



Mimer SQL

System Management Handbook

Version 8.2

Copyright © 2000 Mimer Information Technology AB

Mimer SQL version 8.2 System Management Handbook
Second revised edition

December, 2000

Copyright © 2000 Mimer Information Technology AB.

Published by Mimer Information Technology AB,
P.O. Box 1713,
SE-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 Mimer Information Technology 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 SQL 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 and procedures to use when administering a Mimer SQL database system.

Prerequisites

There are no prerequisites for users of this manual. However, it is advisable for the reader 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 SQL database system, particularly those relevant to the responsibilities of the system administrator.
- Chapter 3** describes how the system administrator establishes a Mimer SQL database.
- Chapter 4** covers the subject of managing a Mimer SQL database server including: a description of various factors and parameters which might affect server performance.
- Chapter 5** describes how the BACKUP and RESTORE functionality provides facilities for taking backups of the database and for restoring data in the event of a system crash.
- Chapter 6** describes how the Databank Check (DBC) functionality is provided to check the physical integrity of databanks.
- Chapter 7** 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 a delay (the actual length of the delay depends on how much data is stored in the databank).
- Chapter 8** describes how the SHADOWING functionality allows simultaneously updated copies of databanks to be created and maintained. These give extra data protection and provide automatic fail-over in the event of loss or damage to the original databank.
- Chapter 9** 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 10** 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 11** describes how the EXPORT/IMPORT functionality permits base tables to be transferred between different databases.

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 SQL publications

- **Mimer SQL Reference Manual** contains a complete description of the syntax and usage of all statements in Mimer SQL and is a necessary complement to this manual.
- **Mimer SQL User's Manual** contains a description of the BSQL 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 SQL platform-specific documents** containing platform-specific information. A set of one or more documents is provided, where required, for each platform on which Mimer SQL is supplied.
- **Mimer SQL Release Notes** contain general and platform-specific information relating to the Mimer SQL release for which they are supplied.

Acronyms, terms and trademarks

API	Application Programming Interface.
BSQL	Facility for using SQL interactively or by running a command file.
Data source	ODBC term for a database.
Databank	A Mimer SQL database consists of the Mimer SQL system databanks and a number of user databanks. Each databank is a single physical file in the host file system.
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.
JDBC	Database connectivity for Java.
MULTIDEFS	A file, on Unix and VMS, containing parameters for controlling the database server for a Mimer SQL 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 SQL databank may have one or more shadows. A shadow is a copy of the original (master) databank and is continuously updated by Mimer SQL.
SQL	Structured Query Language.
SQLHOSTS	A file, on Unix and VMS, containing lookup information for all Mimer SQL 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.....	1-1
1.2	SQL statement execution	1-2
2	THE DATABASE ENVIRONMENT	
2.1	The data dictionary	2-1
2.2	Idents	2-2
2.3	Schemas	2-3
2.4	Databanks	2-3
2.4.1	Mimer SQL system databanks.....	2-3
2.4.2	User databanks	2-4
2.4.3	Locating databank files	2-5
2.4.4	Organizing databank files.....	2-6
2.4.4.1	Allocating disk space.....	2-6
2.4.4.2	Protecting data against loss.....	2-7
2.4.4.3	Balanced I/O.....	2-8
2.4.4.4	Reserved directories	2-8
2.4.4.5	Other performance issues.....	2-8
2.4.5	Altering databank locations.....	2-9
2.4.6	Accessing databank files	2-9
2.4.7	Databank file deletion	2-10
2.5	Transaction control	2-10
2.6	Database security	2-11
2.6.1	The role of idents in database security	2-11
2.6.2	System, object and access privileges.....	2-13
2.6.3	Cascade effects between privileges	2-14
2.6.4	Restriction views	2-15
2.7	Data integrity	2-16
2.7.1	Domains	2-16
2.7.2	Entity integrity.....	2-16
2.7.3	Referential integrity.....	2-17
2.7.4	Table integrity	2-17
2.7.5	View integrity.....	2-17
3	ESTABLISHING A MIMER SQL DATABASE	
3.1	Making the database accessible	3-1
3.2	The local database.....	3-2
3.3	Accessing a database remotely.....	3-3
3.3.1	Client-server interface	3-3
3.4	Mimer SQL License Key	3-4
3.4.1	MIMLICENSE on VMS and Unix.....	3-6
3.5	Generating the Mimer SQL system databanks	3-7
3.5.1	SDBGEN syntax	3-8
3.6	Establishing the ident and data structure.....	3-9
3.7	Managing database connections.....	3-10
3.7.1	Selecting a database	3-10

3.7.2	The default database.....	3-10
3.7.2.1	Defining a node-specific default database	3-11
3.7.2.2	Defining a user-specific default database	3-11
3.7.3	Troubleshooting remote database connect failures	3-12
3.8	Running BSQL and UTIL.....	3-13
3.9	Creating the example database.....	3-14
3.10	Removing a database	3-14
4	MANAGING A DATABASE SERVER	
4.1	System performance.....	4-1
4.1.1	Database server memory areas	4-1
4.1.1.1	Code	4-1
4.1.1.2	Data and thread stacks	4-2
4.1.1.3	Bufferpool	4-2
4.1.1.4	Communication buffers	4-3
4.1.1.5	SQLPOOL	4-3
4.1.2	Number of request threads	4-4
4.1.3	Number of background threads	4-4
4.1.4	Database server system requirements.....	4-4
4.2	Controlling the database server.....	4-5
4.2.1	MIMCONTROL syntax	4-6
4.2.2	MIMCONTROL (/STATUS/DCL or -b)	4-8
4.2.3	MIMCONTROL examples	4-9
4.2.4	MIMCONTROL exit codes	4-10
4.3	System information - using MIMINFO.....	4-10
4.3.1	MIMINFO syntax.....	4-11
4.3.2	The users list	4-12
4.3.3	The Performance report	4-12
4.3.3.1	Performance report example.....	4-16
4.3.4	Bufferpool report	4-18
4.3.5	SQLPOOL report	4-18
4.4	Database server log.....	4-18
4.5	Runtime malfunctions	4-19
4.6	Several installations on one machine	4-19
5	BACKUP AND RESTORE	
5.1	Background information	5-1
5.1.1	Database consistency.....	5-2
5.1.2	LOGDB and TRANSDB importance.....	5-2
5.1.3	Updates recorded in LOGDB.....	5-3
5.1.4	TRANSDB considerations	5-4
5.1.5	SQLDB considerations.....	5-5
5.1.6	Databank backups	5-5
5.1.7	System vs. online backups.....	5-6
5.1.8	SQL statements for backing up databanks.....	5-6
5.2	Backup and restore of databanks	5-8
5.2.1	Online backups using the SQL statements	5-8
5.2.2	System backups using the host file system	5-9
5.2.3	Restoring a databank	5-10
5.2.4	Restoring SYSDB	5-11
5.2.5	Re-creating TRANSDB, LOGDB and SQLDB	5-11
5.2.5.1	Creating a new LOGDB	5-12
5.2.5.2	Creating a new TRANSDB	5-13
5.2.5.3	Creating a new SQLDB	5-13
6	DATABANK CHECK FUNCTIONALITY	
6.1	DBC syntax.....	6-1
6.2	Functions	6-2

6.3	Authorization	6-2
6.4	Result file contents.....	6-2
6.4.1	Example of result file	6-5
6.4.2	Error messages	6-5
7	DBOPEN FUNCTIONALITY	
7.1	DBOPEN syntax	7-1
7.2	Functions.....	7-2
7.3	Authorization	7-2
7.4	DBOPEN output example.....	7-3
8	SHADOWING FUNCTIONALITY	
8.1	General description	8-1
8.1.1	Databank shadowing	8-1
8.1.2	Different levels of data protection	8-2
8.2	Shadowing management	8-4
8.2.1	The shadowing utilities.....	8-4
8.2.1.1	Authorization	8-5
8.2.1.2	Wildcards.....	8-5
8.2.1.3	Create shadow.....	8-5
8.2.1.4	Drop shadow.....	8-6
8.2.1.5	Transform shadow to master.....	8-7
8.2.1.6	List shadowing information	8-8
8.3	Backups from shadows	8-9
8.4	Shadowing system databanks	8-9
8.4.1	SYSDB	8-10
8.4.2	TRANSDB	8-11
8.4.3	LOGDB	8-11
8.4.4	SQLDB.....	8-12
8.4.5	If a shadow for SYSDB, TRANSDB or LOGDB is not accessible.....	8-12
8.5	Testing your data protection	8-12
8.6	Configuring the system	8-13
8.7	Performance aspects of shadowing	8-13
8.7.1	Tuning	8-14
8.8	Troubleshooting	8-14
8.8.1	Operator error messages.....	8-14
8.8.2	Miscellaneous problems	8-16
9	READLOG FUNCTIONALITY	
9.1	Functions.....	9-1
9.2	Authorization	9-1
9.3	Using the READLOG functionality	9-2
9.3.1	List definitions (output control).....	9-2
9.3.2	List restrictions	9-2
9.3.3	List operations.....	9-4
9.3.4	Change program id	9-5
9.4	Output format.....	9-5
10	DATABASE STATISTICS	
10.1	Statistical information	10-1
10.2	Authorization	10-1
10.3	The SQL statistics statements	10-2
10.3.1	Statistics for the entire database	10-2
10.3.2	Statistics for specified idents	10-2
10.3.3	Statistics for specified tables	10-2
10.3.4	Secondary index consistency	10-2
10.4	When to use the SQL statistics statements	10-3

11	EXPORT/IMPORT	
11.1	Functions	11-1
11.2	Export options.....	11-2
11.2.1	Extent and authorization.....	11-2
11.2.2	Exported files	11-3
11.3	Import options.....	11-3
11.3.1	Import - object creation.....	11-3
11.3.2	Import - data load.....	11-6
11.3.3	Authorization	11-7
11.4	Load and Unload functions.....	11-7
11.4.1	Data file formats.....	11-7
11.4.2	Load operation	11-8
11.4.3	Unload operation.....	11-9
11.4.4	Authorization	11-10
11.5	Enter and Leave program ident.....	11-10
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-2
A.4	The SINGLEDEFS parameter file.....	A-4
B	THE SQLHOSTS FILE ON VMS AND UNIX	
B.1	The SQLHOSTS file.....	B-1
B.1.1	DEFAULT section	B-3
B.1.2	LOCAL section	B-3
B.1.3	REMOTE section.....	B-3
B.1.4	Local client/server access.....	B-5
C	THE MULTIDEFS FILE ON VMS AND UNIX	
C.1	The MULTIDEFS parameter file.....	C-1
C.1.1	MULTIDEFS parameters.....	C-2
D	UNIX SPECIFICS	
D.1	Automatic database start and stop.....	D-1
D.2	Using raw device partitions	D-1
D.2.1	Creating a raw disk partition	D-2
D.2.2	Arranging for putting a databank on a raw device.....	D-3
D.2.3	Arranging for taking a databank off a raw device	D-3
E	DATA DICTIONARY TABLES	
	Summary of data dictionary tables.....	E-2
	API_FUNCTION.....	E-3
	AST_CODES.....	E-3
	AST_SOURCES.....	E-4
	BACKUPS.....	E-4
	CHAR_SETS.....	E-4
	CHECK_CONSTRAINTS	E-5
	COLLATIONS	E-5
	COLUMNS.....	E-5
	COLUMN_OBJECT_USE.....	E-9
	COLUMN_PRIVILEGES	E-9
	DATABANKS.....	E-10
	DOMAINS.....	E-10
	DOMAIN_CONSTRAINTS	E-13
	FIPS_FEATURES	E-13
	FIPS_SIZING	E-14
	KEY_COLUMN_USAGE.....	E-14
	LEVEL2_RESTRICT.....	E-15

LEVEL2_VIEWCOL	E-15
LEVEL2_VIEWRES	E-15
MANYROWS.....	E-15
MESSAGE.....	E-16
MODULES	E-16
OBJECTS	E-17
OBJECT_COLUMN_USE	E-18
OBJECT_OBJECT_USE	E-18
ONEROW	E-19
PARAMETERS	E-19
REFER_CONSTRAINTS.....	E-21
ROUTINES.....	E-21
SCHEMATA	E-22
SEQUENCES	E-23
SERVER_INFO.....	E-23
SEVERITY	E-24
SOURCE_DEFINITION.....	E-24
SPECIFIC_NAMES	E-25
SQL_LANGUAGES.....	E-25
SYNONYMS.....	E-25
TABLES	E-26
TABLE_CONSTRAINTS	E-27
TABLE_PRIVILEGES.....	E-28
TABLE_TYPES	E-28
TRANSLATIONS	E-29
TRIGGERED_COLUMNS	E-29
TRIGGERS.....	E-30
TYPE_INFO.....	E-31
USAGE_PRIVILEGES	E-32
USERS	E-33
VIEWS.....	E-33

1 INTRODUCTION

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

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

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 SQL 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 SQL ident called SYSADM is created when a database is installed and this ident should be used whenever database administration activities are to be undertaken. The ident name SYSADM may not be changed.

Some of the Mimer SQL functionality requires privileges and access rights which are initially granted only to the SYSADM user, but which may be passed on to other Mimer SQL users.

SYSADM has SELECT access on the internal Mimer SQL 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.

1.2 SQL statement execution

At several places in this manual there are guidelines where SQL statements are shown. These statements can be executed from any ad-hoc SQL query tools, such as Mimer's BSQL, SQL Query Builder, etc.

2 THE DATABASE ENVIRONMENT

This chapter describes the database environment which is composed of a set of Mimer SQL 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 (schemas, tables, views, domains, sequences, 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 is automatically maintained by Mimer SQL. 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 BSQL (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 they have access. The BSQL 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 E 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 SQL 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 connect to a Mimer SQL database, by using the `CONNECT` statement in an application program or by entering the correct ident name and password in an interactive environment. Any privileges a user ident holds may be exercised once the ident has logged on. User idents are generally associated with specific physical individuals authorized to connect to the database.
- OS_USER** is an ident type which allows the user currently logged in to the operating system to connect to a Mimer SQL database without providing a username or password. If an `OS_USER` ident is defined with a password in Mimer SQL, the ident may connect to Mimer SQL in the same way as any other user ident (i.e. by providing a username and password). An `OS_USER` ident is subject to the same access restrictions as any other user ident.
- PROGRAM** idents may not initiate a connection to a Mimer SQL database, but may be entered from within an application program or interactive environment by using the `ENTER` statement. A connection to the database should have been established before the `ENTER` statement is used. The ident using the `ENTER` statement must hold `EXECUTE` privilege on the program ident. Entering a program ident is analogous to logging on as a user ident, in that the program ident gains access to the system and any privileges the ident holds become applicable. Program idents are generally associated with specific functions within the system, not with physical individuals. For an example showing the use of a program ident see [Section 3.2 of the Mimer SQL User's Manual](#).
- GROUP** idents 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 ident can be a member of as many groups as required, and one group can include any number of members. Group idents provide a facility for organizing the privilege structure in the database system. For examples showing the use of a group ident see [Section 8.2.2 of the Mimer SQL User's Manual](#).

USER, OS_USER and PROGRAM idents are authorized users of the system. Every USER and PROGRAM ident has a unique ident name and a private password which must be correctly supplied to the CONNECT or ENTER statement in application programs. An OS_USER may access the database without explicitly providing a username or password on condition that the username for the user currently logged in to the operating system correspond to the definition of an OS_USER in the Mimer SQL database.

When Mimer SQL is installed, the user ident SYSADM (used for database administration) is automatically created. The password for SYSADM is defined when the system is installed (see [Section 3.5](#)). All idents in the system belong to a logical group which is specified by using the keyword PUBLIC in Mimer SQL statements. Privileges granted to PUBLIC by any user are global to the system.

2.3 Schemas

A schema defines a local environment within which private database objects can be created. The ident creating the schema has the right to create objects in it and to drop objects from it.

When a USER, OS_USER or PROGRAM ident is created, a schema with the same name can also be created automatically and the created ident becomes the creator of the schema. This happens by default unless WITHOUT SCHEMA is specified in the CREATE IDENT statement.

When a private database object is created, the name for it can be specified in a fully qualified form which identifies the schema in which it is to be created. The names of objects must be unique within the schema to which they belong, according to the rules for the particular object-type.

If an unqualified name is specified for a private database object, a schema name equivalent to the name of the current ident is assumed.

2.4 Databanks

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

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

2.4.1 Mimer SQL system databanks

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

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

If any one of the system databanks is damaged or missing, attempts to log on to Mimer SQL will fail. Backup and restore procedures for the system databanks are described in Chapter 5.

The Mimer SQL 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 SQL system and the access rights each ident has.
- TRANSDB** this databank stores information that allows the database server to bring the database into a consistent state after a system failure (see [Section 2.5](#)).
- LOGDB** this databank records all write operations performed within transactions on SYSDB and user databanks which have been defined with the LOG option. The information in this databank is used by the Backup and Restore facilities (see [Chapter 5](#)) to restore the contents of a database in the event of a system failure. The READLOG facility is provided (see [Chapter 9](#)) to allow the information in this databank to be examined.
- SQLDB** this databank is used by the transaction handling mechanism to store row data read from the database (see [Section 2.5](#)) and is used by Mimer SQL for temporary storage of result tables.

2.4.2 User databanks

User databanks contain the tables in the database created by the users of the system. Typically these databanks are created by the system administrator and TABLE privilege is granted to the users of the system.

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 5](#)).

If a databank is dropped from the database, the tables stored in the databank are also dropped and the physical databank file is deleted from disk.

The following is an example where a databank and a two users (idents) are created. The idents are then given the authority to create tables within the databank:

```
SQL> CREATE DATABANK DEVELOP OF 1000 PAGES IN 'dev.dbf' WITH LOG OPTION;  
SQL> CREATE IDENT DEVUSER AS USER IDENTIFIED BY 'devuser';  
SQL> CREATE IDENT JOE AS OS_USER;  
SQL> GRANT TABLE ON DEVELOP TO DEVUSER;  
SQL> GRANT TABLE ON DEVELOP TO JOE;
```

2.4.3 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 3.2](#)).

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 **exactly as specified when the databank was created**. 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.

If a databank file specification appears in the data dictionary without an absolute 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 or 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 3.2](#)).

The first directory in the search path list must be the database home directory, where SYSDB is located. Other databank files can be located in any of the directories in the search path list.

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 the databank file is 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.

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.

2.4.4 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 SQL system.

2.4.4.1 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 SQL pages. The size of a Mimer SQL page is 2 kilobytes.

The size of the databank file will be extended automatically by the database server during the lifetime of the databank as more space is required for data storage.

Unix

Under Unix, the environment variable MIMER_EXTEND can be set to the number of Mimer SQL pages by which all databank files will be extended. The default setting is 128.

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 in use by the database server or accessed in single user mode when this command is used.

