



MIMER

System Management Handbook

Version 8.1

Copyright © 1998 Sysdeco Mimer AB

MIMER version 8.1 System Management Handbook

August, 1998

Copyright © 1998 Sysdeco Mimer AB.

Published by Sysdeco Mimer AB,

P.O. Box 1713,

S-751 47 Uppsala, Sweden.

Tel +46(0)18-18 50 00.

Fax +46(0)18-18 51 00.

Internet: <http://www.mimer.com>

Produced by Sysdeco Mimer AB, Uppsala, Sweden.

All rights reserved under international copyright conventions.

The contents of this manual may be printed in limited quantities for use at a Mimer installation site. No parts of the manual may be reproduced for sale to a third party.

FOREWORD

Documentation objectives

This manual is a general handbook for system administrators, describing in detail the various areas of responsibility when administering a MIMER database system.

Prerequisites

There are no prerequisites for users of this manual. However, it is to the user's advantage to be familiar with, and have a working knowledge of, the host computer operating system.

Organization of this manual

- Chapter 1** is a brief introduction to this manual.
- Chapter 2** describes the components of a MIMER database system, particularly those relevant to the system administrator's responsibilities.
- Chapter 3** describes how the system administrator establishes a MIMER database.
- Chapter 4** covers the subject of managing a MIMER database server including: a description of various factors and parameters which might affect server performance.
- Chapter 5** describes how the EXPORT/IMPORT functionality permits base tables to be transferred between different installations.
- Chapter 6** describes how the BACKUP and RESTORE functionality provides facilities for taking full or incremental backups of the database and for restoring data in the event of a system crash..
- Chapter 7** describes how the READLOG functionality reads the contents of the LOGDB databank, which can provide an audit trail or other relevant information in the event of a system failure.
- Chapter 8** describes how the STATISTICS functionality maintains statistical information in the data dictionary concerning the usage of tables and indexes. This information is used by the MIMER/SQL compiler in optimizing access paths for data manipulation statements.
- Chapter 9** describes how the Databank Check (DBC) functionality is provided to check the physical integrity of databanks.
- Chapter 10** describes how the DBOPEN functionality opens all the databanks in the database. This facility can be used after a databank has not been closed properly (e.g. because of a system failure). If this is not done, the implicit databank check performed when a user first opens the databank can cause an appreciable delay.

The manual also contains the following appendices:

- Appendix A** describes matters relevant when accessing a database in single user mode.
- Appendix B** describes the contents and use of the SQLHOSTS file used on UNIX and VMS.
- Appendix C** describes the contents and use of the MULTIDEFS file used on UNIX and VMS.
- Appendix D** describes the data dictionary tables.

Related MIMER publications

- **MIMER/SQL Reference Manual** contains a complete description of the syntax and usage of all statements in MIMER/SQL. The Reference Manual is a necessary complement to this manual.
- **MIMER/SQL User's Manual** contains a description of the Batch SQL facilities. A user-oriented guide to the SQL statements is also included, which may provide help for less experienced users in formulating statements correctly (particularly the SELECT statement, which can be quite complex).
- **MIMER/SQL Programmer's Manual** contains a description of how MIMER/SQL can be used within the context of application programs, written in conventional programming languages.
- **MIMER Shadowing Handbook** describes the optional Shadowing module, which allows several concurrent copies of a databank to be updated at the same time. This provides extra resilience to disk crashes and allows "backup on the fly". The SQL statements which are part of the Shadowing API are described in the SQL Reference Manual.
- **MIMER Guides** comprise documentation containing platform-specific information. A set of one or more guides is provided, where required, for each platform on which MIMER is supplied.
- **MIMER Release Notes** contain platform-specific information relating to the MIMER release for which they are supplied.

Acronyms and trademarks

BSQL	Batch SQL, a program used for execution of SQL statements which are read from a command file or entered interactively.
Data source	ODBC term for a database.
Database home directory	The directory where the system databank file containing the data dictionary and some other files that form part of the database are located.
DCL	Digital Command Language.
Embedded SQL	The term used for SQL statements when they are embedded in a traditional host programming language.
MIMKEY8	A file, on Unix and VMS, containing the MRS license key (refer to the <i>MIMER Guides</i> for the exact file name).
MULTIDEFS	A file, on Unix and VMS, containing parameters for controlling the database server for a MIMER database.
ODBC	Microsoft's Open Database Connectivity, a specification for a database API in the C language, independent of any specific DBMS or operating system.
PSM	Persistent Stored Modules (i.e. Stored Procedures).
Shadow	A MIMER databank may have one or more shadows. If the databank is shadowed, there will be one file for each shadow. A shadow is a copy of the original (master) databank and is continuously updated by MIMER. If the master databank is lost, operations will continue without interruption by automatically using the shadow databank. The MIMER/SQL backup facilities will automatically access the shadow of a databank if one is present. (If database shadowing is being used, the <i>MIMER Shadowing Handbook</i> should be consulted).
SQL	Structured Query Language.
SQLHOSTS	A file, on Unix and VMS, containing lookup information for all MIMER databases accessible from the current node.
Unix	Unix is a trademark registered by the X/Open Company.
VMS	VMS is a trademark registered by Compaq.
Windows	Windows is a trademark registered by Microsoft.

(All other trademarks are the property of their respective holders.)

CONTENTS

1	INTRODUCTION	
1.1	System management responsibilities	1-2
2	THE DATABASE ENVIRONMENT	
2.1	The data dictionary	2-1
2.2	Idents	2-3
2.3	Databanks	2-4
2.3.1	MIMER system databanks	2-4
2.3.2	User databanks	2-5
2.3.3	Locating databank files	2-6
2.3.4	Organizing databank files	2-8
2.3.4.1	Allocating disk space	2-8
2.3.4.2	Protecting data against loss	2-9
2.3.4.3	Balanced I/O	2-10
2.3.4.4	Disk speed	2-10
2.3.4.5	Reserved directories	2-10
2.3.5	Altering databank locations	2-11
2.3.6	Accessing databank files	2-11
2.3.7	Databank file deletion	2-12
2.4	Transaction control	2-12
2.5	Database security	2-13
2.5.1	The role of idents in database security	2-13
2.5.2	Access rights and privileges	2-15
2.5.3	Recursive interactions between privileges	2-17
2.5.4	Restriction views	2-18
2.6	Data integrity	2-18
2.6.1	Domains	2-19
2.6.2	Entity integrity	2-19
2.6.3	Referential integrity	2-20
2.6.4	Table integrity	2-20
2.6.5	View integrity	2-21
3	ESTABLISHING A MIMER DATABASE	
3.1	General information	3-1
3.2	Making the database accessible	3-2
3.3	Local databases	3-3
3.4	Remote databases	3-4
3.4.1	Client-server interface	3-4
3.5	Generating the MIMER system databanks	3-5
3.6	Establishing the ident and data structure	3-7
3.7	Managing database connections	3-7
3.7.1	Selecting a database	3-8
3.7.2	The default database	3-9
3.7.2.1	Defining a node-specific default database	3-9
3.7.2.2	Defining a user-specific default database	3-10

3.7.3	Connection names.....	3-10
3.7.4	Troubleshooting remote database connect failures.....	3-12
3.8	MRS license key	3-13
3.9	Running BSQL and UTIL	3-15
3.10	Creating the example database	3-16
3.11	Removing a database.....	3-17
4	MANAGING A DATABASE SERVER	
4.1	System performance.....	4-2
4.1.1	Database server memory areas	4-2
4.1.1.1	Code.....	4-2
4.1.1.2	Data and thread stacks	4-3
4.1.1.3	Bufferpool.....	4-3
4.1.1.4	Communication buffers.....	4-4
4.1.1.5	SQLPOOL	4-5
4.1.2	Number of request threads.....	4-5
4.1.3	Number of background threads	4-6
4.1.4	Database server system requirements	4-6
4.2	Controlling the database server	4-8
4.2.1	MIMCONTROL syntax.....	4-10
4.2.2	MIMCONTROL examples.....	4-12
4.2.3	MIMCONTROL exit codes.....	4-13
4.3	System information - using MIMINFO	4-14
4.3.1	MIMINFO syntax	4-14
4.3.2	The users list.....	4-15
4.3.3	The MIMSERV report.....	4-15
4.3.3.1	MIMSERV output example	4-20
4.3.4	MIMDUMP report	4-22
4.4	Database server log	4-22
5	EXPORT/IMPORT	
5.1	The main menu.....	5-2
5.2	Export options.....	5-2
5.2.1	Functions	5-2
5.2.2	Authorization.....	5-3
5.2.3	Exported files	5-3
5.3	Import options.....	5-4
5.3.1	Import - object creation	5-4
5.3.2	Import - data load	5-7
5.3.3	Authorization.....	5-8
5.4	Load and Unload functions	5-8
5.4.1	Data file formats	5-9
5.4.2	Load operation.....	5-10
5.4.3	Unload operation	5-10
5.4.4	Authorization.....	5-11
5.5	Enter and Leave program ident	5-11
6	BACKUP AND RESTORE	
6.1	Background information	6-3
6.1.1	Database consistency.....	6-4
6.1.2	Databank backups.....	6-5
6.1.3	Databank incremental backups	6-6
6.1.4	Backup versus Incremental Backup.....	6-6
6.1.5	Maximizing data security.....	6-7
6.1.6	SQL statements for backing up databanks	6-8

6.2	Backup and restore of databanks.....	6-10
6.2.1	Making a databank backup.....	6-10
6.2.1.1	Using the SQL system management statements.....	6-10
6.2.1.2	Using the host file system.....	6-11
6.2.2	Restoring a databank.....	6-14
6.3	Backup and restore of system databanks.....	6-15
6.3.1	SYSDB.....	6-15
6.3.2	LOGDB.....	6-17
6.3.3	TRANSDB and SQLDB.....	6-17
6.3.4	Re-creating TRANSDB, LOGDB and SQLDB.....	6-18
7	READLOG FUNCTIONALITY	
7.1	Functions.....	7-2
7.2	Authorization.....	7-3
7.3	Using the READLOG functionality.....	7-3
7.3.1	List definitions (output control).....	7-3
7.3.2	List restrictions.....	7-5
7.3.3	List operations.....	7-6
7.3.4	Change program id.....	7-7
7.4	Output format.....	7-8
8	DATABASE STATISTICS	
8.1	Statistical information.....	8-1
8.2	Authorization.....	8-2
8.3	The SQL statistics statements.....	8-2
8.3.1	Statistics for the entire database.....	8-3
8.3.2	Statistics for specified idents.....	8-3
8.3.3	Statistics for specified tables.....	8-4
8.4	When to use the SQL statistics statements.....	8-4
9	DATABANK CHECK FUNCTIONALITY	
9.1	Runtime malfunctions.....	9-1
9.2	The DBC program.....	9-1
9.2.1	General description.....	9-1
9.2.2	Authorization.....	9-2
9.2.3	User communication.....	9-3
9.2.4	Result file contents.....	9-4
9.2.5	Example of result file.....	9-6
9.2.6	Error messages.....	9-7
10	DBOPEN FUNCTIONALITY	
10.1	Functions.....	10-2
10.2	Authorization.....	10-2
10.3	DBOPEN output example.....	10-3
A	EXECUTING IN SINGLE-USER MODE	
A.1	File protection in single- and multi-user mode.....	A-2
A.2	Specifying single-user mode access.....	A-2
A.3	Accessing in single-user mode.....	A-3
A.4	The SINGLEDEFS parameter file.....	A-4
B	THE SQLHOSTS FILE ON VMS AND UNIX	
B.1	The SQLHOSTS file.....	B-2
B.1.1	DEFAULT section.....	B-3
B.1.2	LOCAL section.....	B-4
B.1.3	REMOTE section.....	B-4

C	THE MULTIDEFS FILE ON VMS AND UNIX	
C.1	The MULTIDEFS parameter file	C-1
C.1.1	MULTIDEFS parameters	C-2
D	DATA DICTIONARY TABLES	
	Summary of data dictionary tables	D-2
	ACCESS	D-3
	ACCCOL	D-4
	BACKUP	D-4
	CODE	D-4
	COLUMN	D-5
	COMMENT	D-6
	DATABANK	D-7
	DBFILE	D-7
	DBNAME	D-7
	DEFAULT	D-8
	DOMAIN	D-8
	FUNCTION	D-9
	IDENT	D-10
	INDEX	D-10
	INDEXCOL	D-11
	MESSAGE	D-11
	OBJECT	D-12
	OBJECT_REF	D-12
	PARAMETER	D-13
	PRIV	D-14
	RESTRICT	D-14
	RESULT	D-15
	ROUTINE	D-16
	SERVINFO	D-16
	SEVERITY	D-16
	SOURCE_DEF	D-16
	SQLLANG	D-17
	SQLMODULE	D-17
	SYNONYM	D-17
	TABLE	D-18
	TABLEX	D-18
	VIEWCOL	D-19
	VIEWDEP	D-19
	VIEWRES	D-19

1 INTRODUCTION

MIMER is an advanced database management system developed by Sysdeco Mimer AB. This manual describes how to establish, manage and maintain a MIMER database.

The information contained in this handbook generally applies to all the platforms supported by MIMER.

From time to time platform-specific notes appear in the general description, presented as follows:

Unix	Denotes information that applies specifically to Unix platforms.
VMS	Denotes information that applies specifically to VMS platforms.
Win	Denotes information that applies specifically to Windows platforms.

There are also some Appendices at the end of this handbook which contain information that applies to specific platforms.

1.1 System management responsibilities

Installation of a MIMER system and the initial creation of the database environment is performed by a specially privileged operating system user, referred to as the **system administrator**. In an established system, the system administrator is also responsible for the maintenance of the installation - system tuning, troubleshooting, backup/restore, and so on. The system administrator must have a good working knowledge of the host computer operating system.

In addition to system administration, there are certain management activities which are performed within the database, i.e. database administration. A specially privileged MIMER ident called SYSADM is created when a database is installed and this ident should be initially used to log in to the database whenever database administration activities are to be undertaken. The ident name SYSADM may not be changed. In some organizations, database administration activities may be carried out by a group of people rather than a single individual.

Some of the MIMER functionality requires privileges and access rights which are initially granted only to the SYSADM user, but which may be passed on to other MIMER users. There may however only be one SYSADM ident in a given database.

SYSADM has SELECT access on the internal MIMER data dictionary tables, permitting direct reading of the meta-data describing the system. In examining the contents of the data dictionary with the functionality provided, the user ident SYSADM has, by default, wider access rights than other users.

The user ident SYSADM does not, however, have general access to the contents of the database. Information stored in user-defined tables may only be accessed by other users if the creator of the table explicitly grants permission. In this context, SYSADM is treated just as any other database user.

2 THE DATABASE ENVIRONMENT

This chapter describes the database environment which is composed of a set of MIMER system databanks, one or more idents authorized to connect to the database and the databanks created by the idents. It also describes database security and data integrity.

The objects which are created in the database (tables, views, domains, modules, procedures, synonyms and indexes) are described in the *MIMER/SQL User's Manual*.

2.1 The data dictionary

The database environment is controlled through a central **data dictionary**, stored in the system databank SYSDB and automatically maintained by MIMER. The data dictionary contains meta-data describing all the objects in the database. System access to the data dictionary tables is performed by internal routines and is transparent to the user.

Restricted facilities for examining the contents of the data dictionary are available to all users through the LIST and DESCRIBE functions in Batch SQL (see the *MIMER/SQL User's Manual* for a more complete description of these facilities). In general, a user may read data dictionary information for database objects to which he or she has access. The Batch SQL facilities use pre-defined views on the data dictionary tables to present the information in a structured form (see the *MIMER/SQL Reference Manual* for documentation on the data dictionary views available to all users).

The SYSADM user ident may read the contents of the data dictionary tables directly, and may grant SELECT access on the tables to any other user ident. The organization of the data dictionary tables is documented in Appendix D of this manual.

No individual user, including SYSADM, may update data dictionary tables directly. All write operations in the data dictionary are performed by internally controlled routines, to ensure consistency within the dictionary.

2.2 Idents

An **ident** in a MIMER system is an authorized user of the system, or the collective identity of a group of users sharing common privileges. Four types of idents are recognized:

- | | |
|---------|--|
| USER | idents are authorized to log on to the MIMER system. User idents are generally associated with specific physical individuals authorized to use the system. |
| OS_USER | idents are a type of user ident with the same username as an operating system account. If a user logged in to the operating system is also defined as an OS_USER, the user may log in to MIMER without providing an additional username and password. |
| PROGRAM | idents may not log on to MIMER, but may be entered from within an application program or interactive environment by using the ENTER statement. Once a program ident is entered, the privileges held by the program ident apply within the application program. Program idents are generally associated with specific functions within the system, not with physical individuals. |
| GROUP | idents are not as such authorized to use the system, but are collective identities for groups of user or program idents. Any privileges granted to or revoked from a group ident automatically apply to all members of the group. Any user or program ident can be a member of as many groups as is required, and a group can include an unlimited number of members. Group idents provide a facility for organizing the privilege structure in the database system. |

When MIMER is installed, the **user** ident SYSADM (used for database administration) and the **group** ident PUBLIC are automatically created. The password for SYSADM is defined when the system is installed (see Section 3.5). All idents in the system belong to the group PUBLIC, so privileges granted to PUBLIC by any user are global to the system. Membership in the group PUBLIC cannot be revoked.

2.3 Databanks

A MIMER database consists of the MIMER system databanks and a number of user databanks.

Each databank is a single physical file in the host file system.

2.3.1 MIMER system databanks

The MIMER system databanks are fundamental to the functioning of a MIMER database and they are created during the initial process of installing a database.

System databanks are not used for storing user-defined information and are not accessed directly by users.

If any one of the system databanks is damaged or missing, attempts to log on to MIMER will fail. Backup and restore procedures for the system databanks are described in Chapter 6.

The MIMER system databanks are:

- | | |
|----------------|---|
| SYSDB | this is the most important system databank as it stores the tables that make up the data dictionary (see Section 2.1). Among other things, the data dictionary holds information about the other databanks that make up the database, the tables each user databank contains, the users (idents) that are known to the MIMER system and the access rights each ident has. |
| LOGDB | this system databank records all write operations performed within transactions on user databanks and SYSDB which have been defined with the LOG option. The information in this databank is used by the Backup and Restore facilities (see Chapter 6) to restore the contents of a database in the event of a system failure. The contents of this databank is in an internal format and does not contain tables in the normal sense. It cannot, therefore, be read by normal database access requests. The READLOG facility is provided (see Chapter 7) to allow the information in this databank to be examined. |
| SQLDB | this system databank stores the local read sets used in transaction handling (see Section 2.4) and is used by MIMER/SQL for temporary storage of result tables. The information in this databank is stored in an internal format and cannot be read by normal database access requests. |
| TRANSDB | this system databank stores the local write sets used in transaction handling (see Section 2.4). The information in this databank is stored in an internal format and cannot be read by normal database access requests. |

2.3.2 User databanks

User databanks contain the tables in the database created by the users of the system. These databanks are created by the user(s) responsible for defining the data contained in the database.

The CREATE DATABANK statement is used to create user databanks (see the *MIMER/SQL Reference Manual*).

Except at the point when tables are created, the existence of databanks is transparent to users and application programs. When access is requested to a table, information in the data dictionary is used by MIMER/SQL to locate the table and make it available if permissible.

The division of a database into databanks is made on the basis of file handling considerations from the operating system viewpoint and on the basis of transaction control considerations from the database viewpoint. The use of databanks allows considerable flexibility in the physical placement of data on the computer system.

Databanks may be defined with the LOG, TRANS or NULL options which determine transaction handling and logging behavior, as follows:

LOG	All operations on the databank are performed under transaction control. All transactions are logged.
TRANS	All operations on the databank are performed under transaction control. No transactions are logged.
NULL	All operations on the databank are performed without transaction control (even if they are requested within a transaction), and are not logged.

Note: Operations performed on databanks with the TRANS or NULL option cannot be restored in the event of system failure (see Chapter 6).

If a databank is dropped from the database, the tables stored in the databank are no longer accessible and the physical databank file is usually automatically deleted from disk.

2.3.3 Locating databank files

The system databank SYSDB is always stored in a file located in the directory defined as the home directory for the database (see Section 3.3).

The file locations of all the other system databanks and the user databanks are stored in the data dictionary. The file specification for the databank file is stored in the data dictionary exactly as specified when the databank was created.

If a databank file specification appears in the data dictionary without a directory name, the database home directory will be used to complete the file specification. This substitution is applied whenever the location of the databank file must be determined, (i.e. when the databank is created, altered and whenever tables stored in it are accessed).

Subsequent redefinition of the database home directory or any variables used in the file specification will, therefore, alter the expected location of such databank files.

Unix

Databases on Unix platforms may be set up with a directory search path instead of a single home directory (see Section 3.3). The first directory in the search path list must be the database home directory and subsequent directories are those in which databank files may be located.

Databanks created without specifying the directory in the file specification may be moved between any of the directories in the search path list without the need to alter anything in the data dictionary. The database server should be stopped while databank files are being moved.

VMS

Whenever a databank file is specified without a directory name under VMS, it must be located in the database home directory.

If a logical name is included in the file specification, this will be recorded in the data dictionary and will be used whenever the location of a databank file must be resolved.

Any logical names used in databank file specifications must be created as GROUP or SYSTEM wide logical names so that the database server process has access to them.

Win

Whenever a databank file is specified without a directory name under Windows, it must be located in the database home directory.

If a databank file specification is given in full, it is unambiguously specified and no variable factors are involved in resolving the location of the file.

Storing file specifications in the data dictionary that are incomplete or contain substitutable elements can provide increased organizational flexibility in that databank files can be moved in the future without the need to change the information held in the data dictionary. For details on how to alter the location information stored in the data dictionary, see Section 2.3.5.

The flexibility achieved by not using full databank file specifications must be weighed against the loss of explicitly specified information from the data dictionary. In addition, the centralized use of mechanisms such as environmental variables or logical names in a complex system requires careful and disciplined management.

In particular, it is necessary for the database server process to have access to all relevant environmental variables and logical names in order to use them when accessing the databanks.

2.3.4 Organizing databank files

There are a number of factors involved in the organization of physical databank files that are important to database security and the overall performance of the MIMER system.

2.3.4.1 Allocating disk space

Whenever possible, pre-allocate file space for databanks early in the lifetime of the databank file system.

The databank creation facilities allow the initial size of a new databank file to be specified in terms of the number of MIMER pages. The size of a MIMER page is 2 kilobytes.

The size of the databank file will be extended automatically by the operating system during the lifetime of the databank as more space is required for data storage. The size by which databank files are incremented is configurable on most platforms.

Unix

Under Unix, the environmental variable `MIMER_EXTEND` can be set to the number of MIMER pages by which all databank files will be extended. The default setting is 25.

VMS

By default, under VMS, databank files will be extended by 1000 VMS blocks at a time. The extend size for a databank file can be altered by using the following DCL command: `SET FILE/EXTENSION=extensionsize file.DBF`. The databank file must not be accessed by the database server or in single user mode when this command is used.

Win

Under Windows, the number of MIMER pages by which all databank files will be extended is determined by the MIMER system and is not configurable.

An attempt to extend a file will fail if the disk is full or any imposed disk quota is exceeded.

Having a large file extension size will result in fewer extension operations on a file and will improve performance slightly if the databank is growing rapidly. However, it will waste disk space since the file will generally contain more unused disk blocks.

Having a small file extension size will reduce wasted disk space since the file size follows the actual size required more closely, but the greater number of file extensions may cause disk fragmentation leading to reduced I/O performance. In addition, if the databank is growing rapidly, the frequently occurring file extension operations may have a negative effect on performance.

A databank file which is created with the size it will actually need in production will generally be accessed more efficiently than one created with a small initial size and then incrementally extended.

The SQL statement `ALTER DATABANK ADD... PAGES` can be used to increase the size of a databank file by a specified number of pages (refer to the *MIMER/SQL Reference Manual* for details).

MIMER databank files are organized internally into 2 kilobyte databank blocks. Accessing an internal databank block which is physically split over two distinct areas of allocated disk will require two disk read operations. To avoid the risk of fragmenting the internal databank blocks, ensure that the number of disk blocks allocated for databank file extensions maps onto a whole number of 2 kilobyte databank blocks. This will optimize disk I/O efficiency.

VMS

Disk blocks under VMS are 512 bytes in size, therefore a disk cluster size which is a multiple of 4 will avoid fragmenting the 2 kilobyte databank blocks. The cluster size is set when formatting a disk. Use the `SHOW DEVICE/FULL` command to check the cluster size of a disk that is already formatted.

Win

On Windows machines, disk clustering effects are hardware dependent and are not configurable. Disks are typically configured in terms of an even number of 512 byte or 1024 byte disk blocks and will therefore always work efficiently with MIMER databank files.

2.3.4.2 Protecting data against loss

For data security reasons, in case of a disk failure, it is strongly recommended that LOGDB is located on a disk unit that is physically separate from that on which the other databanks are located. See Section 6.1 for more information.

Ideally, TRANSDB and LOGDB should always be located on different physical disks which are served by separate disk controllers and no other databank files should be located on either disk.

The ordinary maintenance procedures for any computer system must involve backup and restore. A strategy, structure and procedure must be set up to include the MIMER databases in the system backup routines. See Chapter 6 for a detailed discussion on backup and restore.

Note: A system without a complete and valid backup and restore procedure runs the risk of losing valuable data.

2.3.4.3 Balanced I/O

If several physical disk units are available, the various databanks should be distributed across the available disk units in order to balance the system I/O load.

To optimize the distribution of I/O across disks, place databanks on physical disks in such a way that databanks which are likely to be accessed at the same time are on different disk units. It is generally the case that TRANSDB will be accessed at the same time as other databanks during a transaction.

