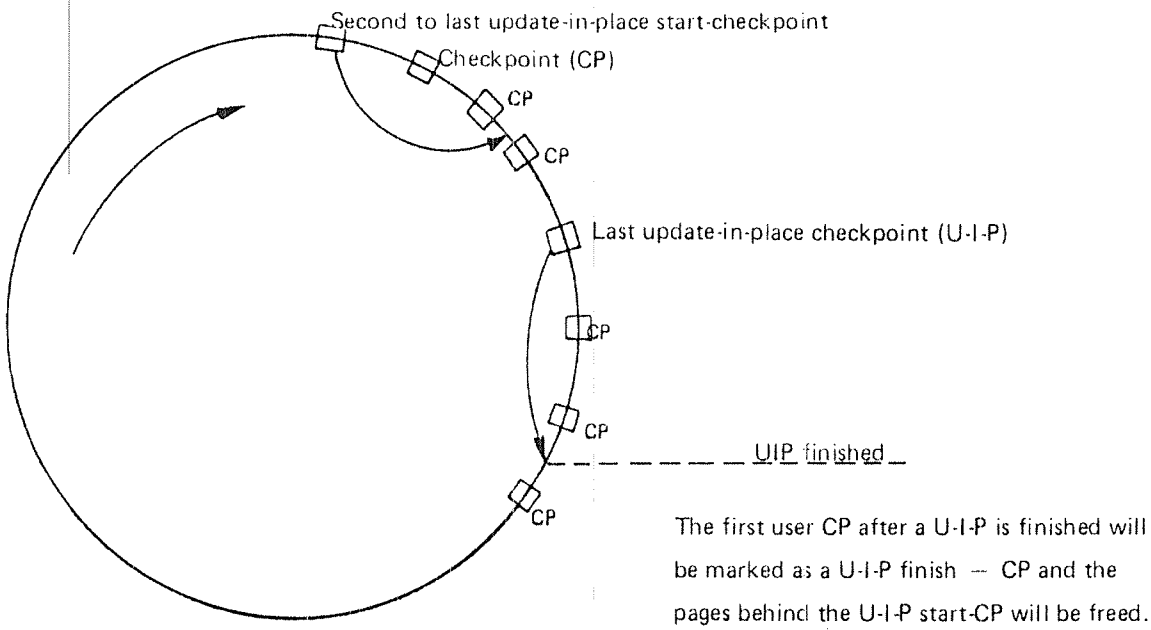


Figure 5.6.

Checkpoint and update-in-place may be called by users at any time, regardless of the validity of the database. It is important that an update-in-place is not initiated while any run-unit is within a single SIBAS call or critical sequence. It is normally the responsibility of the database administrator (or TPS) to halt concurrent run-units in order to initiate update-in-place outside the scope of critical sequences.

An update-in-place should be initiated at fixed intervals to prevent the update file from becoming full. An automatic update-in-place will be initiated if the file is full, but in that case SIBIO will hang until the update-in-place is finished, causing SIBAS and all applications accessing SIBAS to hang for a while.

5.3.4.3 Taking Checkpoints

The user defines the checkpoint identification (normally the SINTRAN time and a sequence number). The identification of the checkpoint must be greater than the previous one.

An immediate start of an update-in-place will also take a checkpoint with the user defined checkpoint identification.

SIBIO also takes its own checkpoints in special cases:

— when starting an update-in-place if the update if full

— when closing the database

These internal SIBIO checkpoints are given a checkpoint identification equal to the last used checkpoint identification + 1 basic unit.

5.3.4.4 Update-In-Place (not relevant for a SIBAS-500 process)

It is the responsibility of the database administrator (or TPS) to initiate an update-in-place. When and how often this is necessary depends on the size of the update file, the degree of changes in the database and how these changes are distributed in time. But with the help of statistics from SIBIO this should be easy to decide. The statistics can be fetched by a @SIBAS-SERVICE command (GET-UPDATE-STATISTICS).

An update-in-place will always increase the load of SIBIO and should therefore, if possible, be initiated when the load of the system is low. The ratio between the updating function and other functions of SIBIO can be decided by the user when initiating an update-in-place.

If an update-in-place is active when closing the database, then the update will be completed before the database will be closed.

The database itself is only open for write access for the duration of the update. Then it is closed and reopened for read access. This means that a backup of the database can be taken by @COPY-FILE while transactions update the database (i.e., the update file).

The backup reflects the situation of the database at the time the last update-in-place was initiated. In other words, a database can be opereated on 24 hours a day.

An update-in-place can either be initiated by the SIBAS-SERVICE program or by a SIBAS call (SUPLA) from an application program. There are three different ways of starting an update-in-place:

1. Take a checkpoint and start update immediately
2. Start update-in-place at next user checkpoint
3. Start update-in-place at close-data-base

The last one (3) implies that the update file will be empty of SIBAS updates when the database is closed and a complete backup of the data base can be taken (SIBIO gives a message on the log console when the file is empty).

If the database is to be used by SIBAS without SIBIO or by the @SIB-DBM or the
@SIB-DRL modules, the update file must be empty of SIBAS updates.

5.3.4.5 Roll-Back

When the roll-back routine is called, SIBIO will perform a quick roll-back to the given checkpoint. If the checkpoint identification given to SIBAS is equal zero (0, 0, 0, 0, 0, 0, 0), SIBIO will perform a roll-back to the most recent checkpoint and return its identification.
A roll-back cannot cross the checkpoint of an update-in-place (see Figure 5.6).

If an update-in-place was active at the checkpoint the data base is being rolled back to, the update-in-place will be reactivated and continue from the state it was in at that time.

5.3.5 Before Image Logging

The Before Image logging is a limited alternative to delayed updating. With this method, a copy of a page is recorded on the Before Image area in the SIBAS SYSTEM REALM just before the page is updated. When a checkpoint is taken, all buffers are flushed out, the state of the SIBAS process is recorded and the Before Image area is emptied.

Should the system crash at any time, the database can be rolled back to the state it was in at the latest checkpoint by copying all "Before Images" out to the database and restoring the state of the SIBAS process.

Checkpointing may be done directly by the user (GCHPO/SCHPO) or be automatically triggered when approaching the end of the Before Image Area. The maximum size of the Before Image Area is limited to approximately 2000 pages (4 Mbytes).

5.3.6 Backup

A full copy (called BACKUP) of the database must be taken at regular intervals. It is often advantageous to combine it with the system backup. A number of older backups and logs may also be retained, to give the possibility to reconstruct the database even if the current database and the backup are damaged.

Full copies of the routine logs and/or update files can also be taken for the same purpose.

5.3.7 System Failure/Restart

Experience to date shows that at one time or another the system will go down. It must be restarted without (too much) loss of information. This is not a trivial matter. It involves:

— Backup frequency, secondary storage capacity, dead time allowable

— Degree of concurrency possible between the run units (this must be taken into account at the application design stage). A high degree of concurrency often implies complicated restart.

— Restart strategy

— Operating procedures

Each of these aspects have sizeable economic consequences.

5.3.7.1 Restart from a Backup Copy and a Routine Log

In case of a system failure, a number of actions must be taken to restart the database:

1. Depending upon the failure; garbage collection, forced close of the data base for all users. Use @SET-UNAVAILABLE and @MAIL to request users to disconnect.

2. Copy the backup to the database.

3. Initiate SIBAS for a reprocessing of the routine log (SREPR call). If critical sequences are in use (BSEQU/ESEQU calls), they must be skipped (see the SICON call).

4. Normal operation again. Use @SET-AVAILABLE and @MAIL to signal to users that the database is operational again.

5.3.7.2 Restart from a Database with Update File/Before Image and Routine Log

1. Depending upon the failure, garbage collection, @MAIL, etc.

2. Since the update file/before image is in use, the database and SIBAS must be rolled back to the most recent checkpoint. Initiate SIBAS for a reprocessing of the routine log from the latest checkpoint. If critical sequences are in use, they must be skipped.

3. Normal operation again.

5.4 DETAILED DESCRIPTION OF THE CALLS

The calls described in this section are concerned with the operation of the SIBAS process rather than manipulating data to or from the database itself. More specifically, many of the statements operate at the SIMULATOR/LIBRARY or the INTERFACE level without any action at a lower level (see Figure 5.2 and 5.3). These calls should be executed through SIBAS-SERVICE (see 6.2). When they should not be,the opposite is explicitly indicated below.

Some parameters have the same meaning in several calls. A detailed description of them will be given here to avoid tedious repetition.

<status>

A single integer returned by the SIBAS process to indicate the result of an operation.

1 means a successful execution

< 0 means an unsuccessful execution. A list of the possible values and their meaning is given in the chapter ERROR REPORTING.

<run-id>

A single integer, used by the SIBAS process to identify a run-unit (a user) in its user table. Normally the start address of the RT-DESCRIPTION of the run-unit is used. This parameter will often be zero indicating all run-units. To get the run-ids, you may run REPROCESS-ROUTINE-LOG with print-option 3 (print only).

Remark: When using @SIBAS-SERVICE, input of run-id is octal and must be terminated by "B", for example, 23456B.

<sequence-name>

A field of 8 bytes containing the name of a critical sequence. The name is freely chosen by the programmer and consists of 1 - 8 characters.

<time>

A field of 7 single integers which has the following format:

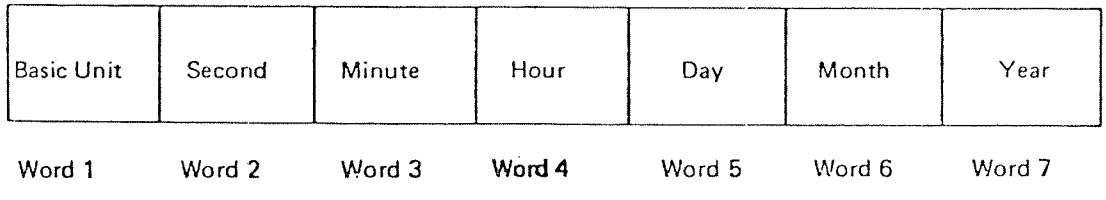


Figure 5.7.

<checkpoint-id>

A field of 7 integers used by SIBAS to identify a checkpoint. It has the same format as <time>.

<owner>

A field of 8 bytes identifying the user that "owns" the database files and the routine log file.

<log directory>

A field of 4 bytes specifying on which directory the routine log file is to be found. If this field contains spaces, the default directories will be used.

5.4.1 START/STOP SIBAS/GET-STATE

START

Function:

Make a SIBAS process ready to be run, i.e., change SIBAS from the READY state to the DBA state.
Note that the SIBAS process must be activated before execution of this CALL. We recommend activating it by giving system no. in SIBAS-SERVICE (see 6.2), but the command @RT SIB2x from user RT also does it. (Here x must be one of the letters A-F)

Format:

CALL START (owner, data-base-name, work-area-size, status)

Rules:

"work-area-size" is the size of the buffer (in Kilo-words) the SIBAS process will use. SIBAS performances are directly related to this parameter. The bigger it is the better, provided there is enough physical storage (as opposed to virtual). It must be between 7 and 32.

"Work-area-size" is dummy on SIBAS-500.

STOP

Function:

Change SIBAS from the DBA state to the READY state.

Format:

CALL STOPS (status)

GET-STATE

Function:

Read the SIBAS STATE code (see Figure 5.3).

Format:

CALL STGET (state, status)

Note that a succesful execution of a STGET call using a Sibas-500 process will return STATUS = 500 (Sibas-100 will return STATUS = 1)
Return state has the following meaning:

"state" = 0 if READY
1 if DBA
2 if RUNNING
3 if RECOVER

5.4.2 RUN/PAUSE/RECOVER/FINISH/SET PASSIVE/ REPRO-STATUS

RUN

Function:

Change the SIBAS process from the DBA state to the RUNNING state.

Format:

CALL SRUN (flag word, status)

"Flag word" contains flag bits with the following meaning:

Bit 0 = 1 returned values from SGET will not be logged
Bit 1 = 1 the OFLOG/ONLOG calls are allowed
Bit 2 = 1 SRRLM, SFRLM are allowed when log is turned off
Bit 3 = 1 SLOCK, SUNLK are allowed when log is turned off
Bit 4 = 1 SINSR, SREMO are allowed when log is turned off
Bit 5 = 1 SCONN, SDCON, SCONA, SCONB are allowed when log is turned off
Bit 6 = 1 STORE is allowed when log is turned off
Bit 7 = 1 SMOFY, SEREL are allowed when log is turned off
Bit 8 = 1 SRASE is allowed when log is turned off
Bit 9 = 1 GCHPO is allowed when log is turned off
Bit 10 = 1 SEXMC is allowd when log is turned off
Bit 12 = 1 SIBAS is set unavailable
Bit 13 = 1 no buffering of the routine log
Bit 14 = 1 all user logged regardless of the <mode> parameter in SOPDB
Bit 15 = 1 database updating is not allowed.

Flag word = —1 if the old flag word is to be used.

PAUSE

Function:

Change the SIBAS process from the RUNNING state to the DBA state.

Format:

CALL SPAUS (status)

RECOVER

Function:

Change the SIBAS process from the DBA state to the RECOVERY state.

Format:

CALL SRECO (status)

FINISH

Function:

Change the SIBAS process from the RECOVERY state to the DBA state.

Format:

CALL SFINI (status)

Rules:

SFINI is not correctly executed until all reprocessing error codes are read using the call to STREP.

REPRO-STATUS

Function:

Get the status of reprocessing (from D version this call is DUMMY).

Format:

CALL STREP (status)

Rules:

This call can be repeatedly used from your program until SIBAS has returned all status information about the reprocessing. Calls to SFINI will give error status until all the reprocessing statuses are read by STREP.
From D version STREP will always return status = 1, but the call SREPR (reprocess) returns reprocessing status.

SET-PASSIVE

Function:

Change the SIBAS process from the READY state to the PASSIVE state.

Format:

CALL SPASS (status)

5.4.3 INITIATE-LOG

Function:

Define/remove or reset the log files a SIBAS process will use.

Format:

CALL INLOG (owner, database name, code, log directory, type, number of pages, status)

Rules:

"number of pages" gives the size of the log file in the number of 1K word pages.

"code" = 1 init routine log
2 reset routine log
3 remove routine log
4 init delayed update page log (not available for SIBAS-500)
5 init before image page log
6 remove page log

"type" is used only when code = 1:

1 magnetic tape routine log
2 direct routine log (noncircular)
3 circular routine log

In case circular routine log is *not* selected, the system will stop if the log becomes full.

5.4.4 BEGIN/END SEQUENCE

Function:

These statements have to do with recovery. They signal in your program the beginning and the end of a sequence of statements which are interdependent. If the system goes down in the middle of a sequence, it is possible when reprocessing to undo the partly executed sequence or sequences specified by the SICON call.

Format:

CALL BSEQU/ESEQU (sequence name, status)

Rules:

Routine logging must be in effect and the database opened for update, otherwise a negative status is returned.

"Sequence name" is an 8 character field just as other SIBAS names. This name is chosen by the user and identifies the critical sequence.

The "sequence name" and the time will be logged on the routine log file and the current log buffer will be written on the disk, ensuring a consistent basis if recovery is needed.

Those calls make it possible for SIBAS to edit the routine log during recovery. Critical sequences cannot be nested within the same run-unit. This technique requires little overhead, but to work properly one must carefully analyse the applications in order to avoid interactions between concurrent sequences. See the sections on Concurrent Processing in Chapter 2.

5.4.5 SET ROUTINE LOGGING ON/OFF

Function:

These statements have to do with recovery. They may be used in your program to lower the volume of the routine log and consequently speed up recovery if reprocessing is needed.

Format:

```
CALL OFLOG (status)
CALL  ONLOG (status)
```

Rules:

OFLOG defines the start of a section and ONLOG defines the end of a section in which the logging is not in effect for the calling run-unit. Routine logging must be in effect, otherwise an error status is returned.

The use of OFLOG/ONLOG imposes severe restrictions on the run-unit logic. A section that starts with an OFLOG and terminates with ONLOG, must be completely unrelated to sections using logging. Reprocessing without the section must give the same database changes as reprocessing with the section.

Remembered and current records and remembered and current search regions set up before and within the scope of an OFLOG/ONLOG section cannot be after the section and are automatically removed from currency table by the ONLOG call.

The ONLOG statement is automatically executed when CLOSE-DATA-BASE is executed.

Several "updating" calls can be made legal in OFLOG mode by the setting of the run flag (see RUN).

5.4.6 LOG MESSAGE

Function:

This statement has to do with recovery. The given message will be written on the routine log file and it will be printed in case of recovery.

Format:

CALL SMESS (length, message, status)

Rules:

If the R-log is not active, the message will only be written out on the SIBAS error device. The status will be 1.

"Length" is the number of words the "message" contains. The message will also be printed on the SIBAS console.

This statement may be used from your program to signal the status of different tasks and may simplify the recovery procedure.

5.4.7 WRITE-LOG-BUFFER-ONTO-ROUTINE-LOG

Function:

This statement has to do with recovery. It forces the writing of the log buffer onto the routine log file when used in your program.

Format:

CALL UTBLK (status)

Rules:

Routine logging must be in effect. The routine log file is buffered for performance reasons. Such a buffer may contain 10 to 30 calls. If the system crashes, the buffer is lost. In some situations, it is required to write out the buffer to ensure consistency of the database in case of recovery.

This statement is automatically executed at BEGIN/END SEQUENCE, CLOSE DATABASE or CHECKPOINT and when SIBAS is normally stopped.

5.4.8 CHECKPOINT

Function:

The CHECKPOINT statement defines a point on the log file(s) where the data base is consistent. In case of a fatal error, the database may later on be returned to the state it had when the checkpoint was taken.

Format 1:

CALL SCHPO (checkpoint-id, status)

Format 2:

CALL GCHPO (checkpoint-id, status)

Rules:

Delayed update or before image must be in effect when these statements are executed, otherwise only the routine log buffers are written out (UTBLK).

The run-unit must have update access ("mode" parameter in SOPDB)

In the case of format 1, SCHPO will return a "checkpoint-id" generated by the DBCS.

In the case of format 2, GCHPO will accept a "checkpoint-id" generated by the run-unit.

When the delayed update option is in effect, the last run-unit executing a CLOSE DATABASE will trigger the execution of a CHECKPOINT statement.

This statement is costly and should not be used too often from your program. It forces the writing of all modified database pages from the buffer area to the disk.

Synchronization of concurrent run-units and restart strategies of run-units are not the topic of SIBAS. However, Norsk Data offers a comprehensive Transaction Processing System (ND TPS) which deals with such questions.

Note: the database is not necessarily valid at checkpoint time because some transactions may not be finished.

Return-status = 0 means that BIM-log, or Delayed update, is not active. R-log has been, nevertheless, checkpointed.

The runflag must allow the GCHPO to be executed. (See SRUN, Section 5.4.2)

5.4.9 ROLL-BACK

Function:

This statement reestablishes the database state at a previous checkpoint.

Format:

CALL SROLL (checkpoint-id, dbname, password, status)

Rules:

The delayed update option or before image log must be in effect, otherwise the statement will be ignored.

If the "checkpoint-id" is equal to 7 zeros, the database will be rolled back to the latest checkpoint. If the "checkpoint-id" is not null, the database will be rolled back to a checkpoint with the given time and date or the closest earlier checkpoint. If the routine log is in effect, it is also rolled back and will be ready for reprocessing from that point.

If the before image log is in effect, there is only one checkpoint on the log, the latest one.

5.4.10 SET-CONDITIONS-FOR-REPROCESSING

Function:

Specify which calls on the routine log are to be included in reprocessing. This must be done for each individual run-unit. The reprocessing condition for each run-unit is put into the reprocessing control table which will be used when the SREPR call is given (see below). The specified calls will then be processed as indicated by the SREPR call. (See 5.4.11)

Format:

CALL SICON (code, run-id, time, sequence-name, status)

Rules:

"Code" defines the condition to be set. Different conditions may be set up after each other by calling SICON as many times as necessary.

"Time" is an array of 7 words containing the time and date of a critical sequence or a checkpoint.

"Sequence-name" is the name of a critical sequence.

"Code" may take different values:

0 release control table entry for the run-id (to remove earlier SICON call settings for the run-id)

1 remove the sequence identified by "time" for the run-id

2 remove all the sequences identified by "sequence name" and run-id

3 remove all the sequences identified by "sequence name" and run-id executed after "time"

4 remove all calls for the run-id

5 remove all calls for the run-id from beginning of "sequence-name"

6 remove all calls for the run-id from the beginning of the sequence identified by "time"

7 remove uncompleted sequences for the run-id or for all run-units when run-id = 0.

A call with code 7 must be executed prior to any other recovery action.

Then you may call SICON for specific run-ids (code 1-6). If you want to call another SICON call for one of the run-ids, you must first release its entry by a SICON call with code 0. Only the last SICON call for a specific run-id will be processed by SREPR.

NB! A SICON call with code 7 can *not* be executed after rollback.

5.4.11 REPROCESS-ROUTINE-LOG

Function:

Process, reprocess and/or print the calls which were recorded on the routine log file according to the conditions specified in the call and the conditions set up by the preceding SICON calls. Usually the STANDARD-REPROCESS command in SIBAS-SERVICE can be used instead of SICON and SREPR calls (see 6.2.2).

Format:

CALL SREPR (condition, mode, time, no.-call, print-option, run-id, remove-flag, status)

Rules:

After some conditions for reprocessing have been set up (by SICON) one can reprocess and/or print all or parts of the routine log file. Further options may be specified:

"condition"

0 = process to end of file or "no. call"
1 = process but remove all critical sequences starting after "time"
2 = process up to a checkpoint identified by "time" or later
3 = process up to a log block written by "time" or later.

"mode"

0 = continue processing (with "mode" parameter as before the previous reprocessing stopped)
1 = start reprocessing without print
2 = start reprocessing and print
3 = start print only (not reprocess)
4 = start print short (not reprocess)
—1, —2, —3, —4 as 1, 2, 3, 4 but means continue processing from where the previous processing stopped, but with new "mode" parameter. *Note: The "continue processing" facilities could be dangerous!*

"no. call"

0 = means all the calls
not 0 = specifies the maximum number of calls to reprocess

"print-option"

specifies print-option *if printing is on* (see "mode")
1 = print only candidates to remove/reinsert
2 = print all the calls
3 = print only checkpoints
4 = print begin and end sequence and checkpoints

"run-id"

Select one run-unit to print. If the value is zero, all run-units are processed according to the selected options.