Win

Under Windows, the number of Mimer SQL pages by which all databank files will be extended is determined by the Mimer SQL 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 small file extension size 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 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 SQL databank files are organized internally into 2, 16 and 64 kilobyte databank blocks. Accessing an internal databank block which is physically split over two or more 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 64 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 following command to check the cluster size of a disk that is already formatted:

```
§ SHOW DEVICE/FULL
```

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 SQL databank files. Use of disk fragmentation utilities may improve performance for large block I/O's.

2.4.4.2 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 5.1](#) 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 SQL databases in the system backup routines. See [Chapter 5](#) 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.

2.4.4.3 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.

2.4.4.4 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.

It is very common practice to reserve entire disks for databanks to allow for the ultimate size of the files.

2.4.4.5 Other performance issues

The placement of databanks on physical disk units will depend on exactly how they will be used when the database system is in operation.

The following issues generally have a more significant effect on database performance than the disk I/O factors relating specifically to physical layout of the Mimer SQL database:

- the amount of virtual memory paging
- the speed of the disk
- the involvement of unnecessary network communication.

For example, to enhance performance, frequently accessed databanks such as TRANSDB may be placed on separate, high performance disks and sufficient memory should be allocated to avoid paging.

2.4.5 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.

Example:

1. Disconnect the databank from the system.

```
SQL> SET DATABANK databank_name OFFLINE;
```

2. Move or copy/delete the databank file to its new location.

3. Alter the databank file name in the data dictionary and reconnect the databank to the system.

```
SQL> ALTER DATABANK databank_name INTO 'new_filename';  
SQL> SET DATABANK databank_name ONLINE PRESERVE LOG;
```

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 5.2.5](#)).

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

The location of a databank cannot 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.

Unix

Databases on Unix platforms may be set up with a directory search path instead of a single home directory (see [Section 2.4.3](#)).

A databank 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. Before being moved, the databank should be set offline to ensure that the file is not locked by the database server.

2.4.6 Accessing databank files

The databank files in a Mimer SQL 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 SQL 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 SQL 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.

2.4.7 Databank file deletion

If a databank or shadow is dropped, the corresponding file will also be deleted from disk. Remember that dropping a Mimer SQL 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 and must be manually deleted.

2.5 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 5.1.1](#)).

Mimer SQL 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 sees 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 behavior in application programs and a number of the system facilities (notably Backup and Restore - see [Chapter 5](#)) is controlled at the databank level 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 SQL 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.

2.6 Database security

Mimer SQL 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
- system, object and access privileges
- restriction views

2.6.1 The role of idents in database security

Access to the Mimer SQL 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 SQL installation process creates one user ident, for use in database administration, with the name SYSADM. The SYSADM ident has all the system privileges (BACKUP, DATABANK, IDENT, SCHEMA, SHADOW and STATISTICS - see Section 2.6.2), with the ability to grant 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.

Re-creation of system databanks can only be done by SYSADM, however, 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.

All idents created in the system automatically belong to a logical group (specified using the keyword PUBLIC in Mimer SQL statements) which is intended to be used for granting global privileges.

The following general recommendations are 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 cascade effects (see [Section 2.6.3](#)). (Creation of domains requires no special privilege and may thus be performed by any ident with a schema. 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 cascading revocation of privileges (see [Section 2.6.3](#)).

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 cascade effects.

2.6.2 System, object and access privileges

Each ident is given privileges within the system which determine the operations the ident is permitted to perform.

Note: In addition to holding any relevant privilege(s), an ident must also be the creator of at least one schema before the ident is able to create private database objects (i.e. domains, functions, indexes, modules, procedures, sequences, synonyms, tables, triggers and views) - see Section 2.3.

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:

BACKUP	gives the right to perform backup and restore operations
DATABANK	gives the right to create databanks
IDENT	gives the right to create idents and schemas
SCHEMA	gives the right to create schemas
SHADOW	gives the right to create shadows and perform shadow control operations
STATISTICS	gives the right to execute the UPDATE STATISTICS statement.

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 execute a function or procedure or the right to enter (become) a specified program ident
MEMBER	makes an ident a member in the specified GROUP
USAGE	gives the right to specify the named domain where a data type would normally be specified (in contexts where use of a domain is allowed) or the right to use a specified sequence.

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 or view. 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 (this privilege may be limited to specified columns within 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 alternate keys of the table as a foreign key from another table (this privilege may be limited to specified columns within the table).

In addition to the five access privileges listed above, the 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.

Access privileges are initially granted to the creator of the table with the WITH GRANT option. The privileges may be passed on to other idents with or without the WITH GRANT OPTION.

Certain operations are not controlled by explicit privileges, but may only be performed by the creator of the object involved. These operations include ALTER (with the exception of ALTER IDENT, which may be performed by either the ident itself or by the creator of the ident), DROP and COMMENT. Privileges may only be explicitly revoked by their grantor, however cascade effects may go wider.

2.6.3 Cascade effects between privileges

Dropping an object from the database or revoking a privilege from an ident may have cascade effects on other objects and idents, depending on the way the database is organized. The keywords CASCADE and RESTRICT may be used in the DROP and REVOKE statements. When using RESTRICT (the default), the operation will fail with no changes being made if any cascade effects result from it. When using CASCADE, the following operations have the consequences described:

- If an ident is dropped, all objects created by the ident are dropped and all privileges granted by the ident are revoked.
- If a databank is dropped, all tables in the databank are also dropped.
- If a table is dropped, all views and synonyms based on the table are dropped. Also, triggers and routines that references the table are dropped.
- If a privilege with the WITH GRANT OPTION is revoked from an ident, all instances of that privilege granted to other idents under the authorization of that WITH GRANT OPTION are also revoked. The WITH GRANT OPTION can be revoked separately.
- If SELECT privilege on a table is revoked from an ident, views created by the ident under the authorization of that SELECT privilege are dropped.

If DATABANK privilege is revoked from an ident, existing databanks created under that privilege are **not** dropped.

The cascade effects of revoking privileges only occur when the last instance of the privilege is revoked (a new instance of the privilege is created each time the privilege is granted to the same ident on the same object). An ident grants privileges, creates views and so on under the authorization of the most recently received valid instance of the WITH GRANT OPTION, SELECT or other relevant privilege.

The data dictionary keeps a record of the specific instance of an authorization under which an operation was performed. The cascade effects apply only to privileges granted or objects created under the specific instance of the authorization which is being revoked.

This is illustrated in the example cases that follow:

CASE 1

1. A grants with grant option to M
M grants to X
2. B grants with grant option to M
M grants to Y
3. A revokes from M
Both X and Y keep privileges
4. B revokes from M
Both X and Y lose privileges

CASE 2

1. A grants with grant option to M
2. B grants without grant option to M
M grants to X
M grants to Y
3. A revokes from M
M loses grant option
Both X and Y lose privileges
4. B revokes from M
M loses privilege

2.6.4 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.

2.7 Data integrity

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

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

2.7.1 Domains

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

The ident defining a table column must hold USAGE privilege on any domain used.

A default value may also be defined for a domain. The domain default value is inserted into a column defined using the domain when data is inserted without an explicit column value being specified. If the default value for the domain is defined outside the range of restriction values for the column, attempts to store the default value in a column using the domain 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 currently be altered after it has been defined.

2.7.2 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).

Entity integrity can only be enforced if a primary key constraint or unique constraints are applied.

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.

2.7.3 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 FOREIGN KEY clause in the CREATE TABLE statement. The properties of a FOREIGN KEY are as follows:

- The columns defined as a foreign key must correspond exactly in number and data type to the primary key or unique key columns in the referenced table.
- Data inserted into the foreign key columns (by either INSERT or UPDATE operations) must either already be present in the primary key or a unique key of the reference table or include at least one NULL column.
- A primary or unique key value that is referenced by a foreign key must not be removed by an update operation. It may be possible to remove such a value with a delete operation provided the ON DELETE rule is used to update the referencing table in a way that preserves the referential integrity.

Note: Referential integrity constraints are effectively checked at the end of the INSERT, DELETE or UPDATE statement. Both the table containing the foreign key reference and the referenced table must be stored in a databank with either the TRANS or LOG option.

2.7.4 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.

2.7.5 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 SQL DATABASE

Once the Mimer SQL software is installed, the database environment must be created. This involves the following activities:

- register the name 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 configure 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 SQL system databanks SYSDB, TRANSDB, LOGDB and SQLDB as well as the database administration ident called SYSADM.
- create the ident and data objects in the database using the data definition statements in Mimer SQL.

3.1 Making the database accessible

In a network environment, the name of a database must be registered on each node from which it is to be accessed. A database is created 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

On Unix and VMS nodes, the name of a database is registered 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 SQL databases.

VMS

Win

On a Windows node, the name of a database is registered by running the Mimer Administrator. The Mimer Administrator adds information about Mimer SQL databases to the Windows registry. Refer to the online Windows help provided with the Mimer Administrator for details on how to use it.

3.2 The local database

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

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

Unix

VMS

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.

Win

The definition of a local database under Windows is created by running the Mimer Administrator and specifying the required parameters, including those that control the database server, which are stored in the Windows registry. 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 all parameters except the database name and home directory.

A fully created local database, complete with its Mimer SQL License Key (see [Section 3.4](#)), Mimer SQL system databanks (see [Section 3.5](#)), user databanks and the ident and data structure contained in them (see [Section 3.6](#)) constitutes a Mimer SQL database.

A completely created local database can only be accessed from the machine on which it resides. 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.

3.3 Accessing a database remotely

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

The purpose of the remote database definition is to define 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 involves creating an entry in the `/etc/sqlhosts` file - see [Appendix B](#) for information on the parameters involved.

On the database server computer the mimer service should be defined in `/etc/services` and a port dispatcher should be defined in `/etc/inetd.conf`. This is usually done automatically by the `dbinstall` command as follows:

`/etc/services:`

```
mimer 1360/tcp
```

`/etc/inetd.conf:`

```
mimer stream tcp nowait root /usr/bin/mimtcp mimtcp -l
```

The `mimtcp` program is started when a connection attempt is made to the mimer service. It establishes a connection to the desired database on the node. The `mimtcp` logfile is found as `/etc/mimtcp.log`. (For manual re-initiation of the `inetd` daemon, the command “`kill -HUP inetd_pid`” should be used).

VMS

The definition of a remote database under VMS involves creating an entry in the **SQLHOSTS** file - see [Appendix B](#) for information on the parameters involved.

Win

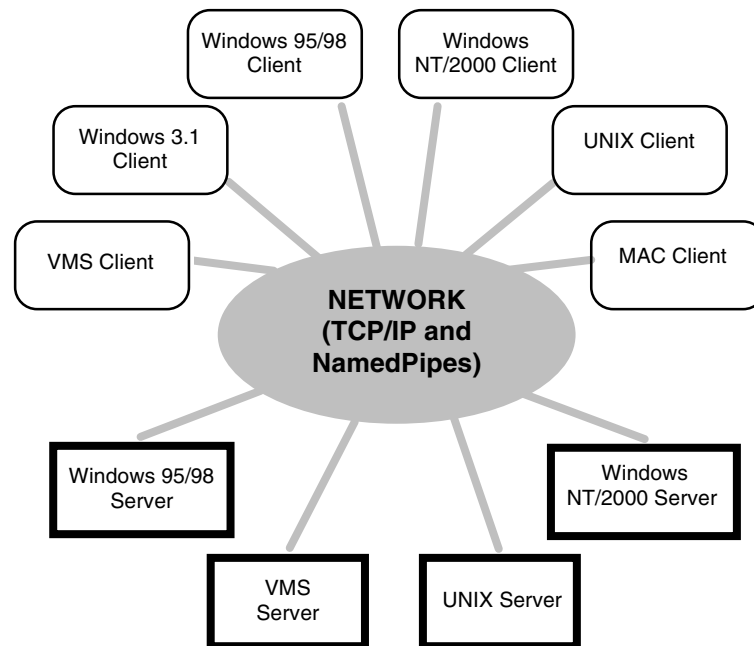
The definition of a remote database under Windows is set up by running the Mimer Administrator and specifying the required parameters. On-line Windows help is provided with the Mimer Administrator to guide you through the creation of a remote database.

3.3.1 Client-server interface

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

The Mimer SQL client/server communications interface is integrated into the database server. The database server process manages all connections. All Mimer SQL 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 SQL client/server protocol is identical on all Mimer SQL platforms. This means that a Mimer SQL client on any machine type may access a Mimer SQL server for a remote database on any of the platforms on which Mimer SQL is implemented.



Mimer SQL client/server communications.

3.4 Mimer SQL License Key

To start the database server and to establish connections to the database, a license key is required. (A key valid for development and evaluation only is included in the Mimer SQL distribution.)

Whenever a user connects to a Mimer SQL database, the computer identification and the license key 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 Mimer SQL license key contains the following (encrypted) information:

- The maximum number of users that may use the database servers running on the same computer node at any one time.
- The maximum number of network users that may use the database servers running on the same computer node at any one time.
- The node name of the computer (in the case of a specific key) or a lifeboat key which is valid for any computer of the platform type for which it was issued (e.g. any Unix machine).
- Version number.
- Expiration date for the key.

The keys for Mimer SQL version 8.2 and later are not valid for earlier versions of Mimer SQL and a key issued for a version of Mimer SQL earlier than version 8.2 is not valid for Mimer SQL version 8.2 and later. The key data is case insensitive and space characters are ignored.

Unix

The **mimlicense** application is used to administrate the license key file. See [Section 3.4.1](#)) for information on how to use MIMLICENSE.

VMS

Win

The Mimer Administrator is used to enter the Mimer SQL license key. The key is distributed in a .mcfg file. When double-clicking on a .mcfg file the Mimer Administrator is automatically invoked to install the key. Refer to the online Windows help provided with the Mimer Administrator for details on how to manually enter the Mimer SQL license key.

The Mimer SQL license key is provided by your Mimer SQL distributor. In order to be able to generate the key, your Mimer SQL distributor must know the node name of the computer on which the database server will run.

Unix

The host name of a Unix machine is obtained by using 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: [MIMERxxxx]MIMSETUP8
```

If the VMS node is part of a cluster, the scsnode 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 a Mimer SQL 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 SQL users is increased or new Mimer SQL functionality is added to the site, a new Mimer SQL license key will be provided.

The Mimer SQL license key uses the node name of the computer to link Mimer SQL to the computer it is authorized to run on. This allows for hardware replacement in the 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 Mimer SQL license key remains valid for the new hardware.

3.4.1 MIMLICENSE on VMS and Unix

Unix

The mimlicense application is currently available on VMS and Unix only.

VMS

Win

On Windows, the Mimer Administrator is used to administrate the license keys.

The MIMLICENSE application is used to administrate the license key file. Keys may be added, removed or updated by using MIMLICENSE. MIMLICENSE may also be used to list and describe the contents of the key file.

The MIMLICENSE program is controlled by flagged information specified on the command-line. The overall syntax (expressed in Unix-style) is as follows:

```
mimlicense [-s] -f filename
           [-s] -a hexcode
           [-s] -d keyid
           -l
           -c
           -r
```

mimlicense command-line arguments

Unix-style	VMS-style	Function
-a <i>hexcode</i>	/ADD= <i>hexcode</i>	Add a license key.
-f <i>file-name</i>	/FILE= <i>file-name</i>	Add a license key from a .mcfg file.
-d <i>key-id</i>	/DELETE= <i>key-id</i>	Delete the specified key.
-r	/REMOVE	Remove all keys. (Each key must be verified.)
-l	/LIST	List the contents of the key file.
-c	/COMBINED	Describe what the combined keys permits.
-s	/SILENT	Silent mode, i.e. execution with no output.

Unix

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

The default name for the key file is **/etc/mimerkey**.

VMS

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

The default name for the key file is

SYSSSPECIFIC:[SYSMGR]MIMERKEY.DAT.

3.5 Generating the Mimer SQL system databanks

The Mimer SQL system databanks SYSDB, TRANSDB, LOGDB and SQLDB are generated by the SDBGEN program. SDBGEN will load the system tables (see Appendix E) and defines the data dictionary views detailed in [Chapter 7 of the Mimer SQL Reference Manual](#).

Note: A databank created for one SYSDB cannot be accessed by using a different SYSDB even if identical data dictionary definitions are created in it.

The initial size for each of the Mimer SQL system databanks can be specified. The size for the databanks is specified in Mimer SQL pages. The size of a Mimer SQL 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.

Unix

A local database is set up on a Unix node by running the dbinstall command (see the dbinstall man-page) and SDBGEN is started automatically when required.

The databank files are by default created in the database home directory which is not the ideal arrangement from a security and performance point of view (see [Section 5.2.5](#)).

If there is more than one disk available on the system, it is recommended that directories be created on those disks specifically for locating databanks. When created, the LOCAL definition for the database should be updated in the /etc/sqlhosts file by changing the single home directory path to a directory path list that includes these directories. The list is colon separated as can be seen in the following example:

```
hotel      /usr/db/hotel:/extra/db/hotel:/extra2/db/hotel
```

After this update is made, the database server can be stopped and the selected databank files can be moved from the database home directory to their new locations. When moved, the database server can be started again.

VMS

In order to create the Mimer SQL system databanks for a local database on a VMS node, an entry for the database must be specified in the SQLHOSTS file. SDBGEN should then be executed to create the actual database. See [Section 3.5.1](#)) for information on how to run SDBGEN.

Win

You set up a local database on a Windows node by running the Mimer Administrator. The Mimer Administrator invokes the SDBGEN program in order to create the system databanks when required. The system databanks are distributed automatically, depending on the number of disks available. On-line Windows help is provided with both the Mimer Administrator and SDBGEN to guide you through setting up a local database.

3.5.1 SDBGEN syntax

Unix

On Unix, the dbinstall utility is used to create the system databanks.

Win

On Windows, the Mimer Administrator is used to create the system databanks.

The SDBGEN command has two purposes. Either to create a new set of system databank files, or to upgrade database files created in an earlier version of Mimer SQL to version 8.2. (Upgrade can be done for databank files created by Mimer SQL version 7.1 and later.)

The SDBGEN program is controlled by flagged information specified on the command-line. The syntax (expressed in VMS-style) for creating databank files is as follows:

```
SDBGEN [/PASSWORD=password] [dbase] [syssz] [tfn] [tsz]
        [lfn] [lsz] [sfn] [ssz]
```

sdbgen command-line arguments (when creating databanks files)

Unix-style	VMS-style	Function
-p password	/PASSWORD=password	Password for SYSADM
<i>dbase</i>	<i>dbase</i>	Database name
<i>syssz</i>	<i>syssz</i>	Size of SYSDB
<i>tfn</i>	<i>tfn</i>	Filename for TRANSDB
<i>tsz</i>	<i>tsz</i>	Size of TRANSDB
<i>lfn</i>	<i>lfn</i>	Filename for LOGDB
<i>lsz</i>	<i>lsz</i>	Size of LOGDB
<i>sfn</i>	<i>sfn</i>	Filename for SQLDB
<i>ssz</i>	<i>ssz</i>	Size of SQLDB

If the password parameter **is omitted**, the SDBGEN command will prompt for all parameters that are missing, including the password for the SYSADM user.

If the password parameter **is given**, the SDBGEN command will **not** prompt for **any** missing parameters, but use default values.

If the `dbase` parameter is missing, the environment variable `MIMER_DATABASE` is used to determine which database the databank files should be created for.

The `SDBGEN` program syntax (expressed in VMS-style) for upgrading databank files to Mimer SQL version 8.2 is as follows:

```
SDBGEN /UPGRADE [dbase]
```

sdbgen command-line arguments (when upgrading databanks)

Unix-style	VMS-style	Function
<code>-u</code>	<code>/UPGRADE</code>	Upgrade existing database to Mimer SQL version 8.2
<code>dbase</code>	<code>dbase</code>	Database name

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 SQL VMS Guide* for more details.

3.6 Establishing the ident and data structure

Once the local database environment has been created for a Mimer SQL database (database name, 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 allows the execution of a sequential file which can then be used as a permanent record of the `CREATE` statements used to create the database objects (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 which could be run again at a later date. BSQL, however, only has limited support for error handling.

An application program using embedded SQL, JDBC 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, such a 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 3.1](#)). Some applications which are part of the Mimer SQL 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 3.3.1](#)).

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

The default database will be used if the CONNECT TO DEFAULT statement is used, or if the database name in the CONNECT statement is specified as an empty string.

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.

Programs supplied as part of the Mimer SQL distribution (e.g. BSQL) will use the default database when database is not specified on the command line.

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, which will be chosen in preference to the node-specific one.

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.

3.7.3 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 3.2](#)) and as a remote database on the client node (see [Section 3.3](#)) and that the name of the remote database is the **same** as that of the local database.

Unix

- Verify that the inetd daemon is listening to the mimer TCP/IP service by using the “netstat -a” command.
- 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:

```
telnet server_node 1360
```

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 SQL version 7.3 client to a Mimer SQL 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 server node, start a NamedPipes server which listens to service “MIMER” so that it can redirect communications to the correct named database server.
- If using NamedPipes to connect a Mimer SQL version 8 client to Mimer SQL 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 Running BSQL and UTIL

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

The overall syntax for BSQL and UTIL (expressed in Unix-style) is as follows:

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

BSQL and UTIL command-line arguments

Unix-style	VMS-style	Function
-m	/MULTI	Connects to the database in multi-user mode.
-s	/SINGLE	Connects to the database in single-user mode.
<i>database</i>	<i>database</i>	Specifies the name of the database to access. If a 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.

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 SQL VMS Guide* for more details.

Win

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

3.9 Creating the example database

BSQL input files are included in the Mimer SQL distribution which will create and then manipulate data in an example database. The “crehotdb.sql” file creates the data structure for the example database and the “examples.sql” file contains various operations to manipulate data in the example database.

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

In the examples below the database named “HOTEL” will be the database wherein the example environment will be created:

VMS

```

BSQL HOTEL
Username: SYSADM
Password:
SQL> READ 'MIMEXAMPLES8:CREHOTDB.SQL';
SQL> EXIT;

```

Unix

```

bsql hotel
Username: SYSADM
Password:
SQL> read 'crehotdb.sql';
SQL> exit;

```

Under Unix the command mimexampledb is available for creating the example environment (see the mimexampledb man-page).

Win

```

bsql hotel
Username: SYSADM
Password:
SQL> read 'crehotdb.sql';
SQL> exit;

```

The “examples.sql” file contains various SQL and PSM statements which may be examined and executed interactively, statement by statement, in order to follow the operations being performed. Many 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. See also the “cjdate.sql” file.

3.10 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: xxxxxxx
SQL> SELECT DATABANK_FILENAME FROM SYSTEM.DATABANKS;
SQL> EXIT;
```

Unix

For a Unix installation, databanks can be located in any directory specified in the pathlist given for the local database definition in /etc/sqlhosts. For a complete list of the databank files included in a database the command mimdbfiles can be used (see the mimdbfiles man-page).

Using the list generated in the previous step, locate and delete all the physical databank files. If no directory is specified, the directory will be the home directory of the database (see [Section 3.2](#)). Delete any directories that have been specifically created to hold databank files for the database.

- Remove the database from the list of accessible databases (see [Section 3.1](#)). 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.

VMS

On VMS the SQLHOSTS file have to be edited.

Win

On a Windows system the Mimer Administrator is used for this operation.

Unix

On Unix The SQLHOSTS file have to be edited. This can be done by using the mimhosts program (see the mimhosts man-page).

- 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.

Win

Under **Windows NT**, when the local definition of a database is removed, the associated service is also automatically removed.

Unix

For a Unix installation, this is usually the rc or init.d facilities.

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.

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.

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 SQL 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.
- list SQLPOOL parameters

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 SQL 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 (perhaps less on some platforms).

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 SQL database management and contains data pages from the databank files. It is a local memory area in the database server process.

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 SQL 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 SQL 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 3.2](#)).

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

buffer space in kilobytes = $Pages2K * 2 + Pages16K * 16 + Pages64K * 64$

Note: The bufferpool contains a variety of other data, therefore 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 3.2](#)) after the Mimer SQL 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 SQL bufferpool size affects the performance of both Mimer SQL and other applications (because it reserves memory for a Mimer SQL database server), it is advisable to perform regular routine checks on the bufferpool statistics in an operational system by generating a Performance report (see [Section 4.3.3](#)).

Note: The Windows NT performance monitor can also be used to monitor a database server running on any platform. Refer to the documentation supplied by Microsoft for the Windows NT operating system for details.

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 SQL and the operating system usually differ, and forced cooperation between the two will often detract considerably from Mimer SQL's performance.
- If more than about 2% of all Mimer SQL page requests result in a page fault, the bufferpool is too small. Statistics for page requests and faults are presented in the Performance 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 SQL 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 3.2](#)).

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 3.2](#)). The SQLPOOL area grows dynamically when the database server needs more space. The local database parameter *MaxSQLPool* controls the maximum size (in Kb) of the SQLPOOL. The value for *MaxSQLPool* is $2000 * (Users + RequestThreads)$ by default.

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 SQL 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. If there are more concurrent requests than threads, the database server will start scheduling requests to improve response times. Increasing the number of request threads in a situation like this may improve performance.

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 SQL database server perform tasks such as:

- Recording transactions in LOGDB.
- Updating master and shadow databanks.
- Securing data on disk.

Refer to the details on “Transaction count” and “Pending background thread requests” in the “Background threads” section of the Performance 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 3.2](#)), whose initial default values are determined by looking at the amount of installed 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.

Unix

The virtual memory handling on Unix platforms is platform specific - refer to the documentation for the specific Unix operating system you are using. (Often a paging file used).

VMS

The MIMCONTROL command sets the paging 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.

Win

If you get a message box saying the system is running out of virtual memory you may need to increase the size of your paging file. This done by using the Virtual Memory option in the Performance section of the System control panel.

Global Pages

VMS

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.

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 can be controlled using the **mimadmin** command (see the mimadmin man-page). This command invokes the MIMCONTROL program, and other programs, as required. The MIMCONTROL command can also be used directly under Unix.

A database server on Unix can be administered by the owner of it or by the superuser "root". To change ownership of a database the mimdbfiles command is used (see the mimdbfiles man-page).

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 created 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.

The Mimer Administrator can also be used to control database servers on Windows platforms.

Windows NT/2000: 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 98: Database servers for the local databases on a Windows 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 SQL 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 [-bcdekstwA] [-l chan] [database]
```

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

MIMCONTROL command-line arguments

Unix-style	VMS-style	Function
-b	/STATUS/DCL	Output status information about the specified database server as a single-line list suitable for use in a script. For details about the output string resulting from this option see Section 4.2.2 .
-c	/STATUS	Output status information about the specified database server. This option can be combined with the -s option (see Section 4.2.3).
-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 4.2.3).
-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 4.2.3).

Unix-style	VMS-style	Function
-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 4.2.3).
-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 4.2.3).
-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).
<i>database</i>	<i>database</i>	Specifies the name of the database to access. If a database name is not specified, the default database will be controlled. The default database is determined by the setting of the MIMER_DATABASE environment variable. The DEFAULT setting in SQLHOSTS is not used for MIMCONTROL.

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 SQL VMS Guide* for more details.

Win

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

4.2.2 MIMCONTROL (/STATUS/DCL or -b)

The “`mimcontrol -b`” (“MIMCONTROL/STATUS/DCL”) command is a special form of the “`mimcontrol -c`” (“MIMCONTROL/STATUS”) command which returns the database server status information in the form of a single string containing a comma-separated list which is useful when writing scripts.

Unix

On Unix, the string returned from “`mimcontrol -b`” can be piped and processed as required. The Unix command `cut` can be used to extract the list elements, e.g. the following command which will print the second list element:

```
# mimcontrol -b | cut -f2 -d ','
```

VMS

On VMS, the MIMCONTROL command is silent and sets the DCL symbol MIMER_STATUS to the value of the status string. The lexical function F\$ELEMENT() can be used to extract the list elements.

Win

On Windows, the string returned from “mimcontrol -b” can be piped and processed as required, depending on the script language being used.

The status string has the following components:

`server-state, logins, db-directory, connections, server-pid, start-date`

Each component is described below:

server-state

The *server-state* value shows the state of the database server. The value is one of “stopped”, “starting”, “running”, “stopping” or “crashing”.

logins

The *logins* value is either “enabled” if new logins are permitted or “disabled” if the server has been ordered to reject new logins.

db-directory

The *db-directory* field shows the home directory for the database.

connections

The *connections* field shows the number of clients connected to the server.

server-pid

The *server-pid* field gives the process id of the database server process.

start-date

The *start-date* field gives the date and time when the database server was started.

If the database server is not operational, the status string will contain only the first three fields.

4.2.3 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):

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

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

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

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

- 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
```

- 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
```

- 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
```

4.2.4 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.

The following return codes are used:

Unix/Windows	VMS	Usage
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 SQL database server which is useful for system control, system tuning and troubleshooting analysis.

Information can be generated from an active Mimer SQL 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.
- Performance report this provides information useful for monitoring performance parameters (MIMSERV).
- Bufferpool report this produces a report which is useful to Mimer Support personnel when investigating system problems (MIMDUMP).
- SQLPOOL report display SQLPOOL parameters

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

MIMINFO command-line arguments

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 Performance report (MIMSERV)
-m	/MIMDUMP	Produce Bufferpool report (MIMDUMP)
-s	/SQLPOOL	Display SQLPOOL parameters
<i>database</i>	/DATABASE = database	Take information from the specified database. If a database name is not specified, the default database will be accessed (see Section 3.7.2).
-f	/FILE	Take information from a dump file (for a users list, a dump file called "sqlpool.mdmp" is expected to exist otherwise a dump file called "bpool.mdmp" is expected to exist)



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 SQL VMS Guide* for more details.

Win

The Unix-style command-line flags can be used if the miminfo program is run from a Command Prompt window. The shortcut MIMER Info can also be used to run the program and interactive selections can then be made in the program.

A detailed description of each of the MIMINFO reports follows.

4.3.2 The users list

```
miminfo [-o file] -u [database] | -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, the channel number used by the connection, the state of the connection, transaction number, the name of the operating user, the network communication protocol (or 'local') and node identification information for connected machine. The channel number may be used in conjunction with MIMCONTROL to kill a user.

The following is an example of a users list report:

Username	Channel	State	Trans. no	OS user	Prot	From
SYSADM	16387	Busy	3		TCP	204.71.200.67
SYSADM	16388	Busy		SARA	Local	00019120

Total of 2 users

4.3.3 The Performance report

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

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

The Performance 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 Performance report is used as an aid to system tuning, it is important that the report is 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 Performance 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 is usually made automatically. It can be made manually 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 (see [Section 4.4](#)) 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 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 buffers per sorter

Total number of Mimer SQL 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 page partitions

Each region in the bufferpool is divided into separate partitions. Each partition can be accessed concurrently by the Mimer SQL 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 buffer region since the 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. (For a recently started system this figure may be higher than 2% due to initial bufferpool read operations).

No. of pages swapped out

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

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

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

No. of read commits

Total number of successful read-only transaction commits since the 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 the 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 that the background thread is flushing one or more transactions to disk and "unused" means that the background thread is allocated but not currently running (i.e. the thread is not started or closed down).

Trans-no

The number of the transaction currently being processed.

Trans-count

The number of transactions processed by the background thread.

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 DATABANK_SYSID and DATABANK_SEQNO in the data dictionary table SYSTEM.DATABANKS.

Type

The databank option NULL, TRANS, or LOG. (see [Section 5.2.5](#)).

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.

DB-Check

The DB-Check field indicates the progress of a databank check. The possible values are "Init", "Working" (foreground processing, typically index check), "Wait B." (foreground ready, waiting for background entrance), "Backgr." (background processing), "Aborting", "Aborted" or "Complete".

Note: After the DB-Check field, a field for additional information may show up. The values here can be the shadow state "offline", or the online backup states "backup in progress" or "backup completion".

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 3.2](#)) or possibly by a limit in the operating system.

Databank verification count

Databank verification is automatically performed when a databank is reopened without having been correctly closed. Each time this happens a log entry is written to the database server log. 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.

Running background verifications...
 Pending background verifications...

These information messages indicate the number of active databank verifications.

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 3.2](#)).

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 3.2](#)).

4.3.3.1 Performance report example

The following example output is from a Mimer SQL 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 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 (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.

```
# miminfo -p hotel
M I M E R / M I M I N F O
Version 8.2.1A Oct  9 2000

*** MIMER/DB runtime statistics 2000-10-18 15:48:12 ***

Hardware, operating system UNIX 32
Current MIMER/DB version is 8.2.1A Oct  9 2000
Database name: HOTEL
System started at 2000-10-18 15:47:54
System status is up

Error count                :          4
No. of request threads    :          2
No. of background threads :          2
No. of I/O threads        :          0

Page management statistics
=====
No. of pages written to disk :      1774
No. of file extend operations :          0
```

Buffer size 2K

```

-----
No. of page buffers           :      2614
No. of page buffers per sorter :      135
No. of remaining sorters      :         2
No. of page partitions        :         9
No. of page requests          :     40859
No. of page faults            :      432    1 %
No. of pages swapped out      :         0
    
```

Buffer size 16K

```

-----
No. of page buffers           :      164
No. of page buffers per sorter :       72
No. of remaining sorters      :         2
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           :        54
No. of page buffers per sorter :       17
No. of remaining sorters      :         2
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
          253 Inactive slave           0      11985
          267 Inactive                 0      1394
    
```

Pending background thread requests : 0

Databank statistics

```

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

```

No. of databanks currently open : 6
Max number of databanks open concurrently: 100
Databank verification count : 0
Running background verifications : 0
Pending background verifications : 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 Bufferpool report

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

The Bufferpool report (MIMDUMP) is used by Mimer Support personnel for trouble-shooting when database problems are reported by customers.

4.3.5 SQLPOOL report

```
miminfo [-o file] -s [database] | -f
```

A SQLPOOL report is useful when the server is malfunctioning due to a memory problem. SQL pool memory allocated is the amount of memory allocated from the operating system for the SQLPOOL. A part of that memory is in use by the server and is displayed on the row SQL pool memory used.

The following is an example of a SQLPOOL report:

```
SQLPOOL report
=====
SQL pool memory allocated (KB):      1656
SQL pool memory used (KB):          554
```

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/2000: Database server events are logged in the EventLog which may be examined using the Windows NT event viewer.

Windows 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.

4.5 Runtime malfunctions

In addition to the database server log, some database server errors are recorded in the database server log file. 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 SQL.

4.6 Several installations on one machine

Unix

Under Unix, a host computer may have several Mimer SQL installations, of the same and different versions, installed simultaneously.

If several Mimer SQL version 8 installations are available, only one of them can be linked to /usr/lib and /usr/bin at the same time. To access an installation that is not linked to these locations, the environment variables PATH and LD_LIBRARY_PATH (or corresponding to located shared libraries) must be used explicitly.

VMS

Under VMS, a host computer may have several Mimer SQL installations, of the same and different versions, installed simultaneously.

Win

Only one Mimer SQL installation can exist on a computer running Windows.

5 BACKUP AND RESTORE

This chapter discusses backup and restore of Mimer SQL databanks. Two types of backup procedures are described:

- **System Backup**, i.e. backing up the databank files from the host operating system. When using host operating system tools for doing databank file backup, the database server must be stopped in order to keep the database consistent.
- **Online Backup**, i.e. using the SQL system management statements. The main advantage of online backup is that the database server can continue to operate (backup operations are performed in the background).

Some of the discussion in this chapter refers to Shadowing (see [Chapter 8](#)), which is an optional Mimer SQL facility that allows one or more copies of a databank to exist on different disks. Shadowing provides a high level of protection from disk failure because the system will automatically use a databank shadow if the master databank is lost, thus allowing normal database activity to continue without interruption. Databank shadows also allow a copy of a databank to be temporarily set offline (e.g. to be backed up) without interrupting normal system use.

Several references to transaction handling are made in this chapter. If you are not familiar with transaction handling in Mimer SQL 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.

The backup and restore functionality in the UTIL program is still supported for backward compatibility, but is not recommended for production systems.

5.1 Background information

A Mimer SQL 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.

Note: 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.

5.1.1 Database consistency

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

Physical consistency means that the tables are readable by Mimer SQL. 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 SQL's transaction handling. All transactions are saved in the TRANSDB databank during build-up and are applied to 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 cannot commit a non-read only transaction which has read data that is being concurrently 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 SQL 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 for automatic completion. This happens in the background 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 7](#)) can be used to open all databanks in one operation and thus achieve transaction consistency quickly.

5.1.2 LOGDB and TRANSDB importance

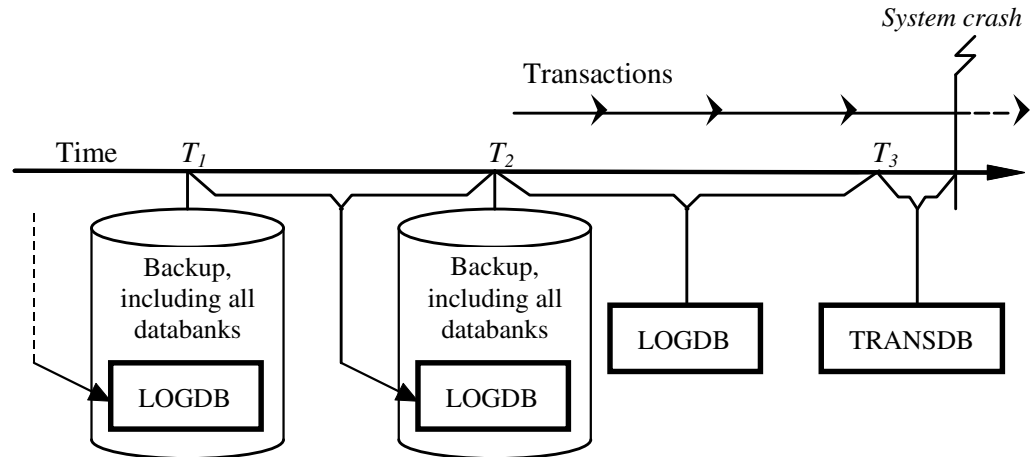
The transaction information in the LOGDB system databank is vitally important because it is used in the event of a disk crash to fully recover a destroyed databank. The LOGDB information will contain data on all the changes made to the databank from the time the backup copy of the databank was taken until the time of the disk crash. This information is used if a backup copy of a databank file is to be restored.

Note: Data changes that are not logged cannot be restored by this process, therefore it is important to consider the issue of transaction logging carefully.

If a databank becomes unavailable (because the Mimer SQL system is stopped 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 when the databank becomes available again. 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.

It is recommended practice to back up all the databanks of the database at the same time, and to ensure that LOGDB is always backed up whenever other databanks are backed up, because the LOGDB information provides the transaction data which links the previous backup copy of a databank with the databank as it exists at the current point in time.

Thus, when restoring a databank it should be brought to a state consistent with the latest backup. This is done either by using the latest backup copy of the databank or by using backed up LOGDB information with an older backup copy of the databank. The current LOGDB system databank is then used to restore the final changes made between the time of the latest backup and the time the databank was lost.



This picture describes a scenario which ends up in a system crash. To recover from this situation the common operation is to start from the most recent backup (T_2) and then use the current LOGDB to recover data up to the state at T_3 . When the system is restarted, the current TRANSDB is used to automatically recover up to the moment of the crash. If the most recent backup cannot be used, an older backup has to be brought in (T_1). This backup is restored up to the consistent state at T_2 by using the LOGDB stored in the backup at T_2 . Then the current LOGDB and TRANSDB are used to restore the transactions committed after the backup at T_2 .

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. LOGDB and TRANSDB should always be located on different physical disks which are ideally 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. (Refer to [Section 2.4.4](#) for more details on data security and databank files.)

5.1.3 Updates recorded in LOGDB

The LOGDB system databank contains logged update information for each databank with LOG option.

It is recommended that all databanks, including LOGDB, are backed up at the same time and that LOGDB is cleared after backup by resetting the log. Thus, the backed up LOGDB will contain the information required to make databanks from the preceding backup consistent with the current backup.

This will provide double backup protection by allowing a lost databank to be recovered in one of the two ways listed below:

- restore the databank from the most recent backup and apply the updates currently held in LOGDB, or
- restore the databank from an earlier backup, then sequentially use the LOGDB files from each subsequent backup to make the databank consistent with the most recent backup, and finally apply the updates currently held in LOGDB.

The records in LOGDB should be cleared **after** a complete backup, in order to maintain consistency between the backup and LOGDB. This ensures that LOGDB only contains information about changes made to a databank since the last backup of it was taken. (It is possible to backup databanks without clearing LOGDB records, although care must be taken as this leaves the backup and LOGDB in an inconsistent state).

The ability to restore databank updates from a backup copy of LOGDB replaces the databank 'incremental' backups which were supported in previous versions of Mimer SQL. These are still supported for backward compatibility but it is now recommended that LOGDB backups always be taken to offer the same double protection.

Caution! If, for any reason, the LOGDB databank is lost, 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 of all databanks 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 LOGDB was destroyed.

5.1.4 TRANSDB considerations

TRANSDB requires backup protection since it nearly always contains unfinished transactions. If TRANSDB is lost before the Mimer SQL system is restarted, the database will be left in a logically inconsistent state.

Possible effects of losing TRANSDB before the database server is restarted are described in the following scenarios:

- If TRANSDB is lost, some of the databank updates that apply to the most recently committed transactions may have been made while others remain unfinished.
The only safe operation to do to avoid a logically inconsistent database is to bring up the most recent backup copy and restore from LOGDB. In this case, the only loss is those transactions that were not completely written to LOGDB.
- If both TRANSDB and LOGDB are lost, the restoration, as described in the previous bullet, cannot be accomplished.

In the case where a restore is not possible, the best solution is to repair the inconsistency immediately after restarting the database server. This is done by using a tool such as BSQL for manual verification and update of data. This is usually possible if the user who initiated the interrupted transaction can be identified and contacted. (Many applications maintain a parallel audit log file for tracking purposes which can be used as a basis for repair work).

An alternative solution if both LOGDB and TRANSDB are lost, is to start over from the most recent backup of your databanks and reprocess all transactions since that time. This may be a 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.

A TRANSDB shadow is another possible security enhancement (see [Chapter 7](#)).

Note: The TRANSDB system databank must **never** be deliberately deleted, because uncompleted transactions nearly always remain saved in the databank even if the database server is currently stopped. If a TRANSDB file containing uncompleted transactions is deleted, inconsistency will occur because the information required to complete those transactions when the database server is re-started will have been lost.

5.1.5 SQLDB considerations

The contents of SQLDB is transient, so this databank does not need backup protection.

However, it may be convenient to have SQLDB included in the backup so that a complete system can be restored easily, without any additional operations to recreate an empty SQLDB.

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 (see [Section 5.2.5.3](#)).

5.1.6 Databank backups

A databank backup is a copy of the databank file.

A databank backup 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).

The backup can be taken either by using the Mimer SQL system administration statements for online backup (see [Section 5.2.1](#)) or by using the host file system utilities in a system backup (see [Section 5.2.2](#)).

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 for online backup are used.

The DBC program (see [Chapter 6](#)) should be executed for each databank in the backup operation in order to validate the physical consistency of the databank.

For a system backup, the backup copies of the databanks should always be taken when the system is closed and the databanks are in a logically consistent state. That is, no uncompleted transactions should exist and all databanks should be backed up at the same time to safeguard database consistency.

5.1.7 System vs. online backups

The main advantage of online backup is that all databanks, including the system databanks, can be backed up while the system can remain operational. The backup is initiated and executed by use of SQL statements only. The disadvantage can be that there must be enough disk space available to copy the complete database.

If the disk space is limited, a system backup can be preferable. For a system backup, the database server must be stopped. A system backup needs certain SQL statements to be used together with operating system commands for file copying, etc.

5.1.8 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.

Online backup commands

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

START BACKUP	starts a backup transaction and clears the log records for the databank backups taken within it.
CREATE BACKUP	creates a backup within a backup transaction. By default an online backup is created, but optionally an exclusive backup can be initiated, which will lock the databank from other users.
COMMIT BACKUP	commits a started backup transaction.
ROLLBACK BACKUP	aborts a backup transaction and ensures that all log records are preserved.

To use these statements to take a databank backup, the user must either be the creator of the databank, or have BACKUP privilege.

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

The use of a backup transaction ensures that the backups taken within the transaction are consistent with one another, as each backup is effectively taken at the same point in time. Log records are cleared for successfully backed up databanks when the backup transaction is committed. If LOGDB is included in the backup transaction all log records are cleared.

Online/offline commands

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

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 DATABANK OFFLINE	sets a databank offline and the databank pages are updated with all changes made by committed transactions so far. The databank file is closed (except SYSDB, which always remains open as long as the database server is running) so the file can be copied, and it becomes unavailable to database users.
SET DATABANK ONLINE	sets a databank online, making it available, optionally clearing records from LOGDB.
SET SHADOW OFFLINE	sets a list of shadows offline, making them unavailable.
SET SHADOW ONLINE	sets a list of shadows online, making them available, optionally clearing records from LOGDB.

A user setting the database online/offline, must have BACKUP privilege and must be the only user accessing the database.

A user setting a databank or a shadow online/offline, must either be the creator of the databank or have BACKUP privilege.

Restore command

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

ALTER DATABANK RESTORE used to restore a databank from a backup copy by using a LOGDB backup and/or the information currently in the LOGDB system databank.

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

5.2 Backup and restore of databanks

This section describes procedures for taking databank backups and for restoring a databank in the event of it being damaged or destroyed.

Refer to the [Mimer SQL Reference Manual](#) for a full description of the SQL system management statements.

5.2.1 Online backups using the SQL statements

The procedure for taking databank backups using the SQL system management statements is detailed below.

A CREATE BACKUP statement is executed for each databank to be backed up and databank consistency is ensured by starting a backup transaction using the START BACKUP statement.

The backup transaction is committed by using the COMMIT BACKUP statement, which will perform the backup and clear the relevant LOGDB records. The ROLLBACK BACKUP statement can be executed to abort the backup transaction, which will preserve LOGDB.

Note: The databank check functionality (the DBC program) should be run before archiving the backup copies of the databank files (e.g. copying them to tape) to verify the physical integrity of the databank files.

To backup databanks online, do the following:

1. Perform SQL statements for initiating and executing the backup.

```
SQL> START BACKUP;
SQL> CREATE BACKUP IN 'backup-file-name' FOR DATABANK databank-name;
.
.   (repeat for each databank to be backed up)
.
SQL> CREATE BACKUP IN 'backup-file-name' FOR DATABANK logdb;
SQL> CREATE BACKUP IN 'backup-file-name' FOR DATABANK sysdb;
SQL> CREATE BACKUP IN 'backup-file-name' FOR DATABANK transdb;
SQL> CREATE BACKUP IN 'backup-file-name' FOR DATABANK sqldb;
SQL> COMMIT BACKUP;
SQL> EXIT;
```

2. Verify the backup copies from the operating system command line using the DBC program.

```
dbc backup-file-name report-filename
.
. (repeat for each backup file created above)
.
```

3. Archive the verified backup copies (e.g. copy to tape).

The START BACKUP statement will start a backup transaction which will ensure that all the backups taken are consistent with one another (they are effectively backed up at the same point in time).

The CREATE BACKUP statement will only create an empty backup copy file. The entire contents of the specified databank file is copied to the specified file by COMMIT BACKUP.

The COMMIT BACKUP statement will clear all the LOGDB records that apply to the databanks backed up in the backup transaction. (If any of the backups fail, the ROLLBACK BACKUP statement can be executed to ensure that the log records are preserved.)

Databank backup file names are subject to the same restrictions that apply to the SQL statement CREATE DATABANK - see the [Mimer SQL Reference Manual](#).

5.2.2 System backups using the host file system

The procedure for taking databank backups using the host file system is detailed below.

It is recommended that a backup of all databanks, including SYSDB, LOGDB, SQLDB and TRANSDB, always be taken.

Note: The database server must be **stopped** in order to close the SYSDB databank file for a host system backup. This unlocks SYSDB and ensures that no operations are performed between taking copies of the databanks and dropping the log. (However, if using shadowing, databank shadows allow a copy of a databank to be temporarily set offline, e.g. to be backed up, without interrupting normal system use.)

To backup databanks using the system backup method, do the following:

1. Set the database offline using the following command:

```
SQL> set database offline;
```

2. Stop the database server so that the system databanks are closed and can therefore be backed up

```
mimcontrol -t database
```

3. Run the DBC program on each databank to verify the physical integrity of the databank files

```
dbc databank-file report-file-name
```

4. Perform the backup, e.g. copy all databank files to tape (including the system databanks SYSDB, LOGDB TRANSDB and SQLDB).

5. Start the database server

```
mimcontrol -s database
```

6. Set the database online again using the following command to clear all log records:

```
SQL> set database online reset log;
```

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 the backup **fails**, the PRESERVE LOG option should be used when setting the databank online to leave LOGDB unaltered.

It is essential that all databanks are backed up at the same time to ensure logical consistency between them.

It is also important that transactions are in a consistent state which is ensured by using the SET DATABASE OFFLINE statement. The statement 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 all background processing done by the database server has completed.

5.2.3 Restoring a databank

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

Note: Data need not be restored in the event of a power failure or system shutdown 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.

Any databank restore operation must start with a backup copy of the databank file that is not damaged or corrupt. This is generally the copy taken during the last backup, either taken by the host operating system or by using the SQL system management statements for online backup. Usually, the host file system is used to copy the backup file from the backup media to disk. The file is generally placed in the normal location for the databank file (as recorded in the data dictionary, SYSDB). However, in certain circumstances it may be necessary to place it in an alternative location, e.g. if the disk is unavailable.

The procedure for restoring a databank is as follows, please note that step 2 and 3 are only required during certain circumstances:

1. Bring a valid backup copy of the databank from the backup media to disk.
2. If the file has been placed in a location that is different to the location of the original databank file, alter the databank to reference the new file location using the following command:

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

3. If restoring from an older backup, i.e. not the latest one, information should be restored from the LOGDB included in the following backup (that was taken **after** the time the backup restored in step 1 was taken). For each LOGDB backup file, the information recorded in it should be applied to the databank using the following command:

```
SQL> ALTER DATABANK databank-name RESTORE USING 'logdb-backup-file-name'
```

4. Finally, apply the updates made since the most recent backup(s) restored in the preceding steps were taken. These updates are currently recorded in LOGDB and they are restored using the following command:

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

5.2.4 Restoring SYSDB

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

If SYSDB is lost or corrupted, a 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 LOGDB information. This is done using the backup and restore functionality in the UTIL program. Start the program and login as SYSADM, or another user with BACKUP privilege. 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
```

5.2.5 Re-creating TRANSDB, LOGDB and SQLDB

No Mimer SQL applications can be run if LOGDB, TRANSDB or SQLDB is missing. In this event, starting the UTIL program and logging in as 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 recorded filenames in the case where the physical files were moved.

The following example shows how to re-create LOGDB for a database where this system databank is missing (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

```

If a database has been operational for some time, a situation may arise where one or more of the system databanks LOGDB, TRANSDB or SQLDB has grown so large that it is desirable to delete the file and create a new empty databank. But, since these databanks (at least TRANSDB and LOGDB) are so vital to the system consistency it is a strong recommendation that these files are kept intact whenever possible.

A complete backup of the entire database should be made before any system databanks are recreated.

The following sections describe how to re-create each of the respective system databanks.

5.2.5.1 Creating a new LOGDB

The following steps should be followed to create a new LOGDB:

1. Shut down the database server (if not already stopped).
2. Run dbcheck on SYSDB and all the user databank files to ensure that none are corrupted.
3. Take a valid backup of the whole database.
4. Archive a copy of the LOGDB databank file and delete the original file from disk.
5. Start the database server.
6. Start the UTIL program, logging in as SYSADM, and when prompted, select the CREATE option and specify a pathname and a size for the new LOGDB databank file.

5.2.5.2 Creating a new TRANSDB

The following steps should be followed to create a new TRANSDB:

1. Shut down the database server (if not already stopped).
2. Ensure that all pending transactions have been flushed to the user-databank files on disk by successful execution of DBOPEN in single-user mode.
3. Archive a copy of the TRANSDB databank file and delete the original file from disk.
4. Start the database server.
5. Start the UTIL program, logging in as SYSADM, and when prompted, select the CREATE option and specify a pathname and size for the new TRANSDB databank file.

5.2.5.3 Creating a new SQLDB

The following steps should be followed to create a new SQLDB:

1. Shut down the database server (if not already stopped).
2. Delete the SQLDB file from disk.
3. Start the database server.
4. Start the UTIL program, logging in as SYSADM, and when prompted, select the CREATE option and specify a size for the new SQLDB databank file.

6 DATABANK CHECK FUNCTIONALITY

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 internal organization of a databank file and display statistical information on the databank.

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.

6.1 DBC syntax

The overall syntax for the DBC program is:

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

DBC command-line arguments

Arguments	Function
<i>databank_filename</i>	Filename for the databank to check. The default filename extension is “.dbf”.
<i>result_filename</i>	Sequential file created by DBC that contains the result of the verification. The default filename extension is “.dbc”. If no result filename is specified, output is written to the screen.

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.

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 following message is shown:

```
No errors found
```

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

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

The result file should be examined to investigate the nature of the errors (see [Section 5.2.2](#)).

It should be noted that the DBC 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.

6.2 Functions

When performing a backup, it is recommended that DBC is used to check the validity of the backed-up databanks. If errors are detected, the databank can be restored from the previous backup copy.

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 SQL 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.

6.3 Authorization

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

6.4 Result file contents

Any errors detected in the databank file are written to the result file directly after the identification record for the table affected.

The result file begins with the following information:

- Databank file name.
- Time when the DBC operation was performed.
- Version of Mimer SQL under which the databank was created (if this is not the same as the current version there will also be a “converted to” message).
- A backup timestamp.
- Structure level.
- System identifier for the databank.
- Databank sequence number (0 = master databank, >0 = a shadow).
- Check flag.
- Number of bitmap pages (starting at 0).

- Root pages (starting at 1).
- Number of pages allocated.
- Number of pages used.

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

* No bitmap checking against databank!

If the check flag indicates that the databank was not properly closed, the database server will not allow the databank to be accessed and it will, therefore, not be possible to perform any operations.

DBC Table Information

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.

DBC Table Information for LOGDB, TRANSDB and SQLDB

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.

6.4.1 Example of result file

The following example illustrates the result of running DBC on the HOTELDB databank file. The fact that the file was not properly closed is flagged.