2.3.4.4 Disk speed

The placement of databanks on physical disk units will depend on exactly how they will be used when the database system is in operation. Factors such as the speed of disks and the amount of virtual memory paging will often have a greater impact on overall system performance than the disk I/O factors relating specifically to physical layout of the MIMER database.

For example, to enhance performance, frequently accessed databanks such as TRANSDB may be placed on separate, high speed, disks or on RAM disks if sufficient memory is available.

Note: A RAM disk without battery backup will lose its contents in the event of a power failure. For TRANSDB, this will almost certainly lead to an inconsistent database. Do not use a RAM disk if the survival of the data it contains cannot be guaranteed in the event of a power or system failure.

2.3.4.5 Reserved directories

The structure of the databank file system and procedures such as backup and restore are generally simplified if databank files are placed in directories reserved solely for that purpose. The system administrator should create and maintain a directory structure that best suits the local system.

2.3.5 Altering databank locations

User databanks may be relocated by moving the physical file using operating system commands and then changing the file location stored in the data dictionary by using the ALTER DATABANK statement to specify the new file specification (see the *MIMER/SQL Reference Manual* for the statement syntax). The ALTER DATABANK statement may only be issued by the owner of the databank.

Facilities for changing the file specifications stored in the data dictionary for the system databanks, other than SYSDB, are provided by the UTIL program (see Section 6.3.4).

SYSDB must always be located in the home directory for the database.

Note: If all databank file specifications are created without explicit directory specifications, there is no need to alter the information stored in the data dictionary when databank file locations are altered (see Section 2.3.3).

The location of a databank should not be altered while the database server is accessing it or while it is being accessed in single-user mode.

Note: Databanks cannot be moved between databases by copying the databank file and using the facilities to alter the databank location recorded in the data dictionary. The EXPORT/IMPORT utility must be used for this.

2.3.6 Accessing databank files

The databank files in a MIMER database are accessed by the database server regardless of the user running the applications. The operating system privileges that apply to accessing the databank files are associated with the database server.

If a MIMER database is accessed in single-user mode (see Appendix A) the user must have the appropriate operating system level privileges in order to access the databank files.

Ownership of the databank files should not be confused with the creator of the databanks, which is internal to the MIMER data dictionary. It is quite possible that a user who has created databanks is denied direct access at the operating system level to the files for those databanks.

2.3.7 Databank file deletion

If a databank or shadow is dropped, the corresponding file will also be deleted from disk. Please note that dropping a MIMER ident will also drop all objects, including databanks, that the ident has created.

When a databank is dropped, all shadows of the databank will also be dropped.

Note: If the databank is OFFLINE when it is dropped, the databank file (and any shadow files) will remain on disk in the file system.

2.4 Transaction control

Transaction control provides a means of protecting the database from corruption which might arise from two users attempting to change the same information at the same time and also provides the basis for ensuring database consistency (see Section 6.1.1).

MIMER transaction management uses Optimistic Concurrency Control, which is described in the *MIMER/SQL Programmer's Manual*. This type of concurrency control overcomes many of the problems that can occur with conventional locking techniques (e.g. deadlocks and locks being retained by defunct connections). Superior performance is achieved because there is no need for the overhead of a deadlock detection mechanism, since deadlocks cannot occur.

A transaction is an “atomic” operation which means that all the changes that form the transaction are applied to the database, or none of them are applied. Three transaction phases exist: *build-up*, during which the database operations are requested, *prepare*, during which the transaction is validated, and *commitment*, during which the operations performed in the transaction are written to disk.

The transaction begins by taking a snapshot of the database in a consistent state. During build-up, changes requested to the contents of the database are kept in a *write-set* and are not visible to other users of the system. This allows the database to remain fully accessible to all users. The application program in which build-up occurs may see the database as though the changes had already been applied. Changes requested during transaction build-up become visible to other users when the transaction is successfully committed.

During build-up, a *read-set* records the state of the database as seen at the time of each operation (including intended changes). If the state of the database at commitment is inconsistent with the read-set, a conflict is reported and the transaction is rolled back (i.e. the write-set is erased and no changes are made to the database). This can happen if, for instance, a transaction asks to update a row which is deleted by another user after build-up has started but before the transaction is committed. The application program is responsible for taking appropriate action if a transaction conflict occurs.

Transaction control in application programs and a number of the system facilities (notably Backup and Restore - see Chapter 6) work at the databank level and is controlled by setting the option (LOG, TRANS or NULL) for the databank.

Only operations performed in databanks set up with the LOG option are logged in the MIMER system databank LOGDB. Write operations against tables in LOG and TRANS databanks must be performed under transaction control (i.e. within a transaction).

Refer to Chapter 6 of the *MIMER/SQL Programmer's Manual* for further details on transaction handling and database security.

2.5 Database security

MIMER supports a sophisticated system of access rights and privileges, which permit detailed control of database security. The main components of the database security system are:

- idents
- access rights and privileges
- restriction views

2.5.1 The role of idents in database security

Access to the MIMER system as a whole is managed through the use of idents and privileges. Careful advance planning of the hierarchical structure of idents in the database is vital to the long-term viability of the system. A poorly planned ident structure can easily become impossible to follow and control after a relatively short period of system use.

The MIMER installation process creates one user ident, for use in database administration, with the name SYSADM. The SYSADM ident has the system privileges to create new databanks and idents, with the ability to allocate these privileges to other idents (i.e. the privileges are held with the WITH GRANT OPTION). The SYSADM ident also has SELECT access on all tables in the data dictionary, again, with the WITH GRANT OPTION. The SYSADM user is ultimately responsible for the structure of the whole system.

Certain system functions can only be run by SYSADM. Note however that in other respects SYSADM is just an ordinary user ident in the system. It is quite possible (and may be advisable, especially in large systems) that SYSADM does not have access to the actual contents of the database; the database administration role should be concerned with objects in the system, not the actual data.

The initial installation of MIMER also includes a group ident called PUBLIC. All idents created in the system are automatically granted membership in the PUBLIC group, which is intended to be used for granting global privileges.

The following general recommendations can be made for structuring the idents in a system:

- Create PROGRAM idents for functional roles within the system. These are not coupled to any physical individual or group of individuals and thus have a lifetime independent of the turnover of personnel. (Database administration is an example of a functional role, but it is represented by a user ident rather than a program ident for practical purposes - see Section 2.2 for details on idents).
- Create USER or OS_USER idents for physical users of the system. These may be dropped when the person concerned should no longer have access to the database. Do not grant privileges directly to user idents, other than membership to groups. Administration is much simpler if privileges are granted through groups.
- Use GROUP idents to represent logical classes of users in the system. Grant privileges to groups rather than to individuals. This makes the granting of access rights to the system easier to organize and a clearer overview of the privilege structure within the system is maintained. It also means that new idents can be granted suitable privileges efficiently through membership in one or more groups.
- Grant the privilege to create objects (DATABANK, IDENT and TABLE privileges) to program idents only. In this way, individual user idents may be dropped with no recursive effects (see Section 2.5.3). (Creation of domains requires no special privilege and may thus be performed by any ident. Creation of views requires only SELECT access to the table on which the view is based).
- Use the WITH GRANT OPTION sparingly and try to minimize the number of levels in the ident hierarchy. This reduces the risk of recursive revocation of privileges (see Section 2.5.3).

If these recommendations are followed, the maintenance of the ident structure in the system will be much more straightforward. Access to the contents of the database will be granted to relatively few group idents instead of many individual program or user idents. When a physical individual should no longer have access to the database, the corresponding user ident can be dropped with no recursive consequences.

2.5.2 Access rights and privileges

Each ident is given privileges within the system which determine the operations the ident is permitted to perform. Privileges may be granted either directly or by making the ident a member of a group ident. The privileges are classified as follows:

System privileges give the right to create global objects in the database. There are the following system privileges:

DATABANK gives the right to create databanks
 IDENT gives the right to create idents.

System privileges are granted to SYSADM at installation time and may be passed on to other idents with or without the WITH GRANT OPTION. An ident receiving a privilege with the WITH GRANT OPTION may pass the privilege on to another ident.

Object privileges give rights associated with certain specified objects in the system. There are the following object privileges:

TABLE gives the right to create tables in a given databank
 EXECUTE gives the right to enter a given program ident or to call a specified stored procedure
 MEMBER makes an ident a member in the specified group ident
 USAGE gives the right to specify a given domain where a data type would normally be used in contexts where the use of a domain is allowed.

Object privileges are initially, automatically, granted only to the creator of the object (e.g. the creator of a databank automatically has TABLE privilege on the databank). The privileges may be passed on to other idents with or without the WITH GRANT OPTION.

Access privileges give rights of access to the contents of a specified table. There are the following access privileges:

SELECT gives the right to read the table contents
 INSERT gives the right to add new rows to the table
 DELETE gives the right to remove rows from the table
 UPDATE gives the right to change the contents of existing rows in the table (this privilege may be limited to specified columns within the table)
 REFERENCES gives the right to use the primary or alternative key of the table as a foreign key from another table (this privilege may be limited to specified columns within the table). This facility applies only to tables created with SQL.

In addition to the five specific privileges listed above, keyword ALL may be used as a shorthand method of specifying all the privileges possessed by the granting ident. For example, if an ident has only SELECT and UPDATE privileges on a table and ALL is granted on that table to a new ident, the new ident will only be given SELECT and UPDATE.

CASE 2

1. A grants with grant option to M
2. M grants to X
3. B grants with grant option to M
4. M grants to Y
5. B revokes from M X keeps privilege (authorization A)
Y keeps privilege (authorization A)

CASE 3

1. A grants with grant option to M
2. B grants with grant option to M
3. M grants to X
4. M grants to Y
5. A revokes from M X keeps privilege (authorization B)
Y keeps privilege (authorization B)

CASE 4

1. A grants with grant option to M
2. M grants to X
3. B grants **without** grant option to M
4. M grants to Y
5. A revokes from M X loses privilege (authorization A)
Y loses privilege (authorization A)

2.5.4 Restriction views

Views are a powerful tool for restricting user access to specific parts of the database and they complement the use of access privileges in maintaining database security. By defining restriction views (i.e. views based on one table but restricted only to specific rows and/or columns in the table), access may be provided to a subset of the contents of a table without affecting the physical database structure. In this way, the database may be designed optimally according to the relational model, while user access can be defined according to actual data retrieval requirements.

2.6 Data integrity

The following facilities are available for ensuring the integrity of a MIMER database:

- domains
- entity integrity (non-NULL primary keys)
- referential integrity (foreign keys)
- table integrity
- view integrity

2.6.1 Domains

Domains define sets of permissible values. By assigning a table column to a domain when the table is created, the values which the column may contain are restricted to those defined in the domain. Any number of columns may belong to any given domain.

A default value may also be defined for a domain, which is inserted into the column belonging to the domain if no explicit value is given. If the default value for the domain is defined outside the range of restriction values for the column, attempts to insert the default will fail. In such a case an explicit value must always be specified when inserting data into the column.

The use of domains in table definitions is recommended, since this can provide an automatic check on the validity of data inserted into the column. However, domain definitions should be carefully planned, since a domain definition cannot be altered after it has been defined.

2.6.2 Entity integrity

Entity integrity refers to the requirement that every row in a table must be uniquely identified and that no row in a table may be identified by NULL (i.e. by an unknown value). According to E. F. Codd, entity integrity should be enforced in all true relational database systems.

All primary key columns in tables created by MIMER/SQL are defined as NOT NULL, thus ensuring entity integrity. Other (i.e. non-primary key) columns may also be defined as NOT NULL as required.

2.6.3 Referential integrity

Referential integrity refers to the requirement that data entered into a table in the database must already be present in another table (e.g. a component may not be entered in a parts list if it does not already exist in the set of known components in the database). MIMER/SQL supports referential integrity through the optional FOREIGN KEY clause in the CREATE TABLE statement. The properties of a FOREIGN KEY are as follows:

- Data inserted into the foreign key columns (by either INSERT or UPDATE operations) must either already be present in the primary key or an alternate key of the reference table or include at least one NULL column.
- The columns defined as a foreign key must correspond exactly in number and definition to the primary key or alternate key columns in the reference table.
- Rows may not be deleted from the reference table and the alternate key columns in the reference table may not be updated if the values of the primary key columns are present in any foreign key.

The use of foreign keys should be carefully planned, since referential integrity constraints may not be added to or removed from existing tables. If referential integrity is to be added to an existing database structure, the tables containing the foreign keys must be dropped and redefined.

Note: Tables with foreign key definitions may not be stored in databanks defined with the NULL option.

2.6.4 Table integrity

Table integrity refers to the facility in MIMER/SQL of defining CHECK clauses in a table definition, whereby the contents of one column is checked against the contents of one or more other columns in the same row of the table. Data may only be entered into the table if the CHECK constraint is not violated.

CHECK constraints may not be added to or removed from existing tables.

2.6.5 View integrity

View integrity refers to the facility in MIMER/SQL of including a WITH CHECK OPTION clause in a view definition. If a view is defined with a WITH CHECK OPTION, data which violates the definition of the view may not be entered into the view by INSERT or UPDATE operations.

When a view is defined with a CHECK OPTION, any views defined on that view will inherit the CHECK OPTION.

3 ESTABLISHING A MIMER DATABASE

3.1 General information

Procedures for installing the MIMER software differ between the different machines and are described in the *MIMER Guides* for the platform you are using.

Once the MIMER software is installed, the database environment must be established. This involves the following activities:

- establish the identity of the database on each node in the network from which it is to be accessed. A database must be set up as a local database on the node it resides on. The local definition for a database involves specifying parameters that control the database server that is started for it. A database must also be identified as a remote database on each other node in the network from which it is to be accessed. The remote definition for a database involves specifying network communication parameters.
- generate the MIMER system databanks SYSDB, TRANSDB, LOGDB and SQLDB as well as the database administration ident called SYSADM.
- establish the ident and data structure in the database using the data definition statements in MIMER/SQL.

3.2 Making the database accessible

In order to make a database accessible, via a database server or in single-user mode, its identity must be defined so that connections can be made to it.

In a network environment, the identity of a database must be established on each node from which it is to be accessed. A database is set up as a local database on the node where it resides and it is defined as a remote database on each other node in the network from which access to it is required.

Unix

VMS

On Unix and VMS nodes, the identity of a database is established by creating an entry for it in the **SQLHOSTS** file - see Appendix B for details about this file. All users must have read access to the **SQLHOSTS** file on the machine they are using in order to run applications and utilities accessing MIMER databases.

Win

On a Windows node, the identity of a database is established by running the MIMER Administrator. The MIMER Administrator adds information about MIMER databases to the Windows registry file. Refer to the online Windows help provided with the MIMER Administrator for details on how to use it.

3.3 Local databases

A local database is one that resides on the local machine (i.e. the system databank file containing the data dictionary exists on a local disk).

A local database definition establishes the database identity by specifying a name (which is not case sensitive) and a home directory for the database. It also involves specifying various parameters which control the database server that is started for the database.

Unix

The definition of a local database under Unix and VMS involves specifying the *database name* and the *database home directory* in the **SQLHOSTS** file (see Appendix B). Parameters that control the database server are specified in the **MULTIDEFS** file which is located in the database home directory - see Appendix C for details about this file and the parameters it contains. The **MULTIDEFS** file is automatically created with appropriate default values for all parameters when the database server is first started.

VMS**Win**

The definition of a local database under Windows is established by running the MIMER Administrator and specifying the required parameters, including those that control the database server, which are stored in the Windows registry file. On-line Windows help is provided with the MIMER Administrator to guide you through the creation of a local database. Default values are supplied for most of the parameters.

A fully established local database, complete with its MIMER system databanks (see Section 3.5), user databanks and the ident and data structure contained in them (see Section 3.6) constitutes a physical MIMER database.

A fully established local database can be accessed from the local machine. If the database is to be accessed from a remote node, connected to the local machine via a network connection, a remote database definition for the database must be created on the remote node.

3.4 Remote databases

A remote database is one that resides on a network node that is not the local machine (i.e. the database must be accessed from the local machine via a network connection).

In order to access a database that resides on another network node, the database must be established as a local database on the node it resides on and a remote database definition must be established on the node from which the database is to be accessed.

The purpose of the remote database definition is to set up a link with a database that resides elsewhere on the network. The name used for the remote database definition must be the same as that given to the local database it represents. The definition for the remote database contains the communication parameters required for accessing the database over the network.

Unix

The definition of a remote database under Unix and VMS involves creating an entry in the **SQLHOSTS** file - see Appendix B for information on the parameters involved.

VMS

Win

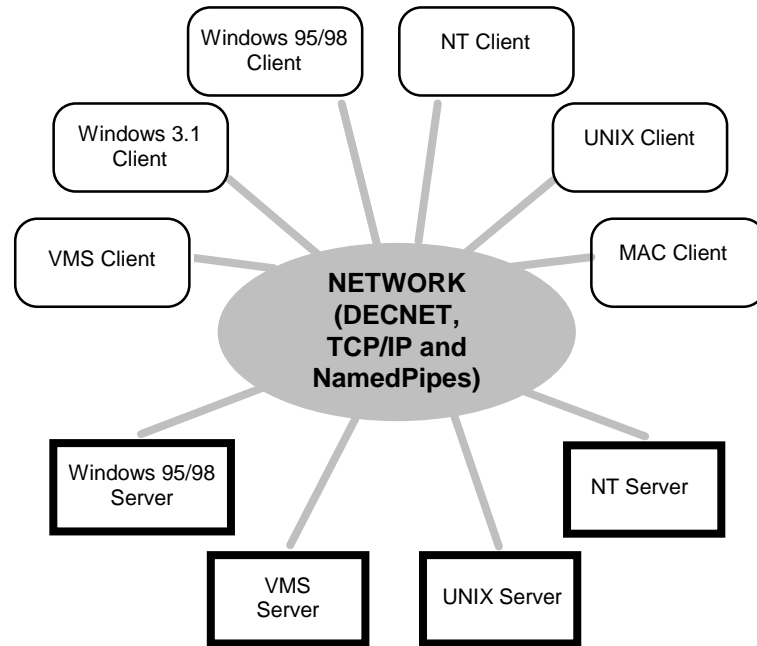
The definition of a remote database under Windows is established by running the MIMER Administrator and specifying the required parameters. On-line Windows help is provided with the MIMER Administrator to guide you through the creation of a remote database.

3.4.1 Client-server interface

Once the remote database definition has been created and provided that the MIMER client/server communications have been established correctly, access to a database that resides on a remote machine is performed transparently.

The MIMER client/server communications interface is integrated into the database server. The database server process manages all connections. All MIMER applications may use the client/server interface without having to make any special provision in the application code. The client/server interface is automatically activated whenever a remote database is targeted.

The MIMER client/server protocol is identical on all MIMER platforms. This means that a MIMER client on any machine type may access a MIMER server for a remote database on any of the platforms on which MIMER is implemented.



MIMER client/server communications

3.5 Generating the MIMER system databanks

The MIMER system databanks SYSDB, TRANSDB, LOGDB and SQLDB are generated by the SDBGEN program.

The SYSDB databank is placed in the current working directory when SDBGEN is run and it also allows the location of each of the other MIMER system databanks to be specified. The current working directory should be set to the database home directory before running SDBGEN to ensure that SYSDB is correctly located.

Win

The SDBGEN program under Windows takes a database name as an argument and automatically sets the working directory to be the database home directory, thus ensuring that the MIMER system databanks are always correctly located.

The initial size for each of the MIMER system databanks can be specified. The size for the databanks is specified in MIMER pages. The size of a MIMER page is 2 kilobytes.

The database administration ident SYSADM is also created and a password (passwords are case-sensitive) must be specified for this ident. For security reasons, the password specified for SYSADM is not echoed on the screen when it is entered. It should be chosen carefully and changed at appropriate intervals using MIMER/SQL with the ALTER IDENT statement.

Note: Care should be taken to safeguard the SYSADM password, because if it is lost it cannot be retrieved from the system and it is not possible to set a new one.

SDBGEN will load the system tables and defines the ODBC views, MIMER system views and the INFORMATION_SCHEMA views used by the MIMER system.

Unix

A local database is set up on a Unix node by running dbinstall (refer to the *MIMER Guides* for Unix for details about dbinstall) and SDBGEN is started automatically when required.

VMS

In order to create the MIMER system databanks for a local database on a VMS node, SDBGEN should be invoked in the directory where the SYSDB databank is to be placed, i.e. in the database home directory, as specified by manually creating an entry for the database in the SQLHOSTS file. An example of how to run SDBGEN on VMS is shown below:

```
$ SET DEFAULT disk:[database_home_directory]
$ RUN MIMEXE8:SDBGEN
```

Win

A local database is set up on a Windows node by running the MIMER Administrator and SDBGEN is started automatically when required. On-line Windows help is provided with the MIMER Administrator and SDBGEN to guide you through the creation of a local database.

Note: SDBGEN is intended for initial databank creation only. It cannot be used to upgrade existing MIMER installations, since SDBGEN will not run if any of the MIMER system databanks already exist. If it is to be used to re-generate the MIMER system databanks for any reason, the old ones must be removed first. If the MIMER system databanks are removed, all the data in the database will be effectively lost because the data dictionary will no longer exist. Without the data dictionary, the location of the user databanks will no longer be known and the data contained in each will not be understood by the system. Note that a databank created for one SYSDB cannot be accessed by using a different SYSDB even if identical data dictionary definitions are created in it.

3.6 Establishing the ident and data structure

Once the local database environment has been created for a MIMER database (database identity, server parameters, system databanks, the SYSADM ident plus the system tables and views), the data structure for the database (idents, user databanks, tables, and so on) can be created using MIMER/SQL data definition statements.

BSQL, used interactively or in batch mode, allows the execution of a sequential file which can then be used as a permanent record of the CREATE statements used to establish the database (see the *MIMER/SQL User's Manual*).

BSQL also supports the saving of input and/or output to a log file (using the LOG command), so this facility could be used to create a permanent record of an interactive BSQL session.

An application program using embedded SQL or ODBC can also be used, but this requires more work on the part of the programmer and it provides a less concise record of the ident and data structure in the database.

Third party SQL tools are also available which may be used to create the database data structure.

Note: A sequential file intended for non-interactive execution through the BSQL facility may include username and password information relating to any CONNECT statements used. For security reasons, the batch file should be well protected in the operating system, preferably with any username and password edited out of any permanent copy of the file.

3.7 Managing database connections

This section describes how users connect to a database and how several simultaneous connections from an application can be handled.

The following SQL statements are used for connection management (see the *MIMER/SQL Reference Manual* for details):

```
CONNECT
DISCONNECT
SET CONNECTION
```

3.7.1 Selecting a database

Applications establish database connections with the CONNECT statement, which specifies the database by name. An application may connect to any of the databases which have been made accessible from the node where the application is running (see Section 3.2). Some applications which are part of the MIMER distribution allow the database name to be specified as a command-line argument.

An application may connect to several databases simultaneously. By using the SQL statement SET CONNECTION the application may switch between active connections. However, a transaction may use only one connection.

The database may be located on the same machine as the application program (a local database), or on a remote machine accessed over a network (a remote database). The network connection is handled by the MIMER/SQL software and this is completely transparent to the application program (see Section 3.4.1).

A database is normally accessed by one or more users via the database server. It is also possible for one user to access a local database directly in single-user mode, provided the database server for it is not running and the operating system user has the appropriate access rights to the database files (see Appendix A).

3.7.2 The default database

In MIMER/SQL, if the CONNECT TO DEFAULT statement is used, or if the database name in the CONNECT statement is specified as an empty string, a connection will be established to the default database .

The default database can be any of the local or remote databases that are accessible from the node the application program is running on.

The database that is actually selected by a default connection depends on whether a node-specific or user-specific default database is defined at the time the connection is attempted. If a user-specific default database has been defined, it will be chosen in preference to a node-specific default database.

3.7.2.1 Defining a node-specific default database

One default database can be defined for each node in a network.

Unix

The default database for Unix and VMS nodes is defined by specifying the name of the database in the DEFAULT section of the **SQLHOSTS** file (see Appendix B).

VMS

Win

The default database for a Windows node is defined by using the MIMER Administrator to create a System Wide MIMER ODBC Data Source with the name "default" and associating it with the selected database. Refer to the online Windows help provided with the MIMER Administrator for details on how to create System Wide MIMER ODBC Data Sources.

3.7.2.2 Defining a user-specific default database

There may be times when an individual user may wish to override the default database defined for the local machine. This is done by defining a user-specific default database.

Unix

A user-specific default database is defined under Unix and VMS by setting the environment variable or logical name called **MIMER_DATABASE** to be the name of the required local or remote database, as stated in the **SQLHOSTS** file.

VMS

If the **MIMER_DATABASE** variable is set, all default connections will be made to the database it identifies. If the **MIMER_DATABASE** variable is not set, default connections will be made to the node-specific default database for the local machine.

Win

A user-specific default database is defined under Windows by using the MIMER Administrator to create a User Specific MIMER ODBC Data Source with the name "default" and associating this with a database selected by the user. Refer to the online Windows help provided with the MIMER Administrator for details on how to create User Specific MIMER ODBC Data Sources.

When a User Specific MIMER ODBC Data Source exists with the same name as a System Wide MIMER ODBC Data Source, the user-specific one takes precedence.

Programs supplied as part of the MIMER distribution (e.g. BSQL etc.) will look at the MIMER_DATABASE environment variable and if this has been set it will take precedence over any MIMER ODBC Data Sources called "default".

3.7.3 Connection names

Whenever an application connects to a database a connection name is associated with the connection. This name may be specified explicitly in the CONNECT statement. Otherwise the connection name is the same as the database name. Several connections can be made to the same database by specifying different connection names. Connection names are not case-sensitive.

The connection name is used in the SET CONNECTION statement for switching between different connections and in the DISCONNECT statement for closing connections.