"remove-flag"

= 1 remove the calls according to the reprocessing control table (while reprocessing)
= —1 reinsert the calls according to the reprocessing control table (when they have been previously removed)

5.4.12 UPDATE-DATA-BASE-IN-PLACE

Function:

Apply the updates on the update file to the database files and release space on the update file.

Format:

CALL SUPLA (update-ratio, trigger-code, checkpoint-id, status)

Rules:

Not available for a SIBAS-500 process.

All the pages which were changed up to the checkpoint will be moved from the update file to the original database files. Before this operation is initiated, a checkpoint is automatically taken, insuring the consistency of all disk files.

"Update-ratio" indicates the number of pages to be updated while a certain load of normal activity is carried on. A high number will almost hang up any other activity but the updating process will terminate sooner.

"Trigger-code" indicates when the update will be initiated

0 immediately
1 at the next checkpoint
2 at the physical close of the database (the last user that closes the data base

SIBAS may be running at a lower speed than usual while the data base is being updated.

5.4.13 SET SIBAS SYSTEM NUMBER

Function:

Set the SIBAS system number to be accessed by the subsequent calls in the run-unit.

Format:

CALL SETDV (SIBAS system number)

Rules:

"SIBAS system number" is an integer value specifying which SIBAS process the run-unit will use. If this call is not given, the default SIBAS process accessed is 0 (SIB2A). It is a good programming practice to include this call as the first SIBAS in all your application programs. It is necessary to include it in case your database was not connected to process 0 (SIB2A) when the SIBAS process was called.

5.4.14 RESERVE/RELEASE SIBAS

Function:

Reserve SIBAS for exclusive use for this run-unit; other run-units will not be allowed to access SIBAS facilities at all. RELEASE-SIBAS signals the other run-units that SIBAS may now be accessed.

Format:

CALL RESIB/RELSI (status)

Rules:

These calls permit the elimination of concurrent processing problems since the whole database becomes unavailable to other run-units. These calls should be used very carefully. If the run-unit has reserved SIBAS but does not release it for some reason (a coffee break is one), all other run-units will hang. See the section "Concurrent Processing".

SCLDB will automatically release SIBAS.

5.4.15 EXECUTE-MACRO

Function:

SIBAS statements may be extended by a user written subroutine which may in turn call normal DML statements, thus providing a way to complement user defined "MACROs". An execute-macro statement will be executed uninterrupted by other run-units. Another advantage of using this facility is that communication overhead is reduced.

Format:

CALL SEXMC (input, length of input, output, length of output, status)

Rules:

SIBAS-500 Macros: See Section 5.5.4.

The use of this statement implies that the SIBAS process is loaded with a user written subroutine which has the same name (SEXMC) and parameters as described in the format. The loading of the SIBAS process is specified in the SIB-SYS-GEN: BATC file.

"Input" contains the values which will be passed to the user subroutine loaded with SIBAS. "Length of input" is the number of words the values occupy altogether. "Output" will contain the values returned by the user macro to the run-unit. "Length of output" is set by the user defined macro. "Status" is also set by the macro.

A number of restrictions apply when writing such a subroutine:

- It must be written in FORTRAN and compiled reentrant except for SIBAS-500. (For SIBAS-500 the source code should be included in the load-file, which in turn will trigger the compiling.)

- There is a limited size to both the code and stack requirements. The limits are given in the SIB2-LOAD: BATC file (only for ND-10, ND-100, SIBAS-100).

- There can be no terminal input or output.

- The subroutine requires exclusive use of SIBAS and while it is executed other users must wait.

Note:

More than one "macro" can easily be implemented by using one of the input values as switch parameter, choosing a selected execution path.

5.4.16 DBA Calls

Function:

Maintenance and timing.

Format:

```
CALL CHCOM (status)
CALL STRLG (terminal-number, mode, status) (not available for SIBAS-500)
CALL SISTA (values, status)
CALL SERVC (function, input-values, length-of-input, output-values,
length-of-output, status)
CALL RBLAN (index, output value, status)
CALL ZTRB (length, index, output value, status)
```

Rules:

CHCOM makes SIBAS switch to an alternative communication procedure and releases the old one (for TPS use only).

STRLG turns on/off the printing of the trace of SIBAS calls.

"Mode" = 0, turn off the terminal log.
"Mode" = 1, turn on the terminal log
"Mode" = 2, turn on the terminal log and a special internal SIBAS trace and debug. Mode 2 uses the console device for input; this device must therefore be free (logged out).

SISTA, SERVC, RBLAN, and ZTRB are privileged calls used by @SIBAS-SERVICE.

5.4.17 FORCE-CLOSE Database

Function:

Performs close database for given run-unit or all run units.

Format:

```
CALL SABOR (database name, status, user-id)
```

Rules:

User-id = —1 means all users with database opened.

Remark:

This call is logged and executed as one or more calls to SCLDB.

5.5 SPECIAL SIBAS-500 FEATURES

In this chapter we will present some of the special features concerning SIBAS-500. *Most of these features are available only on SIBAS-500* at present.

5.5.1 Calls with Different Functions

Applications which need to know whether they are using a SIBAS-100 or a SIBAS-500 process may get this information through the STGET call ("get-sibas-state"). The returned *status* will always be 500 if the SIBAS process is a SIBAS-500 process and the call was successfully executed (not successful implies status <: 1).For SIBAS-100 the returned status of a successfully executed STGET call will always be 1.

The "work-area-size" in the START call has no function at all. Any legal number will do, but it will not have any functional meaning since 32K will always be used on SIBAS-500. (SIBAS-service requires the "work-area-size' to be specified in the START-DATABASE and SUPER-START commands. Any dummy number will do if SIBAS-500 is used).

SETDV may be used as a function when called from a Fortran program. The function value will be 1 if the call was successfully executed. This is a useful solution since SETDV has no returned status-parameter. As a good programming practice, programmers are advised to always include SETDV as the first SIBAS-call in their applications.

As we already have pointed out, the length-of-value-parameter *"value-length"* (the last parameter in SFTCH, SFEBL, SFLBL, SMDFY and STORE) specifies length in number of SIBAS-words, i.e., number of 16 bit words. This implies no differences for the same application running on the 100 or 500 CPU.

5.5.2 Calls not Available

The following calls are at present *not* available from SIBAS-500:

ACCFD	% accumulate floating
STRLG	% printing of trace
SIBIO-calls	% all calls conserning SIBIO

5.5.3 Exceeding the Size of a Direct Routine Log

Whenever the maximum limit of a *direct* routine log (call log/R-log) is exceeded on a SIBAS-500 process, i.e., the routine log is full, no more users will be allowed to open the database. Negative status will be returned. (See 4.2.1) The DBA will get a special message through the DATABASE-STATUS command in SIBAS-service telling him to reset or remove the routine log, and a message will be printed on the SIBAS error-device. Applications running (with the database opened) will be allowed to finish their work until close-data-base is called.

5.5.4 SIBAS-500 Macros

SIBAS macros (SEXMC) can very easily be used in conjunction with SIBAS-500. Macros are normally used to include two or more normal DML statements (e.g., SFTCH and SMDFY) as one unit.

An execute-macro statement in an application program (i.e., CALL SEXMC) will be executed uninterrupted by other run-units, and there will be no intermediate communication between SIBAS and the application transmitting the call. Thus communication overhead will be reduced. The user-written SEXMC Fortran source-routine has to be included in the appropriate SIBAS-500 load file, instead of the default DUMMY-SEXMC routine.

Let us say that a user-written SEXMC routine, residing on the file USER-SEXMC:SYMB, is to be included into SIB2A-500. The only operations required would be to simply substitute all occurences of DUMMY-SEXMC with USER-SEXMC in the file SIB2A-500:LOAD, and then (re)run that mode-file. All formal parameters should be declared (default) INTEGER. Compilation inside the mode-file will force the correct mode.

The actual parameters, *input* and *output,* must be declared INTEGER * 2 in the application program (i.e. they are handled as *value buffers*).

Note that 'length-of-value-parameters' (i.e., "key-length" and "value-length") must be omitted for DML statements residing in a SIBAS macro. (For a detailed description of parameters etc., please refer section 5.4.15).

5.6 HOW TO INSTALL SIBAS

The procedure is explained on the sheets attached to the diskettes.

6 UTILITIES

6.1 DATABASE MAINTENANCE MODULE

6.1.1 Introduction

The DBM module is a tool which enables the Database Administrator to control the efficient and reliable use of the database. The functions included in the DBM module are shown in the figure below.

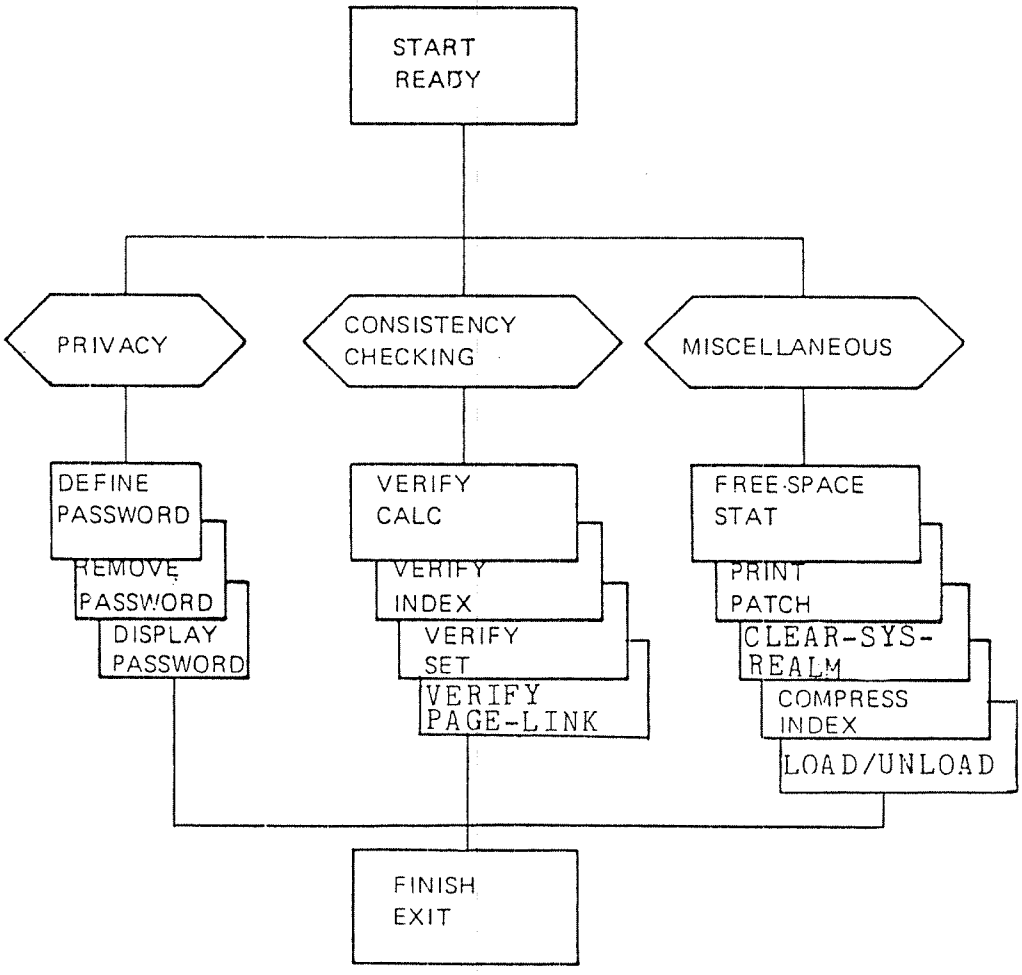


Figure 6.1: Database Administration Functions

The DBM module requires exclusive use of the whole database and accesses the realms directly without using a SIBAS process at all. In fact, the DBM module and the SIBAS processes mutually exclude each other when attempting to open the database.

The use of any of the DBM functions is controlled by the START statement. In this statement a DBA password may be provided, and the validity of this password is checked on the database. This prevents the unauthorized use of the DBM module on a particular database.

Syntax Description

The database maintenance statements are written in a syntax where key words and parameters must appear in a defined order in the same way as for COBOL.

The syntax is phrase oriented and all statements must be terminated by period "." and carriage return.

Throughout this chapter, wherever a statement is described, these conventions are used:

KEY	the key word KEY must be present
KEY	the key word KEY is optional
|A| |B|	the key words A or B must be present
<par>	"par is a required parameter
(<par>)	"par" is an optional parameter

Parameter Values

Parameter values may be SIBAS names, integers or pointer values.

Abbreviation Lookup

All key words (not parameters) can be abbreviated. However, ambiguity is not handled. The first match is always used.

Octal Numbers

All octal numbers must have 0 as first digit. Otherwise, the typed number is treated as decimal.

Pointers

All pointers contain two machine words, typed as two octal numbers separated by "*".

Example:

000400 * 012345

6.1.2 START

Function:

The function of this statement is to indicate the user's intention to process database maintenance statements and to check that the user is allowed to do so.

Format:

START <database-name> (<dba-password>) .

Rules:

1. "Database name" is the name of the database as given in OPEN-DATABASE.

2. If privacy is defined for the database, the "dba password" will be checked to decide whether or not the user is allowed to process DBM statements. (See 6.1.9.)

3. The effect of this statement is to physically open the database.

6.1.3 EXIT, STOP THE DBM MODULE

Function:

To prevent the further processing of database maintenance statments apart from START.

Format:

STOP.

or

EXIT.

Rules:

1. The effect of this statement is to physically close the database.

2. Realms previously readied with READY statement are automatically finished by STOP.

6.1.4 READY REALMS

Function:

This statement indicates to the DBM MODULE the user's intention to process records on one or more realms.

Format:

```
READY  ! REALM  <realm-name> !  .
       !        ALL          !
```

Rules:

1. The effect of this statement is to ready the realm "realm name" or all the realms in the database for exclusive update.

2. This statement must be successfully executed before any PRINT, PATCH or VERIFY statement may be executed.

6.1.5 FINISH REALMS

Function:

To prevent further processing of the data on one or all realms.

Format:

```
FINISH  ! REALM  <realm-name> !  .
        !        ALL          !
```

Rules:

1. The effect of this statement is to prevent further use of the referred realms for PRINT, PATCH or VERIFY.

2. The STOP statement automatically finishes all realms.

6.1.6 PRINT

Function:

To print the content of the specified units of information in a formatted dump form on the terminal.

Format:

```
PRINT  ! <number> !  ! RECORD !  ! REALM <realm-name> (<from-unit>) !  .
       ! ALL      !  ! PAGE   !  ! POINTER  <address>                !
                     ! WORD   !
                     ! BUCKET !
```

```
PRINT  POINTER  <address>  .
```

Rules:

1. "number" is an integer. If neither "number" nor ALL are specified, it is assumed that "number" is equal to 1. Note that the maximum number of records printed by one PRINT command is 100.

2. A unit may be specified as a database address or a word address within the realm "realm name".

3. When RECORD is specified, all records within the defined range are printed, deleted records as well as active records.

4. All the realms involved must be readied prior to PRINT.

5. "from-unit" specifies the start for the dump as a BUCKET, PAGE, RECORD, or WORD number. PAGE counting begins at 0. RECORD, BUCKET, and WORD counting begins at 1.

6. "Address" may be specified in decimal or octal, an "address" starting with 0 (zero) being treated as octal.

6.1.7 PATCH

Function:

To replace one word in the database.

Format:

```
PATCH  ! REALM <realm-name> <page-no> !  <word-disp>
       ! <address>                    !
```

Rules:

1. The use of this statement implies a very good knowledge of how a SIBAS database is built up internally and should only be used in extreme cases.

2. The page containing the word to be replaced is identified either as an absolute address in the database or by giving the realm-name and the page number within the realm.

3. "Word-disp" identifies the word to be patched in the page. Word displacement starts at zero.

4. Carriage return (↵) must be given after the page and word are identified. The value of the identified word will then be printed as follows:

```
PATCH AT POINTER: aaaaaa*aaaaaa
 OLD VALUE: vvvvvv  OCTAL NEW VALUE:
```

   where aaaaaa xaaaaaa = the absolute address and vvvvvv = the old value.

   The new value must then be given followed by ↵. The replacement will be made and the value of the next word printed. This word can then be given a new value, etc.

5. The new value must be specified as a 6 digit octal number, for example 000001 for 1.

6. Giving just ↵ for the new value will give a new value of zero.

7. Giving . (period/full stop) and ↵ for the new value will terminate the patching session.

8. Since no logging takes place while the DBM module is under execution, it may be necessary to take new copies of all or part of the data base after use of the PATCH function.

6.1.8 RESET-ERROR-FLAGS

Function:

To reset all the "DATABASE IN ERROR MODE" flags.

Format:

RESET-ERROR-FLAGS.

Rules:

1. This statement is reserved. Note that it does not repair the database which might still be in error.

6.1.9 PRIVACY

6.1.9.1 GENERAL

The privacy system enables the DBM to restrict the use of the database to authorized users. This is done by defining passwords for the data base or a part of the database. Privacy can be defined on 2 levels:

1. Privacy on the database level.
2. Privacy on the record occurrence level.

The privacy functions of the DBM module are used to define and give values to passwords on the database (Figure 6.2). The data definition/redefinition language is used to define privacy on the record occurrence level, and the data manipulation language is used to give values to the privacy items in each record occurrence (Figure 6.3).

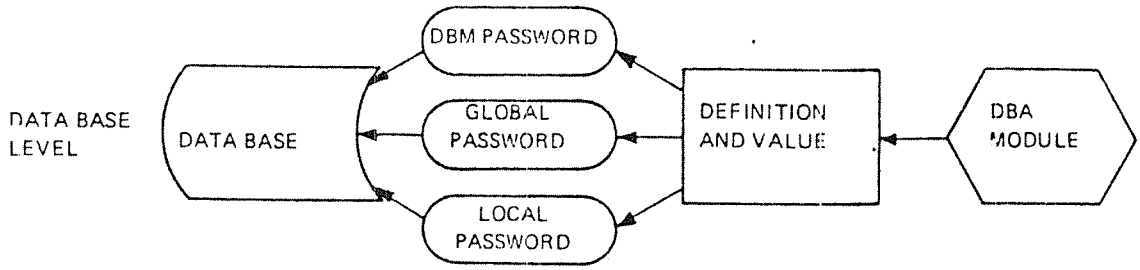


Figure 6.2: Defining and Giving Values to Passwords on the Database Level

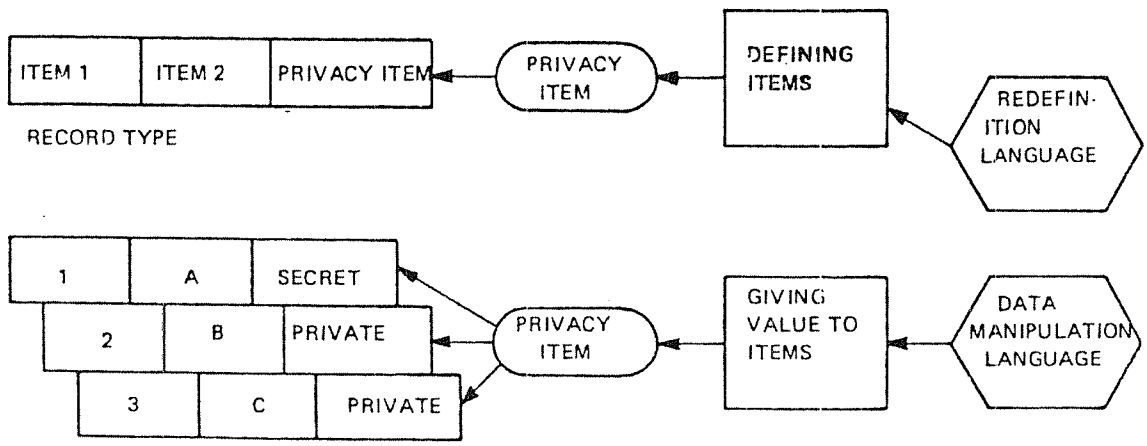


Figure 6.3: Defining and Giving Value to Privacy Items on the Record Occurrence Level

If privacy is defined on the database level for a database, each run unit must give in a password with the OPEN-DATA-BASE statement. The validity of this password will be checked and, if it is valid, it will remain "current password" for the run unit until CHANGE-PASSWORD is used to update the current password.

If privacy is defined on the record occurrence level, each run unit's current password will be checked for validity when the run unit attempts to execute a MODIFY, ERASE, CONNECT, INSERT or GET on a record. The current password must match the value of the privacy item in the record. In case of ERASE, all records to be erased in a single ERASE statement are checked.

A DBA password may be defined in addition to other passwords. The DBA password will allow a user (database administrator) to execute START and to perform any of the functions included in the DBM MODULE and any DML statement.

Table 6.1 shows how privacy restrictions on a database are defined, how and when passwords may be defined and modified, and when the privacy checks are performed by the SIBAS data manipulation routines.

Type of privacy	How privacy is defined	How passwords are given values	How passwords are modified	When the validity of current password is checked
DBA password	Using DBM module	Using DBM module	Using DBM module	At execution of open data base, ready realm, start DBM module
Database level	Using DBM module	Using DBM module	Using DBM module	At execution of open data base, ready realm
Record occurrence level	Using definition/redefinition language	When a record occurrence is stored	When a record occurrence is modified	At execution of modify, get, erase, connect/disconnect, insert/remove

Table 6.1: Defining and Controlling Passwords

Local and Global Passwords

Passwords defined on the database level can be either local or global.

A local password on the database level is valid for OPEN-DATA-BASE only. If privacy is defined on record occurrence level a new current password may have to be given before GET, MODIFY, ERASE, CONNECT, DISCONNECT, INSERT or REMOVE can be executed.

A global password on the database level is valid for OPEN-DATA-BASE. In addition, it will allow the run-unit to execute READY REALM, with the USAGE MODE and PROTECTION MODE defined for the password, on any realm in the database. It will also allow the run-unit to execute other DML statements, regardless of the value of the privacy item in each record (MODIFY and ERASE can only be executed if the realm was readied with usage mode CONNECT, DISCONNECT, INSERT, and REMOVE, can be executed if the realm was readied with usage mode LOAD).

The DBA PASSWORD is a global password on the database level with usage mode UPDATE and protection mode EXCLUSIVE.

Usage Mode and Protection Mode

USAGE MODE and PROTECTION MODE must be defined for global passwords on the database level. The possible usage modes are RETRIEVAL, LOAD and UPDATE. The possible protection modes are NON-PROTECTED and EXCLUSIVE.

Table 6.2 gives a summary of the functions allowed for different types of passwords, assuming that privacy is defined on both levels.