```
MIMER Databank Check Utility
Version 8.2.1

Databank file: HOTELDB
Time: 2000-06-23 16:29:09

Version created:      812 Backup timestamp:   1
Structure level:     6 System ident.:      269
Check flag: Databank not properly closed

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
.
.
.
.
No errors found
```

6.4.2 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. In the example below a y-value indicates the byte position in the page (corresponding x-value) where the record holding the reference to the next level starts:

```
B*-tree
Page no:   x1   x2   x3   ....  xn
Byte pos:  y1   y2   y3   ....  yn
```

If the error occurs at an index level, the following additional message is given, and no checks are made at lower levels:

```
Branch is interrupted
```


7 DBOPEN FUNCTIONALITY

The DBOpen functionality is used to make sure that the users can open all databanks quickly. It can take relatively longer to open a databank if it is a large databank that was closed abnormally (for example if the machine crashed) because a databank check is automatically initiated following such a situation.

Note: When `dbopen` is run in multi-user mode, the index pages of the databank are checked before databank access can proceed, but the bulk of the databank contents (contained in the data pages) is checked in the background, thus minimizing the delay.

The checking performed by `dbopen` is performed by the background threads (see Section 4.1.3), therefore increasing the number of concurrent background threads available for use will increase the efficiency of the checking.

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

7.1 DBOpen syntax

The overall syntax for DBOpen is:

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

DBOPEN command-line arguments

Unix-style	VMS-style	Function
<code>-m</code>	<code>/MULTI</code>	Connects to the database in multi-user mode.
<code>-s</code>	<code>/SINGLE</code>	Connects to the database in single-user mode.
<i>database</i>	<i>database</i>	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.

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 SQL VMS Guide* for more details.

Win

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

Note: 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.

7.2 Functions

The DBOPEN functionality opens all available databanks in a system. 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.

7.3 Authorization

Any user who has SELECT access to the data dictionary table SYSTEM.DATABANKS can run DBOPEN.

7.4 DBOPEN output example

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

```
MIMER Databank Open Utility

Username: SYSADM
Password:

Opening databank SYSMSG
Opening databank SYSQL
Opening databank SYSRG
Opening databank SYSRGOUT
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
```


8 SHADOWING FUNCTIONALITY

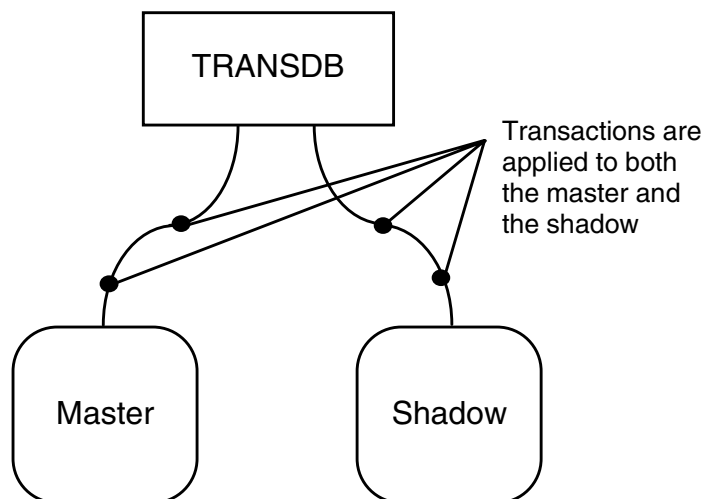
The Shadowing functionality makes it possible to create and maintain one or more simultaneously updated copies of a databank. This allows for a higher degree of data availability by giving extra protection from disk crashes, etc.

This chapter describes the functions and benefits of databank shadowing, how to use the shadowing functionality, and how to handle any problems.

8.1 General description

8.1.1 Databank shadowing

Databank shadowing means updating one or more copies of a databank simultaneously. The master is the “normal” databank file which is accessed for data storage and retrieval. The copies are called shadows. Any changes to the master data is automatically made to the shadows, thus protecting data from a disk crash or other event that might cause a databank to be lost. A databank must support transaction control (see Section 2.4.2) to be shadowed.



All changes made to a shadowed databank are automatically made to all the shadows for the databank.

If a master databank is lost, a shadow will automatically take over from the master and operations can be resumed immediately (assuming the shadow is not also damaged in the disk crash). A shadow can be transformed to a master databank to permanently replace it and this process is much faster than restoring a databank from a backup copy.

It is recommended that conventional backups be taken as a supplement to databank shadows, to protect data in the event of a crash that destroys both the master and the shadow.

Because operations are not interrupted when backups are taken, and because Mimer SQL databanks are automatically reorganized, true 24 hour-a-day operation is possible.

Databank shadowing is entirely invisible to the application. This means that shadows can be added to existing applications. No special handling is needed to access tables in a shadowed databank.

8.1.2 Different levels of data protection

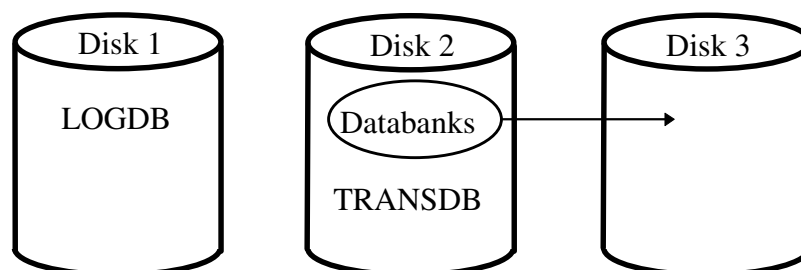
[Section 5.1](#) describes the role of LOGDB and TRANSDB, when used in conjunction with backup and restore, in protecting data against loss. Databank shadowing provides an even higher level of protection. Listed below are the different ways in which data can be protected from loss (from the least amount of protection to the highest):

- **All databanks on one disk and no logging**

If a databank is lost with this level of protection, it is only possible to continue operations from the last backup copy (all changes since the last backup was taken are lost). Databanks can be lost due to accidental deletion, disk crashes (which can destroy all files on a disk), etc. This level of protection is not recommended except for “trash” databanks with unimportant contents.

- **Logging, with LOGDB and TRANSDB on a separate disk from the data**

LOGDB and TRANSDB are vital databanks if the system stops or if any databanks are lost. Because of this, LOGDB and TRANSDB should be stored on separate disks, as shown in the following figure. Application data should be stored on the TRANSDB disk if it cannot be stored separately.



If a databank is lost, it can be restored to the state it had when it was lost by applying the transactions in LOGDB and TRANSDB to a restored backup copy of the databank. This may take some time, especially if the databank is large and if there is a lot of transaction information stored in LOGDB.

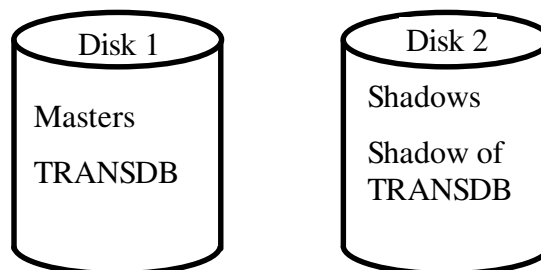
Warning! If the databank disk and the TRANSDB/LOGDB disk are handled by the same disk controller, a disk controller failure may cause both disks to crash. If this happens, the databanks can only be restarted from the state of the last backup copy. Therefore, it is advisable to use separate disks with separate disk controllers.

This security level gives a high degree of security and is recommended for databanks containing important data used in a system where the delay before the system is restored after a crash is not critical. To assure this high degree of security the backup copies should always be stored on separate removable media (e.g. tapes).

- **Shadowing, with shadows on a separate disk**

Shadows should always be stored on a separate disk from the masters to protect them from a total disk crash that could destroy both the master and shadow databanks. It is also advisable to use separate disk controllers to assure that a corrupt disk controller does not destroy the disks holding both the masters and the shadows.

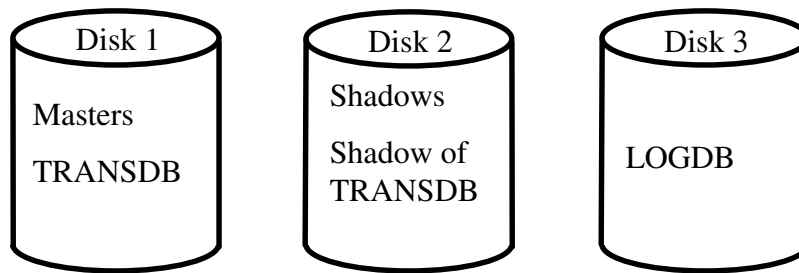
If a databank is lost, its shadow can be transformed into a master and the shadow automatically takes over with no loss of data. Since the shadows are updated after the master, and the operations are saved in TRANSDB until the shadow is updated, it is important to have a good copy of TRANSDB when a shadow is transformed. To assure this, it is advisable to shadow TRANSDB. It is strongly recommended that TRANSDB and its shadows are stored on separate disks, as shown in the following diagram.



This arrangement gives a high degree of security and is recommended for databanks containing important data used in a system where it is vital to be able to get the system running again quickly after a disk crash.

- **Shadowing and logging**

Combining shadowing and logging (see [Chapter 5](#)) gives the highest level of data protection. If logging is not used, the data is not protected if both the master and the shadow disks become corrupted. However, when shadowing is combined with logging (with LOGDB on a third disk) and backups are regularly taken onto separate media (tape, etc.), then data is always protected if any two of the disks crash.



Of course, additional disks can be used, just as long as the databanks that are separated above are not placed onto the same disks. If you only have two disk drives available and all the databanks are shadowed, then logging is of little value. Shadowing of LOGDB will not increase data protection significantly in this configuration.

Different degrees of data security can be used for different databanks, depending on the importance of the data. It is however important that all inter-dependent databanks (because of foreign key relationships, etc.) have the same level of protection. Otherwise logical inconsistencies may result if there is a disk crash.

8.2 Shadowing management

All shadowing management operations can be handled from the shadowing utilities in the UTIL program (see [Section 8.2.1](#)) or by using the SQL commands: [CREATE SHADOW](#), [DROP SHADOW](#), [ALTER SHADOW](#) and [SET SHADOW](#) (described in the [Mimer SQL Reference Manual](#)).

8.2.1 The shadowing utilities

This section describes the options of the shadowing utilities in the UTIL program, which contains functions for the maintenance of a shadowing system. These utilities are used to create or drop a shadow, transform a shadow to a master, list information about shadows and databanks, and enter or leave a program ident. The shadowing utilities main menu looks like this:

```
-- SHADOW utilities --
1. Create shadow
2. Drop shadow
3. Transform shadow to master
4. List shadowing information
5. Enter program ident
6. Leave program ident

0. Exit
```

8.2.1.1 Authorization

Any user can start the UTIL program. The separate functions are restricted by special privileges. The following table shows the privilege that a user needs to have to do specific tasks:

<i>Tasks</i>	<i>Privilege</i>	
	BACKUP	SHADOW
Create a shadow ^{*)}		X
Drop a shadow ^{*)}		X
Transform a shadow ^{*)}		X
List shadow info ^{**)}	X	X
Backup and restore ^{**)}	X	
Set shadow offline ^{**)}	X	
Set shadow online ^{**)}	X	

^{*)} The user must have exclusive use of the databank. This means that no other user can access the databank until the operation is finished.

^{**)} A user who has SHADOW or BACKUP privilege can list shadowing information for any or all databanks. Users can always backup and restore, set shadows offline and online, and list shadowing information for any databank that they have created.

SYSADM is initially granted SHADOW and BACKUP privilege with the WITH GRANT OPTION.

8.2.1.2 Wildcards

Where noted in the following descriptions, wildcard characters following the SQL standard can be used in databank names. The percent character (%) represents one or more characters, and underscore (_) represents one and only one character. Thus DB% means every databank name beginning with DB, and DB_ specifies all databank names that consist of the letters DB followed by one (and only one) alphanumeric character or blank space. A percent character alone is shorthand for all databanks (excluding TRANSDB).

8.2.1.3 Create shadow

When the “Create shadow” option is chosen, you are prompted for the name of the databank to be shadowed, the name for the new shadow, and the filename for the new shadow. Databanks can have more than one shadow.

Databanks with NULL option cannot be shadowed, because shadowing requires transaction handling. The databank to be shadowed cannot be used by any other users while a shadow is being created.

The shadow name cannot be the same as the name of the master databank, of any other shadow, or of any shadow that has been transformed to a master.

Creating a shadow for a large databank may take some time, and thus should be carefully planned.

Example (user input is shown in bold):

```

-- SHADOW utilities --

1. Create shadow
2. Drop shadow
3. Transform shadow to master
4. List shadowing information
5. Enter program ident
6. Leave program ident

0. Exit

Select: 1

-- Create shadow --

Databank name          : hoteldb
Create shadow for databank HOTELDB

Shadow name           : hoteldb_s
Filename for shadow   : /mnt/hoteldb_s.sdw
Shadow HOTELDB_S created and copied from databank HOTELDB

```

A shadow databank is created and all tables and indexes in the databank are copied to the shadow.

Shadows can also be created in SQL with the [CREATE SHADOW](#) command (described in the [Mimer SQL Reference Manual](#)).

8.2.1.4 Drop shadow

When “Drop shadow” is chosen, you are prompted for the databank name for which a shadow should be dropped. Wildcard characters are allowed in the databank name. A listing of all shadows for the specified databank is displayed. Enter the name of the shadow you want to drop.

When a shadow is dropped, the file where the databank is stored is usually deleted in the operating system. Otherwise, removing the file requires that it is deleted in the host operating system.

The databank cannot be used by any other users when a shadow is being dropped.

Example (user input is shown in bold):

```

-- SHADOW utilities --

1. Create shadow
2. Drop shadow
3. Transform shadow to master
4. List shadowing information
5. Enter program ident
6. Leave program ident

0. Exit

Select: 2

```

```

-- Drop shadow --

Databank name           : hoteldb

DATABANK
SHADOW
OFFLINE
FILE
=====
HOTELDB
HOTELDB_S
N
/mnt/hoteldb_s.sdw
---
One shadow found

Name of shadow to drop (<CR> = skip): hoteldb_s

Shadow HOTELDB_S dropped

```

Shadows can also be dropped in SQL with the [DROP SHADOW](#) command (described in the [Mimer SQL Reference Manual](#)).

8.2.1.5 Transform shadow to master

If a master is lost, it is possible to continue operations by transforming a shadow to the master. This operation must be done from the Shadowing utilities menu or with the SQL [ALTER SHADOW](#) statement – it cannot be done in the operating system.

When the “Transform shadow to master” option is chosen from the Shadowing utilities you are prompted for the master databank name (wildcard characters are allowed). After the name is entered, a listing of all shadows for the selected databanks are displayed. Select which shadow to transform by giving the shadow name.

The transform operation only affects the data dictionary. The databank filenames are not changed. The old master databank file is usually deleted in the operating system. Otherwise, removing the file requires that it is deleted in the host operating system.

The databank cannot be used by any other user when a shadow is being transformed into the master (this is not very likely to happen since this function is normally used when the master has been lost or damaged).

Example (user input is shown in bold):

```

-- SHADOW utilities --

1. Create shadow
2. Drop shadow
3. Transform shadow to master
4. List shadowing information
5. Enter program ident
6. Leave program ident

0. Exit

Select: 3

```

```

-- Transform shadow --

Databank name           : hoteldb

DATABANK
SHADOW
OFFLINE
FILE
=====
HOTELDB
HOTELDB_S
Y
/mnt/hoteldb_s.sdw
---
One shadow found

Name of shadow to transform (<CR> = skip): hoteldb_s

Shadow HOTELDB_S is set online before it is transformed to master

Shadow HOTELDB_S transformed to master

```

The SQL commands [ALTER SHADOW](#) and [DROP SHADOW](#) can be used together to transform a shadow to a master.

8.2.1.6 List shadowing information

The “List shadowing information” option lists the following information for both master and shadow databanks: databank name, shadow name (if the databank is a shadow), offline status for the shadows (Y = Offline, N = Online) and the filename.

Enter the master databank’s name that you want to list the shadowing information for. Wildcard characters are allowed. If a databank is specified, shadowing information for that databank is displayed. If a carriage return (<CR>) is entered without specifying a databank, shadow information for all databanks in the system are displayed. The listing is halted when the screen is full and you can decide whether to continue the listing or not.

Example of list all shadows (user input is shown in bold):

```

-- SHADOW utilities --

1. Create shadow
2. Drop shadow
3. Transform shadow to master
4. List shadowing information
5. Enter program ident
6. Leave program ident

0. Exit

Select: 4

-- List shadowing information --

Databank name (<CR> => all databanks):

DATABANK
SHADOW
OFFLINE
FILE
=====
SYSDB
SYSDB
N
sysdb82.dbf
---
```

```

TRANSDB
TRANSDB
N
transdb.dbf
---
LOGDB
LOGDB
N
logdb.dbf
---
SQLDB
SQLDB
N
sqldb.dbf
---
BOOKDB
BOOKDB_S
N
bookdb.dbf
---
ROOMSDB
ROOMSDB
N
roomsdb.dbf
---
HOTELDB
HOTELDB_S
N
/mnt/hoteldb_s.sdw
---
CUSTOMERS
CUSTOMERS_S
Y
/mnt/customers_s.sdw
---
6 databanks found
2 shadows found

```

8.3 Backups from shadows

A backup of a databank can be taken from a shadow instead of the master databank in situations where optimal performance from the databank must be maintained or when it is desired that the databank be set offline while it is being backed up. This allows the backup process to proceed without affecting the users working with data contained in the databank.

If a shadow is set offline, the relevant transactions will be written to the online databank but remain in TRANSDB until the shadow is set online again and the transactions can be written to it.

Backups from shadows can be taken by an online backup, using the SQL system management statements.

8.4 Shadowing system databanks

The system databanks (SYSDB, TRANSDB, LOGDB, and SQLDB) require special handling in some situations. If a problem occurs with these databanks or their shadows, the only permitted login is SYSADM logging into the UTIL program. The UTIL program will then recognize the problem and help you correct it.

System databanks are handled in the same way as other databanks, with the following exceptions:

- If there is a problem with SYSDB, TRANSDB, SQLDB or LOGDB, new users cannot login. Users already active will receive an error message when attempting operations that depend on the affected system databank, while other operations continue to work. The error state is held until Mimer SQL is stopped and the error is corrected.
- If there is a problem with a shadow of SYSDB, TRANSDB or LOGDB, new users cannot login until the shadow is dropped or suspended (described in [Section 8.4.5](#)).
- No users can be connected while a shadow for SYSDB, TRANSDB, or LOGDB is created, altered or dropped.

8.4.1 SYSDB

Because the SYSDB databank holds all the data dictionary information about your database, protecting it with shadowing and/or backups is essential. Otherwise, the whole database will be unreachable if SYSDB is lost.

Transform a SYSDB shadow to a master

If SYSDB is lost or corrupt, any existing shadow of the master SYSDB can be altered to become the master in order to allow Mimer SQL to start again.

The SYSDB shadow file should be renamed and/or moved to the location where the master SYSDB was. Then the UTIL is started and login is performed as SYSADM. Enter the name of the shadow to be transformed into the master, and exit.

Example (user input is shown in bold):

```

M I M E R / U T I L

Username: SYSADM
Password:

MIMER/DB warning -18013 in function CONNECT
MIMER/DB started from SYSDB shadow. Transform SYSDB shadow to master
with UTIL, or restart system from master SYSDB

-- Transform shadow --

DATABANK
SHADOW
OFFLINE
FILE
=====
SYSDB
SYSDB_S
N
sysdb82_s
---
One shadow found

Name of shadow to transform (<CR> = skip): sysdb_s

```

```
Shadow SYSDB_S transformed to master
```

```
-- MIMER UTILITIES --
```

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

```
0. Exit
```

```
Select: 0
```

Note: If a the disk where the SYSDB file is located gets inaccessible, it may be more suitable to redefine the database home directory (to point out the SYSDB shadow) instead of restoring the original directory structure. In this case the ALTER DATABANK statement must be used for all databanks explicitly defined to be located on the halted disk, i.e. with an absolute file specification in the data dictionary.

Restore SYSDB

If SYSDB is lost and no shadows exist, a backup copy of SYSDB can be restored to allow Mimer SQL to start again (an example of how to do this is given in [Chapter 5](#)).

8.4.2 TRANSDB

Shadowing of TRANSDB assures that you can bring your database up-to-date if your ordinary TRANSDB is lost or damaged.

Transform a TRANSDB shadow to a master

If TRANSDB is lost or corrupt, an existing shadow of TRANSDB can be transformed to the master. Start the UTIL program and login as SYSADM. A message is displayed saying that TRANSDB cannot be opened, and a shadow must be transformed to the master (this is similar to the example in [Section 8.4.1](#)). If there are uncompleted transactions they will be completed, as if the “original” TRANSDB was still there.

8.4.3 LOGDB

If some databanks are not shadowed but backup copies of the databanks exist, then a shadow of LOGDB is useful if both an unshadowed databank and the LOGDB master are lost or corrupted.

Transform a LOGDB shadow to a master

If LOGDB is lost or corrupt, an existing shadow of LOGDB can be transformed to the master. Start the UTIL program and login as SYSADM. A message is displayed saying that LOGDB cannot be opened, and a shadow must be transformed to the master (this is similar to the example in [Section 8.4.1](#)). If there are transactions not yet written to the log, this will be done automatically.

8.4.4 SQLDB

Shadowing of SQLDB is not allowed because it only contains temporary data. However, SQLDB is required when a user logs on to Mimer SQL. If SQLDB is corrupt or lost, it must be recreated by logging on to the UTIL program as SYSADM (this automatically recreates SQLDB if the databank is not found).

8.4.5 If a shadow for SYSDB, TRANSDB or LOGDB is not accessible

If a shadow for SYSDB, TRANSDB or LOGDB is not accessible, SYSADM should login to the UTIL program. An error message is given followed by the option to drop the shadow or set it offline. If the shadow is corrupt or missing, it should be dropped. If it is only temporarily unavailable it may be enough to set it offline for a short period of time.

Example (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_S,
file logdb_s not found

Inaccessible shadow encountered. DROP or SET OFFLINE? (D/S): D

-- Drop shadow --

Shadow LOGDB_S dropped

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

0. Exit

Select: 0

```

8.5 Testing your data protection

So far, this chapter has described the facilities that are available for the system manager to maintain the system. But how do you know that you are using the functionality in the right way, and that you will be able to get the system running again if something happens? The answer is **practice**. When you have decided on a certain strategy, a disk crash should be simulated and the databanks restored following the chosen strategy. Check that the contents of all tables are the same as before the simulated crash. Do not forget to simulate a crash of the system databanks (SYSDB, TRANSDB, LOGDB, SQLDB) and then restore or recreate them. When you have decided on a successful strategy, then build command files (scripts) that take “normal” backups, backups on the fly, and create shadows.