When the application switches from one connection to another the previous connection becomes dormant. A dormant connection cannot be accessed until the connection is made current with SET CONNECTION statement. A dormant connection may always be closed down with a DISCONNECT statement without making it current first.

When a DISCONNECT CURRENT has been performed the application has no current connection. The only valid SQL statements in this situation are CONNECT (to establish a new connection), SET CONNECTION (to make a dormant connection active) or DISCONNECT (to close another, dormant, connection).

3.7.4 Troubleshooting remote database connect failures

If an attempt to connect to a remote database fails, the client/server connection can be tested by starting BSQL on the client node and attempting to connect to the database on the server node.

In the event of a connect failure, the following should be checked:

- If the connect was attempting to access the default database, check that a user-specific or node-specific default database is correctly defined on the client node (see Section 3.7.2 for details on how this is done).
- Check that the database been correctly set up as a local database on the server node (see Section 3.3) and as a remote database on the client node (see Section 3.4) and that the name of the remote database is the same as that of the local database.
- Check that the operating system user who is trying to establish the connection can access all required files etc. on the client node:

Unix

Check that the operating system user has read access to the SQLHOSTS file on the client machine.

VMS

- Check that the operating system user who is trying to establish the connection has all the required operating system privileges:

VMS

Check that an operating system user who is trying to use DECNET has TMPMBX and NETMBX privileges enabled.

- If the tcp/ip protocol is being used, check that the server node is reachable from the client node over the network by using the “ping” command:

```
ping server_node
```

- If the tcp/ip protocol is being used, try to telnet to the tcp/ip port. You should get a connection and when <CR> is entered, the connection should be closed by the server.

Win

- If using NamedPipes, the operating system user must have an account set up on both the local machine and on the machine where the remote database resides. Both accounts must be set up with the same password.
- If using NamedPipes to connect a MIMER version 7.3 client to a MIMER version 8 database server, it will be necessary to take certain steps to enable network communication. Under version 7.3 the expected Service name was “MIMER”, but in version 8 the expected Service name is the name of the database. Therefore, one of the following must be performed before a version 7.3 client can communicate with a version 8 remote database server: 1) On each version 7.3 client node, the Service parameter in the remote database definition must be changed to be the name of the database instead of the name “MIMER”, or 2) On the version 8.1 server node, start a NamedPipes server which listens to service “MIMER” so that it can redirect communications to the correct named database server.

Win

- If using NamedPipes to connect a MIMER version 8 client to MIMER version 7.3 database server, the Service parameter in the remote database definition on the version 8 client node must be changed to the name “MIMER” instead of the name of the database.

3.8 MRS license key

Before the MIMER system can be used, a license key must be installed. Whenever a user connects to MIMER, the computer identification and the license key file will be checked to determine access rights. If access is denied, the connect attempt will be aborted and an error message will be shown.

The MRS license key contains the following (encrypted) information:

- The maximum number of users that may use all database servers located on the same computer system at any one time. When a database server is started, it is possible to specify the maximum number of users that can connect to the server simultaneously. The sum of these limits for all database servers on the node may not exceed the number given in the MRS key.
- A list of modules that the site is permitted to use.
- Version number permitted for a specific module. If the MRS key allows execution of a module of a specific version, access to all earlier versions of that module is granted automatically.
- Expiration date for a specific module.

An MRS license key may be valid for a number of releases with the same major version number (your MIMER distributor will advise when a new MRS license is required).

Unix**VMS**

To create the license key file, use any text editor to enter the key data into the MIMKEY8 file. It should be entered exactly as provided, apart from that no line may contain more than 80 characters and that no space characters are allowed.

Win

The MIMER Administrator is used to enter the MRS license key. Refer to the online Windows help provided with the MIMER Administrator for details on how to enter the MRS license key.

The MRS key data is provided by your MIMER distributor and in order to generate the key data, the node name for the computer on which the database server runs must be known.

Unix

The host name of a Unix machine is obtained by the following Unix command:

```
# uname -n
```

VMS

One of the following methods can be used:

The host name is printed by the MIMSETUP8 command procedure:

```
$ @disk: [MIMERxxxxx]MIMSETUP8
```

If the VMS node is part of a cluster, the scsnod parameter describes its name:

```
$ WRITE SYS$OUTPUT F$GETSYI("SCSNODE")
```

If the VMS system is not clustered, the node name parameter may be blank. In this case, you should check the SYS\$NODE logical name instead:

```
$ SHOW LOG SYS$NODE
```

Win

When the dialog which is used to enter an MRS license key is opened in the MIMER Administrator, the node name of the computer will be displayed. Refer to the online Windows help provided with the MIMER Administrator for details on how to open the dialog.

When the number of MIMER users is increased or new MIMER modules are added to the site, a new MRS license key will be provided as a complete replacement to the existing key.

The MRS key uses the node name of the computer to link MIMER to the computer it is authorized to run on. This allows for hardware replacement in event of a failure in the computer system. If a replacement computer is given the same node name as the one it is replacing, the MRS key remains valid for the new hardware.

3.9 Running BSQL and UTIL

The BSQL and UTIL programs, which form part of the MIMER product, support a number of command-line arguments.

The overall syntax for BSQL and UTIL is as follows:

```
bsql  [-m | -s] [database]
util  [-m | -s] [database]
```

BSQL and UTIL command-line arguments

Parameter	Function
-m	Connects to the database in multi-user mode.
-s	Connects to the database in single-user mode.
database	Specifies the name of the database to access.

If the optional database name is not specified, the default database will be accessed (see Section 3.7.2).

If neither `-s` or `-m` is specified for the optional mode flag, the way the database is accessed will be determined by the setting of the `MIMER_MODE` variable (see Section A.2) or, if this is not set, it will be accessed in multi-user mode.

Refer to the *MIMER Guides* for your platform for any additional information that may be required in order to enable the use of these command-line arguments.

3.10 Creating the example database

BSQL input files are included in the MIMER distribution which will create and then manipulate data in an example database. The `CREHOTDB` file creates the data structure for the example database and the `EXAMPLES` file contains various operations to manipulate data in the example database.

Before the data structure in the example database can be constructed, the database environment must be created according to the instructions given earlier in this chapter.

The name of the example database is assumed to be “HOTEL” in the examples shown below:

```
$ bsql HOTEL
```

Sign on to BSQL as SYSADM and do the following:

Unix

```
SQL> READ 'crehotdb.dat';
```

VMS

```
SQL> READ 'MIMEXAMPLES8:CREHOTDB.DAT';
```

Win

```
SQL> READ 'crehotdb';
```

```
SQL> EXIT;
```

The EXAMPLES file contains various SQL and PSM statements which may be examined and executed interactively in order to follow the operations being performed. Most of the operations are presented as examples in the accompanying *MIMER/SQL User's Manual* so they may be studied interactively in conjunction with reading the manual.

3.11 Removing a database

To remove a database, perform the following steps:

- Check that no one is using the database, and that no database server is started against the database you are going to remove.
- Create a list of all databank files by doing the following:

```
$ bsql -s database
Username: SYSADM
Password: xxxxxx
SQL> SELECT * FROM MIMER.DBFILE;
SQL> EXIT;
```

- Using the list generated in the previous step, locate and delete all databank files. If no directory is specified, the directory will be the home directory of the database (see Section 3.3).
- Delete any directories that have been specifically created to hold databank files.
- Delete all files in the database home directory (see Section 3.3) and the directory itself.
- Remove the database from the list of accessible databases (see Section 3.2). This will involve deleting it from the list of local databases on the node where it resides and deleting the remote database definition created for it on each other network node from which it was made accessible.

Win

Under Windows NT, when the local definition of a database is removed, the associated service is also automatically removed, so the next step is not required.

- If any database-specific commands have been added to a system startup or shutdown file, these should be removed.

VMS

Database-specific commands are often added to the VMS system startup file (SYS\$MANAGER:SYSTARTUP_VMS.COM) and the VMS shutdown file (SYS\$MANAGER:SYSHUTDOWN.COM), so these may have to be edited.

4 MANAGING A DATABASE SERVER

The database server allows one or more users to access a database. Each database may have one database server running against it and the server runs on the machine where the database resides (i.e. the database servers running on a machine are for the local databases on that machine).

The parameters which control a database server and which can be tuned to optimize performance are specified as part of the definition of a local database (see Section 3.3).

The MIMCONTROL functionality provides facilities for managing the operation of a database server (e.g. starting, controlled shutdown, etc.).

The MIMINFO functionality provides facilities for getting system management information from a MIMER database server, such as:

- listing details for the users on the system
- monitoring performance parameters
- dumping data for trouble-shooting analysis by MIMER Support personnel.

Operational and error messages generated by a database server are recorded in the database server log (see Section 4.4).

4.1 System performance

In a MIMER installation there are a number of system-wide parameters that have a major impact on the overall system performance. The most important parameters are the bufferpool size and the number of request and background threads.

4.1.1 Database server memory areas

The database server memory requirements involve the following components:

4.1.1.1 Code

The server code requires about 4 Mb.

4.1.1.2 Data and thread stacks

As a rough guideline, assume about 500 Kb data plus 400 Kb for each thread started (the total number of threads started is the number of background threads plus the number of request threads), the actual figures, however, depend on the operating system being used.

4.1.1.3 Bufferpool

The bufferpool is the main primary memory cache used by the basic data access routines in the MIMER database management and contains data pages from the databank files. It is a local memory area in the database server process.

Unix

VMS

Under Unix and VMS, the bufferpool is locked in physical memory by default.

The bufferpool does not grow dynamically, so whenever the bufferpool is full and access to a new page is required, space is released in the bufferpool by swapping out the least-recently-used resident page.

Frequent page replacement operations detract from the overall system performance since access to disk is relatively slow. The best MIMER performance is thus obtained by having **as large a bufferpool as possible** without exceeding the amount of main memory available. In practice, it is always necessary to find a suitable compromise between allocation of memory to the MIMER bufferpool and keeping memory available for user applications and operating system tasks.

The size of the bufferpool depends on the parameters *Pages2K*, *Pages16K* and *Pages64K* which are specified as part of the local database definition (see Section 3.3).

The amount of memory used by the database buffers can be calculated by:

$$\text{buffer space in kilobytes} = \text{Pages2K} * 2 + \text{Pages16K} * 16 + \text{Pages64K} * 64$$

Note: The bufferpool contains a variety of other data, thus the total bufferpool size will be at least 10% greater than the space needed for the database buffers.

The default initial bufferpool size for a database server is based on the memory available on the machine.

Fine tuning of the bufferpool is performed manually by adjusting the parameters in the local database definition (see Section 3.3) after the MIMER system is fully installed and has been functional for a period of time. The fine tuning should be repeated whenever there is a significant change in the computer workload distribution.

Since the MIMER bufferpool size affects the performance of both MIMER and other applications (because it reserves memory for a MIMER database server), it is advisable to perform regular routine checks on the bufferpool statistics in an operational system by generating a MIMSERV report (see Section 4.3.3).

Win

The Windows NT performance monitor can also be used to monitor a database server running on any platform.

Some general guidelines for bufferpool tuning are:

- Whenever main memory is available, it should be allocated, if possible, to the bufferpool.
- Ensure that the bufferpool is not subject to system paging or swapping, since the paging algorithms used by MIMER and the operating system usually differ, and forced cooperation between the two will often detract considerably from MIMER's performance.
- If more than about 2% of all MIMER page requests result in a page fault, the bufferpool is too small. Statistics for page requests and faults are presented in the MIMSERV report (see Section 4.3.3).

It is important to take note of the page fault statistics **for each region** in the bufferpool to ensure that the most appropriate allocation has been made in each. The MIMER system decides which page size is most appropriate for each task to be performed. For example, 16K pages are currently used for transaction data (this may change in the future) and therefore allocating too few 16K pages may currently adversely affect performance even though generous allocations have been made in the other bufferpool regions.

4.1.1.4 Communication buffers

Each communication buffer is about 70 Kb (it varies slightly depending on platform). There is one communication buffer for each user as defined by the *Users* parameter in the local database definition (see Section 3.3).

Unix**VMS**

Under Unix and VMS, the communication buffers are locked in physical memory by default.

All communication buffers reside in shared memory.

4.1.1.5 SQLPOOL

The SQLPOOL area contains information about opened tables and databanks, compiled SQL programs, etc.

The initial size (in Kb) of the SQLPOOL is determined by the *SQLPool* parameter in the local database definition (see Section 3.3). The SQLPOOL area grows dynamically when the database server needs more space. The local database parameter *MaxSQLPool* controls the maximum size of the SQLPOOL. The default value for *MaxSQLPool* is $2000 * Users$.

The SQLPOOL area is not locked in physical memory. This allows the SQLPOOL to grow dynamically and it may become larger than the physical memory of the server process. The operating system generally manages this situation by page-faulting. The page-faults will not affect bufferpool performance if that area is locked in physical memory.

If the amount of operating system page-faulting observed in a database server becomes excessive, it is an indication that the memory required by the server process is much greater than the amount of physical memory allocated to it. In this case, either more memory must be installed on the machine or the local database parameters controlling memory allocation must be adjusted to reduce the memory required by the database server process.

4.1.2 Number of request threads

The MIMER database server process supports a number of separate request threads and background threads, running simultaneously under the operating system. The amount of concurrency the database server can support is dependent on the number of available request threads. Increasing the number of these threads can improve performance.

The number of request threads does not usually correspond to the number of users in the system.

The number of request threads in a database server is defined at system start-up. A change to the number of request threads requires that the system be stopped and re-started.

The maximum number of concurrent request threads is limited by the size of the bufferpool.

4.1.3 Number of background threads

The background threads in a MIMER database server perform tasks such as:

- Recording transactions in LOGDB.
- Updating master and shadow databanks.

Refer to the details on “Transaction count” and “Pending background thread requests” in the “Background threads” section of the MIMSERV report (see Section 4.3.3) for information relevant to fine tuning the number of background threads.

4.1.4 Database server system requirements

From the point of view of the operating system, a database server requires the following system resources:

Physical memory

The amount of physical memory used by the database server process is determined by parameters in the local database definition (see Section 3.3), whose initial default values are determined by looking at the amount of installed memory.

The bufferpool and the communication buffers are usually locked in physical memory.

VMS

For a database server running on a VMS node the amount of physical memory used by the database server process will vary between the VMS process parameters `WSQUOTA` and `WSEXTENT`. The `WSQUOTA` parameter is calculated by `MIMCONTROL` and is set large enough to include the bufferpool, communication buffers, code, and stack data. The `WSEXTENT` parameter is taken from the parameter with the same name in the `MULTIDEFS` file (see Appendix C). The default value for `WSEXTENT` is the `SYSGEN` parameter `WSMAX` (the maximum amount of physical memory a single process may have).

Virtual memory

The amount of virtual memory that the database server process can use is limited by the operating system.

VMS

The `MIMCONTROL` command sets the page file quota of the database server so that it is large enough to contain all memory areas, including the bufferpool and an `SQLPOOL` that has grown to *MaxSQLPool* kilobytes.

It may be appropriate to create larger page files to increase the amount of virtual memory available to the database server.

Global pages

The database server creates a global section for its communication buffers. This global section resides on the page file. The amount of memory a global section may take from a page file is generally controlled by an operating system parameter. If this limit set by the operating system is exceeded, the `MIMCONTROL/START` command will fail with the message:

```
%SYSTEM-E-EXGBLPAGFIL, exceeded global page file limit
```

If this happens, the VMS `SYSGEN` parameter called `GBLPAGFIL`, which limits the amount of memory that global sections may take from the page files, should be increased.

Win

If you get a message box saying the system is running out of virtual memory you need to increase the size of your paging file.

4.2 Controlling the database server

MIMCONTROL functionality is supplied on all platforms as a complete administration tool for managing database servers.

Unix

The database servers on a Unix node are controlled via the **mimadmin** command - refer to the *MIMER Guides* supplied for Unix for details. This command invokes the MIMCONTROL program and other programs as required. The MIMCONTROL command can also be used directly under Unix, using the root account.

When a database server on a Unix machine is started for the first time, MIMCONTROL will create a default **multidefs** file containing appropriate default parameter values, based on the amount of memory installed on the machine. Refer to Section C.1 for details. Database server performance can be fine-tuned later by adjusting the parameters as required.

VMS

The database servers for the local databases on a VMS node are controlled by using the **MIMCONTROL** command directly (as described in this section).

When a database server on a VMS machine is started for the first time, MIMCONTROL will create a default **MULTIDEFS** file containing appropriate default parameter values, based on the amount of memory installed on the machine. Refer to Section C.1 for details. Database server performance can be fine-tuned later by adjusting the parameters as required.

Win

When a local database is established on a Windows machine using the MIMER Administrator, appropriate default values are supplied for the parameters which control the database server and these can be adjusted later to fine tune server performance. On-line Windows help is provided with the MIMER Administrator.

The MIMCONTROL command can be used in a Command Prompt window on a Windows node to control local database servers.

Windows NT: Database servers accessible from a Windows NT node are controlled by using the MIMER Controller utility. Refer to the online Windows help provided with the MIMER Controller for further details. You must belong to the administrators group to control database servers.

It is also possible to use Windows NT commands NET START, NET STOP, etc. to control database server processes.

Windows 95/98: Database servers for the local databases on a Windows 95/98 node are started automatically. By right-clicking or double-clicking on the database server icon you can control the database server. Refer to the online Windows help provided with the MIMER database server for further details.

4.2.1 MIMCONTROL syntax

The MIMCONTROL program is controlled by flagged information specified on the command-line.

The overall syntax for MIMCONTROL (expressed in Unix-style) is:

```
mimcontrol [-cdekstwA] [-l chan] [database_name]
```

If a database name is not specified, the default database will be controlled.

Note: For MIMCONTROL, the default database is determined by the setting of the MIMER_DATABASE environment variable.

If the MIMCONTROL command is invoked without any options it displays help on the command-line options.

The table that follows lists the command-line flags recognized by MIMCONTROL (flags are shown in both Unix-style and VMS-style).

- Unix

The Unix-style command-line flags must be used on a Unix machine.
- VMS

Either the Unix-style or the VMS-style command-line flags may be used on a VMS machine - see the *MIMER Guides* for VMS for more details.
- Win

The Unix-style command-line flags can be used from a Command Prompt window.

MIMCONTROL command-line flags

Unix-style	VMS-style	Function
-c	/STATUS	Output status information about the specified database server. This option can be combined with the -s option (see Section 4.2.2).
-d	/DISABLE	Disable new user connections to the database server. Users already connected are not affected. This option can be combined with the -s, -t and -w options (see Section 4.2.2).
-e	/ENABLE	Enable new user connections to the database server.
-k	/KILL	Kill the database server immediately. This should only be used in emergency situations when a normal stop using the -t option does not work. The next time the database is started, all databanks that were open at the time the server was killed will be automatically checked. Connected users will receive an error the next time they attempt to access the database.
-l chan	/LOGOUT=chan	Force logout of the specified channel number. Use channel numbers displayed by the USERS option of the MIMINFO command (see Section 4.3.2).

-s [timeout]	/START [timeout]	Start the database server. If the server does not become operational within the specified number of seconds, the server will be killed. Default timeout is 600 seconds. This option can be combined with the -c and -d options (see Section 4.2.2).
-t [timeout]	/STOP [timeout]	Stop a database server. Any remaining users will be logged out. If the server does not stop within the specified number of seconds, the server will be killed. The default timeout is 120 seconds. This option can be combined with the -d and -w options (see Section 4.2.2).
-w [timeout]	/WAIT [timeout]	Wait for all connected users to log out. If there are still users connected after the timeout period expires, the command fails. The timeout period should be given in seconds. If no timeout period is specified wait will be performed without any timeout. This option can be combined with the -d and -t options (see Section 4.2.2).
-A	/DUMP	Create a dump directory and produce dumps of all internal database server areas to files in that directory. The files produced can be examined by using the MIMINFO -f command (see Section 4.3).

4.2.2 MIMCONTROL examples

The parameter options can be combined in the following ways (“HOTEL” is the example database used, equivalent examples are given in both VMS-style and UNIX-style):

- 1) Start a database server, but disallow new logins immediately:

```
MIMCONTROL/START/DISABLE HOTEL
mimcontrol -sd hotel
```

- 2) Start a database server and output a status message for the newly started server.

```
MIMCONTROL/START/STATUS HOTEL
mimcontrol -sc hotel
```

- 3) Disable new user connections, then wait for up to three minutes for all users to log out. The exit code from the MIMCONTROL command is “success” if all users logged out within the three minute timeout period. If the timeout period expires and there are still users logged in on the system, the MIMCONTROL command will exit with a “warning” status code.

```
MIMCONTROL/DISABLE/WAIT=180 HOTEL
mimcontrol -dw 180 hotel
```

- 4) This command will wait for all users to log out of the system. When all users are logged out, the system will be stopped. If the wait timeout period expires, the MIMCONTROL command will exit with a “warning” status code without stopping the system.

```
MIMCONTROL/WAIT/STOP HOTEL
mimcontrol -wt hotel
```

- 5) This command is similar to the previous one, but will ensure that no new users log in to the system while waiting for all users to log out.

```
MIMCONTROL/DISABLE/WAIT/STOP HOTEL
mimcontrol -dwt hotel
```

4.2.3 MIMCONTROL exit codes

The MIMCONTROL command returns a status code to the environment executing the command. The status code can be examined by scripts.

VMS

On VMS, the status codes correspond to the VMS condition code severity levels. Use the \$SEVERITY symbol in DCL command procedures.

Three different codes are used:

<u>Unix/Windows</u>	<u>VMS</u>	<u>Usage</u>
0 (success)	1 (success)	This code is used when the MIMCONTROL command has executed all options with no problems.
1 (warning)	0 (warning)	The warning code is used when there was a timeout in one of the options. The complete sequence of options may not have been executed.
> 1 (error)	2 (error)	The error code is used when the specified command could not be executed at all. For instance if there was an illegal combination of options, or if the specified database name was not found. If an “error” status code is returned, an informational error message will also be produced.

4.3 System information - using MIMINFO

The MIMINFO program is used to obtain information from a MIMER database server which is useful for system control, system tuning and trouble-shooting analysis.

Information can be generated from an active MIMER database server as well as from the SQLPOOL and Bufferpool dump files produced by using MIMCONTROL (see Section 4.2).

The output from MIMINFO can be displayed on the screen and may also be directed to a file.

The following reports may be obtained from MIMINFO (further details on each report can be found in the sub-sections that follow):

users list	this lists details of all the users currently connected
MIMSERV report	this provides information useful for monitoring performance parameters
MIMDUMP report	this produces a report which is useful to Mimer Support personnel when investigating system problems.

4.3.1 MIMINFO syntax

The MIMINFO program is controlled by flagged information specified on the command-line.

The overall syntax for MIMINFO (expressed in Unix-style) is:

```
miminfo [-o file] -u | -p [database_name]
miminfo [-o file] -u | -p | -m -f
```

If a database name is not specified, the default database (see Section 3.7.2) will be targeted.

The table that follows lists the command-line flags recognized by MIMINFO (flags are shown in both Unix-style and VMS-style).

Unix

The Unix-style command-line flags must be used on a Unix machine.

VMS

Either the Unix-style or the VMS-style command-line flags may be used on a VMS machine - see the *MIMER Guides* for VMS for more details.

Win

The Unix-style command-line flags can be used from a Command Prompt window. The shortcut to the MIMINFO program can also be used and menu selections can be made from within the program to control it.

MIMINFO command-line flags

Unix-style	VMS-style	Function
-o file	/OUTPUT = file	Send output to the specified file instead of to the screen
-u	/USERS	Display users list
-p	/MIMSERV	Produce MIMSERV report
-m	/MIMDUMP	Produce MIMDUMP report
	/DATABASE = database	Take information from the specified database
-f	/FILE	Take information from a dump file (for a users list, a dump file called "SQLPOOL.DUMP" is expected to exist otherwise a dump file called "BPOOL.DUMP" is expected to exist)

A detailed description of each of the MIMINFO reports follows.

4.3.2 The users list

```
miminfo [-o file] -u [database_name] | -f
```

A users list can be generated from an active database or from a dump file produced using MIMCONTROL. The users list shows the name of each ident connected to the database and the channel number used by the connection. The channel number may be used in conjunction with MIMCONTROL to kill a user.

4.3.3 The MIMSERV report

```
miminfo [-o file] -p [database_name] | -f
```

The MIMSERV report can be used by the system administrator to monitor performance parameters during MIMER use. The MIMSERV report can be generated from an active database or from a dump file produced using MIMCONTROL.

The MIMSERV report presents five kinds of statistical information which may be useful for system tuning (statistics for page management, transactions, background threads, databank and table usage).

Note: When a MIMSERV report is used as an aid to system tuning, it is important that the report be generated when the database is in full use. The output from several executions over a period of a few hours or days can provide valuable information on fluctuations in system usage.

The MIMSERV report contains the following information:

General statistics

Current date and time

Current hardware and operating system

Current MIMER/DB version

Name of database

Starting date and time

System status

If system status is in an error state, a database dump should be made by using the “-A” option in MIMCONTROL. The dump directory created should be saved for use by Mimer Support personnel. The database server can then be restarted. The database server log file should also be inspected to help find the cause of the failure.

Error count

Number of errors that have been written to the database server log. This value should normally be zero.

No. of request threads

Number of request threads started (see Section 4.1.2).

No. of background threads

Number of background threads started (see Section 4.1.3).

No. of I/O threads

Number of I/O threads (typically zero on most machines where separate threads are not needed for I/O processing).

Page Management statistics

No. of page buffers per sorter

Total number of MIMER pages that a request thread performing a sort operation may utilize.

No. of remaining sorters

The initial value specifies the number of concurrent sort/merge steps that are allowed.