TYPE OF PASSWORD / FUNCTION			DBA PASSWORD	DATABASE LEVEL							
				LOCAL	GLOBAL						
					NON-PROTECTED			EXCLUSIVE			
					RETRIEVAL	LOAD	UPDATE	RETRIEVAL	LOAD	UPDATE
OPEN-DATA-BASE			X	X	X	X	X	X	X	X
READY REALM	NONPROTECTED	RETRIEVAL	X		X	X	X	X	X	X
		LOAD	X			X	X		X	X
		UPDATE	X				X			X
	EXCLUSIVE	RETRIEVAL	X					X	X	X
		LOAD	X						X	X
		UPDATE	X							X
STORE			X			X	X		X	X
GET			X		X	X	X	X	X	X
MODIFY, ERASE-ELEMENT			X				X			X
ERASE			X				X			X
INSERT, REMOVE, CONNECT DISCONNECT			X			X	X		X	X
START DBM MODULE			X							

Table 6.2: Functions allowed for Different Types of Privacy

Summary of the Setting of Current Password

Initially, the current password is set for a run-unit when the database is opened. Unless a CHANGE-PASSWORD statement is performed, the value of the current password will remain unchanged. When a READY-REALM statement is performed, the current password must match a password which is defined for the desired mode of operation on the realm. If the run-unit performs a data manipulation statement on records where the value of the privacy item is different from the realm password, the current password for the run-unit must be changed to match the value of the privacy item before the data manipulation statement is successfully executed.

6.1.9.2 DEFINE PASSWORD

Function:

The function of this statement is to register a new password. Passwords can be three different types, and for one of them USAGE MODE and PROTECTION MODE is given with the password.

Format:

This statement has 3 different formats, one for each password type:

```
DEFINE  DBA-PASSWORD  <dba-password>  .

DEFINE  LOCAL-PASSWORD  <password> DATABASE.

DEFINE  GLOBAL-PASSWORD  <password>  DATABASE

                ! RETRIEVAL !                    ! NON-PROTECTED !
USAGE           ! LOAD      !  PROTECTION        ! EXCLUSIVE     !  .
                ! UPDATE    !
```

Rules:

1. LENGTH OF PASSWORDS. All passwords must follow the same conventions as SIBAS names, i.e., up to 8 bytes, starting with a letter, no embedded blanks, but trailing blanks allowed.

2. DBA-PASSWORD. When a "dba-password" is defined for a data base it must always be given with the START statement. The "dba-password" will also serve as a GLOBAL-PASSWORD on a DATABASE with USAGE UPDATE and PROTECTION EXCLUSIVE. This implies that the "dba-password" also allows one to execute any DML statement in addition to the START statement.

3. LOCAL PASSWORD ON DATABASE. "Password-1" will serve as a local password on the database level. The validity of this password is restricted to the OPEN-DATA-BASE statement, and records where the value of the privacy items happens to coincide with the passwords.

4. GLOBAL PASSWORD ON DATABASE. "Password-2" will serve as a global password on the database level. In addition to its use with the OPEN-DATA-BASE statement, the password will be valid for the execution of READY-REALM with the USAGE mode and PROTECTION mode given, and for the execution of any other DML statements covered by the usage mode.

6.1.9.3 REMOVE PASSWORD

Function:

The function of this statement is to remove a single password defined on data-base level or to remove all privacy defined on database level.

Format:

```
REMOVE-PASSWORD   ! <password> !   .
                  ! DATABASE   !
```

Rules:

1. REMOVE-PASSWORD "password". This operation will remove the password given in "password" from the list of passwords on data base level.

2. REMOVE-PASSWORD DATABASE. This option will remove all passwords defined on the database. It will also remove the DBA password.

6.1.9.4 DISPLAY PASSWORD/PRIVACY

Function:

The function of this statement is to print the values and the description of all or some valid passwords on the terminal.

Format:

```
DISPLAY    ! ALL  PRIVACY          !   .
           ! PASSWORD <password>   !
```

Rules:

1. If the ALL option is given, a complete report is printed containing the values of all passwords defined for the database. The report will also contain the type, usage mode and protection mode for each password.

2. If the PASSWORD option is given, the type(s), usage mode and protection mode for the password specified are given. All definitions of the password with the given value will be printed.

6.1.10 Index Compression

Function:

Index tables are dynamically read, written upon or updated by SIBAS. Normal utilization of index tables leads to some disk space waste because random insertion and deletion of keys fills the table to about 60%. Compression of index tables will reorganize them, and achieve a disk space utilization of about 90%.

Format:

```
COMPRESS  INDEX  ! DATABASE                                    !
                 ! REALM  <realm-name>  ( <key-name> )         !
```

Rules:

1. All the realms containing the keys must be readied.

2. If the DATABASE option is given, all indexes in the database will be compressed.

3. When the REALM option is given, if only the "realm name" is specified, all the index tables relative to this realm will be compressed.

4. When the REALM option is given and a "key name" is specified, the named index table will be compressed.

5. In all cases, information is printed on the terminal, showing the amount of saved space.

6.1.11 Consistency Checking

6.1.11.1 GENERAL

The consistency checking functions are a part of the integrity control system for the database. These functions are used to detect integrity breaches. When breaches on the database integrity are detected, the recovery system will normally be used to bring the database back to a consistent state. In some cases, the patch functions can be used to do minor repairs on the database. In some cases, the REGENERATE or automatic REPAIR modes may also be used to make a damaged database readable.

It should be noted that consistency checking does not include validity checking. Validity checking is concerned with the logical contents of the database as viewed by the user, consistency checking is concerned with the physical contents of the database and its consistency vis a vis the database's physical construction.

The types of consistency checking which can be performed in SIBAS are:

- CALC KEY verification
- INDEX KEY verification
- SET verification
- PAGE-LINK verification

If errors are detected, the following information will be given:

Message:
"Message describing the type of error"

Information about the record:
"Realm name", "Item name"
"Physical position of the record (pointer)"
"Item value"
"Comparing value"
"Dump of record"

It must be noted that the item name can be the name of a pointer (see record layout printed from DRL processor).

All realms to be verified must be readied.

A sequence of VERIFY commands may start with the following statement defining the *mode* of later verifications:

Format:

```
                  ! READ-ONLY     !
VERIFY MODE       ! REPAIR        ! .
                  ! REGENERATE    !
```

Rules:

1. In READ-ONLY mode subsequent VERIFY commands will try to detect errors, but do nothing with them if an error is found.

2. In REPAIR mode, subsequent VERIFY commands will try to repair minor damage such as a missing key, unconsistency between keys and records, etc.

3. In REGENERATE mode subsequent VERIFY commands cause large changes: automatic set/index is completely rebuilt. Manual sets/indexes are completely disconnected.

6.1.11.2 CALC KEY VERIFICATION

Function:

This function provides for the verification of calc key consistency. For each calc key verification, the calc key of all records stored in the specified realm will be checked. The value of the calc key is checked against the bucket number of the record.

No attempt is made to correct errors which are detected by calc key verification, regardless of mode. (See 6.1.11.1.) Information about the record and its physical position is printed.

Format:

```
VERIFY  CALC      !  REALM  <realm-name>!  (MAXREC <integer>).
                  !  DATABASE           !
```

Rules:

1. DATABASE. If the DATABASE option is given, all calc keys on the data base will be checked.

2. REALM. If the REALM option is given, the calc keys on the specified realm will be checked.

3. MAXREC. If the MAXREC option is used, the verification process will stop when "integer" records have been checked.

4. ERROR MESSAGE. If one or more inconsistent calc keys are detected, the following message is given for each error:

   CALCULATED KEY DOES NOT CORRESPOND TO RECORD KEY

   Information about the record will be printed for each error detected.

6.1.11.3 INDEX KEY VERIFICATION

Function:

This function checks the consistency of index key values and index table entries.

The command specifications allow for the checking of all index tables in the database, or specified index tables in a realm.

The function of the index key verification is to check the consistency of the key value of each entry in the index table with the corresponding key value in the record for each index key defined. The consistency checks are performed in two ways:

1. By reading all the entries in the index table and finding the corresponding records.

2. By scanning all the records and using the key values to find the corresponding table entries. This check is performed for automatically maintained indexes only.

Format:

```
VERIFY  INDEX  ! REALM  <realm-name> <key-name> (<key-name> ...) ! (MAXREC  <integer>).
               ! DATABASE                                        !
```

Rules:

1. DATABASE. If the DATABASE option is given, all index keys defined for the database will be checked.

2. REALM. If the REALM option is given, all index keys given in the key names will be checked.

3. MAXREC. If the MAXREC option is used, the verification process will stop when "integer" records have been checked.

4. ERROR MESSAGE. The errors that may be detected are:

   a) ENTRY IN INDEX TABLE DOES NOT MATCH RECORD KEY
   b) RECORD HAS NO CORRESPONDING ENTRY IN INDEX TABLE

   Information about the record will be printed for each error detected.

6.1.11.4 SET VERIFICATION

Function:

This function is used for verifying set relationships within the database. This function is performed by traversing records of a set and examining their types and their pointers.

The set verification utility may be requested to vary its domain of examination from a single set occurrence, to all occurrences of a specified set, to all sets of a database, through the specification of the appropriate format of the VERIFY command.

The consistency checks are performed in two ways:

1. By following all chains from the owner records.

2. By scanning all the member records and using the member set item value to find an owner record. This check will be performed for automatically maintained sets only.

Format:

The set verification command has 2 formats:

Format 1:

```
VERIFY  SET    ! <set-name> !   ( MAXREC  <integer> ) .
               ! DATABASE   !
```

Format 2:

```
VERIFY  SET   <set-name>   USING  <owner-item-value> .
```

Rules:

1. DATABASE. This version of Format 1 is used when all occurrences of all sets defined for the database are to be verified. The user is warned that the amount of processing required to accomplish such a function may be considerable.

2. ALL SET OCCURRENCES IN A GIVEN SET. Format 1 with <set-name> is used to verify all occurrences of a given set.

3. SINGLE SET OCCURRENCES. Format 2 is used to verify specified occurrences of a given set. Each set occurrence is identified uniquely by the value of the owner set item.

4. MAXREC. The MAXREC clause is used to specify the maximum number of records to be verified.

5. "OWNER ITEM VALUE". Each item composing "owner-item-value" must be given a value. If it is a character item, it is written as a character string delimited by quotes. If it is an integer item, it is written as an integer number.

6. ERROR MESSAGES. The errors detected may be:

   a) NO OWNER RECORD FOUND WITH GIVEN OCCURRENCE
   b) POINTER POINTS OUTSIDE SET
   c) MEMBER ITEM VALUE NOT EQUAL TO OWNER ITEM VALUE
   d) BACKWARD POINTER IS ERRONEOUS
   e) OWNER POINTS TO ITSELF
   f) MEMBER HAS NO OWNER
   g) LOOP, POINTER POINTS TO A PREVIOUS MEMBER OF SET OCCURRENCE
   h) MEMBER HAS DIFFERENT OWNER
   i) NUMBER OF RECORDS READ VIA SET DOES NOT CORRESPOND TO NUMBER OF RECORDS READ IN PHYSICAL ORDER

For each error detected, information about the record involved is printed out (see Section 6.1.11.1).

Loops in a member chain are not detected if they are more than 504 records long.

For error i), additional information is printed:

Two integers = no. of records read in physical order and no. of records read via set.

6.1.11.5 PAGE-LINK VERIFICATION

Function:

This function is used for verifying or rebuilding the freespace links within a realm.

The effect of VERIFY PAGE-LINK is different depending on whether VERIFY is in READ-ONLY mode or in REGENERATE mode.

1. In READ-ONLY Mode

The function examines how much unused space there is in the realm specified. Information about maximum number of records to be inserted, number of records in the realm up to now, and number of free record space are printed out.

2. In REGENERATE Mode

function checks every data page and rebuilds freespace links.

Error messages and information will be printed out.

Format

VERIFY PAGE-LINK REALM <realm name>.

Rules:

1. The realm specified must be in READY mode.

2. Realm name must be a serial realm.

3. When you want to rebuild freespace pointers (page-links) within a realm, you must do the following:

```
VERIFY MODE REGENERATE.
VERIFY PAGE-LINK REALM <name>.
```

6.1.12 Free-Space-Statistics

Function:

Give an overview of the realm space utilization.

Format:

FREE-SPACE-STAT .

6.1.13 Example

```
@SIB2-DBM

S I B 2 - D B M          SEPT. 79

EXPLANATION ? NO
INTERACTIVE ? YES
     1: START FORDB.
     2: READY ALL
     3: FREE-SPACE-STAT.


REALM      TYPE       PAGES     PAGES      MAX NO.     FREE    + FREED PERCENT
                    RESERVED      USED     RECORDS    RECORDS             USED

FORDB      SYS           192       60         192        122  +      0       32
SYSFILE    SYS           196       14         196        182  +      0        7
PERSON     SERIAL         52        3         260        245  +      0        5
JOBB       SERIAL        120        2         960        944  +      0        1
RAPPORT    CALC          144      123        1440        210  +      0       85

     4: COMPRESS INDEX DATABASE.

          REALM : PERSON     INDEX : AVDPERS              0 PAGES FREED
          REALM : PERSON     INDEX : PERSNAVN             0 PAGES FREED
          REALM : PERSON     INDEX : PERSNR               0 PAGES FREED
          REALM : JOBB       INDEX : JOBBNR               0 PAGES FREED
          REALM : JOBB       INDEX : JOTYPENR             0 PAGES FREED
          REALM : RAPPORT    INDEX : PERJOBB              0 PAGES FREED
          REALM : RAPPORT    INDEX : PERSNR               0 PAGES FREED
     5: VERIFY SET DATABASE.


          SET     : JOBRAP
          NUMBER OF OWNERS  READ :          3
          NUMBER OF MEMBERS READ :          3
          MEMBER READ IN PHYSICAL ORDER :       3  VIA SET :       3

          SET     : PERRAP
          NUMBER OF OWNERS  READ :          6
          NUMBER OF MEMBERS READ :          3
          MEMBER READ IN PHYSICAL ORDER :       3  VIA SET :       3


     6: EXIT

025054 STOP     0
```

6.1.14 Unload/Load

UNLOAD

Function:

To dump the contents of a realm on a SINTRAN-III file.

Format:

UNLOAD REALM <realm-name> ON <file-name> .

Rules:

1. 'Realm-name' must be readied.

2. Default type for 'file-name' is :DATA.

No. of records unloaded and logical record length are written on the terminal. Deleted records will not be unloaded.

LOAD

Function:

To load a realm from a SINTRAN-III file.
LOAD together with the REGENERATE option of VERIFY is substantially faster than standard STORE.

Format:

```
LOAD REALM <realm-name> FROM <file-name>
     RECL <rec-lenght>
```

Rules:

1. 'Realm-name' must be in READY mode.

2. Default type of 'file-name' is :DATA.

3. The input file byte pointer must be correctly set (at end of last record). This is automaticaly done by UNLOAD. 'Rec-length' must be less than or equal to the record length in the realm. If the given 'rec-length' is smaller than the realm record length, the rest of the record is padded out with binary zeros.

4. Automatic index tables and set relationships must be regenerated using VERIFY in REGENERATE mode. The index keys and sets involved are written on the terminal. Remember that manual index tables and set relationships are removed by VERIFY in REGENERATE mode.

5. LOAD assumes that the items in the record have same length and order as in the schema. (Documented by SIB-DRL). Remember that null values are spaces for characters items, otherwise they are binary zeros. Old contents of the realm will be lost after a LOAD.

Hints:

Both LOAD and UNLOAD are faster if the SINTRAN-III file is continous.

6.1.15 Make Modefile for Unload/Load Program

Function:

To make a modefile for running the program "SIB-UNLOAD-LOAD:PROG" (ND-number 10141A) which copies a database by using two SIBAS processes and standard DML calls.

Format:

MAKE-MODEFILE ON <file-name> .

Rules:

1. File 'file-name' must exist and default type is :MODE.

The routine finds the correct sequence of realms to be copied, by taking account of set relationships. Manual sets that lead to a loop in the structure diagram are omitted. These set names are printed on the terminal.

6.1.16 CLEAR SYSTEM—REALM

Function:

The function of this statement is to clear the realm completely, and then rebuild the old indexes as in clear NEW INDEX (see 3.14). That is, the indexes will exist but will be empty after this call.

Format:

CLEAR-SYSTEM-REALM <realm-name>

Rules:

1. REALM NAME. The "realm-name" must identify an existing user system realm (not SIBAS system realm).

2. To be used if sesions errors in the index tables. To rebuild automatic indexes afterwards, use VERIFY INDEX in REGENERATE-mode.

6.2 SIBAS SERVICE PROGRAM

SIBAS service program is an interactive utility which intends to make life a bit easier for the SIBAS operator.

The program can be used to start/stop SIBAS RT processes, get statistics about database space utilization, routine log status, update file status, etc. Most of the calls described in Chapter 5 are directly implemented as commands but only users RT or SYSTEM are allowed to use this program.

Command Syntax:

The command syntax is very near to SINTRAN III with abbreviated look-up, parameter prompting, etc. Only three control characters are implemented:

Control A which deletes the last typed character
Control Q which deletes the last typed line
Control D followed by return which deletes the last typed command

Any line beginning with a space will be treated as comments. If a line begins with "@" the rest of the line will be executed as a SINTRAN III command.

The available commands are:

EXIT	
HELP	list the available commands
OPEN-DATABASE	as SOPDB
CLOSE-DATABASE	as SCLDB
GIVE-CHECKPOINT	as GCHPO
RETURN-CHECKPOINT	as SCHPO
ROLL-BACK	as SROLL
UPDATE-DATABASE-IN-PLACE	as SUPLA
FORCE-CLOSE	as SABOR
CHANGE-COMMUNICATON-PROCEDURE	as CHCOM for TPS
DATABASE-STATUS	as SISTA
FINISH	as SFINI
GET-SIBAS-STATE	as STGET
GET-UPDATE-STATISTICS	restricted call
GIVE-MESSAGE-TO-SIBAS	as SMESS
INITIATE-LOG	as INLOG
PAUSE	as SPAUS
RECOVER	as SRECO
REPROCESS-DATABASE	as SREPR
RUN-DATABASE	as SRUN
SET-CONDITION-FOR-REPROCESSING	as SICON
SET-PASSIVE	as SPASS
SET-ROUTINE-LOGGING-ON/OFF	as OFLOG/ONLOG
START-DATABASE	as START
STOP-DATABASE	as STOPS
TURN-ON/OFF-TERMINAL-LOG	as STRLG
SUPER-START	This command brings SIBAS from the READY state to RUNNING and opens the database.
SUPER-STOP	The opposite of SUPER-START: the database is closed and SIBAS state changed from RUNNING to READY.
STANDARD- REPROCESS	Skip uncompleted sequences with or without ROLL-BACK and reprocess the R-LOG to the end.
EXPLAIN	States the parameter of a command and gives an explanation. Use the EXPLAIN command to get the full documentation of all the commands. The following is an example of one of these.

>>EXPL GIV-CH

GIVE-CHECKPOINT <BASIC UNIT> <SECOND> <MINUTE> <HOUR>
<DAY> <MONTH> <YEAR>

Define a check-point on log file(s) where the database is consistent. If a FATAL ERROR occurs at a later point in time, then the database can be restored to the consistent state when the last CHECKPOINT was taken. "Delayed-update" or "Before-image log" must be in use, otherwise only the "routine-log" buffers would be written.
RUN-FLAG must permit this call (see RUN-DATABASE).
Sibas state: RUNNING

6.2.1 SIBAS-Service Extensions SIBAS-500

So that the SIBAS-service program may be used to service and control SIBAS-500 processes, the program has been expanded with some extra 'SIBAS-500 facilities'. First of all, it must be emphasized that a 'cold start' of a SIBAS-500 process, ie., transfer SIBAS-500 from passive to ready state, will be a more time consuming operation than for SIBAS-100.

As already mentioned, the *work-area-size* in the START call is a dummy on SIBAS-500, and it is not required in the START and SUPER-START commands. If a direct R-log is filled, a message in conjunction with the DATABASE-STATUS command, will be displayed telling the user to reset or remove the R-log. A full back-up should be taken.

An additional feature: When transmitted to a SIBAS-500 process, the DATA-BASE-STATUS command will specify the total number of SIBAS-500 calls executed since start, ie., the total number of calls, including SIBAS-service and recovery calls, executed since the database state was changed from READY to DBA. To notify SIBAS-500, the SIBAS-service program will always display SIBAS-500 STATE: when a SIBAS-500 process is being used, and SIBAS STATE: whenever it is a SIBAS-100 process.

Example (state is running):

```
>> SIBAS-500 STATE:  RUNNING        % SIBAS-500
>> SIBAS STATE:  RUNNING            % SIBAS-100
```

7 ERROR AND EXCEPTION CONDITIONS

Errors occurring when using the SIBAS database control system may be classified in 3 classes:

FATAL ERRORS

Usually associated with serious malfunctions of the software or the hardware. This kind of error is often accompanied by SINTRAN messages and SIBAS run-time messages.

INTERFACE or SIMULATOR ERRORS

Usually associated with errors in operating the database control system. Those errors are signalled with a negative status, a list of which is given later in this chapter.

DML DIAGNOSTICS

By far the most common error or exception conditions. They are associated with "normal" use of the database control system. Those diagnostics are signalled by a status value equal to —1 for error conditions or a status value equal to 0 for exception conditions. An exception condition is not an error, just a warning, for example reaching the end of a search region. In case of DML diagnostics, further information may be obtained from SIBAS to identify the error — the ACCEPT (SDBEC) statement is provided for this purpose. Each error or exception condition is identified by a number, the Database Exception Condition number (DBEC) listed later in this chapter.

7.1 FATAL ERRORS

SIBAS code has many internal checks and if any check fails, it may result in a FATAL ERROR. In this way, malfunctions are detected very soon and damages are not propagated onto the database. A FATAL ERROR stops SIBAS immediately and results in the hanging up of applications trying to access the database. In most cases, recovery actions must be taken.

Recovery actions may also be undertaken after DML diagnostics, usually after an unsuccessful ERASE operation.

DBEC	Meaning
221	Implicitly referenced realm not readied, ERASE partially executed
711	Attempt to erase owner of nonempty set, ERASE partially executed
741	Loop in set structure, ERASE partially executed
991	Privacy breach counter overflow, ERASE partially executed
885	Database left in error mode

In the case of "partially executed ERASE", the database is still consistent, but the erase function did delete some of the records in a set structure, but not all of them. Some corrective action must be taken to delete the rest of the records.

When a SIBAS-500 abnormally terminates, it will automatically open a file under user RT with the name SIB2x:DUMP where x is A,B, ... or F according to the actual SIBAS-500 process terminating.