8.6 Configuring the system

In [Section 8.1.2](#) the various levels of data security are described. When you configure your system, some additional considerations are:

- *How do I divide the work load over several disks to get the best performance?*
TRANSDB and its shadows should preferably be on fast disks (see [Section 8.7](#)).
- *How much disk space is required?*
A shadow file occupies the same amount of disk space as the master databank file.

A problem occurs if TRANSDB or LOGDB run out of disk space. When this happens the system cannot continue. Therefore, it is important to make sure that there is enough space for these databanks.

How large LOGDB grows depends on whether the LOG option is used and how often backups are taken and drop log is performed, since drop log clears LOGDB and releases space (but the file size remain unchanged).

How large TRANSDB grows depends on the number of transactions and how fast the shadow servers are able to perform the transactions on the shadows. When the transactions are performed on the shadows, the space in TRANSDB where they were stored is released.

- *How long does a backup on the fly take?*
During a backup on the fly one or more shadows are set offline, and transactions are buffered in TRANSDB. How large TRANSDB grows depends on the amount of time needed to backup the databanks, and the frequency of the transactions on the databanks with offline shadows.

8.7 Performance aspects of shadowing

Performance is not noticeably affected by shadowing in multi-user systems, even though the shadowing system needs more machine resources because more files need to be updated. Applications do not have to wait for the shadows to be updated because this is performed in the background. Actually all updates to the disk, even to master databanks, are performed by background processes (except for updates to TRANSDB and its shadows).

In single-user systems no background process is used. This means that an application has to wait for the shadows to be updated.

8.7.1 Tuning

If the updating of shadows is delayed it will cause TRANSDB to grow. This can happen for several reasons:

- A shadow has been set offline and forgotten. If this happens transactions will be buffered until it is set online again. To check if a shadow is offline use “list shadowing information” or the [Performance report from MIMINFO](#).
- A shadow is corrupt. Updates on the shadow results in an I/O error, and are buffered in TRANSDB. When this happens the operator is notified by the system. To check it use the [Performance report from MIMINFO](#).
- There are too few shadowing processes to update the shadows, or they get too little machine resources.

8.8 Troubleshooting

This chapter lists the error messages that may appear and the solutions to the different problems. Some of these problems have already been presented earlier in this chapter, but are combined here for convenience.

When there is a problem, first look for the cause of the error by looking in the database server log file. Then, check the following sections for a solution to the problem.

8.8.1 Operator error messages

There are four error messages that can be sent to the operator defined for the database server. The four error messages and their solutions are:

1) Fatal databank error in MIMER/DB multi-user system

Databank <databank_name> has been disabled

In this case a databank has encountered an error which precludes further transactions to be performed on the databank. A common cause for this can be disk space exhausted. The Performance report from MIMINFO [MIMSERV](#) will display an error status for the databank.

<i>Problem</i>	TRANSDB disk space exhausted.
<i>Solution</i>	Allocate more disk space or do the following: Investigate why TRANSDB grows. Start more shadow servers, if required. If there are offline shadows, set them online again.
<i>Problem</i>	LOGDB disk space exhausted.
<i>Solution</i>	Allocate more disk space or do the following: Take new backup copies of all databanks. Remove the LOGDB file. Login to the utility program and create a new LOGDB file. In the future, take backups and drop log more often to prevent LOGDB from growing too large again.

<i>Problem</i>	Disk for master databanks is corrupt.
<i>Solution</i>	Find out which databanks are on the corrupt disk. Transform shadows for corrupt databanks into masters. When possible, create new shadows on another disk (this requires exclusive access to the databank).
<i>Problem</i>	A master is lost and the shadow is suspended.
<i>Solution</i>	Transform the shadow to a master. When possible, create a new shadow (this requires exclusive access to the databank).
<i>Problem</i>	A databank needs to be restored, but LOGDB is lost.
<i>Solution</i>	For this situation there is no satisfactory solution. Use the following emergency procedure to get the system running: Move a backup copy to the location of the databank file. Take new backups of all databanks. Create a new empty LOGDB. Run the DBOPEN utility to open all databanks (this ensures that databank identifications in the new log are up-to-date). Now you can continue from the state of the backup copy. All operations after the backup was taken are lost. If two databanks depend on each other, then both must be restarted from the backups. Note that if data dictionary objects (tables, indexes, etc.) have been created in the databank after the backup was taken, it is advisable to recover all databanks from the backups, since otherwise inconsistency between the databank and data dictionary will occur.

2) Fatal databank error in MIMER/DB multi-user system

Shadow <shadow_name> has been disabled

In this case a shadow has encountered an error. This will cause all transactions performed on the databank to be buffered in TRANSDB.

The first thing that should be done is to set the shadow offline. Then investigate the cause of the error by consulting the database server log file. If the error can be corrected the error status is removed if the shadow can be successfully set online, with the **preserve log** option.

<i>Problem</i>	A shadow has run out of disk space.
<i>Solution</i>	Set the shadow offline and allocate more disk space.
<i>Problem</i>	The disk for shadow databanks is corrupt. Transactions are buffered in TRANSDB, which then grows.
<i>Solution</i>	Drop the shadows on the corrupt disk (this requires exclusive access to the databank). They will only cause TRANSDB to grow. When possible, create new shadows on another disk. Observe that “create shadow” requires exclusive access to the databank during the copying operation to the shadow so this operation will stop operations to the databank during the creation of a shadow. Plan when new shadows can be created.

3) Unrecoverable error in MIMER/DB

Multi-user system terminated

A fatal error has occurred in the database server. This error is usually due to the lack of space in internal structures in the multi-user system. Please consult the database server log file to find out the exact cause of the failure.

4) Unrecoverable internal error in MIMER/DB

Multi-user system terminated

An internal error has been detected in the database server. Store a Bufferpool report from [MIMINFO](#) to a file and then restart the database server. Report the problem to your Mimer service agent.

8.8.2 Miscellaneous problems

The following problems may also occur:

<i>Problem</i>	A restore operation was interrupted.
<i>Solution</i>	Make a new restore from the beginning. To do this you must still have the original backup copy available.
<i>Problem</i>	Operations appear to stop when you try to set a shadow offline.
<i>Solution</i>	There is sometimes a delay while synchronization is performed. Wait for the synchronization to be completed.

9 READLOG FUNCTIONALITY

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).

9.1 Functions

The READLOG 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.

The 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
```

9.2 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 entire database, the user must have BACKUP privilege.

9.3 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
0. EXIT	5. Databank	8. All (no data)	

9.3.1 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. The following examples sets the log file explicitly:

```
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
```

9.3.2 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 200011 as both the starting and finishing time, lists the log from the beginning to the end of November 2000.

A default time value is assumed if no time interval is specified, or may be chosen for starting or finishing time by specifying a "blank" time. If no start time is specified, the time at the beginning of the log is assumed. If no end time is specified, the time at the end of the log is assumed.

If neither a start time nor an end time is specified, 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: 200011
Endtime  : 200012
```

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.

If no databank is specified, the list operation is done for all databanks. If this is the case, 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
```

9.3.3 List operations

Specified tables

This option activates listing of the log for selected tables in the database. As many tables may be specified as are required, with the table name qualified, if necessary, by the name of the schema to which it belongs. If no schema is specified, the schema with the same name as the current ident is assumed.

Databank restrictions selected with option 5 are ignored if specified tables are selected. However, any ident and time restrictions selected with options 3 and 4 are applied.

The ident running the READLOG functionality must have SELECT access on the requested tables, otherwise the following message is displayed for the table in question:

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

If a non-existent table is requested, the following message is displayed:

```
** No such table
```

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, as in the following example:

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

The list operation can be interrupted by entering an exclamation mark ("!").

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 <schema-name.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 BACKUP privilege. If the privilege is not held by the current ident the following error message is displayed:

```
**      AUTHORIZATION FAILURE
```

9.3.4 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.

9.4 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: 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 “Specified tables” or “Tables in 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, “All (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 2000-05-23 10:55 and 2000-05-23 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: 200005231055
Endtime  : 200005231200

```

```

-- 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 2000-05-23 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 2000-05-23 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

618	Create table	
618	Commit	2000-10-23 16:41:35:68 by HOTELADM Read 69
815	Insert	1356, '2000-08-13', 'SKY', 'SDBLB', 'M & D AB', '0350-19738', 'ERIK', 'ANDERSSON', '2000-10-22', '2000-10-25', 'ERIKA', 'ANDERSSON', 'PARKV. 126, SVEDALA', '2000-10-22', <NULL>, 'SKY124', <NULL>
815	Commit	2000-10-23 16:41:55:22 by HOTELADM Read 3
816	Insert	1357, '2000-08-13', 'SKY', 'NSDBLB', 'M & D AB', '0350-19738', 'ERIK', 'ANDERSSON', '2000-10-22', '2000-10-24', 'FRED', 'NILSSON', 'BLOMGATAN 15B, SVEDALA', '2000-10-23', <NULL>, 'LAP201', <NULL>
816	Commit	2000-10-23 16:41:55:36 by HOTELADM Read 3
817	Insert	1369, '2000-08-19', 'WIND', 'NSDBLS', 'ALEX OLSSON', '018-298573', 'ALEX', 'OLSSON', '2000-10-23', '2000-10-25', 'ALEX', 'OLSSON', 'TORSGATAN 12, UPPSALA', '2000-10-23', <NULL>, 'WIND401', <NULL>
817	Commit	2000-10-23 16:41:55:50 by HOTELADM Read 3
818	Insert	1370, '2000-08-19', 'WIND', 'NSDBLS', 'ALEX OLSSON', '018-298573', 'ALEX', 'OLSSON', '2000-10-20', '2000-10-25', 'KERSTIN', 'OLSSON', 'TORSGATAN 12, UPPSALA', '2000-10-20', <NULL>, 'WIND402', <NULL>
818	Commit	2000-10-23 16:41:55:64 by HOTELADM Read 3
1191	Update before	1349, '2000-08-11', 'LAP', 'NSSGLS', 'MIMER AB', '018-185210', 'MATS', 'LINDBLOM', '2000-08-31', '2000-09-01', 'STEFAN', 'HANSEN', 'IDUNGATAN 24, UPPSALA', '2000-08-31', '2000-09-01', 'LAP206', 'EUROCARD'
1191	Update after	1349, '2000-08-11', 'LAP', 'NSSGLS', 'MIMER AB', '018-185210', 'MATS', 'LINDBLOM', '2000-08-31', '2000-09-01', 'STEFAN', 'HANSEN', 'IDUNGATAN 24, UPPSALA', '2000-08-31', '2000-09-01', 'LAP205', 'EUROCARD'
1191	Commit	2000-10-23 16:52:21:05 by HOTELADM Read 2
1192	Update before	1350, '2000-08-12', 'SKY', 'SDBLB', 'SALLY WEBERT', '0760-57609', 'SALLY', 'WEBERT', '2000-10-22', '2000-10-25', 'SALLY', 'WEBERT', 'KRONPARKEN 44, JOKKMOKK', '2000-10-22', '2000-10-23', 'SKY212', 'CASH'
1192	Update after	1350, '2000-08-12', 'SKY', 'SDBLB', 'SALLY WEBERT', '0760-57609', 'SALLY', 'WEBERT', '2000-10-22', '2000-10-25', 'SALLY', 'WEBERT', 'KRONPARKEN 44, JOKKMOKK', '2000-10-22', '2000-10-23', 'SKY210', 'CASH'
1192	Commit	2000-10-23 16:53:04:76 by HOTELADM Read 2

10 DATABASE STATISTICS

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 is still supported for backward compatibility.

10.1 Statistical information

The statistical information includes:

- the total number of rows in each base table, stored in the column TABLE_CARD in the table's row in SYSTEM.TABLES
- the number of distinct values in each column of a table, stored in the column COLUMN_CARD in the column's row in SYSTEM.COLUMNS
- the number of non-NULL values in each column of a table, stored in the column COLUMN_CARD_NONNULL in the column's row in SYSTEM.COLUMNS
- the lowest and highest values in each column of a table, stored in the columns COLUMN_LOW_VALUE and COLUMN_HIGH_VALUE in the column's row in SYSTEM.COLUMNS

10.2 Authorization

The user executing the SQL statistics statements must either have STATISTICS privilege or be the owner of the table(s) or ident(s) for which statistics are being collected. The database administration ident SYSADM holds STATISTICS privilege 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 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.

10.3 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.

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.

Note: The database remains fully accessible while statistics are being collected.

10.3.1 Statistics for the entire database

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

```
SQL> UPDATE STATISTICS;
```

The user must have STATISTICS privilege.

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 times.

10.3.2 Statistics for specified idents

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

```
SQL> UPDATE STATISTICS FOR IDENT list-of-idents;
```

A user requesting statistics for tables belonging to a schema owned by an ident other than himself must have STATISTICS privilege.

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

10.3.3 Statistics for specified tables

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

```
SQL> 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 STATISTICS privilege.

10.3.4 Secondary index consistency

The update statistics facility includes an automatic function which ensures that all secondary indexes on tables contained in databanks with the TRANS or LOG option are in a consistent state.

This function is performed in a way that makes it transparent to other users of the database and it is only performed on secondary indexes created on tables actually selected by the UPDATE STATISTICS statement.

It will take some time to verify the consistency of a secondary index. The data dictionary table TABLE_CONSTRAINTS can be used to determine which secondary indexes are flagged as “not consistent” (shown in the column named IS_CONSISTENT).

An index which is in a consistent state will offer optimal performance when used in a query.

All secondary indexes contained in a databank with the NULL option and those contained in a databank that has been upgraded from Mimer SQL version 7 or 8.1 will be flagged as “not consistent”.

10.4 When to use the SQL statistics statements

Mimer SQL collects basic 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 collect detailed information (this includes information on value distribution and table size).

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

- when the size of a table has changed significantly
- when the maximum/minimum limits on values in a table have altered significantly
- when a databank has been altered from having the NULL option to having the TRANS or LOG option and contains secondary indexes
- when a databank with the TRANS or LOG option contains secondary indexes and has just been upgraded from Mimer SQL version 7 or 8.1.

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

Note: Only the performance, not the result, of an SQL statement is affected by gathering and using the statistical information and by ensuring the consistency of secondary indexes.

11 EXPORT/IMPORT

The Export/Import functionality provides facilities for:

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

11.1 Functions

The functionality is controlled through a series of menus.

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

```

MIMER / UTIL

Username: HOTELADM
Password:

-- MIMER UTILITIES --

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

0. Exit

Select: 2

```

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

```

11.2 Export options

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 SQL database.

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

```
-- Export - definitions and data --  
  
1. Export specific tables  
2. Export all tables for ident  
3. Export all tables in databank  
4. Export all tables in schema  
  
0. Exit
```

The fourth menu option will only appear when connected to a version 8.2 server (or later).

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. In the case of tables selected by ident, schema 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.

11.2.1 Extent and authorization

In general, only the creator of a specific object can export it. Furthermore, a table can be exported by any ident having the SELECT privilege for it. With this background, exporting a complete databank may not necessarily mean that all tables within the databank will be 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. Again, if export is performed for an ident, it is not assured that all tables for that ident are exported (due to privilege restrictions for the ident running the functionality).

Objects that can be exported explicitly are tables and schemas. Indexes, domains and foreign keys associated with an exported table will automatically be included in the export operation as far as access privileges allow.

If a foreign key table cannot be accessed, the foreign key definition will simply be left out from the export operation.

If columns in tables being exported belong to domains, the used domain definitions are exported together with the table definitions. If such a domain is not owned by the ident doing the export operation the domain will be replaced by the data type it is based upon, i.e. the domain is not exported and the use of it is lost.

If a synonym is specified as a table name, the appointed base table will be exported. The synonym itself will not be exported.

Export files created with version 8.1 of the UTIL program (or earlier) can be imported using the version 8.2 UTIL program, but the converse operation is not supported.

The version 8.2 UTIL program can be used towards a version 8.1 database server (or earlier).

11.2.2 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 SQL 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.

11.3 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.

11.3.1 Import - object creation

This phase creates the objects being imported. 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 SQL 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
-----
HOTELADM.BILL                    <----> HOTELDB
HOTELADM.BOOKFORM                <----> HOTELDB
HOTELADM.BOOK_GUEST              <----> HOTELDB
HOTELADM.EXCHANGE_RATE          <----> HOTELDB
HOTELADM.FREEROOMS              <----> HOTELDB
HOTELADM.ROOMSTATUS             <----> HOTELDB

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 name of the schema to which it belongs for clarity, but that the new name may not be qualified by a schema name (the tables will be created in a schema with the same name as the ident performing the import operation, in accordance with the default CREATE TABLE behavior).

```

-- Import - object creation --
File generated by Export/Convout   : BOOKEXP
Import log file                    : BOOKIMP
Listing of couplings between tables and target databanks
TABLE NAME                        DATABANK
-----
HOTELADM.BOOK_GUEST
                                     <---> HOTELDB
Are the couplings okay <Y>?      : Y
Making definitions
Duplicate table name              : HOTELADM.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-object creation:

- 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 by using the ALTER TABLE statement to establish the referential constraints after the tables have been 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 be created without the constraint(s) with a message informing the user of the constraint conflict.