No. of pages written to disk

An indication of the frequency of disk update operations.

No. of file extend operations

The total number of times databank files have been dynamically extended since the latest startup. The value should preferably be as low as possible for performance reasons. It is possible to check databank size usage with the DESCRIBE command in BSQL. A databank can be extended by using the commands ALTER DATABANK ADD... or ALTER SHADOW ADD....

Buffer size 2K (16K, 64K)

The bufferpool is divided into a region with 2K buffers, one region with 16K buffers, and one region with 64K buffers. The following information is given for each region:

No. of page buffers

This is the number of page buffers allocated to this bufferpool region.

No. of page partitions

Each region in the bufferpool is divided into separate partitions. Each partition can be accessed concurrently by the MIMER request threads. In tightly coupled multi-processor systems it is desirable, for performance reasons, to have at least as many partitions as there are CPUs. The number of partitions may be increased by increasing the region size.

No. of page requests

Total number of access operations to pages in the bufferpool since latest system start-up.

No. of page faults

Total number of page access requests that resulted in disk read operations. If this value is more than about 2% of the total number of page requests, performance may be improved significantly by increasing the bufferpool size.

No. of pages swapped out

Total number of pages which were written to disk when they were swapped out of the bufferpool.

Transaction management statistics**No. of transaction commits**

Total number of successful read/write transaction commits since latest system start-up.

No. of read commits

Total number of successful read-only transactions since latest system start-up.

No. of transaction checks

A high proportion of transaction checks in relation to the total number of transactions may indicate ill-designed application programs, with long transactions that are more likely to give rise to transaction conflicts.

No. of transaction aborts

Total number of transactions aborted by the optimistic concurrency protocol since latest system start-up. User requested transaction aborts are not counted.

No. of pending restarts

This is an indication of how much information is stored in TRANSDB. Number of restarts is counted for each databank used in a transaction. This figure grows larger when shadows are set offline.

If all databanks have been accessed and there are no offline shadows there should not be any pending restarts.

Background threads

SWA

Background thread identifier.

State

State of the background thread. If the background thread is currently working with a transaction, “active” is displayed. If the background thread is not doing anything, “inactive” is displayed. “I/O processing” means the background thread is flushing one or more transactions to disk and “unused” means that no background thread is using that slot.

Trans-no

The number of the transaction currently being processed.

Transaction-count

The number of transactions processed by the background thread. This gives an indication of how many background threads are needed in the system. A background thread which is never used indicates that the number of background threads started can be reduced.

Pending background thread requests

This indicates how many transactions have not yet been processed by the background threads. If there are too few background threads this value will grow.

Application waiting for trans-no

For certain operations (SET DATABANK OFFLINE, for example) the application has to wait for the background threads to complete their operations. If there are too few background threads it may take some time before this operation is complete. By comparing this trans-no with the trans-no being handled by the background threads it is possible to see how many transactions are left before the operation is completed. (Note that the example report shown in Section 4.3.3.1 does not show this statistic because the application is not waiting for transactions to complete.)

Databank statistics

Name

The name of the databank or shadow.

DBANKID, SEQNO

Databank identification. These two values correspond to the columns with the same names in the data dictionary table MIMER.DBFILE.

Type

The databank option NULL, TRANS, or LOG. See Chapter 6 of the *MIMER/SQL User's Manual* for a description of these options.

The SQLDB databank has the type WORK, and shadows have the type SHADOW.

Users

Internal user count.

Access

Access mode by which the databank was opened. The possible values are: “Read”, “Write”, “Shared” and “Exclusive”. If the databank is open but not referenced by any active statement, “None” is displayed. If a shadow is offline, it is also indicated here.

No. of databanks currently open

A count of both databanks and shadows opened in the system.

Max number of databanks open concurrently

This is defined by a parameter in the local database definition (see Section 3.3) or possibly by a limit in the operating system.

Databank verification count

Databank verification is automatically performed when a databank is re-opened without having been correctly closed. Each time this happens a log entry is written to the database server log file. When a databank is verified the databank verification count is incremented. The count is cleared when the system is started, and a databank is only verified once per session.

Databank verification is only done on index pages...**Databank verification is performed on all pages...**

These information messages indicate the databank verification mode as defined in the local database definition (see Section 3.3).

Table statistics**No. of tables currently open**

This shows the number of tables open in both master and shadow databanks. Also included are the read and write sets used by each user.

Max number of tables open concurrently

This number is set as a parameter in the local database definition (see Section 3.3).

4.3.3.1 MIMSERV output example

The following example output is from a MIMER system which is not being limited by the bufferpool size (page faults, in 2K region, are only about 1% of page requests). The number of transaction checks is low, indicating either that concurrent user update requests are infrequent or that transaction handling in application programs is well-designed.

The 16K region has been used very little and the 64K region has not been used at all. It is possible to adjust the sizes of the different regions depending on the requirements of the system.

The error count has a value of 4 which indicates that 4 errors have been written to the database server log file (see Section 4.4).

The databank SYMDB has an error status. The database server log will contain further information.

The shadow CASE5_S1 has been set offline as indicated at the end of the line describing the shadow.

*** MIMER/DB runtime statistics 1998-07-23 16:07:58 ***

Hardware, operating system ALPHA/VMS
 Current MIMER/DB version is 8.1.1
 Database name: HOTEL
 System started at 1998-07-17 16:35:27
 System status is up

Error count	:	4
No. of request threads	:	2
No. of background threads	:	3
No. of I/O threads	:	1

Page management statistics
 =====

No. of page buffers per sorter	:	11
No. of remaining sorters	:	2
No. of pages written to disk	:	1774
No. of file extend operations	:	0

Buffer size 2K

No. of page buffers	:	512	
No. of page partitions	:	3	
No. of page requests	:	40859	
No. of page faults	:	432	1 %
No. of pages swapped out	:	0	

Buffer size 16K

No. of page buffers	:	42	
No. of page partitions	:	1	
No. of page requests	:	44817	
No. of page faults	:	7	0 %
No. of pages swapped out	:	0	

Buffer size 64K

No. of page buffers	:	42
No. of page partitions	:	1
No. of page requests	:	0
No. of page faults	:	0
No. of pages swapped out	:	0

Transaction management statistics
 =====

No. of transaction commits	:	1336
No. of read commits	:	0
No. of transaction checks	:	1436
No. of transaction aborts	:	103
No. of pending restarts	:	18

Background threads
 =====

SWA State	Trans-no	Trans-count
175 Inactive	0	11985
189 Inactive	0	1382
203 Inactive	0	12

Pending background thread requests : 0

```

Databank statistics
=====
Name                DBANKID  SEQNO  Type    Users Access
TESTDBM1            310     0  TRANS    1 Shared
TESTDBM2            311     0  TRANS    1 Shared
SYSDB                1       0  LOG      2 Shared
TRANSDB             2       0  TRANS    1 Shared
LOGDB                3       0  LOG      1 Shared
SQLDB                4       0  WORK     1 Shared
SYMDB                285     0  TRANS    1 Shared
Databank error status: -16142
CASE5                279     0  LOG      0 Shared
CASE5_S1             279     1  SHADOW   1 Shared  Offline

No. of databanks currently open      :      6
Max number of databanks open concurrently :    100
Databank verification count          :      0
Databank verification is performed on all pages

Table statistics
=====
No. of tables currently open          :     36
Max number of tables open concurrently :    4000

```

4.3.4 MIMDUMP report

```
miminfo [-o file] -m -f
```

The MIMDUMP report is used by Mimer Support personnel for troubleshooting when database problems are reported by customers. When assistance from Mimer Support personnel is needed, it is important that a MIMDUMP report is generated and included with the error report.

4.4 Database server log

The database server log lists startup and shutdown messages for the database server. It may also contain warning and error messages if such situations have been detected by the database server.

Unix

Under Unix, a log file called **mimer.log** is created when the database server is started for the first time. This file is located in the database home directory. In addition, the *Oper* parameter in the MULTIDEFS file can be set to specify where serious database server messages should be recorded. Such messages always go to the Unix system log.

VMS

Under VMS, a log file called **MIMER.LOG** is created when the database server is started for the first time. This file is located in the database home directory. In addition, the *Oper* parameter in the MULTIDEFS file can be set to specify where serious database server messages should be recorded.

Win

Windows NT: Database server events are logged in the EventLog which may be examined using the operating system tools.

Windows 95/98: A log file called **Mimer.log** is created when the database server for a database is started for the first time. This file is located in the database home directory.

5 EXPORT/IMPORT

The Export/Import functionality provides facilities for:

- loading and unloading data between database tables and sequential files
- moving tables from one database system to another
- generating system databanks for the optional MIMER modules.

The functionality is controlled through a series of menus. These are displayed sequentially at the terminal so that it may be run from a console or printing terminal.

Any MIMER ident may start the Export/Import functionality from the UTIL program as follows (user input is shown in bold):

```

M I M E R / U T I L

Username: SYSADM
Password:

-- MIMER UTILITIES --

1. Backup/restore
2. Export/import, SYSxxGEN
3. SQL statistics
4. Readlog
5. Shadowing

0. Exit

Select: 2

```

5.1 The main menu

Logging on to the Export/Import functionality presents the main menu shown below and a detailed description of the functions follows:

```

-- Export / Import utility --

1. Export - definitions and data
2. Export - definitions only
3. Import - object creation
4. Import - data load
5. Load / Unload table
6. SYSxxGEN
7. Enter program ident
8. Leave program ident

0. Exit

```

5.2 Export options

5.2.1 Functions

From the main menu, the user may export table definitions with or without data. The exported objects and data are stored in sequential files, which may be imported to re-create the tables in another MIMER system.

If columns in tables being exported belong to domains, the appropriate domain definitions are exported together with the table definitions. Indexes defined on exported tables are also exported.

Both choices (“Export - definitions and data” or “Export - definitions only”) display a second menu giving the following options:

```
-- Export --
1. Export specific tables
2. Export all tables for ident
3. Export all tables in databank

0. Exit
```

Depending on the selection made, the user is prompted for a list of table names (the list is terminated by the first “empty” or blank name), an ident name, or a databank name. Specific tables exported in one operation do not need to be owned by the same ident or stored in the same databank. In the case of tables selected by ident or databank, a single set of tables will be exported by each operation.

For all export functions, the user is prompted for the name of the sequential file to be used for the export.

5.2.2 Authorization

The ident performing an export operation must have SELECT privilege on all the tables being exported. The “Enter program ident” option in the main menu provides a facility for specifying an alternative ident to use while performing the export operation. The user running the Export/Import functionality must have EXECUTE privilege on the program ident in order to enter the program.

If export is performed for an ident or a databank, any tables to which the ident running the functionality does not have SELECT privilege are ignored.

5.2.3 Exported files

Table, domain, and index definitions are exported in the form of CREATE statements for re-creating identical objects in the import environment. See the *MIMER/SQL Reference Manual* for the syntax and description of the relevant CREATE statements.

Note: The import function allows naming conflicts to be resolved interactively during import.

Table contents are exported in sequential format.

Export files begin with system information concerning MIMER version, machine and operating system, file record length and data storage formats, and so on. Statistical information at the end of the file records the number of data rows exported for each table.

5.3 Import options

The import functions use the files generated by export as input in order to re-create the exported tables in a new environment.

The import operation is performed in two stages, object creation and data loading. These are represented by the options, “Import - object creation” and “Import - data load” in the Export/Import main menu, and must be performed separately even if the table definitions and data were exported together in the same file.

5.3.1 Import - object creation

This phase creates the tables being imported, together with any domains used in the table definitions and indexes defined on the tables. The newly created objects are owned by the ident performing the import operation. The “Enter program ident” option in the main menu allows imported files to be processed by a program ident.

The user is prompted for the name of the export file to be used, and also for the name of a log file which will be used if any table names are altered from those given in the export file (see below).

Before the imported tables are created, the table-databank couplings in the export file are displayed, and the user is given an opportunity to redirect the tables to new target databanks. If a target databank is specified which does not exist in the import environment, the user is prompted for the filename, size, and options and a new databank is created. If the target databank name is left blank, MIMER will place the table in the databank which is judged best for the purpose (equivalent to using the SQL statement CREATE TABLE with no IN clause).

Once the table-databank couplings have been accepted, the tables are created from the CREATE TABLE statements in the export file.

The following example shows the creation of the import objects (user input is shown in bold):

```

-- Export / Import utility --

1. Export - definitions and data
2. Export - definitions only
3. Import - object creation
4. Import - data load
5. Load / Unload table
6. SYSxxGEN
7. Enter program ident
8. Leave program ident

0. Exit

Select: 3

-- Import - object creation --

File generated by Export/Convout : BOOKEXP

Import log file : BOOKIMP

Listing of couplings between tables and target databanks

TABLE NAME                DATABANK
-----
BILL                       <--->    BOOKDB
BOOKFORM                   <--->    BOOKDB
BOOK_GUEST                  <--->    BOOKDB
EXCHANGE_RATE              <--->    BOOKDB
FREEROOMS                   <--->    BOOKDB
ROOMSTATUS                  <--->    BOOKDB

Are the couplings okay <Y>? : Y

Making definitions
Operation completed

```

Naming conflicts

Naming conflicts will arise during import operations if the names of imported tables or domains already exist in the import environment. If this occurs, the user may choose to rename the object in question, skip creation of the particular object, or quit the object creation operation.

If any tables are renamed during the object creation, the altered names are stored in the import log file from which they are read by the data load operation in the next phase of import.

Domain conflicts need only be resolved once for any import operation. All usage of the domain in the imported table definitions are corrected according to the response to the first reported conflict.

If the user chooses to quit the object creation operation at a naming conflict, any objects created prior to the naming conflict remain in the import environment.

The following example shows how to rename an object (user input is shown in bold). Note that the duplicate table name is shown with the creator name for clarity, but that the new name may not be qualified by a creator name (the ident performing the import operation owns the created tables).

```

-- Import - object creation --

File generated by Export/Convout   : BOOKEXP
Import log file                   : BOOKIMP

Listing of couplings between tables and target databanks

TABLE NAME                        DATABANK
-----
BILL                              <---->   BOOKDB
BOOKFORM                          <---->   BOOKDB
BOOK_GUEST                        <---->   BOOKDB
EXCHANGE_RATE                     <---->   BOOKDB
FREEROOMS                         <---->   BOOKDB
ROOMSTATUS                        <---->   BOOKDB

Are the couplings okay <Y>?      : Y

Making definitions
Duplicate table name              : BOOKADM.BOOK_GUEST

Skip/Quit/Rename <S/Q/R>? R

Give new name                    : BOOKGUEST2
Operation completed

```

Referential conflicts

Referential conflicts can arise in several ways during an import operation:

- depending on the structure of the exported database and the way tables are exported, an attempt may be made during import to create a foreign key for which the reference table does not exist in the import environment
- the order in which the tables were exported may result in an attempt during import to create a foreign key before the reference table is created
- an imported table may have a foreign key which refers to a table which exists in the import environment but which is the “wrong” table (i.e. the primary key of the reference table is not consistent with the foreign key).

It is the responsibility of the user performing the import to avoid referential conflicts in the imported tables, either by importing tables in the correct order or editing foreign key definitions in the export file before the tables are imported. The import functionality does not provide any interactive facilities for correcting conflicts of this type. If referential conflicts do arise, the table(s) affected will not be created and the import will be aborted with a message informing the user of the error.

Note: In a particularly unfortunate circumstance, a table may be imported with a foreign key clause which happens to coincide with an existing table in terms of the reference table name and column definitions in the import environment, but which the user does not actually want to use as a reference table. This situation cannot be detected by the import functionality, since the foreign key reference is syntactically valid. Care is demanded of the user to ensure that this situation does not arise (the risk is minimized by intelligent use of table and column names).

5.3.2 Import - data load

This phase of an import operation loads the imported tables with data from the export file. The operation must be performed separately from the object creation phase. The user is prompted for the name of the export file and for the name of the log file specified during the object creation phase (see above). It is important that data load uses the same log file that was written when the objects were created. The user is also asked if duplicates should be logged. If a duplicate is encountered, a report is written to the session log and a summary appears at the terminal.

Note: Data loaded into tables by the import functionality must be stored in an export file. The import functionality cannot use data files created by the Unload function (see below).

If an error occurs during the data load operation, a message appears on the terminal and the record is logged. If 100 errors occur the import operation is aborted. Tables are loaded in transactions of at most 100 rows each. The progress of the loading is reported by a message every 100th row for the first 10000 rows and thereafter every 1000th row. If the import operation is aborted because of errors, the rows which have been reported as loaded will remain in the table.

Hint: If there is a large amount of data to be imported it can be done faster by setting the databank option to NULL before the load operation takes place. This is because transaction handling affects the performance of the load operation. Do not forget to set the databank back to the desired option after the load is finished. Note, however, that the databank option must not be NULL if foreign or unique constraints are involved (if it is, an error message will be generated).

The following example shows data being loaded from the export file BOOKEXP (user input is shown in bold):

```

-- Import - data load --

File generated by Export/Convout   : BOOKEXP
Log file from 'object creation'    : BOOKIMP
Log duplicates <Y>? N

Loading table : BOOKADM.FREEROOMS
    100 rows loaded
    195 rows loaded

Loading table : BOOKADM.ROOMSTATUS
    20 rows loaded
Operation completed

```

5.3.3 Authorization

The ident performing the object creation phase of the import operation must have TABLE privilege on any databank in which a table is to be created, and also DATABANK privilege if new databanks are to be created. REFERENCES privilege on the appropriate table(s) is required if any foreign keys are to be created.

The ident performing the data load phase must have INSERT privilege on tables being loaded. Normally, the data load phase is performed by the same ident as the object creation phase, in which case INSERT privilege is automatically granted (since the ident owns the tables).

The “Enter program ident” option in the main menu allows a user to act in the capacity of a program ident. The user running the Export/Import functionality must have EXECUTE privilege on the program ident in order to enter it.

5.4 Load and Unload functions

The load and unload functions allow data to be moved between MIMER tables and sequential files. The functionality here is identical to that provided by the LOAD and UNLOAD commands in BSQL, although the interactive screen forms differ between the two environments. (The screens used when the functionality is run from the Export/Import menu are adapted to allow load and unload to be performed from a console or printing terminal).

The term “load” refers to moving data from a file to a table.

The term “unload” refers to moving data from a table to a file.

Choosing the load/unload function from the main menu displays another menu offering four alternatives:

```
-- Load / Unload --  
  
1. Load from file INTO table, default format  
2. Load from file INTO table, user specified format  
3. Unload to file FROM table, default format  
4. Unload to file FROM table, user specified format  
  
0. Exit
```

5.4.1 Data file formats

The user may choose between a default or a user-specified format for the data file. In either case one row of table data (one tuple) corresponds to one record in the sequential file.

The default format allocates space for each column according to the definition of the column in the table, with no extra space between fields in the record.

If a user specified format is chosen, the user must give the start and end positions in the record for each column of the table. Fields in the record do not need to be contiguous. If fields overlap in an unload operation, data from the earlier columns in the table definition will be overwritten in the overlapping positions by data from the later columns.

Both character and numerical data is written to the sequential file in string format. Thus the number 143.6 is unloaded as the string “ 143.6”, occupying 6 bytes (the leading blank is the sign position). Files for loading tables can thus be created with any text editor, and data unloaded from tables could be edited before being reloaded.

NULL values

Before the load or unload operation is performed, the user is asked if a preceding NULL byte is to be used.

If data is unloaded from a table with a preceding NULL byte, an extra byte is added before each field in the sequential record. This byte is assigned an ampersand (&) if the column from which the field is derived contains NULL. Otherwise the byte is blank. At unload, if the NULL byte contains an ampersand, the rest of the field is filled with periods (...).

If data is loaded into a table using a preceding NULL byte, the first byte of each field is read as a NULL flag (an ampersand in this byte indicates NULL; any other value indicates non-NULL). If the NULL byte contains an ampersand, the actual data content of the field is irrelevant.

Note: If the preceding NULL byte is used for the load operation, each field must be one byte longer than the corresponding column.

When the preceding NULL byte is used with a user-specified file format, the starting position given for each field should be the position of the NULL byte. In this case the column data starts at position+1.

5.4.2 Load operation

The load operation transfers data from a sequential file to a table in the order of the records in the sequential file.

If the default file format is used, data from the file record is mapped into the table columns using the definition and order of the columns in the table.

If the user-specified file format is used, the user has full control over where in the record the data for each column in the table is to be taken. The same positions in the record may be used for data inserted into as many columns in one table row as is required. Values for one row in the table may not, however, be taken from more than one record. Any columns in the table omitted from the user-specified file format will be assigned the NULL indicator or a default value as appropriate.

If a string is specified which is longer than the character column width, the following error message appears:

```
Input character string too long.
```

The load operation is performed, conceptually, as a series of INSERT statements, one for each row in the table. If rows with duplicate primary key values exist in the loaded data (or if a loaded row duplicates an already existing row), only the first of the duplicates is retained in the table. The number of rows loaded and the number of duplicates found is reported when the operation is complete.

Hint: If there is a large amount of data to be loaded it can be done faster by setting the databank option to NULL before the load operation takes place. This is because transaction handling affects the performance of the load operation. Note that the databank option may not be NULL if foreign or unique constraints are involved (an error message will be generated). Do not forget to set the databank back to the desired option after the load is finished.

Data may not be loaded into views.

5.4.3 Unload operation

When data is unloaded from MIMER tables, records are written to the sequential file in the ascending primary key order of the table.

All rows are unloaded from the table. If a subset of rows is required, a view can be defined containing the required rows and the data unloaded from the view. (Alternatively, the whole table can be unloaded and excluded records then deleted from the sequential file). The source table remains intact after the unloading.

Selected columns may be unloaded by choosing the user-specified format and listing the required columns. Only the columns listed in the file format will be unloaded.

The sequential file is created at the beginning of the unloading process. It is not possible to append unloaded data directly to a pre-existing file.

When the operation is complete the number of rows unloaded is reported.

5.4.4 Authorization

The ident performing a load operation must have INSERT privilege on the table into which the data is being loaded.

The ident performing an unload operation must have SELECT privilege on the table or view being unloaded.

5.5 Enter and Leave program ident

If a program ident is the creator of a table, the program ident may be entered in advance in order to export or unload the table.

Use “Leave program ident” to conclude the use of the program ident.

6 BACKUP AND RESTORE

This chapter discusses the different ways to backup MIMER databanks:

- by backing up the databanks from the host operating system
- by using the SQL system management statements.

The backup and restore functionality in the UTIL program is still supported for backward compatibility.

How to restore data after a crash is also discussed. Refer to Section 6.2 for instructions on the recommended procedures for handling databank backup and recovery.

Data need not be restored in the event of a power failure or system shut-down that does not damage the databank files, since any transactions that were committed but not completed at the time of the failure are automatically completed when the databank involved is next accessed.

Some of the discussion in this chapter refers to Shadowing, which is an optional MIMER product that allows one or more copies of a databank to exist on different disks. Shadowing provides a high level of protection from disk failure and allows databanks to be backed up while the system is in use (see *MIMER Shadowing Handbook*). The system will automatically start using a databank shadow if the master databank is lost. The SQL system management statements can be used to backup and restore databanks whether you are using MIMER Shadowing or not, but the functionality that applies to Shadowing can only be used if you have purchased the optional Shadowing module.

Reference is made to transaction handling at several points in this chapter. If you are not familiar with transaction handling in MIMER see Chapter 6 of the *MIMER/SQL Programmer's Manual* or Chapter 6 of the *MIMER/SQL User's Manual* for a more detailed description.

6.1 Background information

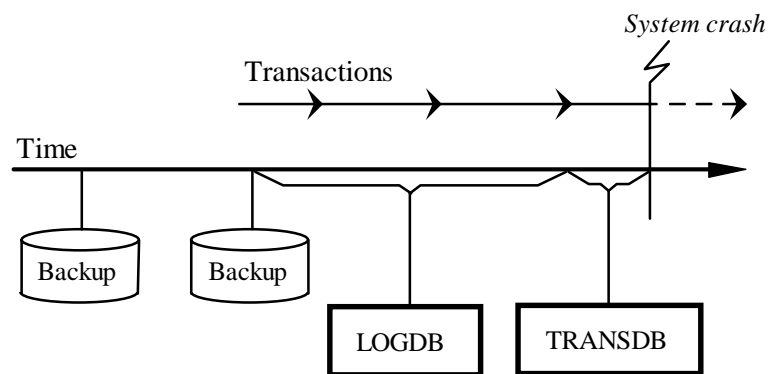
A MIMER database consists of a collection of databanks (each in a separate operating system file) containing tables with data used by the applications. The SYSDB system databank contains a data dictionary describing the different objects in the database. All operations on tables located in databanks with the LOG option are logged in the system databank LOGDB. In the event of a disk crash this information can be used to achieve a full restoration of the information lost from a destroyed databank.

The databank backup facilities delete the records from LOGDB which apply to the information recorded in the databank backup after it has been taken. LOGDB therefore contains a record of all the information that has been added to a databank file since the last backup was taken for it. It is possible to take a backup of a databank without clearing the LOGDB records, however it is usual practice to clear the LOGDB after backing up a databank.

Thus, in the event of a databank being destroyed by machine failure, LOGDB can be used to restore any changes made between the last backup operation and the time when the databank was lost (assuming a proper backup copy exists for that databank). It is important to note that data handling operations that are not logged cannot be restored in the event of a hardware failure.

If a MIMER system is stopped (either deliberately or by a system failure) during the commitment of a transaction, information is retained in the TRANSDB system databank and used to complete the transaction on the next occasion the databanks involved are opened. This is only true for databanks with the TRANS or LOG option. Once information has been successfully written to both LOGDB and the databank file, it is removed from TRANSDB.

The following diagram illustrates the relative functions of LOGDB and TRANSDB in crash protection.