If the file does not exist, it will be created. Therefore, the DBA must ensure that enough space is available for user RT. (One dumpfile will require about 170 pages).

When a SIBAS-500 process terminates itself in this way, the following message will be printed on the actual SIBAS error-device:

```
SIBAS-II / ND-500   INTERNAL ERROR  hh.mm yyyy.mm.dd

(DUMPING ON FILE (RT)SIB2x:DUMP)

>>DUMP WILL TAKE TIME (MINUTES, BE PATIENT . . . -- WAIT --
>>SIBAS-500 DUMPING COMPLETED
```

The last message will appear on the error-device when the dumping is completed and the dump resides on the specified file.

Note that a SIBAS-500 dumping is a 'heavy' affair and may require a considerable amount of time.

7.2 INTERFACE AND SIMULATOR ERRORS

These errors are signalled by a negative value in the status parameter upon return from a SIBAS call. The SIBAS process continues running but some of the errors are so serious that SIBAS should be stopped manually.

Status	*Meaning*
—128	Call is not allowed in current state. SIBAS reserved by another user
—127	Interface buffer full (too many return values). Attempt to send more than 500 words from SIBAS
—126	Interface user table full More than 64 updating users trying to access SIBAS For SIBAS-500: 90 updating users
—125	Illegal to call STOPS when database open
—124	R-log not initiated for this database. Heading of log file does not contain database name
—123	Current time is before current R-log block. The machine time has not been properly set.
—122	Work area specified is inadequate. Try a different value between 7 and 32.
—121	Error in opening new log file SINTRAN message on error device gives more information
—120	Illegal routine number in input packet Communication error or DML-SIB version does not correspond to SIBAS version.
—119	Error when using terminal log device A SINTRAN message on the error device gives more information.
—118	Incorrect number of words in a routine log block The routine log has been damaged
—117	Ready realm for update, or load, invalid after open database for retrieval, or when database is set read-only by run-flag
—116	R-log initiated for another database
—115	Illegal to nest critical sequences or transaction units

Status	*Meaning*
—114	Error in opening log file See error message on SINTRAN error messages
—113	Illegal call when log is turned off Calls that could influence other users are not allowed unless specified in the run call.
—112	Routine log file must contain a positive number of pages
—111	Log is not active
—110	Error status is set recover Recover (and rollback) must be done
—109	Answer mismatch when reprocessing
—108	Max. no. of calls scanned
—107	Scanned to desired checkpoint/log block
—106	Scanning error, STREP must be called to get status
—105	Checkpoint not found on R-log.
—104	Call not allowed with transaction unit
—103	Illegal to remove not empty update file
—102	Illegal routine log type
—101	Illegal code to IN LOG call
—100	Reprocesing control table is full At least 48 entries.
— 99	Time given is before initiation of R-log
— 98	Time given is after last written block
— 97	Time given is before current checkpoint
— 96	User ID is already entered in reprocessing control table
— 95	Illegal control code
— 94	Illegal reprocessing condition
— 93	Illegal scanning mode
— 92	Illegal list option code
— 91	Database is not rolled back Must be done when call log is circular and has gone around.

Status	*Meaning*
— 90	Database rolled back Illegal to remove "uncompleted sequences" after rollback
— 89	Illegal to start scanning once more "Continue scanning" or "SFINI" must be used.
— 88	Illegal to reprocess opened database without rollback Database is not a backup copy.
— 87	Illegal to list only after roll back It is only allowed to reprocess after rollback
— 86	Remove flag must be +1 or —1
— 85	Log already defined
— 84	Entry not found in reprocessing control table
— 83	SCHPO/GCHPO not allowed when db not open for update
— 82	Before Image log size illegal
— 81	GCHPO not allowed by "runflag" parameter.
— 80	SIBAS process unavailable
— 79	Attempt to send more than 508 words to SIBAS Probably missing length parameter in STORE, SMDFY, SFTCH, or SFEBL
— 78	Error in reserving internal devices (user side) SINTRAN is probably not generated for this SIBAS system number.
— 77	Illegal communication procedure
— 76	SIBAS process is passive, from SOPDB, STGET, SDBEC, or SCLDB
— 75	Call is not implemented on SIBAS-500
— 74	Illegal length of item(s) in accumulate call
— 73	Illegal to set bit 14/15 in runflag when database open, i.e., it is not legal to force R-logging for all users, regardless of the "mode" parameter in SOPDB when the database is physically opened (runflag bit 14). Furthermore, it is not legal to set the database read-only when it is already physically opened (bit 15).
— 72	Direct R-log is full, R-logging stopped. Illegal to open the database. DBA should reset or remove the R-log. This status will be returned from SOPDB if a direct R-log is filled.

7.3 DML DIAGNOSTICS, DATABASE EXCEPTION CONDITIONS (DBECS)

110 THE RECORD IS ALREADY LOCKED BY THIS RUN-UNIT

The record identified by "temporary-data-base-key" is already locked by this run-unit.

120 THE RECORD IS ALREADY LOCKED BY ANOTHER RUN-UNIT

The record identified by "temporary-data-base-key" is already locked by another run-unit.

132-145 RECORD UPDATED BY CONCURRENT RUN-UNIT

The record identified by "temporary-data-base-key" has been updated by a concurrent run-unit.

170 PAGE LOGGING NOT ACTIVE

But application program attempts a page logging call such as SUPLA/SCHPO/GCHPO/SROLL.

210 END OF SET OR SEARCH REGION

Find next/prior in set:

There are no more members of this set occurrence.

Find next in search region:

There are no more records in this search region.

The DBCS will set the "status" parameter to the value 0.

220 IMPLICITLY REFERENCED REALM NOT READIED

Find:

The next or prior owner/set member relative to the record identified by "temporary-database-key" is located in a realm which has not been readied.

Store/Modify:

If the record contains a non-null member set item of an automatic set type, then the realm(s) where the owner and the first set member are located must have been readied. If the "item list" contains items which are part of group items defined as member set items, the realms containing owner and members of these set types must also be readied. If a calc key is stored, the realms containing owner and members of all set types defined for this record type must have been readied.

221 IMPLICITLY REFERENCED REALM NOT READIED, ERASE PARTIALLY EXECUTED

When executing ERASE, a record to be erased or updated is encountered on a realm not in ready mode. ERASE could not restore the database to the state before the ERASE was executed. Database recovery should be taken.

225 IMPLICITLY REFERENCED REALM NOT PROPERLY READIED

When executing ERASE, an implicitly referenced realm has not been readied with update usage mode and/or protection mode of exclusive update. Database recovery should be taken.

230 NO CORRESPONDING SET OWNER

Store/Modify:

An attempt is made to store or change a record containing a non-null member set item of an automatic set, type when the owner does not exist.

Connect:

An attempt is made to connect a record in a set but there is no owner.

240 NO RECORD FOUND WITH GIVEN KEY VALUE

A record with the given value of a CALC key or an INDEX key does not exist in the specified realm. The DBCS will set "status" parameter to the value 0.

250 KEY ITEM IS MISSING FROM ITEM LIST

If one of the items of the record type is defined as a CALC key or an INDEX key in the DATABASE SCHEMA, then at least one of the items specified in "item list" must be either a CALC key or an INDEX key.

260 ITEM NOT DEFINED AS INDEX KEY

The name given in "key name" is not defined as an INDEX key for this record type in the DATABASE SCHEMA.

270 CALC KEY ITEM MISSING FROM ITEM LIST

The record type has a CALC key defined in the DATABASE SCHEMA, but the item is missing from the "item list".

290 ATTEMPT TO FIND FIRST OR LAST IN EMPTY SET or FIRST IN EMPTY SEARCH REGION

Find first between limit:

The search region defined by "realm name", "key name", "low limit" and "high limit" is empty. The DBCS will set "status" parameter to the value 0.

Find first in realm:

The realm "realm name" is empty.

Find first/last in set:

There are no members connected to the owner record identified by "temporary-database-key" in the set "set name".

291 REFERENCED RECORD IS NOT LOCATED IN REFERENCED SEARCH REGION

The record identified by "temporary-data-base-key" is outside the search region identified by the "temporary-search-region-indicator".

310 TEMPORARY DATABASE KEY IS INVALID

There is no entry in the remembered list corresponding to the "temporary-database-key" given.

320 TEMPORARY SEARCH REGION INDICATOR IS INVALID

There is no entry in the remembered list corresponding to the "temporary-search-region-indicator" given.

330 NO CURRENT OF RUN-UNIT EXISTS

There is no current of run-unit correponding to the value 0 of "temporary-database-key".

340 NO CURRENT OF SEARCH REGION EXISTS

There is no current search region correponding to the value 0 of "temporary-search-region-indicator".

350 ILLEGAL TO FORGET CURRENT OF RUN-UNIT

The entry in the remembered list corresponding to the "temporary-database-key" given is the current of run-unit. It is not allowed to remove this entry from the list using a FORGET statement.

360 ILLEGAL TO FORGET CURRENT SEARCH REGION

The entry in the remembered list corresponding to the "temporary-search-region-indicator" given is the current search region. It is not allowed to remove this entry from the list using a FORGET statement.

370 ILLEGAL SEARCH-REGION FOR SGETN OR SGIXN

The search region must be an index table i.e. A duplicate index key or a " between-limit" range.

410 SPECIFIED REALM NAME NOT CONSISTENT WITH REALM NAME WRITTEN IN REALM

The realm corresponding to a realm name specified in "realm name" does not contain the correct realm name in the identification area. An incorrect realm has been assigned to the user.

420 INCORRECT DATABASE NAME GIVEN

The parameter given as database name to SCLDB does not match the name of the database. Close is unsuccessful and the database remains open.

430 SPECIFIED REALM NAME NOT DEFINED IN THE DATABASE SCHEMA

A name given in "realm name" is not defined in the DATABASE SCHEMA.

440 SPECIFIED ITEM NAME NOT DEFINED IN THE DATABASE SCHEMA

The name given in "key name" or "item list" is not defined as an item or group item name for this record type in the DATABASE SCHEMA.

450 SPECIFIED SET NAME NOT DEFINED IN THE DATABASE SCHEMA

The name given in "set name" is not defined as a set name in the DATABASE SCHEMA.

460 DATABASE IS NOT OPENED FOR THIS RUN-UNIT

The database has not been opened by this run-unit. If privacy is defined, a valid password must be given when opening the data base.

461 THE USER IS NOT ALLOWED TO ACCESS SYSTEM REALMS

The name given in "realm name" is defined as a system realm in the DATABASE SCHEMA. A user is not allowed to use Data Manipulation Language (DML) statements on a system realm.

510 ATTEMPT TO ERASE ALL ACCESS KEYS AND MEMBER SET ITEMS FOR A RECORD

If one or more non-automatic access keys or member set items are defined for the record type in the DATABASE SCHEMA, at least one of these items must have a non-null value after ERASE ELEMENT has been executed.

520 PROHIBITED DUPLICATE VALUE FOR CALC OR INDEX KEY

One of the item names given in "item list" is defined as a unique CALC key or INDEX key in the DATABASE SCHEMA for this record type. The value given in this item in the "item values" already exists on the database for an occurrence of this record type.

530 NULL VALUE GIVEN ON CALC OR INDEX KEY

Modify:

It is illegal to modify a CALC key or an INDEX key to a null value.

Store:

It is illegal to store a record with all keys having null values. At least one of the keys must have a valid value.

Insert:

It is illegal to insert a key with null value in an index table.

540 NULL VALUE GIVEN ON MEMBER SET ITEM

Modify:

It is illegal to modify a member or owner set item to a null value.

Store:

It is illegal to give a null value to a member or owner set item.

550 MEMBER SET ITEMS NOT CONSISTENT

The member set item identified by "set name" has a different value in the record identified by "temporary-database-key-1" and in the record identified by "temporary-database-key-2".

610 INVALID INPUT PARAMETER VALUE

Get/Store/Modify:

The number given in "no. of items" must be greater than zero and less than or equal to the total number of items and group items defined for the record type corresponding to the "temporary-data-base-key" in the DATABASE SCHEMA.

Ready/Finish Realm:

Invalid value of one of the parameters "no. of realms", "usage mode" or "protection mode".

Erase/Remember/Forget/Lock:
Invalid value of the parameter "option code".

620 PARAMETERS NOT CONSISTENT

The value given in "low limit" is greater than the value given in "high limit".

623 TOO LONG VALUE BUFFER TO RETURN

The value to return from GET/GETN/GIXN exceeds 500 words.

710 ATTEMPT TO ERASE OWNER OF NON-EMPTY SET

An attempt is made to erase the owner of a non-empty set when the "option code" given does not allow this for this set type.

711 ATTEMPT TO ERASE OWNER OF NON-EMPTY SET, ERASE PARTIALLY EXECUTED

When executing ERASE, an owner of a non-empty set was encountered. ERASE could not restore the database to the state before the ERASE was executed. Database recovery should be taken.

720 ILLEGAL ERASE CODE IN MULTI-USER ENVIRONMENT

The value given in "option code" is not allowed in a multi-user environment, unless all realms in the database are readied for exclusive update.

730 RECORD ERASED BY CONCURRENT RUN-UNIT

The record identified by "temporary-database-key" has been erased by a concurrent run-unit.

740 LOOP IN SET STRUCTURE

When executing ERASE, the number of levels in the set structure from which records should be erased exceeded the maximum number given in SIBAS. This number is 15 in the standard version of SIBAS.

741 LOOP IN SET STRUCTURE, ERASE PARTIALLY EXECUTED

When executing ERASE, the number of levels in the set structure from which records should be erased exceeds the maximum number given in SIBAS. Database recovery should be taken. This number is 15 in the standard version of SIBAS.

810 RECORD IS ALREADY CONNECTED TO SET

The record identified by "temporary-database-key" is already connected to a set identified by "set name". The DBCS will set "status" parameter to the value 0.

820 INDEX KEY ITEM IS ALREADY INSERTED IN INDEX TABLE

The INDEX key corresponding to "key name" in the record identified by "temporary-database-key" has already been inserted in the index table. The DBCS will set the "status" parameter to the value 0.

830 RECORD IS NOT CONNECTED TO SET

The record identified by "temporary-database-key" is not connected to a set identified by "set name". The DBCS will set the "status" parameter to the value 0.

835 REFERENCED RECORD IS NOT LOCATED IN SET OCCURRENCE

Find:

If the set type is defined as automatic, the member set item corresponding to the set type identified by "set name" for the record identified by "temporary-database-key" has a null value.

If the set type is defined as manual, the record identified by "temporary-database-key" is not connected to an occurrence of the set type identified by "set name".

Connect:

The record identified by "temporary-database-key-2" is not connected to an occurrence of a set type corresponding to "set name".

840 RECORD TYPE IS NOT MEMBER OF GIVEN SET TYPE

The record type corresponding to "temporary-database-key" is not defined as a member of the set type corresponding to "set name" in the DATABASE SCHEMA.

850 INDEX KEY ITEM IS NOT INSERTED IN INDEX TABLE

The INDEX key corresponding to "key name" in the record identified by "temporary-database-key" has not been inserted in the index table. The DBCS will set the "status" parameter to the value 0.

860 ATTEMPT TO MODIFY OWNER SET ITEM OF NON-EMPTY SET

An attempt is made to modify an owner set item of a non-empty set. All the members of this set occurrence must either have been erased or disconnected if the set type is manual, before the owner set item can be modified.

870 RECORD TYPE IS NOT OWNER OF GIVEN SET TYPE

The record type corresponding to "temporary-database-key" is not defined as an owner of the set type corresponding to "set name" in the DATABASE SCHEMA.

871 SET TYPE IS DEFINED AS AUTOMATIC

The set type corresponding to "set name" is defined as automatic in the DATABASE SCHEMA. Manual operations are therefore illegal.

872 INDEX KEY IS DEFINED AS AUTOMATIC

The INDEX key corresponding to "key name" has been defined as automatic for the record type corresponding to "temporary-database-key" in the DATABASE SCHEMA.

880 ATTEMPT TO FINISH REALM NOT IN READY MODE

One of the realms specified in "realm names" is not in ready mode for this user. The DBCS will set the "status" parameter to the value 0.

881 REALM IS NOT IN READY MODE

The realm identified by "realm name" has not been readied for this run-unit.

882 ATTEMPT TO READY REALM IN READY MODE

One of the realms specified in "realm names" is already in ready mode for this user. The DBCS will set the "status" parameter to the value 0.

884 DATABASE HAS ALREADY BEEN OPENED FOR THIS RUN-UNIT

The database identified by "database name" has already been opened. The DBCS will set the "status" parameter to 0.

885 DATABASE IN ERROR MODE

A realm has not been properly finished before an interrupt. The DBCS indicates that a realm corresponding to a name given in "realm names" has not been properly finished before an interrupt occurred on the database. The database is possibly in error mode and recovery should be taken.

910 SPACE IN REALM IS EXHAUSTED

The maximum space defined for the realm identified by "realm name" in the DATABASE SCHEMA is exhausted for this realm.

920 SPACE IN INDEX TABLE IS EXHAUSTED

The maximum space defined in the DATABASE SCHEMA for the system realm(s) containing the index tables for an INDEX key for the record type identified by "realm name" has been exhausted.

930 MAXIMUM NUMBER OF REMEMBERED TEMPORARY DATA BASE KEYS EXCEEDED

The maximum number or remembered current of run-units for this user is exceeded. A FORGET statement must be executed before further REMEMBER statements can be executed.

940 MAXIMUM NUMBER OF REMEMBERED SEARCH REGION INDICATORS EXCEEDED

The maximum number of remembered current search regions for this user is exceeded. A FORGET statement must be executed before further REMEMBER statements can be executed.

950 REALM NOT READIED FOR THIS USAGE MODE

The realm containing the record identified by "temporary data base key" has not been readied for load or update.

951 REALM READIED FOR EXCLUSIVE UPDATE BY ANOTHER RUN-UNIT

One of the realms specified in "realm names" has been readied for exclusive update by another user. It cannot be readied for load or update until it has been finished by that user.

952 REALM NOT ASSIGNED

One of the realms specified in "realm names" has not been assigned to this run-unit.

953 ATTEMPT TO READY REALM FOR EXCLUSIVE UPDATE WHEN REALM IS READIED FOR LOAD OR UPDATE BY ANOTHER RUN-UNIT

One of the realms specified in "realm names" is requested for exclusive update, but the realm has been readied for load or update by another run-unit.

954 ATTEMPT TO READY REALM FOR EXCLUSIVE UPDATE WHEN RECORDS IN THIS REALM ARE LOCKED TO ANOTHER RUN-UNIT

One of the realms specified in "realm names" is requested for exclusive update, but records in this realm have been locked to another run-unit.

983 PRIVACY BREACH ON RECORD

Current password is not consistent with the value of the privacy item in one of the records to be erased.

984 PRIVACY BREACH ON RECORD, ERASE PARTIALLY EXECUTED

Current password is not consistent with the value of the privacy item in one of the records to be erased. ERASE could not restore the data base to the state before the ERASE was executed. Database recovery should be taken.

990 PRIVACY BREACH COUNTER OVERFLOW

The allowed number of privacy breaches for this run-unit has been exceeded.

991 PRIVACY BREACH COUNTER OVERFLOW, ERASE PARTIALLY EXECUTED

The allowed number of privacy breaches for this run-unit has been exceeded. ERASE could not restore the database to the state before the ERASE was executed. Database recovery should be taken.

7.4 RUN-TIME MESSAGES — Messages from SIBIO

SIBIO writes messages on a log console. These messages can be error messages or useful information about the update file:

INVALID CHECKPOINT TIME

Invalid checkpoint time in rollback checkpoint

INVALID TO CROSS U-I-P CHECKPOINT

Invalid to cross U-I-P checkpoint in rollback

UPDATE FILE IS FULL

Update file is full, an automatic U-I-P will be started

U-I-P ALREADY INITIATED

Trying to initiate a U-I-P while a U-I-P is active (a checkpoint will be taken, however)

UPDATE-IN-PLACE STARTED

A U-I-P has been started

UPDATE-IN-PLACE FINISHED

A U-I-P is finished

UPDATE FILE EMPTY AFTER CLOSE-DATA-BASE

A U-I-P has been performed at close-data-base and the update file is empty.

WAIT TILL U-I-P IS FINISHED

The update file is full and a U-I-P is active. The system will hang until the U-I-P is finished.

All other messages indicate serious errors or illegal use of SIBIO.

7.5 RUN-TIME MESSAGES — FROM SIBAS

USER ERROR 60 SUBERROR xx on SINTRAN ERROR-DEVICE

This is an I/O-ERROR, xx (decimal) is the file-system err-number.

The message is followed by a standard SIBAS error message on SIBAS ERR-DEVICE

```
USER ERROR 59 SUBERROR  5      on SINTRAN ERROR-DEVICE

SIBAS (ddddddd)    ccccccccccccccccccc
'ERROR' REALM   rrrrrrrr   CANNOTR BE READ/WRITE
```

where:	ddddddddd	is the database name
	ccc.ccc	is the clock of the machine
	rrrrrrrr	is the realm involved.

SIBAS will then issue RTOFF and RTWT (put itself in wait state).

The user can then correct the reason for I/O-ERROR and continue without loss of data by:

```
@RTON    SIB2x
@RT      SIB2x
```

USER ERROR 59 SUBERROR y on SINTRAN ERROR-DEVICE

Followed by a message on SIBAS ERROR-DEVICE

y =

1: REALM rrrrrrrr IS FULL

2: SIBAS SYSTEM REALMS FULL
only if before image logging is active

3: REALM rrrrrrrr IS FULL
index table splitting and space exhausted

4: FILE rrrrrrrr CANNOT BE OPENED/CLOSED

5: REALM rrrrrrrr CANNOT BE READ/WRITE

6: SIBIO/BIM WARNING STATUS = s

8: RECORD/PAGE ALLOCATION ERROR AT pppppp'pppppp
the error is fixed a run-time by discarding the free record/page pool. Contact ND support.

9: RECORD FREED/DAMAGED AT pppppp'pppppp
the error is fixed at run-time. Contact ND support.