Important! Wherever possible, LOGDB should be stored on a different disk unit, with a separate disk controller, from the other databanks in order to minimize the risk that a disk crash or damaged disk controller destroys both the log and the other databanks. Ideally, LOGDB and TRANSDB should always be located on different physical disks which are served by separate disk controllers and no other databank files should be located on either disk, since data may be lost if both TRANSDB and LOGDB are destroyed while a committed transaction is being processed. (Refer to Section 2.3.4 for more details on data security and databank files.)

6.1.1 Database consistency

Database consistency is handled on two levels: physical and logical.

Physical consistency means that the tables are readable by MIMER/DB. This is ensured as long as the databank file is not physically damaged.

Logical consistency means that the tables contain valid data. This is ensured by MIMER's transaction handling. All transactions are saved in the TRANSDB databank during build-up and are performed on the databanks when they are committed. To use transaction handling the databank must be created with the TRANS or LOG option.

Transaction handling makes it possible to ensure that a user's operations do not read data that is being updated by another user. If a transaction is successfully committed then all operations in the transaction are performed. If the transaction is aborted due to a conflict, none of the operations in the transaction are performed.

The tables may be logically inconsistent if MIMER is stopped before all operations in a committed transaction have been performed. At some time after the system is restarted all uncompleted transactions will be read from TRANSDB and automatically performed to ensure the system is in a consistent state again. This happens on a per-databank basis, after a databank is first accessed following the restart. Transactions that were not committed before the stop are aborted.

The DBOPEN facility (see Chapter 10) can be used to open all databanks in one operation and thus achieve transaction consistency quickly.

Note: The TRANSDB system databank should never be deliberately deleted, because uncompleted transactions may remain saved in the databank even if the database server is currently stopped. If a TRANSDB file is deleted which contains transactions that are not yet completed, inconsistency will occur because the information required to complete those transactions when the database server is re-started will have been lost.

Some data retrieval requests in MIMER/SQL may require large work areas or transaction handling areas for intermediate processing of the data (for instance, requests to sort or group large result sets will require large work tables in SQLDB). This is particularly relevant to BSQL, where ad-hoc queries may be submitted with little thought for the processing requirements or performance of the query. In systems where files expand automatically, the file for SQLDB can become very large as the result of one badly-planned query. A large SQLDB file can be deleted and a new SQLDB created by using the UTIL program.

It may also be advisable to run the DBC program (see Chapter 9) routinely immediately before each backup operation to check the contents of the databank and make sure that the backup copy will be error-free.

Backup copies of a databank should always be taken when the databank is in a logically consistent state, that is, no uncompleted transactions should exist. If a transaction updates two databanks that depend on each other, both databanks should be in a logically consistent state when the backups are taken.

This is best accomplished by the following steps:

- set the database offline using the following command:

```
SET DATABASE OFFLINE
```
- stop the database server
- check the database server log file for any error messages which might affect the databanks for which you are about to take a backup
- take a backup of the databanks.

6.1.2 Databank backups

A databank backup is a complete copy of a databank file.

It is the starting point for any restore operation, and should be stored in a safe place separate from the working databank files (copied to a different disk or preferably written to backup media and removed from the machine). A databank backup covers all the updates made to the databank, including all those updates recorded in the LOGDB databank file at the time the backup is taken.

This type of backup can be taken either by using the SQL system management statements or by using the host file system.

After a backup is taken, the updates logged for the databank in question should be cleared from LOGDB. This will be done automatically when the SQL system management statements are used to take the backup. If the host file system is used, the databank (or the whole database) must be set offline before the backup is taken and then set online again afterwards with the RESET LOG option to clear LOGDB.

Refer to Section 6.1.6 for more information on use of the SQL system management statements.

6.1.3 Databank incremental backups

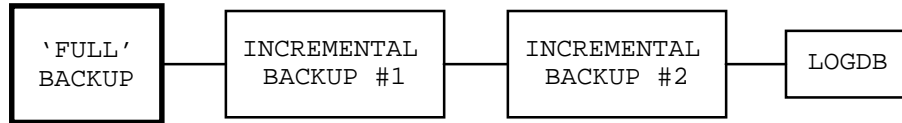
A databank incremental backup is a copy of all the updates recorded for it in the LOGDB system databank file.

This type of backup cannot be performed using the host file system, only the SQL system management statements can be used.

An incremental backup is a record of all the updates made to the tables in a databank since the last backup or incremental backup was taken (or more precisely, since the updates logged in LOGDB for the databank were last cleared). Note that an incremental backup is not a copy of the databank file, the information comes from the updates logged in LOGDB, not from the databank file.

Incremental backups are economical in both time and storage space, but it is essential for the backup function that the chain of incremental backups is intact from the most recent backup of the databank.

A databank may thus be protected against data loss by a sequential chain of incremental backups, originating from a backup of the databank and ending with the updates currently recorded in LOGDB for the databank:



Backups taken by the host file system and those taken by using the SQL system management statements are equally valid as the origin for a chain of incremental backups.

6.1.4 Backup versus Incremental Backup

Any sequence of databank backups must start from a backup, which is a complete copy of the databank file. After that the choice about whether to take an incremental backup or a backup comes down to a trade-off between the time it takes to restore the database in the event of a failure, versus the risk that the contents of a databank file may have been recorded after it got into an invalid state.

A **backup** is a complete copy of the contents of a databank file and thus everything in the databank will be copied, including any data corruption that may exist. Therefore, there is some risk that this type of backup may record data that cannot be used to restore a useable databank. Taking this kind of backup is likely to be slower than taking an incremental backup because a much greater volume of information is typically involved. Restoring a databank from it, however, will be the fastest choice because the databank will be almost completely restored in one step rather than the many steps involved in a restore from incremental backups.

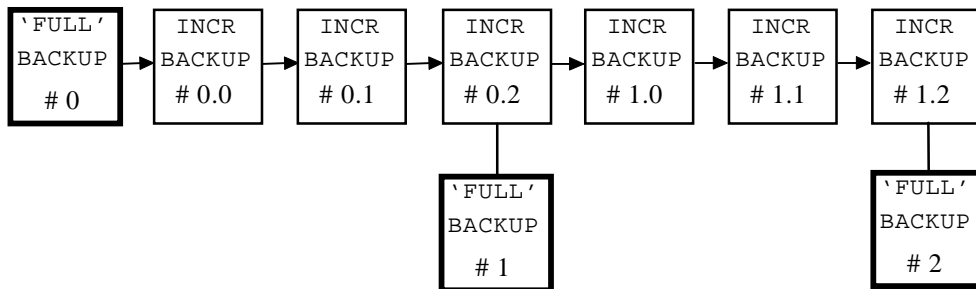
An **incremental backup** is a record of the updates made to the tables in a databank, which have been logged in LOGDB. When this type of backup is used to restore a databank, the updates are re-applied sequentially to the tables in the databank. Therefore, provided the restore of incremental backups starts from a valid backup copy of a databank file, this type of backup can be used to recover from a data corruption situation. Taking an incremental backup is likely to be faster because it only records the table updates made since the last backup was taken, instead of the entire contents of the databank file. It will, however, take longer to recover a databank by restoring it from a series of incremental backups because it involves re-applying each transaction performed on the tables in the database.

6.1.5 Maximizing data security

The scheme that offers the best security, whilst retaining the possibility of a short recovery time, is one that involves an incremental backup being taken at regular intervals (e.g. daily) and a backup with an incremental backup being taken regularly at some slightly longer interval (e.g. weekly).

Note: The incremental backups are not replaced by a backup at the “full” backup interval, but a backup must be taken in addition to an incremental backup at each “full” backup interval. In this way an unbroken incremental backup chain is maintained.

The suggested backup scheme is illustrated below:



The backups serve as potential starting points for a restore operation. If the most recent backup proves to be unusable because it recorded the databank file after it got into an invalid state, an earlier backup plus the chain of intervening incremental backups (which protect against data corruption) can be used to restore the databank to the same point in time.

This is why it is important to keep the entire incremental backup chain unbroken by taking both a backup and an incremental backup in a single operation at each “full” backup interval, as the previous diagram illustrated. If a “full” backup interval consists of a backup being taken without an incremental backup, that backup becomes the only starting point for a restore operation, because the chain of incremental backups leading back to the previous backup will not be complete.

6.1.6 SQL statements for backing up databanks

Refer to the *MIMER/SQL Reference Manual* for a detailed description, and syntax definition, of the SQL system management functions. A brief description of the purpose of each function appears here.

The SQL system management statements which can be used to take backups are:

CREATE BACKUP used to take a backup, with an optional incremental backup being taken at the same time in a single operation.

CREATE INCREMENTAL BACKUP used to take an incremental backup.

The user using these functions to take a databank backup must either be the creator of the databank, or have EXECUTE privilege on MIMER_BR.

When the SQL statements are used to take a backup and/or an incremental backup, the entire process of taking a databank backup is handled automatically.

If a databank shadow is available and accessible, the databank backup functions will use this in preference to the databank itself.

The process used by the databank backup statements is as follows:

- the databank (or the shadow if Shadowing is used) is set offline
- the backup and/or incremental backup information is copied to the specified backup file(s)
- the databank (or the shadow if this was used) is set online again, with the RESET LOG option.

Note: The databank backup process will always automatically clear the LOGDB records when the databank (or its shadow, if that was used) are set online again. If you want to take a backup and an incremental backup at the same time (as recommended in Section 6.1.5), you must use the CREATE BACKUP function to take both at once in a single operation.

The SQL system management statements (typically used when taking databank backups using the host file system) which can be used to set a databank, shadow or the whole database online or offline are:

SET DATABANK OFFLINE	sets a databank offline, making it unavailable.
SET DATABANK ONLINE	sets a databank online, making it available, optionally clearing records from LOGDB.
SET DATABASE OFFLINE	sets all non-system databanks offline, and makes the database unavailable. If one of the databanks cannot be set offline (e.g. because it is being used), the command will fail.
SET DATABASE ONLINE	sets all databanks online, optionally clearing all records from LOGDB and makes the database available.
SET SHADOW OFFLINE	sets a list of shadows offline, making them unavailable.
SET SHADOW ONLINE	sets a list of shadows online, making it available, optionally clearing records from LOGDB.

The user using these functions to set a databank or a shadow of it online or offline must either be the creator of the databank or have EXECUTE privilege on MIMER_BR.

A user setting the database online or offline, must have EXECUTE privilege on MIMER_BR and must be the only user accessing the database.

The SQL system management statements which can be used to recover a databank in the event of it being damaged or destroyed is:

`ALTER DATABANK RESTORE` used to restore a databank from databank backups and/or from information in LOGDB.

A user using this function to restore a databank must be the creator of a databank or have EXECUTE privilege on MIMER_BR.

6.2 Backup and restore of databanks

This section describes procedures for taking databank backups and procedures for restoring a databank in the event of it being damaged or destroyed. The procedures described here do not apply to the system databanks, these are dealt with in Section 6.3.

Refer to the *MIMER/SQL Reference Manual* for a full description of the SQL system management statements.

6.2.1 Making a databank backup

It is possible to take a databank backup using either the SQL system management statements or by using the host file system.

6.2.1.1 Using the SQL system management statements

The procedure for taking a databank backup of a single databank using the SQL system management statements is as follows:

To take a **backup** of a databank, use:

```
CREATE BACKUP IN 'file-name'
FOR DATABANK databank-name
```

This will copy the entire contents of the specified databank file to the specified file-name. If there is a shadow available, this will be copied in preference to the databank itself. During the backup process, the databank (or the shadow if that was used) will be automatically set offline. At the end of the backup process, the databank (or shadow if that was used) will be automatically set online again and records in LOGDB, which apply to the databank, will be cleared.

To take an **incremental backup** of a databank, use:

```
CREATE INCREMENTAL BACKUP IN 'file-name'
FOR DATABANK databank-name
```

This will copy the information contained in the LOGDB records for the databank to the specified file. At the end of the backup, the LOGDB records which have been copied will be cleared from LOGDB.

To take **both** a backup and an incremental backup at the same time, to maximize the security of the databank data (see Section 6.1.5), use:

```
CREATE BACKUP IN 'file-name-1'  
INCREMENTAL BACKUP IN 'file-name-2'  
FOR DATABANK databank-name
```

This will copy the entire contents of the databank file to the file specified by “file-name-1” and will also copy the information contained in all the LOGDB records for the databank to the file specified by “file-name-2”. During the backup process, the databank will be automatically set offline. At the end of the backup process, the databank (or shadow if that was used) will be automatically set online again, and LOGDB records which apply to the databank will be cleared from LOGDB.

Note: The file names specified in connection with these functions are recorded in the database (in the system table MIMER.BACKUP) for use by the backup and restore functionality in the UTIL program. This behavior may, however, be changed in the future so it should not be relied on. The person taking the backup is responsible for keeping a record of the backup file names.

Databank backup file names are subject to the same restrictions that apply to the SQL statement CREATE DATABANK - see the *MIMER/SQL Reference Manual*.

6.2.1.2 Using the host file system

You can use the host file system to take a backup of a specific databank, or to take a backup of all the databanks in the database (see Section 6.3 for specific information regarding backing up the system databanks).

Note: It is recommended that the SYSDB system databank be included in a backup of all the databanks, therefore you must stop the database server in order to close the SYSDB databank file. Note also that setting SYSDB offline does not close the databank file.

You can not use the host file system to take an incremental backup, you can only make copies of entire databank files in the same way as taking a backup using the SQL system management statements.

To take a backup of the whole database and clear LOGDB entirely, do the following:

- set the database offline using the following command:

```
SET DATABASE OFFLINE
```
- stop the database server if you wish to include SYSDB in the backup
- check the database server log for any error messages that might affect the backup
- make backup copies of the databank files (on disk or tape). Normally, all databank files should be backed up at the same time

- start the database server again, if it was stopped in an earlier step
- set the database online again using the following command:

```
SET DATABASE ONLINE RESET LOG
```

Note: If the backup fails the PRESERVE LOG option should be used when setting the databank online to leave LOGDB unaltered.

To take a backup of a specific databank, do the following:

- set the databank offline using the following command:

```
SET DATABANK databank-name OFFLINE
```

- take a copy of the databank file using the host file system commands
- set the databank online again using the following command:

```
SET DATABANK databank-name ONLINE RESET LOG
```

It is important to use the RESET LOG option when setting the databank online again, provided the backup did not fail, so that the LOGDB records which apply to the databank you have just backed-up are correctly cleared from LOGDB (as they would have been if the backup had been taken using the SQL system management statements). If the backup fails, the PRESERVE LOG option must be used.

Note: The RESET LOG option removes all records written to LOGDB since the last backup. This is essential to maintain consistency between the log and the backup.

If two databanks depend on each other it is important that they are backed up at the same time, otherwise logical inconsistency between the backups may cause problems if LOGDB is lost. It is essential that all databanks that are dependent on each other are not used while the backup is being taken. It is also important that transaction consistency in the databanks be achieved before taking the backup.

The purpose of setting the database offline is to ensure that none of the databanks are used. (Additionally, stopping the database server ensures the SYSDB databank is also not used.) Setting the database offline (using the SQL statement SET DATABASE OFFLINE) also achieves transaction consistency because the function will not return until the database has been brought into a consistent state, prior to going offline. In particular, setting the database offline will ensure that TRANSDB is flushed and that all shadow updating has been completed.

6.2.2 Restoring a databank

Restoring a databank after it has been damaged or destroyed will typically involve both the host file system and SQL statements.

Any databank restore operation must start with a copy of the databank file that is not damaged or corrupt. This is generally the copy made during the last backup, whether taken by the host operating system or by using the SQL system management statements.

Restoring this file is done using the host file system to copy it from backup disk or other media. The file may be copied into the correct location for the databank file (as recorded in SYSDB), or may be placed in an alternative location.

If the file is placed in an alternative location, the databank must be altered to point to the new file location, using the following command:

```
ALTER DATABANK databank-name INTO 'new-file-name'
```

The next step is to restore any incremental backup information. For each incremental backup file, the updates recorded in it should be applied to the databank using the following command:

```
ALTER DATABANK databank-name RESTORE  
USING 'file-name'
```

Finally, the updates made since the last backup was taken, which are recorded in LOGDB, should be applied to the databank, using the following command:

```
ALTER DATABANK databank-name RESTORE  
USING LOG
```

This command also automatically sets the databank online, making it available for use.

6.3 Backup and restore of system databanks

System databanks require special procedures for backup and restore. These procedures are described below. Note that backup protection for SYSDB is particularly important for protecting the database, since SYSDB contains all information describing the database structure. If SYSDB is lost, the system must be rebuilt from scratch.

6.3.1 SYSDB

The system databank SYSDB cannot be protected by incremental backups as this is not supported. SYSDB must therefore always be backed up by taking a “full” backup.

If SYSDB is not shadowed, the host operating system must be used when taking backups of it. The database server must always be stopped before a backup copy of SYSDB is made. This ensures that no operations are performed between taking a copy of the databank and dropping the log.

If SYSDB is lost, a backup copy of SYSDB must be restored to allow MIMER to start again. No MIMER-based application may be used before this is done.

If SYSDB is lost or corrupted, the host system backup copy should be copied to the same file location as the original SYSDB. The contents of SYSDB may then be brought completely up to date by restoring the information from LOGDB. This is done using the backup and restore functionality in the UTIL program.

Start the program and login as SYSADM. A message is displayed saying that you have an old version of SYSDB that must be restored. Answer “Y” to the question “Restore SYSDB” to restore the copy of SYSDB. Since SYSDB always has the LOG option, this will restore SYSDB to the state it had before it was lost.

Example, assuming a backup of SYSDB has been copied to the original location (user input is shown in bold):

```
MIMER/DB fatal error -16159 in function CONNECT
      Old version of the databank SYSDB cannot be accessed without
      restoring the databank with the backup and restore utility

      -- Restore databank --

Restore SYSDB?[Y]: Y

Databank SYSDB restored from log
```

6.3.2 LOGDB

If a databank is lost (e.g. due to a disk crash), LOGDB can be used to restore all changes made since the last backup. LOGDB cannot be backed up using the SQL backup statements, host system backup facilities must be used.

Backups of LOGDB are only useful if you are using host system backups for your whole database. A restored LOGDB is useful if a databank must be restored from an older backup (because a later backup recorded it in a corrupt state) and the LOGDB records must be used to bring it up to date.

Caution! If the LOGDB databank is lost for any reason, no problems will be encountered immediately. All changes will have been properly recorded in the application databanks. A new, empty, LOGDB can simply replace the log that was lost. However, a backup of the entire database must be taken immediately. The new LOGDB will be empty, and therefore in a state consistent with a backup having just been taken and all LOGDB records cleared. If a backup is not taken immediately, a later attempt to restore a databank is likely to fail because the restore operation will expect to find information in LOGDB that was lost when it was destroyed.

6.3.3 TRANSDB and SQLDB

The contents of TRANSDB and SQLDB are normally transient, so these databanks do not need backup protection.

However, backing up TRANSDB can be advisable if there are unfinished transactions (due to a system crash, etc.) and TRANSDB is in danger of being lost before the MIMER system is restarted. TRANSDB cannot be backed up with the SQL system management statements.

The following scenarios consider the possible effects of losing TRANSDB:

- If TRANSDB is lost while a transaction is being built but before it is committed, the uncommitted transaction is lost. You can simply create a new TRANSDB and start the build over again. You do not need to restore any files (see Section 6.3.4).
- If TRANSDB is lost while a committed transaction is in progress but before it is completed, some of the changes requested in the transaction may have been made while others are unfinished. As a result, the database as a whole is not internally consistent (but it is not physically corrupt). If you suspect this to be the case, restore the most recent backup of your databanks and use LOGDB to bring the backup copies to the state they were in before the transaction was started.
- If both TRANSDB and LOGDB are lost while a committed transaction is in progress but before it is completed, the restoration described above cannot be accomplished. In such a case, the best solution is to repair the inconsistency immediately after restarting the database server by using a tool such as BSQL. This is usually possible if the user who initiated the interrupted transaction can be identified and contacted. (Many applications maintain a parallel transaction log file for tracking purposes which can be used as a basis for repair work.)

The alternative solution is to start over from the last previous backup of your databanks and reprocess all transactions since that time. This may be a very costly operation.

Keeping TRANSDB and LOGDB on separate disks under separate disk controllers will minimize the risk that both databanks are lost at the same time.

6.3.4 Re-creating TRANSDB, LOGDB and SQLDB

No MIMER applications can be run if TRANSDB, LOGDB or SQLDB is missing. In this event, an attempt to start the UTIL program (you must be SYSADM) will give you an opportunity to re-create the missing databanks with the same file names as the lost databanks, or to alter the filenames in the case where the files were moved. *A complete backup of the system should always be taken before LOGDB is re-created.* Otherwise the old backups will be inconsistent with the new, empty LOGDB.

Warning! If TRANSDB has to be re-created because it has been lost, any uncompleted transactions which were saved in the old TRANSDB will have been lost. Information in TRANSDB is used to complete committed transactions. This means that if a new TRANSDB is created when the transactions are not yet completed, inconsistency will occur because information required to complete transactions will no longer be available.

The following example shows how to re-create LOGDB (user input is shown in bold):

```

M I M E R / U T I L

Username: SYSADM
Password:

MIMER/DB fatal error -16142 in function CONNECT
Cannot open databank LOGDB,
file logdb8 not found

-- Redefinition of system databank --

-- Description of databank name and file --

DATABANK FILENAME
=====
LOGDB logdb8
Re-definition of LOGDB? [Y]: Y
CREATE new file or ALTER filename for LOGDB? (C/A): C
Size [1000] : 5000

Databank LOGDB redefined

```

7 READLOG FUNCTIONALITY

The Readlog functionality is started from the UTIL program as follows (user input is shown in bold):

```

M I M E R / U T I L

Username: SYSADM
Password:

-- MIMER UTILITIES --

1. Backup/restore
2. Export/import, SYSxxGEN
3. SQL statistics
4. Readlog
5. Shadowing

0. Exit

Select: 4
```

7.1 Functions

The READLOG functionality allows the contents of LOGDB to be read, so that logged operations performed on the database since the last backup copy or incremental backup was taken may be checked. This facility may be used as an audit trail or in the event of a system failure to determine which databanks need to be restored (i.e. which databanks have been altered since the last backup). In cases where system failure destroys TRANSDB, reading the most recent contents of LOGDB can reveal whether there are any uncompleted transactions which must be corrected manually.

The READLOG functionality can only read the contents of the current LOGDB, not the contents of incremental backups.

The functionality allows information to be selected from LOGDB on the basis of time interval, ident performing the operation, and specified databanks or tables. This is particularly useful in production systems where LOGDB can contain a large number of entries.

7.2 Authorization

To list the log for selected tables or all tables in a databank, the user must have SELECT access on the tables in question.

To list the log for the whole database, the user must have EXECUTE privilege on the system program ident MIMER_BR.

7.3 Using the READLOG functionality

The READLOG functionality is controlled using a menu from which different listing options may be set before finally performing the read operation. The different listing options are set by using menu selections 1-5, 9 and 10. The menu is redisplayed after selecting any of these so that further options may be set for the listing.

When all the desired listing options have been set in this way, a listing is produced from the log by choosing menu selection 6, 7 or 8 from under "List operations".

List definitions	List restrictions	List operations	Change program id
-----	-----	-----	-----
1. Log list file	3. Time interval	6. Specified tables	9. Enter program
2. Log list width	4. Ident	7. Tables in databank	10. Leave program
	5. Databank	8. All (no data)	
0. EXIT			

7.3.1 List definitions (output control)

Log list file

Choosing this option allows the operator to specify the name of a sequential file into which the listing is to be placed. In systems where the terminal may be addressed by a logical file name, this may be given to display the listing on the terminal. If this option is not selected, a sequential file with the default name RDLOGL will be used.

Example:

```
Select: 1
Log list file: READLOG.DAT
```

Log list width

This option allows the width of the page to be set for the listing output. The default value is 80. The value given must lie between 72 and 132.

Example:

```
Select: 2
Log list width: 120
```

7.3.2 List restrictions

Time interval

This option allows the listing to be restricted to a given time interval, specified as a starting time and a finishing time. Times are given as a single parameter representing year, month, day, hour, minute and second in the format YYYYMMDDHHMMSS. If an incomplete time specification is given (truncated from the right), the remaining parameters are taken as low for the starting time and high for the finishing time. Thus giving 199804 as both the starting and finishing time, lists the log from the beginning to the end of April 1998.

Default values for starting and finishing times are the beginning and end of the log respectively. The default applies if no time interval is selected, or may be chosen for starting or finishing time by specifying a “blank” time. If the default is chosen for both starting and finishing times, the following message is displayed:

```
'** No time restriction'.
```

A selected time interval applies for all subsequent list operations in the current session until the time interval is reset. A time interval of two months has been selected in the following example:

```
Select: 3  
Format : YYYYMMDDHHMMSS  
Starttime: 199804  
Endtime : 199805
```

Ident

Selecting an ident restricts the listing to operations performed by that ident. Only one ident may be selected for a given listing.

The default setting lists operations performed by all idents. The default applies if no ident restriction is selected, or may be chosen by specifying a blank ident. If the default is chosen, the following message is displayed:

```
'** No ident restriction'.
```

A selected ident applies for all subsequent list operations in the current session until the ident is reset.

Example:

```
Select: 4  
Identname: HOTELADM
```

Databank

Selecting a databank restricts the listing to operations performed on that databank. This option must be specified if the list operation 7 (Tables in databank) is to be used. Only one databank may be selected for a given listing.

The default condition lists operations performed on all databanks. The default applies if no databank restriction is selected, or may be chosen by specifying a blank databank. If the default is chosen, the following message is displayed:

```
'** No databank restriction'
```

A selected databank applies for all subsequent list operations in the current session until the databank is reset.

Example:

```
Select: 5
Databank: HOTELDB
```

7.3.3 List operations

Specified tables

This option activates listing of the log for selected tables in the database. The operator may specify as many tables as required, with the table name qualified if necessary by the creator of the table. If no creator is given, the current ident is assumed.

Databank restrictions selected with option 5 are ignored if specified tables are selected. Ident and time restrictions selected with options 3 and 4 are however applied.

The ident running the READLOG functionality must have SELECT access on the requested tables, otherwise the message:

```
'** No select access on table'
```

is displayed for the table in question. If a non-existent table is requested, the message:

```
'** No such table'
```

is displayed. Errors of this type do not abort the listing if valid and invalid requests are mixed in the same operation.

The list operation is activated by giving a blank response to the prompt for a table name when all the required tables have been specified.

Example:

```
Select: 6
Table: HOTELADM.EMPLOYEE
Table: HOTELADM.STAFF
Table: HOTELADM.SALARY
Table:
```

Tables in databank

Operations on all tables in the databank specified under option 5 are listed. If no databank has been selected, the following message is displayed and the user must select a new option:

```
'** Databank not entered'.
```

Time or ident restrictions selected with options 3 or 4 are applied.

Data is listed only for those tables to which the ident running the READLOG functionality has SELECT access. Tables to which access is denied are indicated by the following message in the log list file:

```
'Table <creator.table-name> - No select access'
```

All (no data)

This option lists logged operations without details of data records (see below). The ident running the READLOG functionality must have EXECUTE privilege on the system program ident MIMER_BR. If the privilege is not held by the current ident the following error message is displayed:

```
'** AUTHORIZATION FAILURE'.
```

7.3.4 Change program id

Enter program ident

If a program ident is the creator of a databank, that ident may be entered to read the log for that databank.

Leave program ident

This option leaves the entered program ident.

7.4 Output format

The output from the READLOG functionality is divided into transactions, showing the date and time, the ident performing the transaction (with entered program idents where appropriate) and the number of database records read during the transaction. Note that the output does not contain statements for reconstructing the logged operations - it is simply a documentary record of the transactions performed on the database.

If list operations 6 or 7 (select by table or databank) are selected, the contents of the affected rows in the table are displayed. Insert and delete operations are listed as a single row. Update operations are recorded as the state of the row before and after the update.

If the list operation 8 is selected (no data), the operations are listed without the data records.

The following example uses the READLOG functionality to list all the transactions that have been committed between 1998-05-23 10:55 and 1998-05-23 12:00 and shows extracts from the output for All (no data) and for table BOOK_GUEST (user input is shown in bold):

```

-- Read log --

List definitions      List restrictions    List operations      Change program id
-----
1. Log list file     3. Time interval     6. Specified tables  9. Enter program
2. Log list width    4. Ident             7. Tables in databank 10. Leave program
0. EXIT              5. Databank          8. All (no data)

Select: 1
Log list file: LOGFILE

-- Read log --

List definitions      List restrictions    List operations      Change program id
-----
...

Select: 3
Format   : YYYYMMDDHHMMSS
Starttime: 199805231055
Endtime  : 199805231200

-- Read log --

List definitions      List restrictions    List operations      Change program id
-----
...

Select: 8

```

Extract from output All (no data):

```

Transno Function
-----
106 Update before in HOTELADM.FREEROOMS
106 Update after in HOTELADM.FREEROOMS
106 Delete from HOTELADM.MAINTENANCE
106 Delete from foreign key
106 Commit 1998-05-23 11:06:59:74 by HOTELADM/CHARLIE Read 6
-----
107 Insert into HOTELADM.CHARGES
107 Update before in HOTELADM.BOOK_GUEST
107 Update after in HOTELADM.BOOK_GUEST
107 Commit 1998-05-23 11:08:00:45 by HOTELADM/GEORGE Read 5
-----

-- Read log --

List definitions List restrictions List operations Change program id
-----
1. Log list file 3. Time interval 6. Specified tables 9. Enter program
2. Log list width 4. Ident 7. Tables in databank 10. Leave program
0. EXIT 5. Databank 8. All (no data)

Select: 6
Table: BOOK_GUEST
Table:

```

Extract from output for table BOOK_GUEST:

Table HOTELADM.BOOK_GUEST

```

Transno Function RESERVATION, BOOKING_DATE, HOTELCODE, ROOMTYPE,
RESERVED_BY, TELEPHONE, RESERVED_FOR, ARRIVE, LEAVE,
GUEST, ADDRESS, CHECKIN, CHECKOUT, ROOMNO, PAYMENT
-----
92 Insert 1382, 19980523, 'WINS', 'SGLB', 'JULIO PEREZ',
'6-6549868', 'JULIO SANCHEZ', 19980528, 19980531,
'JULIO SANCHEZ', 'CARLOTA 7, MADRID, SPAIN', <NULL>,
<NULL>, 'WINS301', <NULL>
92 Commit 1998-05-23 10:57:56:49 by HOTELMANAGER Read 3
-----
99 Delete 1353, 19980516, 'SKY', 'DBLB', 'HOT SERVICE LIMITED',
'08-135709', 'BASIL FAWCETT', 19980602, 19980612,
'ALFRED SMITH', '23 BACK NELLY VIEW, ACKWORTH',
<NULL>, <NULL>, 'SKY124', 'CASH'
99 Commit 1998-05-23 10:58:59:74 by HOTELADM/RICHARD Read 1
-----
107 Update before 1348, 19980505, 'SKY', 'SGLS', 'SYSDECO MIMER AB',
'018-185000', 'BENGT PETTERSON', 19980520, 19980523,
'BENGT PETTERSSON', 'MIMERGATAN 64, UPPSALA',
19980520, <NULL>, 'SKY101', 'EUROCARD'
107 Update after 1348, 19980505, 'SKY', 'SGLS', 'SYSDECO MIMER AB',
'018-185000', 'BENGT PETTERSON', 19980520, 19980523,
'BENGT PETTERSSON', 'MIMERGATAN 64, UPPSALA',
19980520, 19980523, 'SKY101', 'EUROCARD'
107 Commit 1998-05-23 11:08:00:45 by HOTELADM/GEORGE Read 5
-----

```


8 DATABASE STATISTICS

8.1 Statistical information

The SQL statistics statements collect statistical information about table and index data in the database and store this information in the data dictionary. The information is used by the SQL compiler in optimizing access paths for SQL data manipulation statements.

The statistics functionality in the UTIL program, provided in previous versions of MIMER to provide a mechanism for statistics collection, is still supported for backward compatibility.

The statistical information includes:

- the total number of rows in each base table, stored in the column CARD in the table's row in MIMER.TABLEX
- the number of distinct values in each column of a table, stored in the column COLCARD in the column's row in MIMER.COLUMN
- the number of non-NULL values in each column of a table, stored in the column COLCNOTN in the column's row in MIMER.COLUMN
- the lowest and highest values in each column of a table, stored in the columns LOW and HIGH in the column's row in MIMER.COLUMN
- the date and time for the last time statistics were gathered for a table is stored in MIMER.OBJECT with the type "TABSTAT". This can be read by the view MIMER.OBJECTS.

Statistics may be collected for the entire database (i.e. all tables in all databanks recorded in the same SYSDB), for tables owned by specified idents, or for specific tables.

8.2 Authorization

The user executing the SQL statistics statements must either have EXECUTE privilege on the system program ident MIMER_SC or be the owner of the table(s) or ident(s) for which statistics are being collected. The database administration ident SYSADM holds EXECUTE privilege on MIMER_SC with the WITH GRANT OPTION, and may thus take responsibility for maintaining statistics for the whole system or delegate the responsibility to selected idents.

Note: A user with statistics privilege (i.e. EXECUTE privilege on MIMER_SC) is not necessarily permitted to read the contents of the databank using data manipulation statements, this privilege only permits access for the collection of statistics.

8.3 The SQL statistics statements

The SQL statistics statements may be used to collect statistical information in the areas described below. Also refer to the *MIMER/SQL Reference Manual* for details on the SQL statistics statements.

8.3.1 Statistics for the entire database

To collect statistical data for all tables in the database, use the following function:

```
UPDATE STATISTICS
```

The user must have EXECUTE privilege on MIMER_SC.

Even in a database of only moderate size, collecting statistical data for all tables is time-consuming. It is recommended that this option in particular is run at off-peak working times, and preferably in batch mode.

8.3.2 Statistics for specified idents

To collect statistics for all base tables owned by a list of specified idents, use the following function:

```
UPDATE STATISTICS FOR IDENT list-of-idents
```

A user requesting statistics for tables owned by an ident other than himself must have EXECUTE privilege on MIMER_SC.

To collect statistics for SYSDB, the pseudo-ident MIMER may be specified.

8.3.3 Statistics for specified tables

To collect statistics for a list of specified tables, use the following function:

```
UPDATE STATISTICS FOR TABLE list-of-tables
```

The user requesting statistics for the tables specified in the list must either be the owner of them or have EXECUTE privilege on MIMER_SC.

8.4 When to use the SQL statistics statements

The SQL statistics statements should be used whenever optimizing performance is an issue of significant importance.

MIMER collects approximate statistics for each table whenever the table is opened. These statistics may suffice for maintaining high performance in many situations. If optimal performance is required for an application, the SQL statistics statements should be used to supplement and refine the automatically collected information.

When this is the case, statistics should be typically updated in the following situations:

- when the size of a table has altered
- when the maximum/minimum limits on values in a table have altered significantly.

The statistics information in the data dictionary is used only by the MIMER/SQL compiler.

Note that only the performance, not the result, of an SQL statement is affected by the statistical information.

9 DATABANK CHECK FUNCTIONALITY

9.1 Runtime malfunctions

Some database server errors are recorded in the database server log. In addition, some operating systems maintain a log file of all errors and other system events. This file, or files, should be investigated in the event of unexpected system interrupts. Contact your MIMER representative if the errors appear to reflect a problem with MIMER.

9.2 The DBC program

9.2.1 General description

The DBC program allows investigation of databank files to ensure that the physical structure is not damaged. The functionality may also be used to examine the organization of databank files and display statistical information on file usage.

Note: The functionality checks only the **physical** condition of the databank. Informational errors such as data inconsistency, invalid data format, and data outside the validation limits of domains are not detected by DBC.

Before performing a backup, it is recommended that DBC is used to check the validity of the databanks to be backed-up. If errors are detected, the databank can be restored from the previous backup copy. This practice is particularly recommended for databanks which are not used intensively in applications, where the chance of detecting errors during normal use is relatively low.

Note: The only satisfactory way to reconstruct a databank damaged by physical storage errors is to perform a databank restore operation from a backup copy (except when MIMER Shadowing is used). **Direct editing of the databank file must never be attempted.**

DBC produces detailed diagnostic information when databank errors are detected. The diagnostic output should always be included with any information you send to MIMER support regarding databank errors.

It should be noted that the databank check program DBC returns an error status to the operating system when an error is encountered. This may be useful when running it from scripts or in batch mode.

9.2.2 Authorization

The DBC program operates directly against the databank file, with no reference to the MIMER database manager. The program may not be run on a file which is currently held open by another user (an error message is displayed in such a case). The system administrator should arrange for exclusive access to the databank file during DBC operations.

9.2.3 User communication

The overall syntax for the DBC program is:

```
dbc [databank_filename [result_filename] ]
```

If the filenames are not specified on the command-line, the program prompts for the name of the databank file and a name for the result file, which is a sequential file created by the DBC program and used for output. If no result filename is specified, output is written to the screen. Default filename extensions are .DBF for the databank file and .DBC for the result file.

If an error occurs when opening the databank file (e.g. file not found or file locked by another user), or while creating the result file, an appropriate error message is displayed.

If no errors are detected in the databank file, the message:

```
'No errors found'
```

is displayed. The result file then contains statistics describing the physical databank organization. Otherwise, error descriptions (see below) are written to the result file, and the message:

```
'* Errors logged in result file'
```

is shown. The result file should be examined to investigate the nature of the errors.

The following message:

```
'* Too many bitmap pages'
```

when displayed during the DBC operation, does not reflect an error in the databank file. Instead, it indicates that a complete bitmap check has not been performed, due to a restriction imposed on the program during installation at your site. Contact your MIMER agent if this situation arises.

9.2.4 Result file contents

The result file contains the databank file name and the time when the DBC operation was performed. This is followed by header information read from the databank, such as the version number under which the databank was created. If this is not the same as the current version there will also be a “converted to” message. Also shown are the backup timestamp, the structure level, the system identification number of the databank, the page number for the bitmap pages (starting at 0) and for the root pages (starting at 1), and the total number of allocated and used MIMER pages. If the databank file was created as a shadow copy the sequence number is also given.

Note: If the databank was not properly closed when MIMER was stopped, there may be an additional message saying:

```
'* No bitmap checking against databank!'
```

An identification record is given for each table in the databank. The following information is shown:

Tabid	The system identification number of the table. This is the number used to identify the table in the data dictionary. The table name is not stored directly in the databank file.
Startp	The page number of the start page for the highest index level. If the number of levels is 1, this is the only data page. If the start page is 0, the table is empty.
Levels	The number of levels in the table storage structure.
Keylen	The length of the primary key in bytes.
Reclen	The record length (row length) of the table.
Type of table	“Base table”, “Base table (VL)”, “Secondary index table” or “Secondary index table (VL)”. (VL) indicates that data pages have variable-length records (i.e. are compressed).
Status of table	“Resident” or “Marked for delete”.
Index page size	Size of the index page in bytes.
Data page size	Size of the data page in bytes.
Number of index pages	Indicates how many index pages were checked.
Number of data pages	Indicates how many data pages were checked.
Required/Allocated datapages	Gives an approximate measure of the space used in the datapages. This is expressed as a percentage, which may be >100% for data pages having variable-length (i.e. compressed) records because the required number of pages is calculated based on uncompressed record sizes.

Reached no of records If no errors are reported, the number of records reached is equal to the total number of records (rows) stored for the table.

For LOGDB, TRANSDB and SQLDB the identification record contains the following information:

Tabid	Ordered identification.
Startpage	First page in the sequential file.
Endpage	Last page in the sequential file.
Type of table	“Sequential table” or “Sequential table (VL)”. (VL) indicates that pages have variable-length records (i.e. are compressed).
Status of table	“Resident” or “Marked for delete”.
Page size	Size of the page in bytes.
No. of pages read	Number of pages checked.
No. of records read	Number of records checked.

Any errors detected in the databank file are written to the result file directly after the identification record for the table affected. Tables with no error indications can be read by MIMER, and copied to sequential files using the UNLOAD function in BSQL. In this way information from intact tables in a damaged databank file can be saved, providing partial security in case the backup routines for the databank are inadequate.

9.2.5 Example of result file

The following example illustrates the result of running DBC on the HOTELDB databank file. No errors were detected.

```

MIMER Databank Check Utility
  Version 8.1.1

Databank file: HOTELDB
Time: 1998-06-23 16:29:09

Version created:      811  Backup timestamp:    1
Structure level:     6   System ident.:       269

Bitmap pages:        0

Root  pages:         1

No. of pages allocated: 160          No. of pages used:      115

Tabid:      270  Startp:      3  Levels:  2  Keylen:  4  Reclen:  25
Base table, Resident
Index page size          2048      Data page size          2048
Number of index pages:   1         Number of data pages:   4
Required/Allocated datapages: 100%  Reached no of records: 255

Tabid:      271  Startp:      5  Levels:  2  Keylen:  8  Reclen:  12
Base table, Resident
Index page size          2048      Data page size          2048
Number of index pages:   1         Number of data pages:   6
Required/Allocated datapages: 83%   Reached no of records:  700

Tabid:      272  Startp:     15  Levels:  2  Keylen:  8  Reclen:  89
Base table (VL), Resident
Index page size          2048      Data page size          2048
Number of index pages:   1         Number of data pages:   92
Required/Allocated datapages: 183%  Reached no of records: 3707

Tabid:      273  Startp:    108  Levels:  2  Keylen:  8  Reclen:  12
Base table, Resident
Index page size          2048      Data page size          2048
Number of index pages:   1         Number of data pages:   2
Required/Allocated datapages: 100%  Reached no of records:  222

Tabid:      274  Startp:      4  Levels:  1  Keylen:  4  Reclen:   8
Base table, Resident
Index page size          2048      Data page size          2048
Number of index pages:   0         Number of data pages:   1
Required/Allocated datapages: 100%  Reached no of records:  166

Tabid:      275  Startp:    112  Levels:  2  Keylen:  4  Reclen:  89
Base table (VL), Resident
Index page size          2048      Data page size          2048
Number of index pages:   1         Number of data pages:   3
Required/Allocated datapages: 266%  Reached no of records:  166

```

9.2.6 Error messages

The following error situations are described by explicit messages in the result file:

Bitmap errors

- * Illegal number of free bits in bitmap

The number of free bits in the bitmap is marked as a negative number or as a number greater than the permissible value.

- * Illegal pointer to first word with free bit in bitmap

The pointer to the first word is either negative or greater than the number of words per page, or points outside the number of allocated pages.

Root page errors

- * Illegal record length in root page

The record length is not valid for the current version or for the site.

- * Pageno. outside databank: x

- * Pageno. not marked as used: x

The reference to the page number (x) is invalid. (Applies only where there is more than one root page.)

Sequential structure errors

- * Pointer to previous page invalid

Error in page linkage.

- * Invalid record length

The record length is either not the same as in the previous page or outside the page limits.

- * Record crosses page boundary

A record stretches over the page limits.

Table structure errors

- * Error in root record

The value for start page, levels, key length or record length is outside the legal values for the site.

- * Page has illegal record length: x

The record length (x) given in the page is not the same as that given for the table.

- * Page has illegal last-record pointer: x

The pointer (x) to data within a page is outside the page limits. The limits are the values of bytes/header and bytes/page.

- * Page has records in wrong order. Pos: x

The records in the page are not correctly sorted. The value of x is the byte position within the page for the start of the wrong record.

- * Page has last-record key > index key. Pos: x

The records within a page include key values greater than those in the index level above. The value of x is the byte position within the page for the start of the last record.

- * Page no. outside databank: x
Reference is made to a page number (x) which is higher than the highest allocated page.
- * Page no. referenced twice
There are at least two references to the same page number (x). One of these references should also give another error, or an error should have been notified earlier in the file.
- * Page no. not marked as used: x
Bitmap pageno: Word: Bit:
A page that is used is illegally marked as free. Continued insertion of data in the databank will result in a double referenced page.

All table structure errors are followed by a listing of the page numbers passed in the B*-tree structure on the way to the error:

B*-tree

Page no: x1 x2 x3 xn

Byte pos: y1 y2 y3 yn

where yn indicates the byte position in the page xn where the record holding the reference to the next level starts.

If the error occurs at an index level, the additional message:

```
'Branch is interrupted'
```

is given, and no checks are made at lower levels.

10 DBOPEN FUNCTIONALITY

The DBOpen functionality is used to make sure that the users can open all databanks quickly. It can take a long time to open a databank if it is a large databank that was closed abnormally (for example if the machine crashed).

The DBOpen functionality should normally be run as soon as the database server has been started.

The overall syntax for DBOpen is:

```
dbopen [-s | -m] [database]
```

DBOPEN command-line arguments

Parameter	Function
-m	Connects to the database in multi-user mode.
-s	Connects to the database in single-user mode.
database	Specifies the name of the database to access.

If the optional database name is not specified, the default database will be accessed (see Section 3.7.2).

If neither `-s` or `-m` is specified for the optional mode flag, the way the database is accessed will be determined by the setting of the `MIMER_MODE` variable (see Section A.2) or, if this is not set, it will be accessed in multi-user mode.

Refer to the *MIMER Guides* for your platform for any additional information that may be required in order to enable the use of these command-line arguments.

It should be noted that the DBOpen program returns an error status to the operating system when an error is encountered. This may be useful when running it from scripts or in batch mode.

10.1 Functions

The DBOPEN functionality opens all available databanks in a system. Each opened databank is closed before the next one is opened. As each databank is opened, the integrity is checked and any transactions that were interrupted by the abnormal close are completed. The integrity check is analogous to running DBC and it is automatically performed when opening a databank that has not been closed normally.

The checks performed when an abnormally closed databank is opened may take some time, particularly if the databank is large. Running DBOPEN means that the checks are performed at a time determined by the system administrator, rather than a time determined by application programs. As a result, users will always have fast access to all databanks.

The databanks are opened in a randomly determined order. Running several DBOPEN sessions in parallel may speed up the checking process for the database as a whole.

10.2 Authorization

Any user who has access to the data dictionary table MIMER.DATABANK can run DBOPEN.

10.3 DBOPEN output example

The following example output is from a MIMER system where two databank files were deleted before the database server was started.

```
MIMER Databank Open Utility

Username: SYSADM
Password:

Opening databank SYSMMSG
Opening databank SYSQQL
Opening databank SYSRGG
Opening databank SYSRGGOUT
Opening databank SYSQF
Opening databank SYSPG
Opening databank CMD
Opening databank SYSSH
Opening databank SYSFM
Opening databank SYSHELP
Opening databank WORKDB
Opening databank HOTELDB
Opening databank BOOKDB
Opening databank ROOMSDB
Opening databank WORKDB
Opening databank L2DB3

MIMER/DB fatal error -16142
      Cannot open databank L2DB3,
      file L2DB3 not found

Opening databank L2DB1

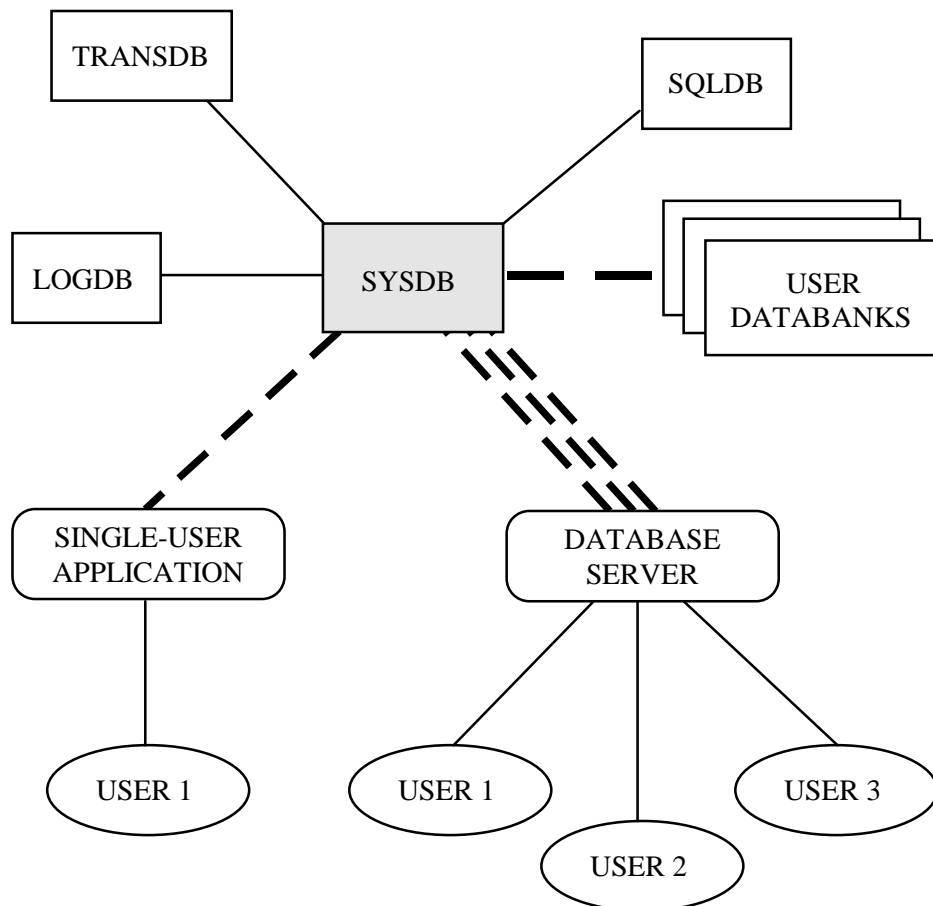
MIMER/DB fatal error -16142
      Cannot open databank L2DB1,
      file L2DB1 not found
Opening databank RECORDS

      2 databanks were not possible to open
```


A EXECUTING IN SINGLE-USER MODE

Normally users access the database via a database server (multi-user mode), but in some situations it may be required that the use of a database be restricted to a single user. Any local database can be opened in single-user mode, provided there is no database server currently running against the database.

Note: An application started in single-user mode will access the databank files directly from within its own process. This means that the operating system user who is running the application must have access, at the operating system level, to all the existing databank files. All new databank files created in single-user mode will be owned by that user.



Single- and multi-user access to a MIMER database .

A.1 File protection in single- and multi-user mode

If a database that has been created in single-user mode is to be used by a database server, or vice versa, certain precautions must be observed with regard to the databank files:

- Files created in single-user mode must be accessible for read and write by a database server if the database is to be subsequently used in multi-user mode, since databank file access is performed by the database server process. The creator of the files should change the protection if necessary.
- Conversely, files created by a database server must be accessible for read and write by the individual operating system users who need to access the database in single-user mode. Since the databank files are created by the database server process, the protection on these files can only be changed by an operating system user who has privileges which are equivalent to those of the database server process.

A.2 Specifying single-user mode access

If the database is to be accessed in single-user mode by default, the environmental variable or logical name called `MIMER_MODE` should be defined as “`SINGLE`”, as shown in the examples that follow.

If `MIMER_MODE` is not set or is set to “`MULTI`”, the database will be accessed in multi-user mode by default.

If `MIMER_MODE` is set to `SINGLE` and the default database (see Section 3.7.2) is set to point to a local database, the database will be opened in single-user mode. (Remote databases will be accessible through client/server interfaces as normal).