APPENDIX A

SUMMARY OF THE DML STATEMENTS

OPEN-DATA-BASE

CALL SOPDB (mode, database name, password, status)

CLOSE-DATA-BASE

CALL SCLDB (database name, status)

READY-REALM

CALL SRRLM (no. of realms, realm names, usage modes, protection modes, status)

FINISH-REALM

CALL SFRLM (no. of realms, realm names, status)

FIND-USING-KEY

CALL SFTCH (realm name, key name, key-value, status, key length)

FIND-FIRST-BETWEEN-LIMITS-USING-KEY

CALL SFEBL (realm name, key name, low limit, high limit, status, key length)

FIND-LAST-BETWEEN-LIMITS-USING-KEY

CALL SFLBL (realm name, key name, low limit, high limit, status, key length)

FIND-FIRST-IN-REALM

CALL SRFIR (realm name, status)

FIND-FIRST-IN-SET

CALL SRFSM (temporary-database-key, set name, status)

FIND-LAST-IN-SET

CALL SRLSM (temporary-database-key, set name, status)

FIND-PRIOR-IN-SET

CALL SRPSM (temporary-database-key, set name, status)

FIND-NEXT-IN-SET

CALL SRNSM (temporary-database-key, set name, status)

FIND-NEXT-IN-SEARCH-REGION

CALL SRNIS (temporary-database-key, temporary search region indicator, status)

FIND-PRIOR-IN-SEARCH-REGION

CALL SRPIS (temporary-database-key, temporary search region indicator, status)

FIND-SET-OWNER

CALL SRSOW (temporary-database-key, set name, status)

GET

CALL SGET (temporary-database-key, no. of times, item list, item values, status)

GETN

CALL SGETN (temporary-database-key, temporary search region indicator, no. wanted, no. of items, item list, item values, no. found, status)

GET-INDEXES

CALL SGIXN (temporary-database-key, temporary search region indicator, no. wanted, item values, no. found, status)

MODIFY

CALL SMDFY (temporary-database-key, no. of items, item list, item values, status, value length)

STORE

CALL STORE (realm name, no. of items, item list, item values, status, value length)

ERASE

CALL SRASE (temporary-database-key, option code, status)

CONNECT

CALL SCONN (temporary-database-key 1, set name, status)

CONNECT-BEFORE

CALL SCONB (temporary-database-key 1, temporary database key 2, set name, status)

CONNECT-AFTER

CALL SCONA (temporary-database-key 1, temporary database key 2, set name, status)

DISCONNECT

CALL SDCON (temporary-database-key, set name, status)

INSERT

CALL SINSR (temporary-database-key, key name, status)

REMOVE

CALL SREMO (temporary-database-key, key name, status)

REMEMBER

CALL SREMB (temporary id, option code, status)

FORGET

CALL SFORG (temporary id, option code, status)

LOCK

CALL SLOCK (temporary-database-key, option code, status)

UNLOCK

CALL SUNLK (status)

CHANGE-PASSWORD

CALL SCHPW (new password, status)

ACCEPT

CALL SDBEC (set name, realm name 1, realm name 2, item name, dml statement code, dbec)

ERASE-ELEMENT

CALL SEREL (temporary-database-key, no. of items, item list, status)

ACCUMULATE

CALL ACCID/ACCFD/ACCDD (temporary-database-key, no. of items, item list, increments, new values, status)
(ACCFD not available for SIBAS-500)

FETCH-GET

CALL SFTGT (realm name, key name, key length, key value, no. of items, item list, item values, status)

GET-SCHEMAS-INFORMATION

CALL SINFO (code, name1, name2, length, array, status)

TRANSACTION BEGIN

CALL SUBEG (run-id, unit type, status)

TRANSACTION END

CALL SUEND (run-id, COMIT or ROLL-BACK, status)

APPENDIX B

SUMMARY OF THE SIB-DRL STATEMENTS

```
START  INITIATION  DATABASE  <db-name> .
       ( SUPPRESS (REALM) (RECORD-TYPE) (ITEM) (SET) (INDEX-TABLE) )
       SIZE  <no-of-64w-pages> .

START  REDEFINITION  DATABASE  <db-name> ( DBA-PASSWORD <password> )
       ( SUPPRESS (REALM) (RECORD-TYPE) (ITEM) (SET) (INDEX-TABLE) )
       SIZE  <no-of-64w-pages>
             ! TEST       !
       MODE  !            !  COPY OF SYSTEM-REALM   <file-name-1>
             ! PRODUCTION !
       SCRATCH-FILE  <file-name-2> .

END REDEF.  or
EXIT

TRACE  (STEP1) (STEP2) (STEP3) (STEP4) .

NO-TRACE.

NEW  OS-FILE  <file-name>  ( PAGESIZE <no-of-words> )
       ( DIRECTORY <abbreviated-dir-name> ) .

NEW  SYSTEM-REALM  <realm-name>  OS-FILE <file-name>
       REALMSIZE  <no-of-pages> .

NEW  SERIAL-REALM <realm-name>  OS-FILE  <file-name>
       REALMSIZE  <no-of-pages>
       RECORD LENGTH  <no-of-words>
       ( MAIN <system-realm>  ( ADDITIONAL <system-realm>
                                (<system-realm>) (<system-realm>))) .
```

```
NEW  CALC-REALM  <realm-name>  OS-FILE  <file-name>
       REALMSIZE  <no-of-pages>
       MAIN-AREA  <no-of-pages>
       RECORD LENGTH <no-of-words>
       CALC-KEY   <key-name>    DUPLICATES  ARE ( NOT )  ALLOWED
         MAIN <system-realm>  ( ADDITIONAL <system-realm>
                                (<system-realm>) (<system-realm>))  .

NEW  ITEM  <realm-name>  <item-name>
            ! INTEGER       !
       TYPE ! FLOATING      !   START  <word-no>
            ! CHARACTER     !
            ! PRIVACY-ITEM  !

                       ! BIT  POSITION <first-bit>  !
       LENGTH   <no>   ! WORD                      ! .
                       ! BYTE POSITION <first-byte> !

NEW  GROUP  <realm-name>  <group-name>
       <item-name>  (<item-name> ....)   .

NEW  SET  <set-name>
       LINK IS  ! SINGLE !
                ! DOUBLE !

       STORAGE-CLASS IS  ! AUTOMATIC !
                         ! MANUAL    !

       OWNER  <owner-set-item>  <realm-name>

       MEMBER <member-set-item> <realm-name> ( <realm-name> .... )  .

NEW INDEX  <realm-name>  <key-name>
       UPDATE IS  ! MANUAL    !   DUPLICATES ARE ( NOT)  ALLOWED
                  ! AUTOMATIC !

      (SYSTEM-REALM   <system-realm-name> )

       ( MIN-VALUE  <value>   MAX-VALUE  <value> )  .
```

```
DELETE  SET       <set-name>  .
DELETE  INDEX     <realm-name>  <key-name>  .
DELETE  ITEM      <realm-name>  <item-name>  .
DELETE  GROUP     <realm-name>  <group-name>  .

CHANGE  SYSTEM-REALM  <Rrealm-name>
                                ( REALMSIZE  <no-of-pages> )  .

CHANGE  SERIAL-REALM  <realm-name>
        ( OS-FILE  <file-name> )  ( REALMSIZE  <no-of-pages> )
        ( RECORD LENGTH  <no-of-words> ).

CHANGE  CALC-REALM  <realm-name>
        ( REALMSIZE  <no-of-pages> )
        ( MAIN-AREA  <no-of-pages> )
        ( RECORD LENGTH <no-of-words> )
        ( CALC-KEY   <key-name>  DUPLICATES ARE (NOT) ALLOWED ).

CHANGE SET  <set-name>
        ( LINK IS   ! SINGLE !   )
                    ! DOUBLE !
        ( STORAGE-CLASS IS   ! AUTOMATIC !   )
                             ! MANUAL    !
        ( MEMBER <member-set-item> <realm-name> ( <realm-name> .... )  )  .
```

APPENDIX C

SUMMARY OF THE SIB-DBM STATEMENTS

```
START  <database-name>  (<dba-password>) .

READY  ! REALM  <realm-name> !  .
       !        ALL          !

FINISH  ! REALM  <realm-name> !  .
        !        ALL          !

STOP .

EXIT .

PRINT  ! <number> !  ! RECORD !  ! REALM <realm-name>  <from-unit>  !  .
       ! ALL      !  ! PAGE   !  ! POINTER  <address>              !
                     ! WORD   !
                     ! BUCKET !

PRINT  POINTER  <address>  .

PATCH  ! REALM <realm-name> <page-no> !  <word-no>  <new-value>
       ! <address>                    !
                                                   <new-value ....>  .

RESET-ERROR-FLAGS.

FREE-SPACE-STAT  .

COMPRESS  INDEX  ! DATABASE                              !  .
                 ! REALM  <realm-name>  ( <key-name> ) !

                 ! READ-ONLY  !
VERIFY MODE      ! REPAIR     !  .
                 ! REGENERATE !

VERIFY  INDEX  ! REALM  <realm-name> <key-name> !  (MAXREC <integer> ) .
               ! DATABASE                       !
```

```
VERIFY  CALC    ! REALM  <realm-name> !  (MAXREC  <integer>)  .
                ! DATABASE            !

VERIFY  SET     ! <set-name>  !    ( MAXREC  <integer> ) .
                ! DATABASE    !

VERIFY  SET   <set-name>   USING  <owner-item-value> .

DEFINE  DBA-PASSWORD  <dba-password>  .

DEFINE  LOCAL-PASSWORD  <password> DATABASE.

DEFINE  GLOBAL-PASSWORD  <password>  DATABASE

          ! RETRIEVAL !                  ! NON-PROTECTED !
  USAGE   ! LOAD      !  PROTECTION     ! EXCLUSIVE     !  .
          ! UPDATE    !

REMOVE-PASSWORD  ! <password> !  .
                 ! DATABASE   !

DISPLAY     ! ALL  PRIVACY          !  .
            ! PASSWORD <password>  !

MAKE-MODEFILE ON <file-name>  .

UNLOAD REALM  <realm-name> ON <file-name>  .

LOAD REALM  <realm-name> FROM <file-name>
     RECL  <rec-lenght>
```

APPENDIX D

SUMMARY OF THE SIB-SERVICE STATEMENTS

HELP
CHANGE-COMMUNICATION-PROCEDURE
CLOSE-DATABASE
EXIT
FINISH
GET-SIBAS-STATE
INITIATE-LOG
DATABASE-STATUS
OPEN-DATABASE
PAUSE
RECOVER
ROLL-BACK
REPROCESS-DATABASE
RUN-DATABASE
SET-CONDITIONS-FOR-REPROCESSING
SET-PASSIVE
SET-ROUTINE-LOGGING-ON/OFF
START-DATABASE
STOP-DATABASE
SUPER-START
SUPER-STOP
GIVE-CHECKPOINT
RETURN-CHECKPOINT
TURN-ON/OFF-TERMINAL-LOG
UPDATE-DATABASE-IN-PLACE
GET-UPDATE-STATISTICS
FORCE-CLOSE
EXPLAIN
STANDARD-REPROCESS

APPENDIX E

SUMMARY OF THE DATABASE EXCEPTION CONDITIONS

DATABASE EXCEPTION CONDITION SUMMARY

CONCURRENCY EXCEPTION CONDITIONS	CODE	110 The record is already locked by this run-unit	120 The record is already locked by another run-unit	132—145 Record updated by another run-unit	170 Page logging not active	180 Run-unit rolled back by other	190 SUEND attempted for wrong user
DML STATEMENT CODE		-1	-1	-1	0	-1	-1
001 FIND USING KEY	(SFTCH)						
002 FIND FIRST BETWEEN LIMITS USING KEY	(SFEBL)						
003 FIND FIRST IN REALM	(SRFIR)						
004 FIND LAST BETWEEN LIMIT	(SFLBL)						
005 GET SCHEMAS INFORMATION	(SINFO)						
006 TRANSACTION BEGIN	(SUBEG)						
007 TRANSACTION END	(SUEND)						
011 FIND NEXT IN SET	(SRNSM)			×			
012 FIND PRIOR IN SET	(SRPSM)			×			
013 FIND FIRST IN SET	(SRFSM)			×			
014 FIND LAST IN SET	(SRLSM)			×			
015 FIND OWNER	(SRSOW)			×			
016 FIND NEXT IN SEARCH REGION	(SRNIS)			×			
018 FIND PREVIOUS IN " "	(SRPIS)						
020 GET/GETN/GET INDEXES	(SGET)			×			
031 STORE	(STORE)						
032 MODIFY	(SMDFY)		×	×			
033 ERASE	(SRASE)		×	×			
034 ERASE ELEMENT	(SEREL)		×	×			
041 CONNECT	(SCONN)		×	×			
042 DISCONNECT	(SDCON)		×	×			
043 CONNECT AFTER	(SCONA)		×	×			
044 CONNECT BEFORE	(SCONB)		×	×			
045 INSERT	(SINSR)		×	×			
046 REMOVE	(SREMO)		×	×			
050 OPEN DATA BASE	(SOPDB)						
051 CLOSE DATA BASE	(SCLDB)						
052 READY REALM	(SRRLM)						
053 FINISH REALM	(SFRLM)						
054 CHANGE PASSWORD	(SCHPW)						
060 REMEMBER	(SREMB)			×			
061 FORGET	(SFORG)						
062 LOCK	(SLOCK)	×	×	×			
063 UNLOCK	(SUNLK)						
072 CHECKPOINT	(SCHPO/GCHPO)						
074 ROLL BACK	(SROLL)						

DATABASE EXCEPTION CONDITION SUMMARY

DATABASE RETRIEVAL EXCEPTION CONDITIONS	CODE	0	210	220	-1	221	-1	230	-1	0	240	-1	250	-1	260	-1	270	0	290	-1	291		1	225	
			End of set or search region		Implicitly referenced realm not readied		Implicitly referenced realm not readied, ERASE partially executed		No corresponding set owner		No record found with given key value		Key item is missing from item list		Item not defined as index key		CALC key item is missing from item list		Attempt to find first or last in empty set or first in empty search region		Referenced record is not located in referenced search region			Implicitly referenced realm not properly readied	
DML STATEMENT CODE																									
001 FIND USING KEY	(SFTCH)										×		×												
002 FIND FIRST BETWEEN LIMITS USING KEY	(SFEBL)										×		×		×				×						
003 FIND FIRST IN REALM	(SRFIR)																		×						
004 FIND LAST BETWEEN LIMIT	(SFLBL)																								
005 GET SCHEMAS INFORMATION	(SINFO)																								
006 TRANSACTION BEGIN	(SUBEG)																								
007 TRANSACTION END	(SUEND)																								
011 FIND NEXT IN SET	(SRNSM)		×		×																				
012 FIND PRIOR IN SET	(SRPSM)		×		×																				
013 FIND FIRST IN SET	(SRFSM)				×														×						
014 FIND LAST IN SET	(SRLSM)				×														×						
015 FIND OWNER	(SRSOW)				×																				
016 FIND NEXT IN SEARCH REGION	(SRNIS)		×																		×				
018 FIND PREVIOUS IN " "	(SRPIS)																								
020 GET/GETN/GET INDEXES	(SGET)																								
031 STORE	(STORE)				×				×				×				×								
032 MODIFY	(SMDFY)				×				×																
033 ERASE	(SRASE)						×																×		
034 ERASE ELEMENT	(SEREL)				×				×																
041 CONNECT	(SCONN)								×																
042 DISCONNECT	(SDCON)																								
043 CONNECT AFTER	(SCONA)								×																
044 CONNECT BEFORE	(SCONB)								×																
045 INSERT	(SINSR)														×										
046 REMOVE	(SREMO)														×										
050 OPEN DATA BASE	(SOPDB)																								
051 CLOSE DATA BASE	(SCLDB)																								
052 READY REALM	(SRRLM)																								
053 FINISH REALM	(SFRLM)																								
054 CHANGE PASSWORD	(SCHPW)																								
060 REMEMBER	(SREMB)																								
061 FORGET	(SFORG)																								
062 LOCK	(SLOCK)																								
063 UNLOCK	(SUNLK)																								
072 CHECKPOINT	(SCHPO/GCHPO)																								
074 ROLL BACK	(SROLL)																								

DATABASE EXCEPTION CONDITION SUMMARY

CURRENCY INDICATOR EXCEPTION CONDITIONS	CODE	310 Temporary data base key is invalid	320 Temporary search region indicator is invalid	330 No current of run-unit exists	340 No current search region exists	350 Illegal to forget current of run-unit	360 Illegal to forget current search region	370 Illegal search-region for SGETN/SIGIXN
DML STATEMENT CODE		-1	-1	-1	-1	-1	-1	-1
001 FIND USING KEY	(SFTCH)							
002 FIND FIRST BETWEEN LIMITS USING KEY	(SFEBL)							
003 FIND FIRST IN REALM	(SRFIR)							
004 FIND LAST BETWEEN LIMIT	(SFLBL)							
005 GET SCHEMAS INFORMATION	(SINFO)							
006 TRANSACTION BEGIN	(SUBEG)							
007 TRANSACTION END	(SUEND)							
011 FIND NEXT IN SET	(SRNSM)	×		×				
012 FIND PRIOR IN SET	(SRPSM)	×		×				
013 FIND FIRST IN SET	(SRFSM)	×		×				
014 FIND LAST IN SET	(SRLSM)	×		×				
015 FIND OWNER	(SRSOW)	×		×				
016 FIND NEXT IN SEARCH REGION	(SRNIS)	×	×	×	×			
018 FIND PREVIOUS IN " "	(SRPIS)							
020 GET/GETN/GET INDEXES	(SGET)	×		×				×
031 STORE	(STORE)							
032 MODIFY	(SMDFY)	×		×				
033 ERASE	(SRASE)	×		×				
034 ERASE ELEMENT	(SEREL)	×		×				
041 CONNECT	(SCONN)	×		×				
042 DISCONNECT	(SDCON)	×		×				
043 CONNECT AFTER	(SCONA)	×		×				
044 CONNECT BEFORE	(SCONB)	×		×				
045 INSERT	(SINSR)	×		×				
046 REMOVE	(SREMO)	×		×				
050 OPEN DATA BASE	(SOPDB)							
051 CLOSE DATA BASE	(SCLDB)							
052 READY REALM	(SRRLM)							
053 FINISH REALM	(SFRLM)							
054 CHANGE PASSWORD	(SCHPW)							
060 REMEMBER	(SREMB)			×	×			
061 FORGET	(SFORG)	×	×			×	×	
062 LOCK	(SLOCK)	×		×				
063 UNLOCK	(SUNLK)							
072 CHECKPOINT	(SCHPO/GCHPO)							
074 ROLL BACK	(SROLL)							

DATABASE EXCEPTION CONDITION SUMMARY

NAME SPECIFICATION EXCEPTION CONDITIONS	CODE	410 Specified realm name not consistent with realm name written in loaded type	430 Specified realm name not defined in the DATABASE SCHEMA	440 Specified item name not defined in the DATABASE SCHEMA	450 Specified set name not defined in the DATABASE SCHEMA	460 Database is not opened for this run-unit	461 The user is not allowed to access system realms	420 Incorrect database name given
DML STATEMENT CODE		-1	-1	-1	-1		-1	-1
001 FIND USING KEY	(SFTCH)		×	×		×	×	
002 FIND FIRST BETWEEN LIMITS USING KEY	(SFEBL)		×	×		×	×	
003 FIND FIRST IN REALM	(SRFIR)		×			×	×	
004 FIND LAST BETWEEN LIMIT	(SFLBL)							
005 GET SCHEMAS INFORMATION	(SINFO)							
006 TRANSACTION BEGIN	(SUBEG)							
007 TRANSACTION END	(SUEND)							
011 FIND NEXT IN SET	(SRNSM)				×	×		
012 FIND PRIOR IN SET	(SRPSM)				×	×		
013 FIND FIRST IN SET	(SRFSM)				×	×		
014 FIND LAST IN SET	(SRLSM)				×	×		
015 FIND OWNER	(SRSOW)				×	×		
016 FIND NEXT IN SEARCH REGION	(SRNIS)					×		
018 FIND PREVIOUS IN " "	(SRPIS)							
020 GET/GETN/GET INDEXES	(SGET)			×		×		
031 STORE	(STORE)		×	×		×	×	
032 MODIFY	(SMDFY)			×		×		
033 ERASE	(SRASE)					×		
034 ERASE ELEMENT	(SEREL)			×		×		
041 CONNECT	(SCONN)				×	×		
042 DISCONNECT	(SDCON)				×	×		
043 CONNECT AFTER	(SCONA)				×	×		
044 CONNECT BEFORE	(SCONB)				×	×		
045 INSERT	(SINSR)			×		×		
046 REMOVE	(SREMO)			×		×		
050 OPEN DATA BASE	(SOPDB)							
051 CLOSE DATA BASE	(SCLDB)					×		×
052 READY REALM	(SRRLM)	×	×			×	×	
053 FINISH REALM	(SFRLM)		×			×	×	
054 CHANGE PASSWORD	(SCHPW)					×		
060 REMEMBER	(SREMB)					×		
061 FORGET	(SFORG)					×		
062 LOCK	(SLOCK)					×		
063 UNLOCK	(SUNLK)					×		
072 CHECKPOINT	(SCHPO/GCHPO)							
074 ROLL BACK	(SROLL)							

DATABASE EXCEPTION CONDITION SUMMARY

ITEM VALUE EXCEPTION CONDITIONS	CODE	510 Attempt to erase all keys or member set items for a record	520 Prohibited duplicate value for CALC or INDEX key	530 Null value given on CALC or INDEX key	540 Null value given on number set item	550 Member set items not consistent
DML STATEMENT CODE		-1	-1	-1	-1	-1
001 FIND USING KEY	(SFTCH)			×		
002 FIND FIRST BETWEEN LIMITS USING KEY	(SFEBL)					
003 FIND FIRST IN REALM	(SRFIR)					
004 FIND LAST BETWEEN LIMIT	(SFLBL)					
005 GET SCHEMAS INFORMATION	(SINFO)					
006 TRANSACTION BEGIN	(SUBEG)					
007 TRANSACTION END	(SUEND)					
011 FIND NEXT IN SET	(SRNSM)					
012 FIND PRIOR IN SET	(SRPSM)					
013 FIND FIRST IN SET	(SRFSM)					
014 FIND LAST IN SET	(SRLSM)					
015 FIND OWNER	(SRSOW)					
016 FIND NEXT IN SEARCH REGION	(SRNIS)					
018 FIND PREVIOUS IN " "	(SRPIS)					
020 GET/GETN/GET INDEXES	(SGET)					
031 STORE	(STORE)		×	×	×	
032 MODIFY	(SMDFY)		×	×	×	
033 ERASE	(SRASE)					
034 ERASE ELEMENT	(SEREL)	×	×	×		
041 CONNECT	(SCONN)				×	
042 DISCONNECT	(SDCON)					
043 CONNECT AFTER	(SCONA)				×	×
044 CONNECT BEFORE	(SCONB)				×	×
045 INSERT	(SINSR)		×	×		
046 REMOVE	(SREMO)					
050 OPEN DATA BASE	(SOPDB)					
051 CLOSE DATA BASE	(SCLDB)					
052 READY REALM	(SRRLM)					
053 FINISH REALM	(SFRLM)					
054 CHANGE PASSWORD	(SCHPW)					
060 REMEMBER	(SREMB)					
061 FORGET	(SFORG)					
062 LOCK	(SLOCK)					
063 UNLOCK	(SUNLK)					
072 CHECKPOINT	(SCHPO/GCHPO)					
074 ROLL BACK	(SROLL)					

DATABASE EXCEPTION CONDITION SUMMARY

SYNTAX ERRORS

DML STATEMENT CODE		CODE -1	610 Invalid input parameter value -1	620 Parameters not consistent -1	623 Too long return value -1
001 FIND USING KEY	(SFTCH)				
002 FIND FIRST BETWEEN LIMITS USING KEY	(SFEBL)			×	
003 FIND FIRST IN REALM	(SRFIR)				
004 FIND LAST BETWEEN LIMIT	(SFLBL)				
005 GET SCHEMAS INFORMATION	(SINFO)				
006 TRANSACTION BEGIN	(SUBEG)				
007 TRANSACTION END	(SUEND)				
011 FIND NEXT IN SET	(SRNSM)				
012 FIND PRIOR IN SET	(SRPSM)				
013 FIND FIRST IN SET	(SRFSM)				
014 FIND LAST IN SET	(SRLSM)				
015 FIND OWNER	(SRSOW)				
016 FIND NEXT IN SEARCH REGION	(SRNIS)				
018 FIND PREVIOUS IN " "	(SRPIS)				
020 GET/GETN/GET INDEXES	(SGET)		×		×
031 STORE	(STORE)		×		
032 MODIFY	(SMDFY)		×		
033 ERASE	(SRASE)		×		
034 ERASE ELEMENT	(SEREL)		×		
041 CONNECT	(SCONN)				
042 DISCONNECT	(SDCON)				
043 CONNECT AFTER	(SCONA)				
044 CONNECT BEFORE	(SCONB)				
045 INSERT	(SINSR)				
046 REMOVE	(SREMO)				
050 OPEN DATA BASE	(SOPDB)				
051 CLOSE DATA BASE	(SCLDB)				
052 READY REALM	(SRRLM)		×		
053 FINISH REALM	(SFRLM)		×		
054 CHANGE PASSWORD	(SCHPW)				
060 REMEMBER	(SREMB)		×		
061 FORGET	(SFORG)		×		
062 LOCK	(SLOCK)		×		
063 UNLOCK	(SUNLK)				
072 CHECKPOINT	(SCHPO/GCHPO)				
074 ROLL BACK	(SROLL)				

DATABASE EXCEPTION CONDITION SUMMARY

ERASE RECORD EXCEPTION CONDITIONS	CODE	710 Attempt to erase owner of non-empty set	711 Attempt to erase owner of non-empty set, ERASE partially executed	720 Illegal erase code in multi-user environment	730 Record erased by concurrent run-unit	740 Loop in set structure	741 Loop in set structure, ERASE partially executed
DML STATEMENT CODE		-1	-1	-1	-1	-1	-1
001 FIND USING KEY	(SFTCH)						
002 FIND FIRST BETWEEN LIMITS USING KEY	(SFEBL)						
003 FIND FIRST IN REALM	(SRFIR)						
004 FIND LAST BETWEEN LIMIT	(SFLBL)						
005 GET SCHEMAS INFORMATION	(SINFO)						
006 TRANSACTION BEGIN	(SUBEG)						
007 TRANSACTION END	(SUEND)						
011 FIND NEXT IN SET	(SRNSM)				×		
012 FIND PRIOR IN SET	(SRPSM)				×		
013 FIND FIRST IN SET	(SRFSM)				×		
014 FIND LAST IN SET	(SRLSM)				×		
015 FIND OWNER	(SRSOW)				×		
016 FIND NEXT IN SEARCH REGION	(SRNIS)				×		
018 FIND PREVIOUS IN " "	(SRPIS)						
020 GET/GETN/GET INDEXES	(SGET)				×		
031 STORE	(STORE)						
032 MODIFY	(SMDFY)				×		
033 ERASE	(SRASE)	×	×	×	×	×	×
034 ERASE ELEMENT	(SEREL)				×		
041 CONNECT	(SCONN)				×		
042 DISCONNECT	(SDCON)				×		
043 CONNECT AFTER	(SCONA)				×		
044 CONNECT BEFORE	(SCONB)				×		
045 INSERT	(SINSR)				×		
046 REMOVE	(SREMO)				×		
050 OPEN DATA BASE	(SOPDB)						
051 CLOSE DATA BASE	(SCLDB)						
052 READY REALM	(SRRLM)						
053 FINISH REALM	(SFRLM)						
054 CHANGE PASSWORD	(SCHPW)						
060 REMEMBER	(SREMB)				×		
061 FORGET	(SFORG)						
062 LOCK	(SLOCK)				×		
063 UNLOCK	(SUNLK)						
072 CHECKPOINT	(SCHPO/GCHPO)						
074 ROLL BACK	(SROLL)						

DATABASE EXCEPTION CONDITION SUMMARY

RELATIONSHIP EXCEPTION CONDITIONS	CODE	0	0	0	0	0	-1	0	-1	-1	-1	-1				
		810 Record is already connected to set	820 INDEX key item is already inserted in index table	830 Record is not connected to set	835 Referenced record is not located in set occurrence	840 Record type is not member of given set type	850 INDEX key item is not inserted in index table	860 Attempt to modify owner set item of non-empty set	870 Record type is not owner of given set type	871 Set type is defined as automatic	872 INDEX key is defined as automatic				
DML STATEMENT CODE																
001 FIND USING KEY	(SFTCH)															
002 FIND FIRST BETWEEN LIMITS USING KEY	(SFEBL)															
003 FIND FIRST IN REALM	(SRFIR)															
004 FIND LAST BETWEEN LIMIT	(SFLBL)															
005 GET SCHEMAS INFORMATION	(SINFO)															
006 TRANSACTION BEGIN	(SUBEG)															
007 TRANSACTION END	(SUEND)															
011 FIND NEXT IN SET	(SRNSM)				×	×										
012 FIND PRIOR IN SET	(SRPSM)				×	×										
013 FIND FIRST IN SET	(SRFSM)								×							
014 FIND LAST IN SET	(SRLSM)								×							
015 FIND OWNER	(SRSOW)				×	×										
016 FIND NEXT IN SEARCH REGION	(SRNIS)															
018 FIND PREVIOUS IN " "	(SRPIS)															
020 GET/GETN/GET INDEXES	(SGET)															
031 STORE	(STORE)															
032 MODIFY	(SMDFY)							×								
033 ERASE	(SRASE)															
034 ERASE ELEMENT	(SEREL)							×								
041 CONNECT	(SCONN)	×				×				×						
042 DISCONNECT	(SDCON)			×		×				×						
043 CONNECT AFTER	(SCONA)	×			×	×				×						
044 CONNECT BEFORE	(SCONB)	×			×	×				×						
045 INSERT	(SINSR)		×								×					
046 REMOVE	(SREMO)						×				×					
050 OPEN DATA BASE	(SOPDB)															
051 CLOSE DATA BASE	(SCLDB)															
052 READY REALM	(SRRLM)															
053 FINISH REALM	(SFRLM)															
054 CHANGE PASSWORD	(SCHPW)															
060 REMEMBER	(SREMB)															
061 FORGET	(SFORG)															
062 LOCK	(SLOCK)															
063 UNLOCK	(SUNLK)															
072 CHECKPOINT	(SCHPO/GCHPO)															
074 ROLL BACK	(SROLL)															

DATABASE EXCEPTION CONDITION SUMMARY

READY MODE EXCEPTION CONDITIONS	CODE	880 Attempt to finish realm not in ready mode	881 Realm is not in ready mode	882 Attempt to ready realm in ready mode	884 Database has already been opened for this run-unit	885 Database in error mode. A realm has not been properly finished before an interrupt
DML STATEMENT CODE		0	-1	0	0	-1
001 FIND USING KEY	(SFTCH)		×			
002 FIND FIRST BETWEEN LIMITS USING KEY	(SFEBL)		×			
003 FIND FIRST IN REALM	(SRFIR)		×			
004 FIND LAST BETWEEN LIMIT	(SFLBL)					
005 GET SCHEMAS INFORMATION	(SINFO)					
006 TRANSACTION BEGIN	(SUBEG)					
007 TRANSACTION END	(SUEND)					
011 FIND NEXT IN SET	(SRNSM)					
012 FIND PRIOR IN SET	(SRPSM)					
013 FIND FIRST IN SET	(SRFSM)					
014 FIND LAST IN SET	(SRLSM)					
015 FIND OWNER	(SRSOW)					
016 FIND NEXT IN SEARCH REGION	(SRNIS)					
018 FIND PREVIOUS IN " "	(SRPIS)					
020 GET/GETN/GET INDEXES	(SGET)					
031 STORE	(STORE)		×			
032 MODIFY	(SMDFY)					
033 ERASE	(SRASE)					
034 ERASE ELEMENT	(SEREL)					
041 CONNECT	(SCONN)					
042 DISCONNECT	(SDCON)					
043 CONNECT AFTER	(SCONA)					
044 CONNECT BEFORE	(SCONB)					
045 INSERT	(SINSR)					
046 REMOVE	(SREMO)					
050 OPEN DATA BASE	(SOPDB)				×	
051 CLOSE DATA BASE	(SCLDB)					
052 READY REALM	(SRRLM)			×		×
053 FINISH REALM	(SFRLM)	×				
054 CHANGE PASSWORD	(SCHPW)					
060 REMEMBER	(SREMB)					
061 FORGET	(SFORG)					
062 LOCK	(SLOCK)					
063 UNLOCK	(SUNLK)					
072 CHECKPOINT	(SCHPO/GCHPO)					
074 ROLL BACK	(SROLL)					

DATABASE EXCEPTION CONDITION SUMMARY

RESOURCE ALLOCATION EXCEPTION CONDITIONS	CODE	910 Space in realm is exhausted	920 Space in index table is exhausted	930 Maximum number of remembered temporary database keys exceeded	940 Maximum number of remembered search region indicators exceeded
DML STATEMENT CODE		-1	-1	-1	-1
001 FIND USING KEY	(SFTCH)				
002 FIND FIRST BETWEEN LIMITS USING KEY	(SFEBL)				
003 FIND FIRST IN REALM	(SRFIR)				
004 FIND LAST BETWEEN LIMIT	(SFLBL)				
005 GET SCHEMAS INFORMATION	(SINFO)				
006 TRANSACTION BEGIN	(SUBEG)				
007 TRANSACTION END	(SUEND)				
011 FIND NEXT IN SET	(SRNSM)				
012 FIND PRIOR IN SET	(SRPSM)				
013 FIND FIRST IN SET	(SRFSM)				
014 FIND LAST IN SET	(SRLSM)				
015 FIND OWNER	(SRSOW)				
016 FIND NEXT IN SEARCH REGION	(SRNIS)				
018 FIND PREVIOUS IN " "	(SRPIS)				
020 GET/GETN/GET INDEXES	(SGET)				
031 STORE	(STORE)	x	x		
032 MODIFY	(SMDFY)	x	x		
033 ERASE	(SRASE)				
034 ERASE ELEMENT	(SEREL)				
041 CONNECT	(SCONN)				
042 DISCONNECT	(SDCON)				
043 CONNECT AFTER	(SCONA)				
044 CONNECT BEFORE	(SCONB)				
045 INSERT	(SINSR)		x		
046 REMOVE	(SREMO)				
050 OPEN DATA BASE	(SOPDB)				
051 CLOSE DATA BASE	(SCLDB)				
052 READY REALM	(SRRLM)				
053 FINISH REALM	(SFRLM)				
054 CHANGE PASSWORD	(SCHPW)				
060 REMEMBER	(SREMB)			x	x
061 FORGET	(SFORG)				
062 LOCK	(SLOCK)				
063 UNLOCK	(SUNLK)				
072 CHECKPOINT	(SCHPO/GCHPO)				
074 ROLL BACK	(SROLL)				

DATABASE EXCEPTION CONDITION SUMMARY

DML STATEMENT CODE	CODE	PROTECTION EXCEPTION CONDITIONS: -1	950 Realm not readied for this usage mode	951 Realm readied for exclusive update by another run-unit	952 Realm not assigned	953 Attempt to ready realm for exclusive update when realm is readied for load or update by another run-unit	954 Attempt to ready realm for exclusive update when records in this realm are locked to another run-unit
001 FIND USING KEY	(SFTCH)						
002 FIND FIRST BETWEEN LIMITS USING KEY	(SFEBL)						
003 FIND FIRST IN REALM	(SRFIR)						
004 FIND LAST BETWEEN LIMIT	(SFLBL)						
005 GET SCHEMAS INFORMATION	(SINFO)						
006 TRANSACTION BEGIN	(SUBEG)						
007 TRANSACTION END	(SUEND)						
011 FIND NEXT IN SET	(SRNSM)						
012 FIND PRIOR IN SET	(SRPSM)						
013 FIND FIRST IN SET	(SRFSM)						
014 FIND LAST IN SET	(SRLSM)						
015 FIND OWNER	(SRSOW)						
016 FIND NEXT IN SEARCH REGION	(SRNIS)						
018 FIND PREVIOUS IN " "	(SRPIS)						
020 GET/GETN/GET INDEXES	(SGET)						
031 STORE	(STORE)		×				
032 MODIFY	(SMDFY)		×				
033 ERASE	(SRASE)		×				
034 ERASE ELEMENT	(SEREL)		×				
041 CONNECT	(SCONN)		×				
042 DISCONNECT	(SDCON)		×				
043 CONNECT AFTER	(SCONA)		×				
044 CONNECT BEFORE	(SCONB)		×				
045 INSERT	(SINSR)		×				
046 REMOVE	(SREMO)		×				
050 OPEN DATA BASE	(SOPDB)						
051 CLOSE DATA BASE	(SCLDB)						
052 READY REALM	(SRRLM)			×	×	×	×
053 FINISH REALM	(SFRLM)						
054 CHANGE PASSWORD	(SCHPW)						
060 REMEMBER	(SREMB)						
061 FORGET	(SFORG)						
062 LOCK	(SLOCK)			×			
063 UNLOCK	(SUNLK)						
072 CHECKPOINT	(SCHPO/GCHPO)						
074 ROLL BACK	(SROLL)						

DATABASE EXCEPTION CONDITION SUMMARY

PRIVACY EXCEPTION CONDITIONS	CODE	-1	982 Privacy breach on realm	-1	983 Privacy breach on record	-1	984 Privacy breach on record. ERASE partially executed	-1	990 Privacy breach counter overflow	-1	991 Privacy breach counter overflow. ERASE partially executed
DML STATEMENT CODE											
001 FIND USING KEY	(SFTCH)										
002 FIND FIRST BETWEEN LIMITS USING KEY	(SFEBL)										
003 FIND FIRST IN REALM	(SRFIR)										
004 FIND LAST BETWEEN LIMIT	(SFLBL)										
005 GET SCHEMAS INFORMATION	(SINFO)										
006 TRANSACTION BEGIN	(SUBEG)										
007 TRANSACTION END	(SUEND)										
011 FIND NEXT IN SET	(SRNSM)										
012 FIND PRIOR IN SET	(SRPSM)										
013 FIND FIRST IN SET	(SRFSM)										
014 FIND LAST IN SET	(SRLSM)										
015 FIND OWNER	(SRSOW)										
016 FIND NEXT IN SEARCH REGION	(SRNIS)										
018 FIND PREVIOUS IN " "	(SRPIS)										
020 GET/GETN/GET INDEXES	(SGET)			×				×			
031 STORE	(STORE)										
032 MODIFY	(SMDFY)			×				×			
033 ERASE	(SRASE)			×		×		×		×	
034 ERASE ELEMENT	(SEREL)			×				×			
041 CONNECT	(SCONN)			×				×			
042 DISCONNECT	(SDCON)			×				×			
043 CONNECT AFTER	(SCONA)			×				×			
044 CONNECT BEFORE	(SCONB)			×				×			
045 INSERT	(SINSR)			×				×			
046 REMOVE	(SREMO)			×				×			
050 OPEN DATA BASE	(SOPDB)										
051 CLOSE DATA BASE	(SCLDB)										
052 READY REALM	(SRRLM)		×					×			
053 FINISH REALM	(SFRLM)										
054 CHANGE PASSWORD	(SCHPW)										
060 REMEMBER	(SREMB)										
061 FORGET	(SFORG)										
062 LOCK	(SLOCK)										
063 UNLOCK	(SUNLK)										
072 CHECKPOINT	(SCHPO/GCHPO)										
074 ROLL BACK	(SROLL)										

APPENDIX F

SUMMARY OF THE DML ROUTINE NUMBERS

Table of Routine Numbers

The SIBAS routine logging uses one set of routine numbers. SIBAS error reporting uses another set as seen in the database exception condition summary. The former is logged on the routine log. The latter is returned by calls to SDBEC.

CALL	MEANING	STATE	LOG NO.	DML NO.	INDEX
ACCDD	accumulate-double-integer	run	48	20	4.2.23
ACCFD	accumulate-floating	run	46	20	4.2.23
ACCID	accumulate-integer	run	47	20	4.2.23
BSEQU	initiate-critical-sequence	run	44		5.3.3
CHCOM	change-communication-modus	dba	98		5.4.16
ESEQU	end-critical-sequence	run	45		5.3.3
GCHPO	give-checkpoint (to SIBAS)	run	55	72	5.4.8
INLOG	initiate-logging (all kinds)	rdy	96		5.4.3
OFLOG	set-routine-logging-off	run	87		5.4.5
ONLOG	set-routine-logging-on	run	88		5.4.5
RBLAN	read-SIBAS-common	run	36		5.4.16
RELSI	release-SIBAS	run	59		5.4.14
RESIB	reserve-SIBAS	run	58		5.4.14
SABOR	forced-close	dba	104	51	5.4.17
SCHPO	return-checkpoint (from SIBAS)	run	32	72	5.4.8
SCHPW	change-password	run	28	54	4.2.20
SCLDB	close-database	run	22	51	4.2.2
SCONA	connect-after	run	27	43	4.2.12
SCONB	connect-before	run	26	44	4.2.12
SCONN	connect	run	16	41	4.2.12
SDBEC	accept	run	29		4.2.21
SDCON	disconnect	run	18	42	4.2.13
SEREL	erase-element	run	30	34	4.2.22
SETDV	set-SIBAS-system-number	run			5.4.13
SEXMC	execute-macro	run	56		5.4.15
SFEBL	find-first-between-limits	run	23	2	4.2.5
SFINI	finish (recovery)	rec	89		5.4.2
SFLBL	find-last-between-limits	run	53	4	4.2.5
SFORG	forget	run	13	61	4.2.17
SFRLM	finish-realm	run	21	53	4.2.4
SFTCH	find-using-key	run	1	1	4.2.5
SFTGT	fetch-get	run	33		4.2.24
SGET	get	run	7	20	4.2.8
SGETN	get-n-records	run	34		4.2.8
SGIXN	get-indexes	run	43		4.2.8
SIBIO	priviliged	all	103		5.4.16
SICON	set-conditions-for-reprocessing	rec	92		5.4.10
SINFO	get-schemas-information	run	35	5	4.2.25
SINSR	insert	run	15	45	4.2.14
SISTA	set-SIBAS-state	all	101		5.4.16
SLOCK	lock	run	12	62	4.2.28
SMDFY	modify	run	8	32	4.2.9
SMESS	log-message	run	49		5.4.6
SOPDB	open-database	run	20	50	4.2.1
SPASS	set-passive	rdy	97		5.4.2
SPAUS	pause	run	94		5.4.2
SRASE	erase	run	10	33	4.2.11
SRECO	recover	dba	102		5.4.2
SREMB	remember	run	11	60	4.2.16
SREMO	remove	run	17	46	4.2.15
SREPR	reprocess-routine-log	rec	91		5.4.11

CALL	MEANING	STATE	LOG NO.	DML NO.	INDEX
SRFIR	find-first-in-realm	run	24	3	4.2.5
SRFSM	find-first-in-set	run	2	13	4.2.6
SRLSM	find-last-in-set	run	4	14	4.2.6
SRNIS	find-next-in-search-region	run	25	16	4.2.6
SRNSM	find-next-in-set	run	3	11	4.2.6
SROLL	roll-back	rec	90	74	5.4.9
SRPIS	find-prior-in-search-region	run	54	18	4.2.6
SRPSM	find-prior-in-set	run	5	12	4.2.6
SRRLM	ready-realm	run	19	52	4.2.3
SRSOW	find-set-owner	run	6	15	4.2.7
SRUN	run-database	dba	99		5.4.2
START	start-database	rdy	95		5.4.1
STGET	set-SIBAS-state	all	57		5.4.16
STOPS	stop-database	dba	106		5.4.1
STORE	store	run	9	31	4.2.10
STREP	get-status-from-reprocessing	rec	93		5.4.2
STRLG	set-terminal-log (on/off)	dba	105		5.4.16
STRLG	turn-on/off-terminal-log	dba	105		5.4.16
SUBEG	transaction-unit-begin	run	39	6	4.2.26
SUEND	transaction-unit-end	run	40	7	4.2.26
SUNLK	unlock	run	14	63	4.2.19
SUPLA	update-database-in-place	dba	100	75	5.4.12
UTBLK	write-log-buffer-onto-r-log	run	31		5.4.7
ZTRB	priviliged	run	38		5.4.16

*

APPENDIX G

CONSTANTS AND LIMITATIONS

CONSTANTS and LIMITATIONS

SIB-DRL	Max. number of lines in an initiation run	ca. 2600 lines
	Max. number of lines in a redefinition run	ca. 500 lines
	Maximum number of OS-FILEs in one database (including the object schemas)	12 OS-FILEs
	Maximum number of REALMs in one database (including the SIBAS system realm)	63 REALMs
	Maximum number of ITEMs in one REALM (including GROUP items)	ca. 100 ITEMs
	Maximum number of ITEMs in one GROUP	ca. 50 ITEMs
	Maximum number of INDEXes on one REALM	all ITEMs are indexes
	Maximum number of SETs in one database	49 SETs
	Maximum number of pages in an OS-FILE	65533 pages
	Maximum number of pages in a REALM	65533 pages
	Maximum page size for an OS-FILE	with BIM: 2 Kw without BIM: 8 Kw
	Maximum record size	realm page size: 2
	Maximium number of records by page	254 records
	Maximum size of an index key item	sysrealm p. size: 5
	Maximum size of an item	500 words
DML	Maximum number of SIBAS processes by machine	6 processes
	Maximum number of concurrently updating run-units on the same database	ND100: 63 run-units ND500: 90 run-units
	Maximum size of Routine Log (R-LOG)	32000 1K-pages
	Maximum size of the Before Image area (BIM log)	2000 1K-pages
	Maximum number of temporary database keys by run-unit	30 tdbk
	Maximum number of temporary search-region indicators by run-unit	5 tsri
	Maximum size of SIBAS-DML work area	31 Kw
	Minimum size of SIBAS-DML work area	10 Kw

COMMENTS TO INDEX

References are given for key words only where relevant information for the word is given. Lower case letters are used throughout except for commands and certain abbreviations, like SIBAS, CSRI etc.