Note: Many of the programs which are part of the MIMER distribution support the command-line flags “`-s`” and “`-m`” which control whether they access a database in single-user or multi-user mode (see Section 3.9).

A.3 Accessing in single-user mode

MIMER applications that connect to a local database server when single-user access is indicated, will dynamically include a shared library when activated. This library holds all functionality that normally is provided by the database server program.

Example

The following example session first connects to the INVENTORY database in single-user mode and then connects to the STAFF database, administered by a running database server process:

Unix

```
$ MIMER_DATABASE=INVENTORY
$ MIMER_MODE=SINGLE
$ export MIMER_DATABASE MIMER_MODE
$ bsql
.
.
SQL> exit;
$ unset MIMER_MODE
$ MIMER_DATABASE=STAFF
$ export MIMER_DATABASE
$ bsql
.
.
SQL> exit;
```

VMS

```
$ DEFINE MIMER_DATABASE INVENTORY
$ DEFINE MIMER_MODE SINGLE
$ RUN MIMEXE8:BSQL
.
.
SQL> exit;
$ DEASSIGN MIMER_MODE
$ DEFINE MIMER_DATABASE STAFF
$ RUN MIMEXE8:BSQL
.
.
SQL> exit;
```

Win

```
C:\> SET MIMER_DATABASE=INVENTORY
C:\> SET MIMER_MODE=SINGLE
C:\> BSQL
.
.
SQL> exit;
C:\> SET MIMER_MODE=
C:\> SET MIMER_DATABASE=STAFF
C:\> BSQL
.
.
SQL> exit;
```




A.4 The SINGLEDEFS parameter file

The use of a SINGLEDEFS parameter file is optional (and is not supported the Windows platform).

When a single-user mode connection is established, the SQLPOOL and BUFFERPOOL data areas are dynamically created.

The initial size of the SQLPOOL in single-user mode is 200 Kbytes. The SQLPOOL area will grow dynamically if more space is needed (see Section 4.1.1.5).

Normally, the size of the BUFFERPOOL area is determined by parameters in the MULTIDEFS file. If this size is acceptable, no SINGLEDEFS file will need to be created (see Section 4.1.1.3 for more details on the BUFFERPOOL).

To change the size of the BUFFERPOOL in single-user mode a SINGLEDEFS file, similar to the MULTIDEFS file, should be created in the database home directory. A template of this file, showing the default values for the relevant parameters, can be found in the examples directory:

```
--
-- Parameters for single-user system
--
Databanks      100      -- (40-1000)      Max number of databanks
Tables         4000     -- (500-1000000)  Max number of tables
Pages2K        128     -- (22-1000000)   Size of 2k bufferpool region (pages)
Pages16K       32      -- (22-1000000)   Size of 16k bufferpool region (pages)
Pages64K       32      -- (22-1000000)   Size of 64k bufferpool region (pages)
```

Note: When changing parameters in the SINGLEDEFS file, always change the copy in the database home directory. Never change the template file in the examples directory.

B THE SQLHOSTS FILE ON VMS AND UNIX

This appendix applies to the VMS and Unix platforms only.

It describes the SQLHOSTS file which is used to list all the databases that are accessible to a MIMER application from the node on which it resides. For general information on how to make databases accessible, refer to Section 3.2.

Unix

On a Unix node, the pathname of SQLHOSTS file is `/etc/sqlhosts`. The program called **mimhosts** can be used to manage the contents of the SQLHOSTS file instead of editing it manually. When the `dbinstall` program is used to install a local database on a Unix node, an entry for it is automatically added to the LOCAL section (see Section B.1.2) of the SQLHOSTS file on that node. If the file is not found, a default SQLHOSTS file is automatically generated.

VMS

On a VMS node, the SQLHOSTS file can have any name and is located by translating the logical name `MIMER_SQLHOSTS`. The `MIMSETUP8` command defines this logical name to be `SYSS$SPECIFIC:[SYSMGR]SQLHOSTS.DAT`. A default SQLHOSTS file is generated by the installation of the MIMER software.

B.1 The SQLHOSTS file

A line of text beginning with the character sequence “--” is interpreted as a comment in the SQLHOSTS file.

The SQLHOSTS file contains three sections, called DEFAULT, LOCAL and REMOTE. The names of the local databases on the current node are listed in the LOCAL section (see Section B.1.2) and the names of the remote databases accessible from the node are listed in the REMOTE section (see Section B.1.3).

One of the local or remote databases can be set to be the default database for the node by specifying its name in the DEFAULT section (see Section B.1.1).

Each database listed in the SQLHOSTS file must have a unique name.

Database names may, in general, be up to 64 characters long and are case-insensitive.

VMS

The maximum length for the name of a database on a VMS node is 30 characters.

When the MIMER system is installed on a node, the following default SQLHOSTS file is automatically generated:

```

-----
--
--  S Q L H O S T S
--  =====
--
--  This file contains a list of all databases, local and remote, accessible
--  from the node where the file resides.
--
--  The DEFAULT label
--  -----
--  Name of default database. Can be either a REMOTE or LOCAL database name.
--  Can be overridden by setting MIMER_DATABASE to the name of a database.
--
--  The LOCAL label
--  -----
--  A list of all local databases on the current node, containing the
--  database name and a directory specification (Path).
--
--  Path -      database home, and directory path for databank lookup.
-----

--  Path -      database home.
-----

--
--  The REMOTE label
--  -----
--  A list of all remote databases containing the database name, the database
--  node, the protocol to be used, the protocol interface and the protocol
--  service to be used.
--
--  Protocol, Interface and Service may be defaulted by entering ''.
--
--  Node -      network node name for computer on which the database resides.
--
--  Protocol -  currently tcp is supported. (tcp or '' should be specified)
-----

--  Protocol -  currently tcp and decnet are supported. The default is tcp.
-----

--  Interface - currently not used ('' should be specified).
--  Service -   corresponds to the port number used in TCP/IP. The port number
--             Default is 1360, i.e. the port number reserved for MIMER.
--
--             (The port number may either be a number or a name of a service
--             stored in the /etc/services file).
-----

--             (If using DECNET, the service fields corresponds to a NCP object
--             or command procedure used by the network server.
--             Default is the database name.)
-----

--
--  =====
DEFAULT:
--
--  Database
--  -----
--  SINGLE
--  =====
LOCAL:
--
--  Database      Path
--  -----
--  SINGLE      .
--  =====
REMOTE:
--
--  Database      Node      Protocol Interface Service
--  -----
--  example_database  server_nodename  ''      ''      1360

```

Unix

VMS

Unix

VMS

Unix

VMS

B.1.1 DEFAULT section

The DEFAULT section contains a single line that specifies the default database which will be used by an application that does not explicitly specify a database to connect to (see Section 3.7.2).

The default database can be one of those listed in the LOCAL or REMOTE sections.

B.1.2 LOCAL section

The LOCAL section contains a list of all the local databases residing on the current machine (see Section 3.3).

Each line under the “LOCAL” keyword should contain two fields, separated by one or more blanks or tab characters. The first field specifies the database name.

Unix

On a Unix node, the second field may be a colon (“:”) separated search path specification. The first directory in the search path is taken as the database home directory and the other directories in the search path will be used to locate databank files which have a file specification stored in the data dictionary without an explicit directory.

VMS

On a VMS node, the second field specifies a directory which will be the home directory for the database.

The MIMER system databank SYSDB will be located in the database home directory and other databanks will typically be located relative to it (see Section 2.3.3).

B.1.3 REMOTE section

The REMOTE section contains a list of all accessible databases that reside on other nodes in the network environment (see Section 3.4).

Access to these databases is provided by using either DECnet or TCP/IP to establish a client/server connection to the remote machine.

Each entry in the REMOTE section contains up to five fields, separated by spaces and/or tab characters.

The **DATABASE** field specifies the name of the remote database.

The **NODE** field should specify the network node name of the remote machine. If the TCP/IP interface is used, the IP address may be specified here.

Unix

The **PROTOCOL** field should specify “TCP” or " (two single quote characters).

VMS

The **PROTOCOL** field may specify “DECNET” or “TCP” depending on the type of network protocol that should be used to create the client/server connection. The default, specified by " (two single quote characters), is TCP.

The **INTERFACE** field is currently not used. Specify " (two single quote characters) here.

If using TCP/IP, the **SERVICE** field specifies the TCP/IP port number the database server uses. The default is 1360, which has been reserved by Sysdeco Mimer AB for MIMER client/server communication.

Unix

When TCP/IP is used under Unix, the port number in the **SERVICE** field may be a number or the name of a service stored in the /etc/services file.

VMS

For a MIMER8 database server using DECnet, the **SERVICE** field should contain the database name, which is also the default. The server listens to the network object using the same name as the database. (A MIMER7 database server using DECnet listens to the network object named "MIMER").

The remote section parameters are summarized below, depending on the protocol selected, (the character sequence " is two single quotes and specifies the default value for a parameter):

TCP

DATABASE *remote_database_name*
NODE *TCP/IP_node_name* or *IP_number*
PROTOCOL "
INTERFACE "
SERVICE *TCP/IP_port_number* or *TCP/IP_service_name* or "
 (Note: When " is used to specify the default SERVICE, the TCP/IP port number 1360 will be used.)

VMS
DECNET

DATABASE *remote_database_name*
NODE *decnet_node_name*
PROTOCOL **DECNET**
INTERFACE "
SERVICE *decnet_network_object* or "
 (Note: When " is used to specify the default SERVICE, the value of *remote_database_name* will be used for MIMER version 8 and the value "MIMER" will be used for MIMER version 7.)

C THE MULTIDEFS FILE ON VMS AND UNIX

This appendix applies to the VMS and Unix platforms only.

It describes the MULTIDEFS file which forms part of a local database definition (see Section 3.3) for a database residing on a VMS or Unix node.

This file contains operational parameters for the database server for such a database and these are read when the database server is started. It is not possible to change the parameters for a running database server.

C.1 The MULTIDEFS parameter file

Comments in MULTIDEFS are introduced by the character sequence "--", or by the character "!" or "#".

If the MULTIDEFS file is not found when starting a database server, the MIMCONTROL command will create a new file and fill it with the default values for all parameters.

Unix

On a Unix node, the MULTIDEFS file is located in the database home directory and is called **multidefs**.

VMS

On a VMS node, the MULTIDEFS file is located in the database home directory and is called **MULTIDEFS.DAT**.

The actual default values used may vary and may depend on factors like machine type and the amount of physical memory available on the machine.

The following is an example of the MULTIDEFS parameter file which may be generated by MIMCONTROL:

```
-- MIMER version 8.1.1 parameters generated 1998-07-07 16:38
Databanks      100          # 40-1000      Max # of databanks
Tables         4000        # 500-1000000 Max # of tables
ActTrans       4000        # 500-1000000 Max # of active trans
SQLPool        400         # 10-2000000  Initial SQLPool size (kb)
RequestThreads 5           # 1-100       # of request threads
BackgroundThreads 2       # 1-100       # of background threads
Users          10         # 1-5000      Max # of logged in users
DBCheck        1         # 0-1         DB check (0=index, 1=full)
Pages2K        5779       # 22-1000000  # of 2K bufferpool pages
Pages16K       409        # 22-1000000  # of 16K bufferpool pages
Pages64K       91         # 22-1000000  # of 64K bufferpool pages
```

Unix

Oper		#	Receivers for messages
DumpPath	.	#	Path for dump directory
TCPPort	inted	#	TCP/IP port

VMS

Oper	OPER	#	Receivers for messages
DumpPath	<>	#	Path for dump directory
TCPPort	1360	#	TCP/IP port

VMS

MaxSQLPool	30000	#	10-2000000	Max size of SQLPool (kb)
MemLock	1	#	0-1	Lock bpool in memory?

DECPort	example_database	#	Decnet network object	
ProcName	example_database	#	Process name prefix	
NetUsers	5000	#	1-5000	Max # of network users
ServPrio	5	#	0-16	VMS prio for server process
Cleanup	60	#	10-10000	Cleanup interval (seconds)
WSExtent	70000	#		WSEXTENT for server process

C.1.1 MULTIDEFS parameters

Databanks Specifies the maximum number of databank files that the database server can have open at any one time.

Tables Specifies the maximum number of tables that can be accessed simultaneously by the database server.

ActTrans Specifies the maximum number of transactions that can be active in the database server.

SQLPool Initial size of the SQLpool area in Kbytes. This area contains information about each session, i.e. opened tables and databanks, compiled SQL programs, etc. The SQLpool area will expand automatically if it is too small, but it will not be larger than *MaxSQLPool*.

RequestThreads The number of threads in the database server that can serve client requests.

BackgroundThreads The number of background threads in the database server.

<i>Users</i>	The maximum number of users that are allowed to connect to the database server. This parameter should not exceed the number of users specified in the MRS key. This number is also used to calculate the size of the shared memory region used for local database server communication. About 70 Kbytes of shared memory will be allocated for each user.
<i>DBCcheck</i>	A number which specifies whether a full check (1) or only an index page check (0) should be performed when a databank is opened which previously was not closed properly. A full databank check provides for more secure operations, but may take much longer to execute than an index page check. Databank checks can be avoided by always shutting down the database server properly with the MIMCONTROL command, especially prior to shutting down the machine.
<i>Pages2K</i>	The number of 2 Kbytes pages in the bufferpool area containing pages from the databank files. The default value of this parameter is 12.5% of the total RAM memory in the machine.
VMS	(There may be a VMS limit set for the amount of memory a process may allocate, this limit will not be exceeded. Among the various VMS parameters, WSMAX is likely to be of primary interest in connection with this limit).
<i>Pages16K</i>	The number of 16 Kbytes pages in the bufferpool area containing pages from the databank files. The default value of this parameter is 8.33% of the total RAM memory in the machine.
<i>Pages64K</i>	The number of 64 Kbytes pages in the bufferpool area containing pages from the databank files. The default value of this parameter is 5% of the total RAM memory in the machine.
<i>Oper</i>	This parameter gives a list of host system users that should receive notification of serious problems with the database server.
VMS	On VMS, you can also specify "OPER". This will enable that notification messages are sent to the "central operator", i.e. processes that have set \$REPLY/ENABLE=CENTRAL.
<i>DumpPath</i>	This parameter may specify an alternate path for the dump directories. The default is to create dump directories under the database home directory.

TCPPort

Specifies how the database server should handle incoming TCP/IP connection requests. If this parameter is set to “-” (a single dash), the TCP/IP capability will be disabled for the database server.

Unix

The *TCPPort* parameter is, by default, set to “inetd” which means that the TCP/IP port server program, *mimtcp*, will be used for establishing a connection to any MIMER version 8 database server. In this case clients may connect to the port to which *mimtcp* listens, usually 1360, and the handshake will be passed over to the requested MIMER database server.

If a TCP/IP port number is specified, the database server will listen directly to that port.

VMS

The *TCPPort* parameter is, by default, set to the TCP/IP port number “1360”. The TCP/IP port server program, *MIMTCP*, will automatically be started to listen to the given port, serving all MIMER version 8 database servers set up to use that port.

MaxSQLPool

The maximum size (in kilobytes) of the SQLpool. The SQLpool memory area grows dynamically, but the size will never exceed this parameter. Use this parameter to control the maximum virtual size (maximum page file usage) for the database server process.

MemLock

A number which specifies whether the bufferpool and communication buffers should be locked in memory (1) or not locked in memory (0).

VMS*DECPort*

Specifies the DECNET network object that the database server listens to. Each database server must use a unique network object.

The default value is the database name.

If you set this parameter to “-” (a single dash), the DECNET capability will be disabled for the database server.

VMS*ProcName*

This parameter specifies the process name prefix for the database server. (The last part of the process name is always “Srv”).

Specify a maximum of 11 characters. The default value is to use the first 11 characters of the database name.

VMS	<i>NetUsers</i>	<p>This parameter specifies the number of users who can access the database through a network connection.</p> <p>The value used by the system will be the minimum of this parameter and the <i>Users</i> parameter.</p> <p>The default value is 5000. Since the <i>Users</i> parameter can not be larger than 5000, this means that all users may be network users.</p>
VMS	<i>ServPrio</i>	<p>This parameter specifies the VMS priority for the database server process.</p>
VMS	<i>Cleanup</i>	<p>Specifies the time (in seconds) between the cleanup sweeps that check for terminated database clients.</p>
VMS	<i>WSExtent</i>	<p><i>WSExtent</i> specifies the maximum amount of physical memory that the database server may use, expressed in VMS pagelets (512 bytes). The database server has a <i>WSQUOTA</i> that is calculated by <i>MIMCONTROL</i> to include the bufferpool, communication areas and some extra storage. The amount of physical memory used by the database server varies dynamically between <i>WSQUOTA</i> and <i>WSEXTENT</i> depending on global system load.</p> <p>The default value for this parameter is the <i>sysgen</i> parameter <i>WSMAX</i> (the maximum amount of physical memory a single process can use).</p>

D DATA DICTIONARY TABLES

This appendix documents the organization of the data dictionary tables, which are stored in the databank SYSDB.

The tables are created when the system is first installed, and are owned by a system ident called MIMER. The database administration ident SYSADM is granted SELECT access on the dictionary tables with the WITH GRANT OPTION. No other user may read the data dictionary base tables unless authorized to do so by SYSADM.

A set of system views is defined on the data dictionary tables when the system is installed (see Appendix C of the *MIMER/SQL Reference Manual* for documentation of these views). The global group ident PUBLIC is granted SELECT access on these views, so that any user may read the (in many cases restricted) dictionary information presented in the views. Many of the view definitions restrict the information presented to descriptions of objects accessible to the current user. Note that if SYSADM reads the contents of such a view, the result shows only the objects to which SYSADM has access. In order to gain information on inaccessible objects, SYSADM must read the contents of the dictionary base tables directly.

The SYSADM ident may define installation-specific views on the data dictionary tables to supplement the system-defined views. Such views may be tailor-made for the installation or system in use, and SELECT access on the views may be granted to limited user groups if desired.

All maintenance of the data dictionary is performed by internal routines and is invisible to the user. No user, including SYSADM, may alter the contents of the data dictionary directly.

Note: MIMER reserves the right to change the internal organization of the data dictionary, without maintaining backward compatibility with user-written application programs which read the data dictionary tables directly.

Summary of data dictionary tables

Table name	Description
ACCESS	access privileges on tables and views
ACCCOL	access privileges on specific columns
BACKUP	backup operations performed with backup and restore functionality
CODE	conditions in table, view and domain definitions
COLUMN	columns in tables and views
COMMENT	comments on database objects
DATABANK	databanks in the database
DBFILE	databank file names
DBNAME	databank copies by name
DEFAULT	default value strings for domains
DOMAIN	domains in the database
FUNCTION	translation of id to function or module name
IDENT	idents in the database
INDEX	secondary indexes in the database
INDEXCOL	columns in indexes (primary, secondary and foreign)
MESSAGE	message codes and texts
OBJECT	objects in the database
OBJECT_REF	objects referenced in a routine or module
ONEROW	dummy table with one row
PARAMETER	parameters for routines
PRIV	system and object privileges
RESTRICT	domain restrictions valid in level 2
RESULT	result set for routines
ROUTINE	routines in the database
SERVINFO	attributes of current database system or server
SEVERITY	error code severity
SOURCE_DEF	source code for modules
SQLLANG	contains one row for each SQL standard and SQL dialect supported
SQLMODULE	modules in the database
SYNONYM	synonyms in the database
TABLE	tables and views in the database
TABLEX	base, secondary index, and foreign key tables
VIEWCOL	columns in views valid in level 2
VIEWDEP	view dependencies
VIEWRES	view restrictions valid in level 2

ACCESS**Records privileges held by users on tables and views.**

Column name	Data type	Description
TABLEID	INT(10)	System identifier for table or view
GRANTEE	CHAR(18)	Name of the user who holds the privilege
GRANTOR	CHAR(18)	Name of the user who granted the privilege
TIMENO	INT(10)	Internal system time stamp
DEL	CHAR(1)	Privilege to delete rows “N” = not held “Y” = held without grant option “G” = held with grant option
INS	CHAR(1)	Privilege to insert rows “N” = not held “Y” = held without grant option “G” = held with grant option
LOA	CHAR(1)	Privilege to load rows “N” = not held “Y” = held without grant option “G” = held with grant option
SEL	CHAR(1)	Privilege to select rows “N” = not held “Y” = held without grant option “G” = held with grant option
REF	CHAR(1)	Privilege to use the primary key of the table as a foreign key reference “N” = not held “Y” = held without grant option “G” = held with grant option
UPD	CHAR(1)	Privilege to update rows “N” = not held “Y” = held without grant option “G” = held with grant option
UPDCOL	CHAR(1)	Restricted update privilege “N” = privilege applies to all columns (except primary key columns) “Y” = privilege applies only to columns with matching rows in ACCCOL
REFCOL	CHAR(1)	Restricted reference privilege “N” = privilege applies to all columns “Y” = privilege applies only to columns with matching rows in ACCCOL
SELCOL	CHAR(1)	Restricted select privilege “N” = privilege applies to all columns “Y” = privilege applies only to columns with matching rows in ACCCOL
INSCOL	CHAR(1)	Restricted insert privilege “N” = privilege applies to all columns “Y” = privilege applies only to columns with matching rows in ACCCOL
Primary key: TABLEID, GRANTEE, GRANTOR, TIMENO		
Secondary index: GRANTEE		

ACCCOL**Records access privileges held by users on individual columns of a table or view.**

Column name	Data type	Description
TABLEID	INT(10)	System identifier for the table or view
COLNO	INT(3)	Column no to which the update applies
GRANTEE	CHAR(18)	Name of the user who holds the privilege
GRANTOR	CHAR(18)	Name of the user who granted the privilege
TIMENO	INT(10)	Internal system time stamp
ACCTYPE	CHAR(3)	Privilege type "INS" = insert "REF" = reference "UPD" = update
Primary key: TABLEID, COLNO, GRANTEE, GRANTOR, TIMENO, ACCTYPE		
Secondary index: GRANTEE		

BACKUP**Contains one row for each backup operation.**

Column name	Data type	Description
DATABANK	CHAR(18)	Name of the databank
DATEB	CHAR(8)	Date of operation
TIMEB	CHAR(6)	Time of operation
TYPE	CHAR(2)	Function type "BC" = backup copy "BU" = incremental backup "DL" = drop log
FILENO	INT(3)	File number
IDENT	CHAR(18)	Name of the user who invoked the operation
FILE	CHAR(64)	Name of the file in the host file management system
TIMENO	INT(10)	Time order number
Primary key: DATABANK, DATEB, TIMEB, TYPE, FILENO		

CODE**Contains one or more rows for each condition for table, view or domain.**

Column name	Data type	Description
CODEID	INT(10)	System identifier for table, view or domain
TYPE	CHAR(1)	Indicates the type of code "T" = text "A" = abstract syntax tree
VERNO	INT(3)	Version number for AST generation
SEQNO	INT(3)	Sequential order number
LENGTH	INT(5)	Length of code null if SEQNO not equal to 1
DATA	CHAR(200)	Code field
Primary key: CODEID, TYPE, VERNO, SEQNO		

COLUMN**Contains one row for every column in each table or view.**

Column name	Data type	Description
TABLEID	INT(10)	System identifier of the table or view which contains the column
COLUMN	CHAR(18)	Name of the column
COLNO	INT(3)	Ordinal number of the column in the table or view
COLOFF	INT(5)	Offset in row for the column
TYPE	CHAR(1)	Type of column "C" = character "D" = decimal "F" = float "I" = integer "P" = period (interval) "T" = time (datetime) "V" = varchar
SIZE	INT(5)	The length attribute of the column: for character types this is the maximum number of chars for numeric types this is the precision for datetime types this is the length of the datetime string for interval types this is the leading precision
SCALE	INT(2)	For decimal types this is the number of digits after the decimal point. For integer or character types this is 0. For floating point types this is -1. For datetime or interval types this is the number of digits in the fractional seconds component.
LENGTH	INT(5)	The internal length of the column
NULLS	CHAR(1)	Indicates whether the column may contain null values "N" = no "Y" = yes
DEFAULT	CHAR(1)	Indicates whether column is associated with a default value "N" = no "Y" = yes
DOMAINID	INT(10)	System identifier for domain null if no domain exists
UPDATES	CHAR(1)	Indicates whether the column is updatable "N" = no "Y" = yes (The value is "N" if the column is part of a primary key or is derived from a function or arithmetic expression)
COLCARD	INT(10)	Number of distinct values in the column (null if statistics not collected)
COLCNOTN	INT(10)	Number of non-null values in the column (null if statistics not collected)
LOW	CHAR(16)	Lowest value of the column (null if statistics not collected)
HIGH	CHAR(16)	Highest value of the column (null if statistics not collected)

DATATYPE	CHAR(18)	Data type of column: "CHARACTER" "CHARACTER VARYING" "SMALLINT" "INTEGER" "INTEGER(p)" "DECIMAL" "NUMERIC" "REAL" "FLOAT" "FLOAT(p)" "DOUBLE PRECISION" "DATE" "TIME" "TIMESTAMP" "YEAR" "MONTH" "YEAR TO MONTH" "DAY" "HOUR" "MINUTE" "SECOND" "DAY TO HOUR" "DAY TO MINUTE" "DAY TO SECOND" "HOUR TO MINUTE" "HOUR TO SECOND" "MINUTE TO SECOND"
Primary key: TABLEID, COLUMN		
Secondary index: TABLEID, COLNO		
Secondary index: TABLEID, COLOFF		

COMMENT

Contains comments for an object.

Column name	Data type	Description
OBJECTID	INT(10)	System identifier for the object
COLNO	INT(3)	Zero or column number
SEQNO	INT(3)	Sequential order number
LENGTH	INT(5)	Length of comment (null if SEQNO not equal to 1)
COMMENT	CHAR(64)	Comment on the object
Primary key: OBJECTID, COLNO, SEQNO		

DATABANK

Contains one row for each databank (including system, transaction and log databank).