References are separated by '/' if placed on the same line, or frequently to save space, on new lines.

Additional information is given for the key words, for example subsystems to which the key words belong like (DML), or by giving the context in which the word is used, for example checking, verification etc.

The key word is usually repeated if the information on the line is different from the preceding. See for example references for 'SIBAS'.

References with the asterisk prefix '*' mean that the key word is used in the table of contents, and is usually the main reference if more than one is given. See for example reference 'Calc key verification 6.1.11.2'.
Cross references are frequently given, for example as follows:

Temporary database key
Database - temporary * key
Key - temporary database *

where '*' replaces the key word in the context.

The additional information is usually separated from the key word by '-', for ex'Bucket - main 2.1' which here could mean 'Main bucket'. But this is no general rule.

INDEX

ACCDD	— call	(DML)	4.2.23
ACCEPT		(DML)	4.2.21
ACCFD	— call	(DML)	4.2.23
ACCID	— call	(DML)	4.2.23
ACCUMULATE		(DML)	4.2.23
Abbreviation lookup		(DBM)	6.1
Accept — error condition			*4.2.21
Accept — statemment		(DML)	4.2.3
Access — direct			2.4.1.1
Access — random			2.1/2.2.3
Access — relative			2.4.1.1
Access principles			*2.4.1
Access ways			2.4.1.1
Accumulate			*4.2.23
Accumulate —		(DML)	2.4.3.1
Additional sys realm		(DRL)	3.9
Additions — schema		(DRL)	3.3
Algorithm — hashing			2.2.3
Algorithm — random		(DRL)	3.25
Application programs —		load	*4.4
Area — main			2.1/(DRL)3.10
Area — overflow			2.1/(DRL)3.10
Authorized users		(DBM)	6.1.9.1
Automatic maint index			2.2.4/2.4.2.2
Automatic storage class			2.3.2.4/2.4.2.1(DRL)3.13
Automatic update		(DRL)	3.14
Back-up			(DBA)5.1/*5.3.6
Background applications		(DML)	4.4
Before-image logging			*5.3.5
Begin/End sequence			*5.4.4
Bit position		(DRL)	3.11
Bucket			2.1/(DRL)3.25
Bucket — calc			2.2.5
Bucket — main			2.1
Bucket — overflow			2.1
Byte position		(DRL)	3.11
CHANGE CALC-REALM		(DRL)	3.23
CHANGE SERIAL-REALM		(DRL)	3.22
CHANGE SET		(DRL)	3.24
CHANGE SYSTEM-REALM		(DRL)	3.21
CHANGE-COM-PROC		(SERV)	6.2
CHANGE PASSWORD		(DML)	4.2.20
CHCOM	— call	(DBA)	5.4.16
CLOSE-DATA-BASE		(DML)	4.2.2
CLOSE-DATABASE		(SERV)	6.2
COMPRESS		(DBM)	6.1.10
COMPRESS INDEX		(DBM)	6.1.10

CONNECT	(DML)	4.2.12
CONNECT-AFTER	(DML)	4.2.12
CONNECT-BEFORE	(DML)	4.2.12
CRUI	(DML)	4.2.16
CRUI — (find)	(DML)	4.2.5
CRUI — current run-unit ind		2.4.1.2
CSRI	(DML)	4.2.16
CSRI — (find)	(DML)	4.2.5
CSRI — curr.search reg.ind		2.3.1/2.4.1.2
Calc key		2.2.3/2.3.2.1/2.4.1.2
Calc key	(DRL)	3.9/(DML)4.2.5
Calc key-checking	(DBM)	6.1.11.2
Calc key verification	(DBM)	*6.1.11.2/6.1.11.1
Calc location mode		2.2.3
Calc mode		2.2.3/2.2.4
Calc mode location		2.1
Calc realm — change		*3.23
Calc realm — new		*3.10
Calc realms	(DRL)	3.25
Calc records		2.2.3/2.2.5
Calls — subroutine		1.2
Chain — set type		*2.3.2.3
Chain — double link		2.3.2.3
Chain — single link		2.3.2.3
Chaining		2.3.2.3
Change calc-realm	(DRL)	3.1/*3.23
Change serial-realm	(DRL)	3.1/*3.22
Change set	(DRL)	3.1/*3.24
Change system-realm	(DRL)	3.1/*3.21
Change-password	(DML)	2.2.4.3/*4.2.20
Changes — schema	(DRL)	3.3
Character	(DRL)	3.11
Character item		2.2.1
CHECKING CONSISTENCY		5./(DBM)*6.1.11
Checkpoint		*5.3.2
Checkpoint	(DBA)	5.3.4/*5.4.8
Checkpoint-id	(DBA)	5.3.5.4/5.4
Checkpoints -- taking		*5.3.4.3
Class — removal		2.3.2.4
Class — storage		*2.3.2.4
CLEAR-SYSTEM-REALM	(DBM)	6.1.16
Close-data-base		*4.2.2
Cobol — language cons.		*4.3.2
Cobol program (ex)	(DML)	4.3.2
Cobol storage area	(DML)	4.1
Codasyl		2.2/2.3.2.3/2.4
Codasyl report		1.1
Code — location		2.1
Collating sequence (SIBAS)		2.2.4
Compress index table	(DRL)	3.25
Computational (Cobol)		2.2.1

Computational-3 fields		4.3.2
Concept — Database		*2.1/2.2.6
Concurrency except.condition		App E
Concurrent processing		*2.4.3/5.4.13
Concurrent run-units — prot.		2.4.3.4
Conflicts — ready stat.	(DML)	4.2.3
Connect —	(DML)	2.3.2.4
	(DML)	2.4.2.1
Connect — records		*2.4.2.1/*4.2.12
Consistency — schema	(DRL)	3.2
Consistency checking		5./(DBM) *6.1.11
Core space — minimize	(DML)	4.1
Critical sequence		5.4.4
Critical sequence (logging)		*5.3.3.1
Currency indicator exc.cond		App E
Currency indicators		*2.4.1.2
Current password — set	(DBM)	6.1.9.1
Current password — setting		*2.4.4.3
Current run unit ind	(DML)	4.2.5
	(CRUI)	2.4.1.2/2.4.1.1
Current search reg ind	(CSRI)	2.3.1/2.4.1.2
Current-search-reg-ind	(DML)	4.2.5
DATABASE ADMINISTRATION		*5.
DATA MANIPULATION LANGUAGE		*4.
DATABASE-STATUS	(SERV)	6.2
DB definition syntax	(DRL)	3.3
DBA calls	(DBA)	*5.4.16
DBA password	(DBM)	6.1
DBA state	(DBA)	5.2/5.4.1/5.4.2
DBA-password	(DRL)	3.5
DBCS — DB control system		1.2/2.4.4.2
	(DML)	4.1
DBEC — DB exceptionel cond.		2.4.1.1/2.4.3.4
		(DML)4.2/*7.3
DBEC — fatal errors		7.1
DBM — example		*6.1.13
DBM module (mainten.)	(DBM)	*6.1
DBM password	(DBM)	6.1.9.1
DBM statements summary		*App C
DBMS — DB management system		1.1/3.
DDR — data def/redef prog		2.2.6
DEFINE DBA-PASSWORD	(DBM)	6.1.9.2
DEFINE GLOBAL-PASSWORD	(DBM)	6.1.9.2
DEFINE LOCAL-PASSWORD	(DBM)	6.1.9.2
DEFINITION/REDEF LANGUAGE		*3.
DELETE GROUP	(DRL)	3.18
DELETE INDEX	(DRL)	3.16
DELETE ITEM	(DRL)	3.17
DELETE SET	(DRL)	3.15
DISCONNECT	(DML)	4.2.13
DISPLAY	(DBM)	6.1.9.4
DML diagnostics		*7.3

DML resident tables	(DRL)	3.26
DML routine log numbers		*App F
DML statements		2.3.2.4/4.
DML-statements summary		*App A
DRL module	(DRL)	3.2
DRL statements summary		*App B
DRL — examples	(DRL)	3.26
Database		2.2/*2.2.6
Database — close		*4.2.2
Database — define		3.
Database — dimension param		*3.25
Database — document	(DRL)	3.2
Database — force close		*5.4.17
Database — main components		2.2.6
Database — open		*4.2.1
Database — privacy		*2.4.4
Database — redefine		3.
Database — temporary * key		2.4.1.2
Database — update	(DBA)	*5.4.12
Database — utilities		*6.
Database Except Conditions		*App E
Database Maintain, example		*6.1.13
Database Managem. — ex	(DBM)	6.1.13
Database adm.functions	(DBM)	6.1
Database administrator	(DBM)	6.1
Database concept		*2.1/2.2.6
Database contr system	(DBCS)	1.2/(DML)4.1
Database definition	(DRL)	3.2
Database exept.cond	(DBEC)	2.4.1.1
Database exept.cond	(DML)	4.2/*7.3
Database maintenance module		2.4.4.1/*6.1
Database management system		1.1
Database names	(DRL)	3.3
Database numbers	(DRL)	3.3
Database parameters	(DRL)	3.25
Database redefinition	(DRL)	3.2
Database repairs	(DBM)	6.1.11.1
Database reservation		*2.4.3.1
Database system		*1.1
Database unavailable	(DBA)	5.4.14
Data Manipul.Language	(DBM)	6.1.9.1
Data consistency error	(DRL)	3.2
Data division	(DML)	4.3.2
Data independence		3.1
Data manipulation		*2.4
Data relations		1.1/*2.3
Data structure		*2.2
Database-name	(DML)	4.2
Database definition	(DRL)	3.26
Database initiation (ex)	(DRL)	3.26
Deadlock — realm protec		2.4.3.3
Deadlocks (concurr proc.)		2.4.3

Define Database		3.
Define checkpoint	(DBA)	5.4.8
Define password	(DBM)	*6.1.9.2
Defined limits (key value)		2.4.1.2
Definition — Database	(DRL)	3.2
Definition syntax — DB	(DRL)	3.3
Definition/Redef module		*3.2
Delayed update	(DBA)	5.3/*5.3.4
Delete group	(DRL)	3.1/*3.18
Delete index	(DRL)	3.1/*3.16
Delete item	(DRL)	3.1/*3.17
Delete set	(DRL)	3.1/*3.15
Deletions — schema	(DRL)	3.3
Density — packing	(DRL)	3.25
Description of calls	(DBA)	*5.4
Description of manual		*ix
Device — internal		1.2
Dimension Database param		*3.25
Direct access		2.4.1.1
Direct find		*4.2.5
Directory	(DRL)	3.7
Disconnect — records		*2.4.2.1/*4 2.13
Disconnect —	(DML)	2.3.2.4/2.4.2.1
Display	(DML)	4.3.2
Display (Cobol)		2.2.1
Display password/priv	(DBM)	*6.1.9.4
Distribution — bucket	(DRL)	3.25
Documentation — DB	(DRL)	3.2
Double limk	(DRL)	3.13
Double link chain		2.3.2.3
Dump realm (to file)	(DBM)	6.1.14
Duplicate key		2.2.3/2.3.1
Duplicate key value		2.3.2.1
Duplicates	(DRL)	3.9
END	(DRL)	3.6
ERASE	(DML)	4.2.11
ERASE-ELEMENT	(DML)	4.2.22
ERROR AND EXCEPTION CONDS		*7
EXIT	(DRL)	3.6
EXIT	(SERV)	6.2
EXIT	(DBM)	6.1.3
EXPLAIN	(SERV)	6.2
Ecceptional conds — fatal err		7.1
Element — erase		*4.2.22
Empty sets		2.3.2.2
Encoded call from	(DML)	4.
End	(DRL)	3.1
Entry — record	(DRL)	3.25
Erase record		*4.2.11
Erase —	(DML)	2.3.2.4
Erase record except.cond.		App E
Erase — element		*4.2.22

Error — DB example	(DRL)	3.26
Error — data consist	(DRL)	3.2
Error — schema consist	(DRL)	3.2
Error reports	(DML)	4.2
Error-flag — reset	(DBM)	*6.1.8
Errors — fatal		*7.1
Errors — interface		*7.2
Errors — simulator		*7.2
Errors — syntax	(DRL)	3.2
Examples — running DBM		*6.1.13
Examples — DB managem	(DBM)	6.1.13
Examples — Def/Redef	(DRL)	3.26
Exception conditions		*6.
Exclusive update — realm		2.4.3.2
Exclusive-update	(DML)	4.2.3
Execute-macro	(DBA)	*5.4.15
Extended monitor mode		2.4.3.4
FETCH-GET		4.2.24
FIND-FI-BETW-LIM-US-KEY	(DML)	4.2.5
FIND-FIRST-IN-REALM	(DML)	4.2.5
FIND-FIRST-IN-SET	(DML)	4.2.6
FIND-LAST-IN-SET	(DML)	4.2.6
FIND-NEST-IN-SET	(DML)	4.2.6
FIND-NEXT-IN-SEARCH-REG	(DML)	4.2.6
FIND-OWNER	(DML)	4.2.7
FIND-PRIOR-IN-SET	(DML)	4.2.6
FIND-USING-KEY	(DML)	4.2.5
FINISH	(SERV)	6.2
FINISH	(DBM)	6.1.5
FINISH-REALM	(DML)	4.2.4
FORCE-CLOSE	(SERV)	6.2
FORGET	(DML)	4.2.17
FORTRAN		2.2.1
FREE-SPACE-STAT	(DBM)	6.1.12
Facilities — log/recover		*5.3
Fatal errors		*7.1
fetch-get		4.2.24
File — SIBAS I/O update		*5.3.4.2
File — new OS		*3.7
Find	(DML)	2.4.1.1
Find — direct		*4.2.5
Find — relative		*4.2.6
Find-set-owner		*4.2.7
Finish	(DBA)	*5.4.2
Finish-realm		*4.2.4
Finish-realm	(DBM)	*6.1.5
Flag — reset error	(DBM)	*6.1.8
Floating	(DRL)	3.11
Floating item		2.2.1
Force-close database	(DBA)	*5.4.17
Forget	(DML)	2.4.2.1
Forget — record		*4.2.17

Forget — all-records	(DML)	4.2.17
Forget-all-search-reg	(DML)	4.2.17
Forget-record	(DML)	4.2.17
Forget-search-region	(DML)	4.2.17
Fortran — language cons		*4.3.1
Fortran array	(DML)	4.1
Fortran program (ex)	(DML)	4.2.23
Free-space-statistics	(DBM)	*6.1.12
GCHPO — call		5.4.8
GET	(DML)	4.2.8
GET-INDEXES	(DML)	4.2.8
GET-SCHEMAS-INFORMATION		4.2.25
GET-SIBAS-STATE	(SERV)	6.2
GET-UPDATE-STATE	(SERV)	6.2
GETN	(DML)	4.2.8
GIVE-CHECKPOINT	(SERV)	6.2
GIVE-MESS-TO-SIBAS call	(SERV)	6.2
Get		*4.2.8
Get —	(DML)	2.4.1.1
Get indexes		*4.2.8
get-schemas-information		4.2.25
Get-state		*5.4.1
Getn		*4.2.8
Global passwords	(DBM)	6.1.9.1
Group — delete		*3.18
Group — new		*3.12
Group items		2.2/*2.2.2
HELP	(SERV)	6.2
Hashing algorithm		2.2.3
High-limit	(DML)	4.2
Host language considerations		*4.3
Host language program	(DML)	4.1
I/O-table	(DRL)	3.25
INITIATE-LOG	(SERV)	6.2
INLOG — call		5.4.3
INSERT	(DML)	4.2.14
INTRODUCTION TO SIBAS		*1.
Idget — assembly routin	(DML)	4.4
Implementation of SIBAS		*1.2
Index — autom.maintained		2.2.4/2.4.2.2
Index — compress	(DBM)	6.1.10
Index — delete		*3.16
Index — manual.maintained		2.2.4/2.4.2.2
Index — new		*3.14
Index compression	(DBM)	*6.1.10
Index description table	(DRL)	3.25
Index key		*2.2.4/2.3.2.1/2.4.1.2
Index key	(DML)	4.2.5
Index key — checking	(DBM)	6.1.11.3
Index key property	(DRL)	3.16
Index key verification	(DBM)	6.1.11.1/*6.1.11.3
Index levels		2.2.4

Index table		2.1
Index table — compress	(DRL)	3.25
Index table — representation		2.2.4
Index tables — size	(DRL)	3.25
Indexes — get		*4.2.8
Indicator — search region		2.3.1
Indicator — current run-unit		2.4.1.1
Indicator — status	(DML)	4.2.3
Indicators — currency		*2.4.1.2
Information retrievel		2.1
Information storage		2.1
Initiate-log	(DBA)	*5.4.3
Initiation run (ex)	(DRL)	3.26
Initiation steps	(DRL)	3.26
Insert — record		*4.2.14
Inserting — index		*2.4.2.2
Integer	(DRL)	3.11
Integer item		2.2.1
Interface	(DBA)	5.1
Interface errors		*7.2
Intrnal device		1.2/(DBA)5.1
Involuted set		2.1
Involuted set type		2.3.2/2.3.2.3
Involuted set type	(DRL)	3.13
Item		2.2
Item — character		2.2.1
Item — delete		*3.17
Item — floating		2.2.1
Item — integer		2.2.1
Item — member set		2.3.2.1
Item — new		*3.11
Item — owner set		2.3.2.1
Item name	(DRL)	3.11
Item type	(DRL)	3.11
Item value exception cond.		App E
Item-list	(DML)	4.2
Item-values	(DML)	4.2
Items		2.1/*2.2.1
Key		*2.2.4
Key — calc		2.2.3/2.3.2.1/2.4.1.2
Key — calc	(DRL)	3.9
Key — duplicate		2.2.3/2.3.1
Key — duplicate value		2.3.2.1
Key — index		2.3.2.1/2.4.1.2
Key — record		2.1
Key — search		2.4
Key — temporary database*		2.4.1.2
Key — unique		2.2.3/2.3.2.3
Key size	(DRL)	3.25
Key value		2.1/2.4.1.1
Key value — store	(DRL)	3.25
Key-length	(DML)	4.2

Key-name	(DML)	4.2
Key-value	(DML)	4.2
LOAD	(DBM)	6.1.14
LOCK	(DML)	4.2.18
Language — host lang	(DML)	4.1
Language considerations		*4.3
Level — index tables		2.2.4
Levels — protection		2.4.3
Limits — (key value)		2.4.1.2
Link — double	(DRL)	3.13
Link — double chain		2.3.2.3
Link — single	(DRL)	3.13
Link — single chain		2.3.2.3
List — rememberd*		2.4.1.2
Load	(DML)	4.2.3
Load — realm usage mode		2.4.3.2
Load application programs		*4.4
Load records to (ex)	(DML)	4.2.23
Load/unload	(DBM)	*6.1.14
Loading progr w SIBAS	(DML)	4.4
Local passwords	(DBM)	6.1.9.1
Locate record (find)	(DML)	4.2.5
Location — calc mode		2.1
Location code		2.1
Location mode — serial		2.2.3
Lock — record		*4.2.18
Lock records — realm protec		2.4.3.3
Lock out — record level		*2.4.3.3
Log message	(DBA)	*5.4.6
Log-buf-on-rout-log	(DBA)	*5.4.7
Log-directory	(DBA)	5.4
Log-file-name	(DBA)	5.4
Logging — Routine On/Off		*5.4.5
Logging — before-image		*5.3.5
Logging — routine		*5.3.3
Logging facilities	(DBA)	5./*5.3
Logical relationship		2.3.2.3
Low-limit	(DML)	4.2
MAKE-MODEFILE	(DBM)	6.1.15
MODIFY	(DML)	4.2.8
MSI — member set item value		2.4.2.1
Macro		2.4.3.1
Macro — user defined	(DBA)	5.4.15
Main area		2.1/2.2.3
Main area	(DRL)	3.10/3.25
Main bucket		2.1
Main system realm	(DRL)	3.9
Maintenance module		*6.1
Maintenance/timing	(DBA)	5.4.16
Make mode-file	(DBM)	*6.1.15
Manipulation — data		*2.4
Manual description		*ix

Manual storage class		2.3.2.4/2.4.2.1
Manual storage class	(DRL)	3.13
Manual update	(DRL)	3.14
Manually maintained index		2.2.4/2.4.4.2
Manually maintained set		2.4.2.1
Max-value (key)	(DRL)	3.14
Member — set		2.1
Member — single set		2.3.2
Member record		2.3.2.1
Member set item		2.3.2.1
Member set item	(DRL)	3.13
Message — log	(DBA)	*5.4.6
Message — run-time	(SIBAS)	*7.5
Message — run-time	(SIBIO)	*7.4
Message to operator	(DBA)	5.4.6
Min-value (key)	(DRL)	3.14
Minimize core space	(DML)	4.1
Mode (param.desc)	(DML)	4.2
Mode — production	(DRL)	3.2
Mode — test	(DRL)	3.2
Mode — usage/protection		*2.4.3.2
Modefile — for Load/Unl	(DBM)	*6.1.15
Modify	(DML)	2.4.2.1
Modify — record		*4.2.9
Module — Database mainten		*6.1
Module — Definition/Redef		*3.2
Modules available		*PREFACE vii
Monitor mode — extended*		2.4.3.4
Multi-member set		2.3.2
Multi-user	(DBA)	5.1
NEW CALC REALM	(DRL)	3.9
NEW GROUP	(DRL)	3.12
NEW INDEX	(DRL)	3.14
NEW ITEM	(DRL)	3.11
NEW OS FILE	(DRL)	3.7
NEW SERIAL REALM	(DRL)	3.9
NEW SET	(DRL)	3.13
NEW SYSTEM REALM	(DRL)	3.8
Name specification exc.cond		App E
Names — Database	(DRL)	3.3
New OS-file	(DRL)	3.1
New calm realm	(DRL)	3.1
New group	(DRL)	3.1
New index	(DRL)	3.1
New item	(DRL)	3.1
New serial realm	(DRL)	3.1
New set	(DRL)	3.1
New system realm	(DRL)	3.1
New-password	(DML)	4.2
Next — (search)	(DML)	4.2.6
No-found	(DML)	4.2
No-of-items	(DML)	4.2