Column name	Data type	Description
DBANKID	INT(10)	System identifier for the databank
SEQNO	INT(3)	Sequence number
DATABANK	CHAR(18)	Name of the databank
CREATOR	CHAR(18)	Name of the creator of the databank
TYPE	CHAR(1)	Indicates type of transaction handling "L" = transaction handling with logging "T" = transaction handling without logging "N" = transaction handling not used "W" = work databank (SQLDB)
Primary key: DBANKID		

DBFILE

Contains one row for each databank file.

Column name	Data type	Description
DBANKID	INT(10)	System identifier for the databank
SEQNO	INT(3)	Sequence number
FILENO	INT(3)	File number
SHADOW	CHAR(18)	Shadow name
SUSPEND	CHAR(1)	Databank/shadow offline status "Y" = offline, may not be accessed "N" = online, may be accessed
FILE	CHAR(64)	Shadow file name
Primary key: DBANKID, SEQNO, FILENO		

DBNAME

Contains one row for each copy of a databank.

Column name	Data type	Description
DBNAME	CHAR(18)	Databank or shadow name
DBANKID	INT(10)	System identifier for the databank
SEQNO	INT(3)	Sequence number
Primary key: DBNAME		

DEFAULT**Contains default value string connected to a domain or column.**

Column name	Data type	Description
DOMAINID	INT(10)	System identifier for domain or table with column default value
COLNO	INT(3)	Column number (0 if domain)
SEQNO	INT(3)	Sequence number
LENGTH	INT(5)	Length of default value -2 indicates USER -3 indicates CURRENT_DATE -4 indicates CURRENT_TIME -5 indicates CURRENT_TIMESTAMP (NULL if SEQNO not equal to 1)
DEFVALUE	CHAR(32)	Default value (if USER then USER elseif current date/time then CURRENT_DATE or CURRENT_TIME(p) or CURRENT_TIMESTAMP(p))
Primary key: DOMAINID, COLNO, SEQNO		

DOMAIN**Contains one row for each domain.**

Column name	Data type	Description
CREATOR	CHAR(18)	Name of the creator of the domain
DOMAIN	CHAR(18)	Name of the domain
DOMAINID	INT(10)	System identifier for the domain
TYPE	CHAR(1)	Type of column "C" = character "D" = decimal "F" = float "I" = integer "P" = period (interval) "T" = time (datetime) "V" = varchar
SIZE	INT(5)	The length attribute of the column: for character types this is the maximum number of chars for numeric types this is the precision for datetime types this is the length of the datetime string for interval types this is the leading precision
SCALE	INT(2)	For decimal types this is the number of digits after the decimal point. For integer or character types this is 0. For floating point types this is -1. For datetime or interval types this is the number of digits in the fractional seconds component.
LENGTH	INT(5)	The internal length of the column
CODE	CHAR(1)	Indicates whether check option exists "N" = no "Y" = yes
DEFAULT	CHAR(1)	Indicates whether default value exists "N" = no "Y" = yes

LEVELS	CHAR(1)	Indicates whether legal to use for DB level 2 "N" = no "Y" = yes
DATATYPE	CHAR(18)	Data type of column: "CHARACTER" "CHARACTER VARYING" "SMALLINT" "INTEGER" "INTEGER(p)" "DECIMAL" "NUMERIC" "REAL" "FLOAT" "FLOAT(p)" "DOUBLE PRECISION" "DATE" "TIME" "TIMESTAMP" "YEAR" "MONTH" "YEAR TO MONTH" "DAY" "HOUR" "MINUTE" "SECOND" "DAY TO HOUR" "DAY TO MINUTE" "DAY TO SECOND" "HOUR TO MINUTE" "HOUR TO SECOND" "MINUTE TO SECOND"
Primary key: CREATOR, DOMAIN		
Secondary index: DOMAINID		

FUNCTION

Translation of id to function or module name.

Column name	Data type	Description
MODULEID	INT(3)	Module identification
FUNCTION	INT(5)	Function identification
TEXT	CHAR(40)	Module name or routine name
Primary key: MODULEID, FUNCTION		

IDENT**Contains one row for each ident.**

Column name	Data type	Description
IDENT	CHAR(18)	Name of the ident
CREATOR	CHAR(18)	Name of the creator of the ident
TYPE	CHAR(1)	Indicates the type of the ident "U" = user "G" = group "O" = OS-user "P" = program
PASSWORD	CHAR(18)	Enciphered password (null for OS-user without password)
IDENTID	INT(10)	System identifier for the ident
Primary key: IDENT		

INDEX**Contains one row for each secondary index.**

Column name	Data type	Description
CREATOR	CHAR(18)	Name of the creator of the index
INDEX	CHAR(18)	Name of the index
INDEXID	INT(10)	System identifier for the index
TABLEID	INT(10)	System identifier for the table with the index
Primary key: CREATOR, INDEX		

INDEXCOL**Contains one row for every column of an index (primary, secondary or foreign).**

Column name	Data type	Description
TABLEID	INT(10)	System identifier for the base table (the referenced table if IXTYPE = "X")
INDEXID	INT(10)	System identifier for the index table (the base table if IXTYPE = "P" or "G" secondary index table if IXTYPE = "S" or "Q" foreign key table if IXTYPE = "F" or "X" alternate key table if IXTYPE = "A")
IXTYPE	CHAR(1)	Indicates the type of the index "A" = alternate (unique) key "F" = foreign key "G" = generated primary key "P" = primary "Q" = unique secondary index "S" = secondary "X" = cross reference for foreign
INDEXNO	INT(3)	Ordinal number of the column in the index table
COLNO	INT(3)	Ordinal number of the column in the base table
COLOFF	INT(5)	Offset in row for the column in the base table
LENGTH	INT(5)	Internal length of the column
SORTDIR	CHAR(1)	Sort direction for column in index "A" = ascending (used if not secondary index) "D" = descending
RTABLEID	INT(10)	System identifier for referenced table (base table if IXTYPE = "X", null if IXTYPE = "A", "G", "P", "S" or "Q")
RINDEXID	INT(10)	System identifier for referenced index table null if IXTYPE = "A", "G", "P", "S" or "Q")
RCOLNO	INT(3)	Ordinal number of column in referenced table (null if IXTYPE = "A", "G", "P", "S" or "Q")
RCOLOFF	INT(5)	Offset in row for column in referenced table (null if IXTYPE = "A", "G", "P", "S" or "Q")
Primary key: TABLEID, INDEXID, IXTYPE, INDEXNO		

MESSAGE**Connection between message codes and their messages.**

Column name	Data type	Description
MODULEID	INT(3)	Module identification
MESSAGID	INT(5)	Message identification
LANGUAGE	INT(3)	Message text language identification
LINENO	INT(3)	Zero = short message, + = long message
MESSAGE	CHAR(80)	Message text in specified language
Primary key: MODULEID, MESSAGID, LANGUAGE, LINENO		

OBJECT

**Contains one row for each created object,
one special record for updates of system identifiers and
one special record for updates of system timestamp.**

Column name	Data type	Description
CREATOR	CHAR(18)	Name of the creator of the object
OBJECT	CHAR(18)	Name of the object
COLUMN	CHAR(18)	Blank or name of the column
TYPE	CHAR(8)	Indicates the type of the object: "COLUMN" "DATABANK" "DOMAIN" "IDENT" "INDEX" "MODULE" "ROUTINE" "SHADOW" "SYNONYM" "TABLE" "TABSTAT" "VIEW"
DATEC	CHAR(8)	Date of creation in form YYYYMMDD
TIMEC	CHAR(6)	Time of creation in form HHMMSS
OBJECTID	INT(10)	System identifier for the object
COLNO	INT(3)	Zero or column number
Primary key: CREATOR, OBJECT, COLUMN, TYPE		

OBJECT_REF

Contains one row for each object referenced in a routine or module.

Column name	Data type	Description
REFID	INT(10)	System identifier for routine or module which references object
OBJECTID	INT(10)	System identifier for referenced object
COLUMN	CHAR(18)	Name of referenced column, blank if referenced object is domain, routine or ident
PRIVILEGE	CHAR(18)	Privilege associated with usage of object. For table or view, one of "SELECT" "UPDATE" "INSERT" "DELETE" if object is domain "USAGE" if object is routine "EXECUTE ROUTINE" if object is ident "PATH"
Primary key: REFID, OBJECTID, COLUMN, PRIVILEGE		
Secondary index: OBJECTID		

PARAMETER

Contains one row for every parameter in all routines.

Column name	Data type	Description
ROUTINEID	INT(10)	System identifier of the routine which contains the parameter
PARNO	INT(3)	Ordinal number of the parameter in the routine
PARAMETER	CHAR(18)	Name of parameter
MODE	CHAR(5)	One of: "IN" "INOUT" "OUT"
SIZE	INT(5)	The length attribute of the parameter: for character types this is the maximum number of chars for numeric types this is the precision for datetime types this is the length of the datetime string for interval types this is the leading precision
SCALE	INT(2)	For decimal types this is the number of digits after the decimal point. For integer or character types this is 0. For floating point types this is -1. For datetime or interval types this is the number of digits in the fractional seconds component.
DOMAINID	INT(10)	System identifier for domain null if no domain exists
DATATYPE	CHAR(18)	Data type of column: "CHARACTER" "CHARACTER VARYING" "SMALLINT" "INTEGER" "INTEGER(p)" "DECIMAL" "NUMERIC" "REAL" "FLOAT" "FLOAT(p)" "DOUBLE PRECISION" "DATE" "TIME" "TIMESTAMP" "YEAR" "MONTH" "YEAR TO MONTH" "DAY" "HOUR" "MINUTE" "SECOND" "DAY TO HOUR" "DAY TO MINUTE" "DAY TO SECOND" "HOUR TO MINUTE" "HOUR TO SECOND" "MINUTE TO SECOND"
Primary key: ROUTINEID, PARNO		

PRIV**Records system and object privileges held by users.**

Column name	Data type	Description
PRIV	CHAR(1)	Privilege: "I" = Ident "D" = Databank "T" = Table "M" = Member "E" = Execute on program ident "R" = Execute on routine "U" = Usage on domain "X" = Execute on module
OBJECTID	INT(10)	Zero if privilege is Ident or Databank. System identifier for databank if privilege is Table. System identifier for ident if privilege is Member or Execute.
GRANTEE	CHAR(18)	Name of the user who holds the privilege
GRANTOR	CHAR(18)	Name of the user who granted the privilege
TIMENO	INT(10)	Internal system time stamp
GRANTOPT	CHAR(1)	Indicates whether the privilege is held with grant option "N" = no "Y" = yes
Primary key: PRIV, OBJECTID, GRANTEE, GRANTOR, TIMENO		

RESTRICT**Restriction for domains legal for use by DB level 2 (internal information).**

Column name	Data type	Description
DOMAINID	INT(10)	System identifier for domain with restrictions
SEQNO	INT(3)	Restriction order number
LOGOP	CHAR(3)	Logical operator("AND", "OR")
RELOP	CHAR(2)	Relational operator("EQ", ...)
LENGTH	INT(5)	Length of value
RESVALUE	CHAR(64)	Restriction value
Primary key: DOMAINID, SEQNO		

RESULT**Contains one row for every result column in a routine.**

Column name	Data type	Description
ROUTINEID	INT(10)	System identifier of the routine which contains the result
RESNO	INT(3)	Ordinal number of the result column
CORRELATION	CHAR(18)	Correlation name
SIZE	INT(5)	The length attribute of the result: for character types this is the maximum number of chars for numeric types this is the precision for datetime types this is the length of the datetime string for interval types this is the leading precision
SCALE	INT(2)	For decimal types this is the number of digits after the decimal point. For integer or character types this is 0. For floating point types this is -1. For datetime or interval types this is the number of digits in the fractional seconds component.
DOMAINID	INT(10)	System identifier for domain null if no domain exists
DATATYPE	CHAR(18)	Data type of column: "CHARACTER" "CHARACTER VARYING" "SMALLINT" "INTEGER" "INTEGER(p)" "DECIMAL" "NUMERIC" "REAL" "FLOAT" "FLOAT(p)" "DOUBLE PRECISION" "DATE" "TIME" "TIMESTAMP" "YEAR" "MONTH" "YEAR TO MONTH" "DAY" "HOUR" "MINUTE" "SECOND" "DAY TO HOUR" "DAY TO MINUTE" "DAY TO SECOND" "HOUR TO MINUTE" "HOUR TO SECOND" "MINUTE TO SECOND"
Primary key: ROUTINEID, RESNO		

ROUTINE**Contains one row for each routine.**

Column name	Data type	Description
CREATOR	CHAR(18)	Creator of routine
ROUTINE	CHAR(18)	Routine name
ROUTINEID	INT(10)	System identifier for routine
SQLMODULEID	INT(10)	System identifier for module in which routine is defined
OFFSET	INT(10)	Offset of routine declaration in module
LENGTH	INT(10)	Length of source definition
PARAMETERS	INT(3)	Number of parameters for routine
RESULT	INT(3)	Number of result columns for routine
INPAR	INT(3)	Number of in parameters
OUTPAR	INT(3)	Number of out parameters
ACCESS_MODE	INT(3)	Access mode 1 - NO SQL 2 - CONTAINS SQL 3 - READS SQL 4 - MODIFIES SQL
Primary key: CREATOR, ROUTINE		
Secondary index: SQLMODULEID, ROUTINEID		
Secondary index: ROUTINEID		

SERVINFO**Attributes of current database system or server.**

Column name	Data type	Description
SERVATTR	CHAR(18)	Server attribute
ATTRVAL	CHAR(18)	Attribute value
Primary key: SERVATTR		

SEVERITY**Error code severity level and optional module.**

Column name	Data type	Description
MODULEID	INT(3)	Module identification
MESSAGID	INT(5)	Zero or message identification
SEVERITY	INT(1)	Message, warning, error, fatal, internal
MODULE	CHAR(20)	Module where error was encountered
Primary key: MODULEID, MESSAGID		

SOURCE_DEF**Contains one or more rows for each module definition.**

Column name	Data type	Description
SOURCEID	INT(10)	System identifier for source
LINENO	INT(10)	Line number
SOURCE	CHAR(5000)	Source definition
Primary key: SOURCEID, LINENO		

SQLLANG

Contains one row for each SQL standard and SQL dialect supported.

Column name	Data type	Description
SOURCE	CHAR(18)	Organization that defined this SQL version
YEAR	CHAR(18)	Year when the relevant source document was approved
BINDSTYL	CHAR(18)	To envisage future adoption of binding styles
PROGLANG	CHAR(18)	Host language for which the binding style is supported
CONFORM	CHAR(18)	Conformance level to the relevant document, the implementation claims
INTEGRIT	CHAR(18)	Indication of whether the implementation supports integrity enhancement feature
IMPLEMEN	CHAR(18)	Identification of SQL product
Primary key: SOURCE, YEAR, BINDSTYL, PROGLANG		

SQLMODULE

Contains one row for each module.

Column name	Data type	Description
CREATOR	CHAR(18)	Creator of module
SQLMODULE	CHAR(18)	Module name
SQLMODULEID	INT(10)	System identifier for module
LENGTH	INT(10)	Length of source definition
Primary key: CREATOR, SQLMODULE		
Secondary index: SQLMODULEID		

SYNONYM

Contains one row for each synonym of a table or view.

Column name	Data type	Description
CREATOR	CHAR(18)	Name of the creator of the synonym
SYNONYM	CHAR(18)	Synonym name for the table or view
SYNONID	INT(10)	System identifier for the synonym
BCREATOR	CHAR(18)	Name of the creator of the table or view
BTABLE	CHAR(18)	Name of the table or view
Primary key: CREATOR, SYNONYM		

TABLE**Contains one row for each table or view.**

Column name	Data type	Description
CREATOR	CHAR(18)	Name of the creator of the table or view
TABLE	CHAR(18)	Name of the table or view or synonym
TYPE	CHAR(2)	Indicates whether table or view (and synonym): "T" = table "V" = view "TS" = synonym for table "VS" = synonym for view
TABLEID	INT(10)	System identifier for table or view (if synonym then identifier for referenced table or view)
CODE	CHAR(1)	Indicates whether check clause for table, subselect clause for view "N" = no "Y" = yes
UPDATES	CHAR(1)	Indicates whether the view is updatable, null if not view "N" = no "Y" = yes
CHECK	CHAR(1)	Indicates whether check option specified for view, null if not view "N" = no "Y" = yes
LEVELS	CHAR(1)	Indicates whether legal to use for DB level 2 "N" = no "Y" = yes
Primary key: CREATOR, TABLE		
Secondary index: TABLEID		

TABLEX**Contains one row for each base table, secondary index table or foreign key table.**

Column name	Data type	Description
TABLEID	INT(10)	System identifier for the table
TYPE	CHAR(1)	Indicates the type of table: "A" = alternate key table "F" = foreign key table "Q" = unique secondary index table "S" = secondary index table "T" = base table
NOKEYCOL	INT(3)	Number of key columns
KEYLEN	INT(5)	Primary key length
NOCOL	INT(3)	Number of columns
RECLEN	INT(5)	Record length
DBANKID	INT(10)	System identifier for the databank containing the table
CARD	INT(10)	Total number of rows in table null if statistics not gathered
Primary key: TABLEID		
Secondary index: DBANKID		

VIEWCOL

Contains one row for every column of a view acceptable by DB level 2 (internal information).

Column name	Data type	Description
VTABID	INT(10)	View table identifier
VCOLNO	INT(3)	View table column number
BTABID	INT(10)	Base table identifier
BCOLNO	INT(3)	Base table column number
Primary key: VTABID, VCOLNO		

VIEWDEP

Records the dependencies of views on tables or other views.

Column name	Data type	Description
TABLEID	INT(10)	System identifier of the table or view the view is dependent on
TYPE	CHAR(1)	Indicates whether the view is dependent on another view or on a table “T” = table “V” = view
DEPTABID	INT(10)	System identifier for the dependent view
Primary key: TABLEID, TYPE, DEPTABID		

VIEWRES

Records restrictions for DB level 2 (internal information).

Column name	Data type	Description
VTABID	INT(10)	View table identifier
SEQNO	INT(3)	Restriction order number
LOGOP	CHAR(3)	Logical operator(“AND”, “OR”)
BTABID	INT(10)	Base table identifier
BCOLNO	INT(3)	Base table column number
RELOP	CHAR(2)	Relational operator(“EQ”, ...)
LENGTH	INT(5)	Length of value
RESVALUE	CHAR(64)	Restriction value
Primary key: VTABID, SEQNO		

INDEX

A

- access privileges 2-12
- authorization
 - database statistics 8-2
 - DBC program 9-2
 - DBOPEN functionality 10-2
 - export functions 5-2
 - import functions 5-6
 - load and unload 5-9

B

- background threads 4-4
- backup and restore 6-1
 - SQL statements 6-8
 - using the host system 6-9
- backups of databanks 6-4
- balanced I/O 2-7
- BSQL command-line arguments 3-12
- bufferpool 4-2

C

- CHECK constraints in tables 2-16
- CHECK OPTION in views 2-16
- command-line arguments
 - BSQL 3-12
 - MIMCONTROL 4-7
 - MIMINFO 4-11
 - UTIL 3-12
- concurrency control 2-9
- connecting to a database 3-6
- connection names 3-8
- creating MIMER system databanks 3-4

D

- data dictionary 2-1
- data dictionary tables D-1
- data integrity 2-14
- databank check (DBC) functionality 9-1
 - authorization 9-2
 - error messages 9-6
 - error status 9-1
 - output format 9-3
 - syntax 9-2
- databank open (DBOPEN) functionality 10-1
 - authorization 10-2
 - error status 10-1
 - syntax 10-1

- DATABANK privilege 2-12
- databanks
 - allocating disk space 2-6
 - backups 6-4
 - changing location 2-8
 - data security 2-7
 - disk I/O 2-7
 - disk speed 2-7
 - file access 2-8
 - file deletion 2-9
 - incremental backups 6-4
 - locating files 2-4
 - moving between systems 5-1
 - options 2-4
 - organizing 2-6
 - physical integrity check 9-1
 - recreating the system databanks 6-14
 - restoring from backup 6-11
 - system 2-3
 - user 2-4
- database
 - administration 1-1
 - client-server interface 3-3
 - connection 3-6
 - creating 'HOTEL' example 3-12
 - default (node-specific) 3-7
 - default (user-specific) 3-7
 - ident and data structure 3-5
 - listing connected users 4-12
 - local 3-2
 - making accessible 3-1
 - names 3-6
 - remote 3-3
 - removal 3-13
 - security 2-10
 - server control 4-6
 - server log 4-18
 - server memory requirements 4-1
 - single-user mode A-1
 - statistics 8-1
 - system information 4-11
 - system performance parameters 4-1
 - system resource requirements 4-4
 - troubleshooting connect failures 3-9
- database performance parameters 4-1
 - bufferpool 4-2
 - number of background threads 4-4
 - number of request threads 4-4
 - SQLPOOL 4-3
- DBC 9-1
- DBC functionality 9-1
 - authorization 9-2
 - error messages 9-6
 - error status 9-1
 - output format 9-3
 - syntax 9-2

- DBOPEN functionality 10-1
 - authorization 10-2
 - error status 10-1
 - syntax 10-1
- default database 3-7
- DEFAULT section B-3
- DELETE privilege 2-12
- domains 2-15
 - default value 2-15
- drop log 6-8
- dropping objects - recursive effects 2-13
- duplicate rows - handling in import load 5-6

E

- entity integrity 2-15
- EXECUTE privilege 2-12
- export 5-2
 - authorization 5-2
 - file contents 5-2

F

- foreign keys in import 5-5

G

- GRANT OPTION 2-10, 2-13
- group idents 2-2

H

- host system backups 6-9

I

- IDENT privilege 2-12
- idents 2-2
 - GROUP 2-2
 - OS_USER 2-2
 - PROGRAM 2-2
 - recommended organization 2-11
 - USER 2-2
- import 5-3
 - authorization 5-6
 - data loading 5-6
 - naming conflicts 5-4
 - object creation 5-3
 - referential conflicts 5-5
- incremental backups of databanks 6-4
- INSERT privilege 2-12
- integrity
 - domains 2-15
 - foreign key 2-15
 - in view definitions 2-16
 - of databank files 9-1
 - primary key 2-15
 - within tables 2-16

L

- load and unload 5-7
 - authorization 5-9
 - data files 5-7
 - duplicate rows 5-8
 - NULL indicators in data files 5-8

- loading data into tables 5-8
- local database 3-2
- LOCAL section B-3
- locating databank files 2-4
- LOG databank option 2-4
- LOGDB 2-3, 6-2, 6-4
 - backups of 6-13
 - initial creation 3-4
 - reading contents of 7-1

M

- MEMBER privilege 2-12
- MIMCONTROL 4-6
 - syntax 4-7
- mimer log 4-18
- MIMER page size 3-4
- MIMER system databanks 2-3
 - recreating after loss 6-14
- MIMER_MODE variable A-2
- MIMINFO 4-11
 - mimdump report 4-18
 - mimserv report 4-12
 - syntax 4-11
 - users list 4-12
- MRS license key 3-10
- MULTIDEFS file C-1

N

- NULL databank option 2-4
- NULL indicators in load and unload data files 5-8

O

- object privileges 2-12
- optimistic concurrency control 2-9
- optimization of query performance 8-3
- os_user idents 2-2

P

- page size 3-4
- performance
 - database system parameters 4-1
 - query optimization 8-3
- privileges
 - access 2-12
 - object 2-12
 - system 2-12
- program idents 2-2
 - using in import/export 5-7
- PUBLIC group ident 2-2

R

- READLOG functionality 7-1
 - listing restrictions 7-3
 - output destination 7-2
 - output format 7-4
- recursive effects with drop and revoke 2-13
- REFERENCES privilege 2-12
- referential integrity 2-15

- relocating databanks
 - system 2-8
 - user 2-8
- remote database 3-3
- REMOTE section B-3
- request threads 4-4
- reset log 6-8
- restore 6-11
- restriction views 2-14
- revoking privileges - recursive effects 2-13

S

- SDBGEN 3-4
- SELECT privilege 2-12
- shadowing in backup and restore 6-1
- SINGLEDEFS file A-4
- single-user mode access to a database A-1
 - file protection A-2
 - MIMER_MODE variable A-2
 - the singledefs file A-4
- SQL compiler 8-1
- SQL statements for managing databank backups 6-6
- SQLDB 2-3
 - backups of 6-13
 - initial creation 3-4
- SQLHOSTS file B-1
- SQLPOOL 4-3
- statistics on data access 8-1
 - authorization 8-2
 - recommended usage 8-3
- SYSADM 1-1, 2-2
 - creation 3-4
 - privileges 1-2
- SYSDB 2-3
 - backups of 6-12
 - importance of backup protection 6-9
 - initial creation 3-4
- system (machine) administration 1-1
- system databanks 2-3
 - recreating after loss 6-14
- system information - MIMINFO 4-11
- system privileges 2-12

T

- table integrity 2-16
- TABLE privilege 2-12
- threads
 - background 4-4
 - request 4-4
- TRANS databank option 2-4
- transaction control 2-10
- transactions 2-9
 - build-up 2-9
 - read-set 2-10
 - write-set 2-9
- TRANSDB 2-3
 - backups of 6-13
 - initial creation 3-4

U

- unloading data from tables 5-9
- UPDATE privilege 2-12
- USAGE privilege 2-12
- user databanks 2-4
- user idents 2-2
- users list - MIMINFO 4-12
- UTIL command-line arguments 3-12

V

- view integrity 2-16
- views - use in database security 2-14