No-of-realms	(DML)	4.2
No-wanted	(DML)	4.2
Non-protected-realm		2.4.3.2
Non-protection	(DML)	4.2.3
Nonreentrant programs	(DML)	4.4
Notification of change		*2.4.3.4
Null value — key		2.2.4
Null value — privacy item		2.4.4.2
Number — prime		2.2.3
Numbers — Database	(DRL)	3.3
OFLOG — call	(DBA)	5.4.5
ONLOG — call	(DBA)	5.4.5
OPEN-DATA-BASE	(DML)	4.2.1
OPEN-DATABASE	(SERV)	6.2
OS file — delete		*3.20
OS file — new		*3.7
OS-file	(DRL)	3.8/3.9/3.20
OSI — owner set item value		2.4.2.1
Object schema		2.2.6
Object schema		2.2.6/(DRL)3.2
Occurrence of set		*2.3.2.2
Occurrence		2.2
Occurrence — record		2.1
Occurrence — set		2.3.2/2.3.2.3
Octal numbers	(DBM)	6.1
Open — statement		2.2.6
Open-data-base		*4.2.1
Open-data-base	(DBM)	2.4.4.1
Operating system file	(DRL)	3.4
Operator — message to*	(DBA)	5.4.6
Optimum value (buckets)	(DRL)	3.25
Option code (erase)	DML)	4.2.11
Option-code	(DML)	4.2
Organisation — Real-time		*5.1
Overflow area		2.1/2.2.3
Overflow area	(DRL)	3.10
Overflow area — calc		2.2.5
Overflow bucket		2.1
Owerflow page	(DRL)	3.25
Owner	(DBA)	5.4
Owner — fined set*		*4.2.7
Owner — set*		2.1
Owner record		2.3.2.1
Owner set item		2.3.2.1
Owner set item	(DRL)	3.13
PATCH	(DBM)	6.1.7
PAUSE	(SERV)	6.2
PRINT	(DBM)	6.1.6
Packing density	(DRL)	3.25
Page size	(DRL)	3.7/3.25
Parameter descrition	(DRL)	*4.2
Parameter values	(DRL)	6.1

Passive	(DBA)	5.4.2
Passive state	(DBA)	5.2
Password	(DML)	4.2
Password — DBA	(DRL)	3.5
Password — change		*4.2.20
Password — define	(DBM)	*6.1.9.2
Password — privacy		2.4.4
Password — record occurrence		2.4.4.2
Password — remove	(DBM)	*6.1.9.3
Password — setting current		*2.4.4.3
Password/priv — displ	(DBM)	*6.1.9.4
Patch	(DBM)	*6.1.7
Patching Database	(DBA)	5.
Pause	(DBA)	*5.4.2
PLANC		4.3.3.
Pointer — record		2.3.2.3
Pointer — record	(DRL)	3.25
Pointers		1.1/(DBM)6.1
Prerequisite knowledge		*viii
Primary area — calc		2.2.5
Primary key (index)		2.2.4
Prime number		2.2.3
Prime number	(DRL)	3.25
Principles — access		*2.4.1
Principles of SIBAS		*2.
Print	(DBM)	*6.1.6
Prior — (search)	(DML)	4.2.6
Privacy	(DBM)	*6.1.9
Privacy exception cond.		App E
Privacy — Database level		*2.4.4.1
Privacy — Record level		*2.4.4.2
Privacy — tables		2.4.4
Privacy definition	(DBA)	5.
Privacy item	(DBM)	6.1.9.1
Privacy item — record		2.4.4.1
Privacy system		*2.4.4
Privacy table	(DBM)	6.1.9.1
Privacy-item	(DRL)	3.11
Processing — concurrent		*2.4.3
Processing — real-time		1.2
Production mode	(DML)	3.2/3.5
Program examples	(DML)	4.2.23
Programs — Load application		*4.4
Protect — concurr run-units		2.4.3.4
Protection exception cond.		App E
Protection levels		2.4.3
Protection mode	(DBM)	6.1.9.1
Protection mode	(DML)	4.2/4.2.3
Protection mode — Realm*		*2.4.3.2
READY	(DBM)	6.1.4
READY-REALM	(DML)	4.2.3
RECOVER	(SERV)	6.2

RELSI — call	(DBA)	5.4.14
REMBER	(DML)	4.2.16
REMOVE	(DML)	4.2.15
REMOVE-PASSWORD	(DBM)	6.1.9.3
REPROCESS-DATABASE	(SERV)	6.2
RESET-ERROR-FLAG	(DBM)	6.1.8
RESIB — call	(DBA)	5.4.14
RETURN — CHECKPOINT	(SERV)	6.2
RK — record key		2.2.4
ROLL-BACK	(SERV)	6.2
RP — record pointrer		2.2.4
RT-common	(DBA)	5.1
RT-common	(DML)	4.4
RUN-DATABASE	(SERV)	6.2
Random access		2.1/2.2.3
Randomizing alorithme	(DRL)	3.25
Readers		*The READER viii
Ready — statement		2.2.6
Ready conflicts	(DML)	4.2.3
Ready mode except.cond		App E
Ready state	(DBA)	5.4.1/5.4.2/5.2
Ready-realm		*4.2.3
Ready-realm	(DBM)	*6.1.4
Ready-realm —	(DML)	2.4.4.3
Real-time organis.of SIBAS		*5.1
Real-time processing		1.2
Realm		2.2/*2.2.5/2.4.1.2
Realm — SIBAS system	(DRL)	3.5
Realm — delete		*3.19
Realm — finish		*2.4.2
Realm — finish Mainten.mod		*6.1.5
Realm — new calc		*3.10
Realm — new serial		*3.9
Realm — new system		*3.8
Realm — ready		*4.2.3
Realm — ready	(DBM)	*6.1.4
Realm — system		2.2.5
Realm — user		2.2.5
Realm description table	(DRL)	3.25
Realm protection mode		*2.4.3.2
Realm size		2.2.5
Realm size	(DRL)	3.25
Realm space utilization	(DBM)	6.1.12
Realm usage mode		*2.4.3.2
Realm-name	(DML)	4.2
Realms — calc	(DRL)	3.25
Realms — user system*	(DRL)	3.25
Record descript.table	(DRL)	3.25
Record key		2.1/2.2.4
Record length	(DRL)	3.10
Record occurrence		2.1
Record occurrence — privacy		2.4.4

Record pointer		2.2.4
Record types		2.1/2.2/*2.2.3
Records — connect/disconnect		*2.4.2
Recover	(DBA)	*5.4.2
Recovery	(DBA)	5.1/5.4.4
Recovery — message	(DBA)	5.4.6
Recovery — speed up	(DBA)	5.4.5
Recovery facilities	(DBA)	5./*5.3
Recovery state	(DBA)	5.2/5.4.2
Redefine Database		3.
Redefine password		2.4.4
Redefinition run (ex)	(DRL)	3.26
Reentrant applications	(DML)	4.4
Regenerate (Database)	(DBM)	6.1.11.1
Region — search		2.1
Regions for searching		*2.3.1
Related manua s		*ix
Relations — data		1.1
Relations of data		*2.3
Relationship		2.4.1.1
Relationship except.cond.		App E
Relationship — logical		2.3.2.3
Relative access		2.4.1.1
Relative find		*4.2.6
Release SIBAS		2.4.3.1
Release SIBAS	(DBA)	*5.4.14
Remainder		2.2.3
Remember-record	(DML)	*4.2.16
Remember-search-region	(DML)	4.2.16
Remembered list	(CRUI)	2.4.1.2
Remembered list	(CSRI)	2.4.3.4
Removal class		2.3.2.4
Remove — record		*4.2.15
Remove password	(DBM)	*6.1.9.3
Removing — records		*2.4.2.2
Repair (Database)	(DBM)	6.1.11.1
Report — Codasyl		1.1
Repro-status	(DBA)	*5.4.2
Reprocess — set-conds	(DBA)	*5.4.10
Reprocess-routine-log	(DBA)	*5.4.11
Reprocessing		*5.3.3.2
Reprocessing (conds)	(DBA)	5.4.10
Reservation — Database		*2.4.3.1
Reserve SIBAS		2.4.3.1
Reserve SIBAS	(DBA)	*5.4.14
Reset-error-flag Mainten.mod		*6.1.8
Resident tables (DML)	(DRL)	3.26
Resolut. — ready confl	(DML)	4.2.3
Resource allocation exc.cond		App E
Restart — Updat/Rout.	(DBA)	5.3.7.2
Restart Back-up/Routine log		*5.3.7.1
Restart Update file/rout.log		*5.3.7.2

Retrieval	(DML)	4.2.3
Retrieval — realm usage mode		2.4.3.2
Retrieval except.condition		App E
Retrieval — information		2.1
Ring buffers	(DBA)	5.1
Roll-back	(DBA)	5.3.5.6/*5.4.9
Roll-back	(SIBIO)	*5.3.4.5
Roll-back facilities	(DBA)	5.
Routine log — volume	(DBA)	5.4.5
Routine log file — def	(DBA)	5.4.3
Routine log numbers — summary		App F
Routine logging	(DBA)	5.3
Routine logging		*5.3.3
Routine logging On/Off		*5.4.5
Routine-log — reproc	(DBA)	*5.4.11
Run	(DBA)	*5.4.2
Run-id	(DBA)	5.4
Run-time message	(SIBIO)	*7.4
Run-time message	(SIBAS)	*7.5
Run-unit		2.4.1.1
Run-unit — current indicator		2.4.1.1
Run-unit password		2.4.4
Running state	(DBA)	5.2/5.4.2
SCHPO — call	(DBA)	5.4.8
SCHPW — call	(DML)	4.2.20
SCLDB — call	(DML)	4.2.2
SCONA — call	(DML)	4.2.12
SCONB — call	(DML)	4.2.12
SCONN — call	(DML)	4.2.12
SDBEC — call	(DML)	4.2.21
SDCON — call	(DML)	4.2.13
SEREL — call	(DML)	4.2.22
SERVC — call	(DBA)	5.4.16
SET-COND-FOR-REPR	(SERV)	6.2
SET-PASSIVE	(SERV)	6.2
SET-ROUT-LOG-OFF	(SERV)	6.2
SET-ROUT-LOG-ON	(SERV)	6.2
SETDV — call	(DBA)	5.4.13
SEXMC — call	(DBA)	5.4.15
SFEBL — call	(DML)	4.2.5
SFINI — call	(DBA)	5.4.2
SFORG — call	(DML)	4.2.17
SFRLM — call	(DML)	4.2.4
SFTCH — call	(DML)	4.2.5
SFTGT — call	(DML)	4.2.24
SGET — call	(DML)	4.2.8
SGETN — call	(DML)	4.2.8
SGIXN — call	(DML)	4.2.8
SIB-DEM statments summary		*App C
SIB-DRL	(DBA)	5.5
SIB-DRL	(DRL)	3.25
SIB-DRL statements summary		*App B

SIB-SERVICE statements		*App D
SIB-SYS-GEN:batc	(DBA)	5.4.15
SIB2-DML-B-MH		4.4.3
SIB2-DML-R-MH		4.4.3
SIB2A	(DBA)	5.1
SIBAS — installing		*5.5
SIBAS — release		2.4.3.1
SIBAS — reserve		2.4.3.1
SIBAS — start	(DBA)	*5.4.1
SIBAS — stop	(DBA)	*5.4.1
SIBAS Database system		*1.1
SIBAS I/O system	(DBA)	5.3.5.1
SIBAS I/O system	(updating)	*5.3.4.1
SIBAS PRINCIPLES		*2.
SIBAS communication	(DBA)	5.1
SIBAS implementation		*1.2
SIBAS libraries	(DML)	4.4
SIBAS performance	(DBA)	5.4.1
SIBAS process number	(DBA)	5.4.13
SIBAS realms	(DRL)	3.25
SIBAS run-time messages		7.5
SIBAS segments	(DBA)	5.1
SIBAS service program	(DBM)	*6.2
SIBAS states		*5.2
SIBAS system number	(DML)	4.4
SIBAS system number	(DBA)	5.4.13
SIBAS system realm		2.2.6
SIBAS system realm	(DRL)	3.25
SIBAS user	(DML)	4.4
SIBAS-files		*PREFACE vii
SIBAS-service	(DBA)	5.
SIBAS/SIBIO	(DBA)	5.3.5.3
SIBINTER		1.2
SIBIO	(DBA)	5.3.5.1/5.3.5.6
SIBLIB-2N-MH		4.4.3
SIBLIB-1N-MH		4.4.3
SIBLIB-1R-MH		4.4.3
SICON — call	(DBA)	5.4.10
SINFO — call	(DML)	4.2.25
SINSR — call	(DML)	4.2.14
SINTRAN		1.2
SISTA — call	(DBA)	5.4.16
SLOCK — call	(DML)	4.2.18
SMDFY — call	(DML)	4.2.9
SMESS — call	(DBA)	5.4.6
SOPDB — call	(DML)	4.2.1
SPASS — call	(DBA)	5.4.2
SPAUS — call	(DBA)	5.4.2
SRASE — call	(DML)	4.2.11
SRECO — call	(DBA)	5.4.2
SREMB — call	(DML)	4.2.16
SREMO — call	(DML)	4.2.15

SREPR	— call	(DBA)	5.4.11
SRFIR	— call	(DML)	4.2.5
SRFSM	— call	(DML)	4.2.6
SRLSM	— call	(DML)	4.2.6
SRNIS	— call	(DML)	4.2.6
SRNSM	— call	(DML)	4.2.6
SROLL	— call	(DBA)	5.4.9
SRPSM	— call	(DML)	4.2.6
SRRLM	— call	(DML)	4.2.3
SRSOW	— call	(DML)	4.2.7
SRUN	— call	(DBA)	5.4.2
STANDARD-REPROC		(SERV)	6.2
START	— call	(DBA)	5.4.1
START	— call	(DBM)	6.1.2
START INIT DATABASE		(DRL)	3.4
START-DATABASE		(SERV)	6.2
START REDEFINITION		(DRL)	3.4
STOP	— call	(DBM)	6.1.3
STOP-DATABASE		(SERV)	6.2
STOPS	— call	(DBA)	5.4.1
STORE	— call	(DML)	4.2.10
STRLG	— call	(DBA)	5.4.16
STREP	— call		5.4.2
SUBEG	— call	(DML)	4.2.26
SUEND	— call	(DML)	4.2.26
SUNLK	— call	(DML)	4.2.19
SUPER-START		(SERV)	6.2
SUPER-STOP		(SERV)	6.2
SUPLA		(DBA)	5.3.5.5
SUPLA	— call	(DBA)	5.4.12
Schema — object			2.2.6
Schema — object		(DRL)	3.2
Schema — source			2.2.6
Schema additions		(DRL)	3.3
Schema changes		(DRL)	3.3
Schema consist.error		(DRL)	3.2
Schema consistency		(DRL)	3.2
Schema deletions		(DRL)	3.3
Schema traslator			2.2.6
Scratch file		(DRL)	3.5
Search — sequential			2.2.3
Search key			*2.2.4/2.4
Search region			2.1
Search region		(DML)	4.2.5
Search region — temp*		ind	2.4.1.2
Search regions			*2.3.1
Secondary key (index)			2.2.4
Security breach		(DML)	4.2.1
Sequence		(DBA)	5.4.4
Sequence — Begin/End		(DBA)	*5.4.4
Sequence — critical logging			*5.3.3.1
Sequence — statement		(DRL)	3.3

Sequence-name	(DBA)	5.4
Sequential search		2.2.3
Serial location mode		2.1/2.2.3
Serial mode		2.2.3/2.2.4
Serial realm — change		*3.22
Serial realm — new		*3.9
Serial realms	(DRL)	3.25
Service prog — SIBAS	(SERV)	6.2
Set — change		*3.24
Set — delete		*3.15
Set — empty		2.3.2.2
Set — involuted		2.1
Set — multi-member		2.3.2
Set — new		*3.13
Set — single member		2.3.2
Set SIBAS system no	(DBA)	*5.4.13
Set description table	(DRL)	3.25
Set item		*2.3.2.1
Set item — member	(DRL)	3.13
Set item — owner	(DRL)	3.13
Set member		2.1
Set occurrence		2.3.2/*2.3.2.2/2.3.2.3
Set owner		2.1
Set owner — find		*4.2.7
Set type		2.3.2.3
Set type — involuted		2.3.2/2.3.2.3
Set type — involuted	(DRL)	3.13
Set types — (examples)		2.3.2.4
Set verification	(DBM)	6.1.11.1/*6.1.11.4
Set-conds-for-reproc	(DBA)	*5.4.10
Set-name	(DML)	4.2
Set-passive	(DBA)	*5.4.2
Sets		*2.3.2
Setting current password		*2.4.4.3
Short form (statement)	(DML)	4.
Siflg — ext sybol	(DML)	4.4
Simulator	(DBA)	5.1
Simulator	(DML)	4.4
Simulator errors		*7.2
Single link	(DRL)	3.13
Single link chain		2.3.2.3
Single member set		2.3.2
Sintran OS-file	(DRL)	3.4
Size — index tables	(DRL)	3.25
Size — object schema	(DRL)	3.4
Size — realm	(DRL)	3.25
Source schema		2.2.6
Space requirements	(DRL)	3.25
Stack area	(DML)	4.4
Start —	(DBM)	*6.1.2
Start SIBAS	(DBA)	5./*5.4.1
Start initiation	(DRL)	3.1

Start redefinition	(DRL)	3.1
Statement sequence	(DRL)	3.3
Statistics — free-space	(DBM)	*6.1.12
Status	(DBA)	5.4
Status	(DML)	4.2
Status — interface errors		7.2
Status — simulator errors		7.2
Status indicator	(DML)	4.2.3
Stop —	(DBM)	*6.1.3
Stop SIBAS	(DBA)	5./*5.4.1
Storage — information		2.1
Storage class		*2.3.2.4/2.3.2.2/2.4.2.1
Storage class — autom.	(DRL)	3.13
Storage class — automatic		2.3.2.4
Storage class — manual		2.3.2.4
Storage class — manual	(DRL)	3.13
Store — record		*4.2.10
Store —	(DBM)	2.4.2.1
	(DBM)	2.3.2.4
Store randomly	(DRL)	3.25
Structuring principles		2.2
Subroutine calls		1.2
Subschema fasility	(DML)	4.1
Successful execution	(DML)	2.4.1.1
Suppress documentation	(DRL)	3.4/3.5
Syntax — Database def	(DRL)	3.3
Syntax check	(DRL)	3.2
Syntax description	(DBM)	6.1
Syntax errors	(DRL)	3.2
Syntax errors except.cond		App E
System failure/restart		*5.3.7
System realm		2.2.5
System realm (SIBAS)	(DRL)	3.5
System realm — SIBAS		2.2.6
System realm — addit	(DRL)	3.9
System realm — change		*3.21
System realm — new		*3.8
System realm — system	(DRL)	3.9
System realm — user		2.2.6
TP — table pointer		2.2.4
TPS	(DBA)	5.3.5.3
TURN-OFF-TERM-LOG	(SERV)	6.2
TURN-ON-TERM-LOG	(SERV)	6.2
Table — index descript	(DRL)	3.25
Table — realm descript	(DRL)	3.25
Table — record descr	(DRL)	3.25
Table — set descript	(DRL)	3.25
Table pointer		2.2.4
Tables — DML resident	(DRL)	3.26
Tables — index		1.1
Temp-data-base-key	(DML)	4.2
Temp-search-reg-ind	(DML)	4.2

Temporary database key		2.4.1.2
Temporary search region ind		2.4.1.2
Test mode	(DRL)	3.2
Test mode	(DRL)	3.5
Time	(DBA)	5.4
Trailing blanks	(DML)	4.2
Transaction units		4.2.26
Translator — schema		2.2.6
Trigger-code	(DBA)	5.4.12
UNLOAD	(DBM)	6.1.14
UNLOCK	(DML)	4.2.19
UPDATE-DATAB-IN-PL	(SERV)	6.2
UTBLK — call	(DBA)	5.4.7
UTILITIES		*6.
Unconsistent DB name	(DML)	4.2.1
Unique key		2.2.3/2.3.2.3
Unload/load	(DBM)	*6.1.14
Unlock — records		2.4.3.3/*4.2.19
Unsuccessful execution	(DML)	2.4.1.1
Updat-dat-bas-in-plac	(DBA)	*5.4.12
Update — old Database	(DRL)	3.2
Update	(DML)	4.2.3
Update — realm usage mode		2.4.3.2
Update — automatic	(DRL)	3.14
Update — manual	(DRL)	3.14
Update file	(DBA)	5.3.5.2
Update file — SIBAS I/O		*5.3.4.2
Update-in-place	(DBA)	5.3.4
	(DBA)	5.3.5.2
	(DBA)	5.3.5.5
Update-in-place	(SIBIO)	*5.3.4.4
Update-ratio	DBA)	5.4.12
Updating — delayed		*5.3.4
Usage mode	(DBM)	6.1.9.1
Usage mode	(DML)	4.2.3
	(DML)	4.2
Usage mode — Realm		*2.4.3.2
User applic program		2.2.6
User realm		2.2.5
User subroutine	(DBA)	5.4.15
User system realm		2.2.6
User system realms	(DRL)	3.25
VERIFY CALC	(DBM)	6.1.11.2
VERIFY INDEX	(DBM)	6.1.11.3
VERIFY MODE	(DBM)	6.1.11.1
VERIFY SET	(DBM)	6.1.11.4
Value-length	(DML)	4.2
Verification — calc key	(DBM)	*6.1.11.2
Verification — index key	(DBM)	*6.1.11.3
Verification — set	(DBM)	*6.1.11.4
Verfication mode	(DBM)	6.1.11.1

Virtual memory layout	(DBA)	5.1
Warning — (realm protec)		2.4.3.4
Warnings — DB example	(DRL)	3.26
Word	(DRL)	3.11
Word boundary	(DML)	4.1
Work-area-size	(DBA)	5.4.1

The Competitive European Computer Company

NORSK DATA A.S JERIKOVN. 20 P.O. BOX 4 LINDEBERG GÅRD OSLO 10 NORWAY
TEL.: 02 - 30 90 30 - TELEX: 18661