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).

11.3.2 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 have been created by the export utility. The import functionality cannot use data files created by the UNLOAD function (see below).

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 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. 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 performed 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 : HOTELADM.FREEROOMS
    100 rows loaded
    195 rows loaded

Loading table : HOTELADM.ROOMSTATUS
    20 rows loaded
Operation completed

```

11.3.3 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 schema in which the tables are created).

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.

11.4 Load and Unload functions

The load and unload functions allow data to be moved between Mimer SQL tables and sequential files. The functionality here is equivalent to that provided by the LOAD and UNLOAD commands in BSQL, this interface involving prompted responses while the other is command based.

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 the following alternatives:

```

-- Load / Unload --
1. Load from file INTO table, default format
2. Load from file INTO table, user specified format
3. Load from file INTO table, delimited data
4. Unload to file FROM table, default format
5. Unload to file FROM table, user specified format
6. Unload to file FROM table, delimited data

0. Exit

```

11.4.1 Data file formats

The user may choose between a default format, a user-specified format or a delimited 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.

If a delimited format is chosen, the user is prompted for the delimiter character (which is entered via the keyboard). Fields in the record are separated by the delimiter character.

Both character and numerical data is written to the sequential file in string format. Files created by unloading table data can be edited with any text editor before being reloaded.

When the default or a user specified format is used, numerical data is right justified and blank padded to the size of the numeric data. For example, the number 143.6 is unloaded as the string " 143.6", occupying 6 bytes (the leading blank is the sign position).

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.

If the load operation attempts to insert the NULL-value into a NOT NULL column or if some data type conversion error occurs, an error will be raised and the INSERT operation will fail.

11.4.2 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 the delimited file format is used, data from the file record is mapped into the table columns in the order of the columns in the table using the delimiter character to indicate the end of the data for a column.

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 one of the duplicates is retained in the table. The number of rows loaded, the number of duplicates found and the number of rows not loaded because of conversion errors is reported when the operation is complete.

Hint: If there is a large amount of data to be loaded it can be performed 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.

11.4.3 Unload operation

When data is unloaded from Mimer SQL 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 is then unloaded from the view. (Alternatively, the whole table can be unloaded and excluded records are then deleted from the sequential file). The source table remains intact after the unload operation.

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 an existing file.

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

11.4.4 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.

11.5 Enter and Leave program ident

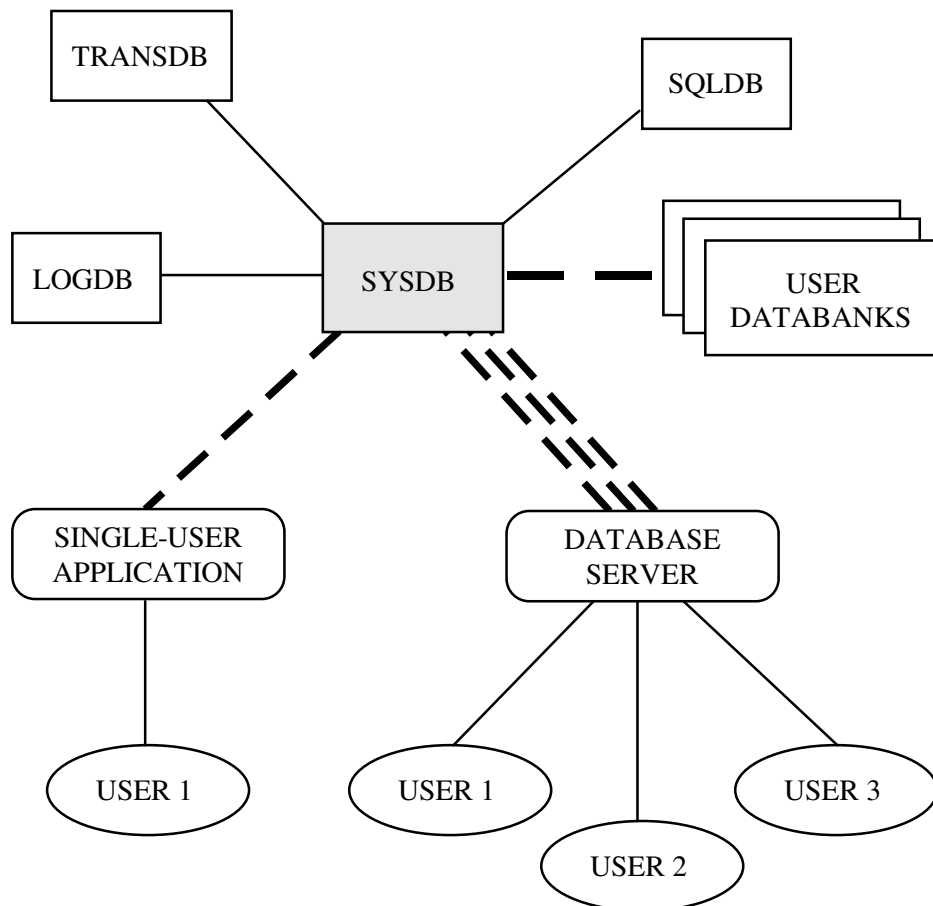
If a program ident is the creator of a table, the program ident may be entered in advance in order to export or unload the table.

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

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 typically be owned in the operating system by that user.



Single- and multi-user access to a Mimer SQL database.

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

If a database which 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. Suitably the database server should have exclusive access to the databank files.
- Conversely, files created in multi-user mode are created by the database server process and will not be accessible by a specific user who needs to access the database in single-user mode. The protection on these files can only be changed by an operating system user who has privileges equivalent to those of the database server process. Individual users should not generally have direct access to databank files.

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 defined or is set to “MULTI”, or the database is a remote one, it 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 the client/server interfaces in multi-user mode).

Note: Many of the programs which are part of the Mimer SQL distribution support the command-line flags “-s” and “-m” (or /SINGLE, /MULTI) which control whether they access a database in single-user or multi-user mode (see [Section 3.8](#)).

A.3 Accessing in single-user mode

Mimer SQL 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 the 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> .
SQL> .
SQL> exit;
$ unset MIMER_MODE
$ MIMER_DATABASE=STAFF
$ export MIMER_DATABASE
$ bsql
SQL> .
SQL> .
SQL> exit;
```

VMS

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

Win

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

A.4 The SINGLEDEFS parameter file

Unix

VMS

The use of a SINGLEDEFS parameter file is optional.

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

The SQLPOOL area will grow dynamically if more space is needed (see [Section 4.1.1.5](#)).

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.

Win

The use of a SINGLEDEFS is not supported on the Windows platform.

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 SQL application from the node on which it resides. For general information on how to make databases accessible, refer to [Section 3.1](#).

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. (See the `mimhosts` and `sqlhosts` man-pages).

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 will define it to be `SYSS$SPECIFIC:[SYSMGR]SQLHOSTS.DAT`. A default SQLHOSTS file is generated by the installation of the Mimer SQL 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](#)).

Database names may, in general, be up to 128 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 SQL 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).
--
```

Unix

```
-- Path - database home, and directory path for databank lookup.
```

VMS

```
-- 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)
--  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.
```

Unix

```
-- (The port number may either be a number or a name of a service
-- stored in the /etc/services file).
```

```
--
--  =====
--  DEFAULT:
--
--  Database
--  -----
--  example_localdb
--  =====
--  LOCAL:
```

```
-- Database      Path
-- -----
-- SINGLE        .
--
-- example_localdb  /directory
```

Unix

```
example_localdb  DISK:[DIRECTORY]
```

VMS

```
--  =====
--  REMOTE:
--
--  Database      Node      Protocol Interface Service
--  -----
--  example_remedb  server_nodename  ''      ''      1360
```

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 should 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 3.2](#)).

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 SQL system databank SYSDB will be located in the database home directory and other databanks will typically be located relative to it (see [Section 2.4.3](#)).

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 3.3](#)).

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 Mimer Information Technology AB for Mimer SQL client/server communication.

Unix

When TCP/IP is used under Unix, the value in the **SERVICE** field may be the actual port number, the name of a service stored in the /etc/services file or " (two single quote characters) for the default value "1360".

VMS

For a Mimer SQL 8 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 Mimer SQL 7 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 SQL version 8 and the value "MIMER" will be used for Mimer SQL version 7.)

B.1.4 Local client/server access

It is possible to use the client/server interface locally, i.e. having a remote database definition pointing to a local database server. This is the recommended access method when using Mimer SQL version 7 tools, such as QL and PG, towards a Mimer SQL version 8 database.

To achieve this, the SQLHOSTS file must be updated in a specific way using a 6th parameter in the remote definition, as shown in the following example (current node is “startrek”, i.e. where the Mimer SQL version 8 database “m8server” runs):

```
--
-- =====
DEFAULT:
--
-- Database
-----
SINGLE
-- =====
LOCAL:
--
-- Database          Path
-----
SINGLE
  m8server          /database/m8server:/mnt1/m8server:/mnt2/m8server
-- =====
REMOTE:
--
-- Database          Node          Protocol Interface Service
-----
  example_database  server_nodename  ''          ''          1360
  m8access          startrek         ''          ''          1360          m8server
```

To access the “m8server” database, the Mimer SQL version 7 application must connect to the database “m8access”.

In this case the “startrek” node must have both a Mimer SQL version 7 license (for the application) and a Mimer SQL version 8 license (for the database server). The Mimer SQL version 8 license must include a specific license in order to be able to accept client/server connections.

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 3.2](#)) 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 8.2.2 parameters generated 2000-12-07 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 8           # 1-100       # of request threads
BackgroundThreads 3       # 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	inetd	#	TCP/IP port

VMS

Oper	OPER	#	Receivers for messages
DumpPath	<>	#	Path for dump directory
TCPPort	1360	#	TCP/IP port

MaxSQLPool	30000	#	10-2000000	Max size of SQLPool (kb)
MemLock	0	#	0-1	Lock bpool in memory?

VMS

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)

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.

<i>BackgroundThreads</i>	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 Mimer SQL license 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 (involving index and data pages) provides for more secure operations, but may take much longer to execute than an index page check. When a full check is done, however, the data pages are checked in the background so there is a minimal negative effect on performance. 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, i.e. operators, 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.

DumpPath

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 SQL 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 SQL 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 SQL 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

<i>DECPort</i>	<p>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.</p> <p>If you set this parameter to “-” (a single dash), the DECNET capability will be disabled for the database server.</p>
<i>ProcName</i>	<p>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.</p>
<i>NetUsers</i>	<p>This parameter specifies the number of users who can access the database through a network connection. The value used by the system will be the minimum of this parameter and the <i>Users</i> parameter. 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>
<i>ServPrio</i>	<p>This parameter specifies the VMS priority for the database server process.</p>
<i>Cleanup</i>	<p>Specifies the time (in seconds) between the cleanup sweeps that check for terminated database clients.</p>

D UNIX SPECIFICS

D.1 Automatic database start and stop

When Mimer SQL is installed, autostart is automatically enabled.

The mimservers program will start or stop all local Mimer SQL v8.2 servers defined in the SQLHOSTS file. Automatic start and stop for local servers is achieved by adding routines to the init.d environment of the operating system that performs calls to the mimservers program. The init.d setup is done by using the mimautoset command, invoked during installation. For details, see the manpage for init.d.

To exclude a server from the automatic start/stop procedure, set the AutoStart parameter in the multidefs file for that server to "0".

Note! For systems for which the method using an init.d environment for automatic start/stop does not apply, no setup will be made. In this case explicit calls to the mimservers program must be added to rc-files, or corresponding.

D.2 Using raw device partitions

The term "raw devices" applies to the character oriented disk device files (as opposed to the block oriented ones) normally found in **/dev**. These device files are a part of the interface between the hardware disks and the Unix system software.

In most Unix systems it is a performance advantage to use raw device files for data storage. By using raw devices, the Unix file system (which uses index pages to locate the file blocks) is bypassed and the operating system is able to perform more effective I/O. Of course the effect also depends on several other factors such as file size, transaction rate, I/O-implementation, etc. Generally, large and/or frequently used databanks should be stored in raw device files.

Note! Familiarity with raw device files is recommended. It is essential that a raw device file be correctly defined, since the normal Unix file handling safeguards do not apply. Some important points to be considered:

- Usually the first cylinder on the disk (cylinder 0), or corresponding, should be avoided when partitioning, since operating system information for the disk is stored there.
 - Overlapping partitions must be manually avoided.
 - It should be verified that backup procedures include raw devices.
 - Always take a complete backup before enabling use of a raw device.
-

The major disadvantage of using a raw device file is that the maximum file size is fixed by the size of the partition. If the partition becomes full, the raw device file must be moved to a larger partition, if one is available. In the worst case, the disk must be reformatted in order to create a larger partition. *Always take a complete backup before reformatting a disk and before setting up any raw device files for use!*

Use of the raw device interface is completely transparent from the point of view of a Mimer SQL user. A user has no way of distinguishing a system using raw device files from a system where ordinary Unix files are used to store databanks.

D.2.1 Creating a raw disk partition

If it is decided to use raw I/O towards a disk and a character special device for the disk partition in question is missing, such a device must be created. This can be done from a Logical Volume Manager (LVM) or on some systems by using the `mknod` command.

LVM

When a logical volume is created a corresponding character oriented raw device is created along with the block oriented one. When using raw I/O towards the character oriented device, make sure the block oriented device is not used (and not mounted at boot).

Mknod

The major and minor numbers for the disk partition must first be identified and then the character special, non-buffering, device can be created:

```
# ls -l sdb2
brw-rw---- 1 root    disk      8, 18 May  5 1998 /dev/sdb2
# mknod rsdb2 c 8 18
# ls -l *sdb2
crw-rw---- 1 root    root      8, 18 Oct 18 08:51 /dev/rsdb2
brw-rw---- 1 root    disk      8, 18 May  5 1998 /dev/sdb2
```

D.2.2 Arranging for putting a databank on a raw device

The database server must be stopped:

```
# mimcontrol -t m82server
```

The databank file should be renamed:

```
# mv transdb.dbf transdb.dbf.ORDINARY
```

Create a soft link to the raw device file, using the original databank file name. Make sure root is the only one with access to the partition:

Note! It must be verified that the partition that corresponds to the raw device is not part of a file system that can be mounted.

```
# ln -s /dev/rdsk/c1t2d0s5 transdb.dbf
# chmod 600 /dev/rdsk/c1t2d0s5
# chmod 600 /dev/dsk/c1t2d0s5
```

Copy the original databank file onto the soft link and restart the database server:

```
# cp transdb.dbf.ORDINARY transdb.dbf
# mimcontrol -s m82server
```

D.2.3 Arranging for taking a databank off a raw device

The database server must be stopped:

```
# mimcontrol -t m82server
```

The raw device databank file must be copied to an ordinary block oriented file, located on a partition large enough to hold the complete raw device partition. Then the soft link can be removed:

```
# cp transdb.dbf /extra/transdb.dbf
# unlink transdb.dbf
```

Now the raw device partition can be used for other purposes. However, to be able to start the database system again, the databank file (in this case the TRANSDB file /extra/transdb.dbf) must be locatable by the database server. This can be achieved by arranging for the databank file to be moved back to its original location. Another solution is to update the /etc/sqlhosts file by adding an optional databank location.

E DATA DICTIONARY TABLES

This appendix documents the organization of the data dictionary tables, which are stored in the databank SYSDB.

The tables are created by the SDBGEN program when the system is first established. The tables are created in a schema called SYSTEM and are thus effectively owned by a pseudo ident called SYSTEM. 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 [Chapter 7 of the Mimer SQL Reference Manual](#) for documentation of these views). The logical group PUBLIC is granted SELECT access on these views, so that any user may read the dictionary information presented in the views. Many of the view definitions restrict the information presented to descriptions of objects and privileges accessible to the current user.

Note: If SYSADM reads the contents of such a view, the result shows only the objects and privileges to which SYSADM has access. In order to gain information on inaccessible objects and privileges, 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
API_FUNCTION	translation of id to function or module name.
AST_CODES	binary representation of the search condition of views in the database (for internal use).
AST_SOURCES	textual representation of view definitions and tables and domains with check constraints.
BACKUPS	databank backups recorded in the database.
CHAR_SETS	character sets in the database.
CHECK_CONSTRAINTS	check constraints defined for tables and domains.
COLLATIONS	character collations in the database.
COLUMNS	columns in tables or views.
COLUMN_OBJECT_USE	columns that depend on other database objects.
COLUMN_PRIVILEGES	instances of privileges granted on a column.
DATABANKS	databanks in the database.
DOMAINS	domains in the database.
DOMAIN_CONSTRAINTS	constraints defined for domains.
FIPS_FEATURES	details about all FIPS features.
FIPS_SIZING	details about FIPS limits.
KEY_COLUMN_USAGE	columns in an index or in a primary, unique or foreign key.
LEVEL2_RESTRICT	restrictions for domains legal for use by DB level 2.
LEVEL2_VIEWCOL	columns of views acceptable by DB level 2.
LEVEL2_VIEWRES	restrictions for DB level 2.
MANYROWS	dummy table with more than one row.
MESSAGE	translation of message code to message text.
MODULES	SQL-server modules in the database.
OBJECTS	objects in the database.
OBJECT_COLUMN_USE	columns referenced by other database objects.
OBJECT_OBJECT_USE	objects that have a dependency on another object.
ONEROW	dummy table containing one row.
PARAMETERS	parameters of routines in the database.
REFER_CONSTRAINTS	referential constraints in the database.
ROUTINES	procedures and user-defined functions in the database.
SCHEMATA	schemas in the database.
SEQUENCES	sequences in the database.
SERVER_INFO	attributes of the current database system or server.
SEVERITY	severity levels and optional module for error codes.
SOURCE_DEFINITION	source definitions for defaults, check constraints, views, modules, procedures, functions and triggers.

SPECIFIC_NAMES	specific names of the routines in the database.
SQL_LANGUAGES	SQL standards and SQL dialects supported.
SYNONYMS	synonyms and shadows in the database.
TABLES	tables and views in the database.
TABLE_CONSTRAINTS	table constraints in the database.
TABLE_PRIVILEGES	instances of privileges granted on a table.
TABLE_TYPES	the types of table supported.
TRANSLATIONS	character translations in the database.
TRIGGERED_COLUMNS	table columns explicitly specified in an UPDATE trigger event.
TRIGGERS	triggers in the database.
TYPE_INFO	information about data types supported.
USAGE_PRIVILEGES	instances of privileges which grant the right to use a database object.
USERS	idents (group, OS_USER, program or user) in the database.
VIEWS	views in the database.

API_FUNCTION

Records translations of id to function or module name.

Column name	Data type	Description
MODULEID	INTEGER	System identifier for module.
API_FUNCTION	INTEGER	System identifier for function.
TEXT	CHAR(40)	Name of the function or module.
Primary key: MODULEID, API_FUNCTION		

AST_CODES

Records the binary representation of the search condition of views in the database (for internal use).

Column name	Data type	Description
AST_SYSID	INTEGER	System identifier for the view.
AST_VERSION	INTEGER	Current AST-revision version number.
SEQUENCE_NO	INTEGER	Sequence number within representation.
AST_LENGTH	INTEGER	Length of binary data in AST_CODE.
AST_CODE	VARCHAR(5000)	Binary representation of the view search condition.
Primary key: AST_SYSID, AST_VERSION, SEQUENCE_NO		

AST_SOURCES

Records the textual representation of view definitions and domains or tables with check constraints in the database (for internal use).

Column name	Data type	Description
ASTS_SYSID	INTEGER	System identifier for the object which the definition represents.
SEQUENCE_NO	INTEGER	Sequence number within representation.
ASTS_LENGTH	INTEGER	Length of representation.
ASTS_SOURCE	VARCHAR(5000)	Textual representation of view, domain or table.
Primary key: ASTS_SYSID, SEQUENCE_NO		

BACKUPS

Records databank backups recorded in the database.

Column name	Data type	Description
DATABANK_NAME	VARCHAR(128)	Name of the databank recorded in the backup.
DATABANK_FILENO	INTEGER	File number of the databank file.
BACKUP_TYPE	CHAR(2)	Type of backup: "BC" = backup copy "BU" = incremental copy "DL" = drop log.
BACKUP_DATE	CHAR(8)	Date the backup was taken.
BACKUP_TIME	CHAR(6)	Time the backup was taken.
BACKUP_USER	VARCHAR(128)	Ident name taking the backup.
BACKUP_FILENAME	VARCHAR(256)	Name of the file containing the backup.
BACKUP_TIMENO	INTEGER	Time order number.
Primary key: DATABANK_NAME, DATABANK_FILENO, BACKUP_TYPE, BACKUP_DATE, BACKUP_TIME		

CHAR_SETS

Records character sets in the database.

Column name	Data type	Description
CHARSET_SYSID	INTEGER	System identifier for the character set.
FORM_OF_USE	VARCHAR(128)	A user-defined name that indicates the form-of-use of the character set.
NUMBER_OF_CHARS	INTEGER	Number of characters in the character set.
DEF_COLLATE_SYSID	INTEGER	System identifier for the character collation.
Primary key: CHARSET_SYSID		

CHECK_CONSTRAINTS

Records check constraints defined for tables and domains in the database.

Column name	Data type	Description
CONSTRAINT_SYSID	INTEGER	System identifier for the check constraint.
CHECK_CLAUSE	VARCHAR(2000)	The text of the search condition of the check constraint. If the text is too long it will be stored in the SOURCE_DEFINITION table.
Primary key: CONSTRAINT_SYSID		

COLLATIONS

Records character collations in the database.

Column name	Data type	Description
COLLATION_SYSID	INTEGER	System identifier for the collation.
CHARSET_SYSID	INTEGER	System identifier for the character set.
PAD_ATTRIBUTE	CHAR(20)	One of the following values: “NO PAD” = the collation has the <i>no pad</i> attribute “PAD SPACE” = the collation has the <i>pad space</i> attribute.
Primary key: COLLATION_SYSID		

COLUMNS

Records table and view columns in the database.

Column name	Data type	Description
TABLE_SYSID	INTEGER	System identifier for the table or view.
COLUMN_NAME	VARCHAR(128)	The name of the table or view column.
ORDINAL_POSITION	INTEGER	The ordinal position of the column in the table. The first column in the table is number 1.
DOMAIN_SYSID	INTEGER	System identifier for the domain used by the column.
COLLATION_SYSID	INTEGER	System identifier for the collation used by the column.

DATA_TYPE	CHAR(30)	Identifies the data type of the column. Can be one of the following: “BIGINT” “BINARY” “BINARY VARYING” “CHARACTER” “CHARACTER VARYING” “DATE” “DECIMAL” “DOUBLE PRECISION” “FLOAT” “INTEGER” “INTERVAL” “NUMERIC” “REAL” “SMALLINT” “TIME” “TIMESTAMP”.
INTERVAL_TYPE	CHAR(30)	For interval data types, this is a character string specifying the interval qualifier for the named interval data type (see Section 4.3.3.2 of the Mimer SQL Reference Manual). For other data types it is the NULL value.
CHAR_MAX_LENGTH	INTEGER	For a character data type, this shows the maximum length in characters. For all other data types it is the NULL value.
CHAR_OCTET_LENGTH	INTEGER	For a character data type, this shows the maximum length in octets. For all other data types it is the NULL value. (For single octet character sets, this is the same as CHAR_MAX_LENGTH).
NUMERIC_PRECISION	INTEGER	For numeric data types, this shows the total number of decimal digits allowed in the column. For all other data types it is the NULL value. NUMERIC_PREC_RADIX indicates the units of measurement.
NUMERIC_SCALE	INTEGER	This defines the total number of significant digits to the right of the decimal point. For INTEGER and SMALLINT, this is 0. For CHARACTER, CHARACTER VARYING, DATETIME, FLOAT, INTERVAL, REAL and DOUBLE PRECISION data types, it is the NULL value.
NUMERIC_PREC_RADIX	INTEGER	For numeric data types, the value 10 is shown because NUMERIC_PRECISION specifies a number of decimal digits. For all other data types it is the NULL value. NUMERIC_PRECISION and NUMERIC_PREC_RADIX can be combined to calculate the maximum number that the column can hold.

DATETIME_PRECISION	INTEGER	For DATE, TIME, TIMESTAMP and interval data types, this column contains the number of digits of precision for the fractional seconds component. For other data types it is the NULL value.
INTERVAL_PRECISION	INTEGER	For interval data types, this is the number of significant digits for the interval leading precision (see Section 4.3.3.2 of the Mimer SQL Reference Manual). For other data types it is the NULL value.
IS_NULLABLE	CHAR(3)	One of: "NO" = the column is not nullable, according to the rules in the international standard "YES" = the NULL value is allowed in the column.
IS_UPDATABLE	CHAR(3)	One of: "NO" = the column is not updatable "YES" = the column is updatable.
IS_INSTEAD_OF	CHAR(3)	One of: "NO" "YES".
INTERNAL_LENGTH	INTEGER	Length in bytes of value.
INTERNAL_OFFSET	INTEGER	Offset for value in record.
INTERNAL_VC_OFFSET	INTEGER	Offset for field containing actual length for a varying length item.
COLUMN_CARD	BIGINT	Number of unique values in column.
COLUMN_CARD_NONULL	BIGINT	Number of values in column that are not null.
COLUMN_LOW_VALUE	CHAR(16)	Lowest value in column.
COLUMN_HIGH_VALUE	CHAR(16)	Highest value in column.

COLUMN_DEFAULT	VARCHAR(2000)	<p>This shows the default value for the column.</p> <p>If the default value is a character string, the value shown is the string enclosed in single quotes.</p> <p>If the default value is a numeric literal, the value is shown in its original character representation without enclosing quotes.</p> <p>If the default value is a DATE, TIME or TIMESTAMP, the value shown is the appropriate keyword (e.g. DATE) followed by the literal representation of the value enclosed in single quotes (see Section 4.3.3.2 of the Mimer SQL Reference Manual for a description of DATE, TIME and TIMESTAMP literals).</p> <p>If the default value is a pseudo-literal, the value shown is the appropriate keyword (e.g. CURRENT_DATE) without enclosing quotes.</p> <p>If the default value is the NULL value, the value shown is the keyword NULL without enclosing quotes.</p> <p>If the default value cannot be represented without truncation, then TRUNCATED is shown without enclosing quotes and the text will be stored in the SOURCE_DEFINITION table.</p> <p>If no default value was specified then its value is the NULL value.</p> <p>The value of COLUMN_DEF is syntactically suitable for use in specifying <i>default-value</i> in a CREATE TABLE or ALTER TABLE statement.</p>
REMARKS	VARCHAR(254)	A comment on the column. If no comment has been defined a zero length string is stored.
Primary key: TABLE_SYSID, COLUMN_NAME		
Unique constraint: TABLE_SYSID, ORDINAL_POSITION		
Unique constraint: TABLE_SYSID, INTERNAL_OFFSET		

COLUMN_OBJECT_USE

Records table or view columns that depend on other objects in the database.

Column name	Data type	Description
OBJECT_TYPE	CHAR(20)	One of: "DOMAIN" "SEQUENCE" "COLLATION".
OBJECT_SYSID	INTEGER	System identifier for the object on which the column depends.
TABLE_SYSID	INTEGER	System identifier for the table or view.
COLUMN_NAME	VARCHAR(128)	The name of the table or view column.
Primary key: OBJECT_TYPE, OBJECT_SYSID, TABLE_SYSID, COLUMN_NAME		
Secondary index: TABLE_SYSID, COLUMN_NAME		

COLUMN_PRIVILEGES

Records table or view columns that depend on other objects in the database.

Column name	Data type	Description
TABLE_SYSID	INTEGER	System identifier for the table or view.
COLUMN_NAME	VARCHAR(128)	The name of the table or view column.
PRIVILEGE_TYPE	CHAR(20)	One of: "INSERT" "REFERENCES" "SELECT" "UPDATE".
GRANTEE	VARCHAR(128)	The name of the ident to which the privilege was granted.
GRANTOR	VARCHAR(128)	The name of the ident granting the privilege.
IS_GRANTABLE	CHAR(3)	One of: "NO" = WITH GRANT OPTION is not held for the privilege "YES" = WITH GRANT OPTION is held for the privilege.
IS_INSTEAD_OF	CHAR(3)	One of: "NO" = The privilege has been granted explicitly "YES" = The privilege is granted because an instead of trigger has been defined on the view to which the column belongs.
Primary key: TABLE_SYSID, COLUMN_NAME, PRIVILEGE_TYPE, GRANTEE, GRANTOR		
Secondary index: GRANTEE		
Secondary index: GRANTOR		

DATABANKS

Records the databanks in the database.

Column name	Data type	Description
DATABANK_SYSID	INTEGER	System identifier for the databank.
DATABANK_SEQNO	INTEGER	The shadowing sequence number for the databank.
DATABANK_FILENO	INTEGER	The file number of the databank file.
DATABANK_NAME	VARCHAR(128)	The name of the databank.
SHADOW_NAME	VARCHAR(128)	The shadow name if the databank is a shadow.
DATABANK_FILENAME	VARCHAR(256)	The pathname for the databank file.
DATABANK_TYPE	CHAR(20)	Indicates type of transaction handling: "LOG" = transaction handling with logging "TRANS" = transaction handling without logging "NULL" = transaction handling not used "WORK" = work databank (SQLDB).
IS_MASTER	CHAR(3)	One of: "NO" = the databank is a shadow "YES" = the databank is a master.
IS_ONLINE	CHAR(3)	One of: "NO" = the databank is OFFLINE "YES" = the databank of ONLINE.
Primary key: DATABANK_SYSID, DATABANK_SEQNO, DATABANK_FILENO		
Unique constraint: SHADOW_NAME, DATABANK_FILENO		
Unique constraint: DATABANK_FILENAME		
Secondary index: DATABANK_NAME, DATABANK_SEQNO		

DOMAINS

Records the domains in the database.

Column name	Data type	Description
DOMAIN_SYSID	INTEGER	System identifier for the domain.
COLLATION_SYSID	INTEGER	System identifier for the collation for a domain associated with a character set.
DATA_TYPE	CHAR(30)	Identifies the data type of the domain. Can be one of the following: "BIGINT" "BINARY" "BINARY VARYING" "CHARACTER" "CHARACTER VARYING" "DATE" "DECIMAL" "DOUBLE PRECISION" "FLOAT" "INTEGER" "INTERVAL" "NUMERIC" "REAL" "SMALLINT" "TIME" "TIMESTAMP".

INTERVAL_TYPE	CHAR(30)	For interval data types, this is a character string specifying the interval qualifier for the named interval data type (see Section 4.3.3.2 of the Mimer SQL Reference Manual). For other data types it is the NULL value.
CHAR_MAX_LENGTH	INTEGER	For a character data type, this shows the maximum length in characters. For all other data types it is the NULL value.
CHAR_OCTET_LENGTH	INTEGER	For a character data type, this shows the maximum length in octets. For all other data types it is the NULL value. (For single octet character sets, this is the same as CHAR_MAX_LENGTH).
NUMERIC_PRECISION	INTEGER	For numeric data types, this shows the total number of decimal digits allowed in the column. For all other data types it is the NULL value. NUMERIC_PREC_RADIX indicates the units of measurement.
NUMERIC_SCALE	INTEGER	This defines the total number of significant digits to the right of the decimal point. For INTEGER and SMALLINT, this is 0. For CHARACTER, CHARACTER VARYING, DATETIME, FLOAT, INTERVAL, REAL and DOUBLE PRECISION data types, it is the NULL value.
NUMERIC_PREC_RADIX	INTEGER	For numeric data types, the value 10 is shown because NUMERIC_PRECISION specifies a number of decimal digits. For all other data types it is the NULL value. NUMERIC_PRECISION and NUMERIC_PREC_RADIX can be combined to calculate the maximum number that the column can hold.
DATETIME_PRECISION	INTEGER	For DATE, TIME, TIMESTAMP and interval data types, this column contains the number of digits of precision for the fractional seconds component. For other data types it is the NULL value.
INTERVAL_PRECISION	INTEGER	For interval data types, this is the number of significant digits for the interval leading precision (see Section 4.3.3.2 of the Mimer SQL Reference Manual). For other data types it is the NULL value.
INTERVAL_LENGTH	INTEGER	Length in bytes of value.

DOMAIN_DEFAULT	VARCHAR(2000)	<p>This shows the default value for the domain.</p> <p>If the default value is a character string, the value shown is the string enclosed in single quotes.</p> <p>If the default value is a numeric literal, the value is shown in its original character representation without enclosing quotes.</p> <p>If the default value is a DATE, TIME or TIMESTAMP, the value shown is the appropriate keyword (e.g. DATE) followed by the literal representation of the value enclosed in single quotes (see Section 4.4.5 of Mimer SQL Reference manual for a description of DATE, TIME and TIMESTAMP literals).</p> <p>If the default value is a pseudo-literal, the value shown is the appropriate keyword (e.g. CURRENT_DATE) without enclosing quotes.</p> <p>If the default value is the NULL value, the value shown is the keyword NULL without enclosing quotes.</p> <p>If the default value cannot be represented without truncation, then TRUNCATED is shown without enclosing quotes and the text will be stored in the SOURCE_DEFINITION table.</p> <p>If no default value was specified then its value is the NULL value.</p> <p>The value of DOMAIN_DEFAULT is syntactically suitable for use in specifying <i>default-value</i> in a CREATE TABLE or ALTER TABLE statement.</p>
ANY_CONSTRAINT	CHAR(3)	<p>One of:</p> <p>“NO” = There are no constraints for the domain</p> <p>“YES” = There are constraints for the domain.</p>
IS_LEVEL2_APPROVED	CHAR(3)	<p>One of:</p> <p>“NO” = Tables using this domain can not be used with the level 2 interface</p> <p>“YES” = The use of this domain does not prohibit the table being used with the level 2 interface.</p>
Primary key: DOMAIN_SYSID		

DOMAIN_CONSTRAINTS

Records the constraints defined for domains in the database.

Column name	Data type	Description
DOMAIN_SYSID	INTEGER	System identifier for the domain.
CONSTRAINT_SYSID	INTEGER	System identifier for the constraint.
IS_DEFERRABLE	CHAR(3)	One of: "NO" = the constraint is not deferrable. "YES" = the constraint is deferrable.
INITIALLY_DEFERRED	CHAR(3)	One of: "NO" = the constraint is not initially deferred "YES" = the constraint is initially deferred.
Primary key: DOMAIN_SYSID, CONSTRAINT_SYSID		

FIPS_FEATURES

Lists details of all FIPS features.

Column name	Data type	Description
FEATURE_ID	INTEGER	Identification number of the FIPS feature.
FEATURE_NAME	CHAR(50)	The name of the FIPS feature.
CLASSIFICATION	CHAR(12)	The conformance level of the feature. One of: "TRANSITIONAL" "INTERMEDIATE" "FULL".
IS_SUPPORTED	CHAR(3)	One of: "YES" = Mimer SQL supports the feature "NO" = Mimer SQL does not support the feature.
IS_VERIFIED	CHAR(3)	One of: "YES" = Mimer SQL support for the feature has been tested and verified "NO" = Mimer SQL support for the feature has not been verified.
FEATURE_COMMENTS	VARCHAR(500)	Comments about the feature.
Primary key: FEATURE_ID		

FIPS_SIZING

Lists details about FIPS limits.

Column name	Data type	Description
SIZING_ID	SMALLINT	Identification number for limit.
DESCRIPTION	CHAR(50)	Description of limit.
ENTRY_VALUE	INTEGER	Limit for entry SQL.
INTERMEDIATE_VALUE	INTEGER	Limit for intermediate SQL.
VALUE_SUPPORTED	INTEGER	Limit for Mimer SQL.
SIZING_COMMENTS	VARCHAR(500)	Comments for limit.
Primary key: SIZING_ID		

KEY_COLUMN_USAGE

Records columns in an index or in a primary, unique or foreign key.

Column name	Data type	Description
TABLE_SYSID	INTEGER	System identifier for the table.
CONSTRAINT_SYSID	INTEGER	System identifier for the table column.
ORDINAL_POSITION	INTEGER	The ordinal position of the column in the table. The first column in the table is number 1.
COLUMN_NAME	VARCHAR(128)	The name of the table column.
KEY_TYPE	CHAR(20)	One of: "PRIMARY_KEY" "UNIQUE" "FOREIGN KEY" "INDEX" "INTERNAL_KEY".
IS_UNIQUE	CHAR(3)	One of: "NO" = unique values are permitted in the column "YES" = the column is constrained to contain only unique values.
IS_ASCENDING	CHAR(3)	One of: "NO" = the column is indexed in descending value order "YES" = the column is indexed in ascending value order.
Primary key: TABLE_SYSID, CONSTRAINT_SYSID, ORDINAL_POSITION		
Unique constraint: CONSTRAINT_SYSID, COLUMN_NAME		

LEVEL2_RESTRICT

Records restrictions for domains legal for use by DB level 2.

Column name	Data type	Description
DOMAINID	INT(10)	System identifier for the 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 the value.
RESVALUE	CHAR(64)	Restriction value.
Primary key: DOMAINID, SEQNO		

LEVEL2_VIEWCOL

Records columns of views acceptable by DB level 2.

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		

LEVEL2_VIEWRES

Records restrictions for DB level 2.

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 the value.
RESVALUE	CHAR(64)	Restriction value.
Primary key: VTABID, SEQNO		

MANYROWS

Dummy table with more than row.

Column name	Data type	Description
C	INT(3)	The values column.
Primary key: C		

MESSAGE

Records translations of message codes to message text.

Column name	Data type	Description
MODULEID	INTEGER	Identification number for Mimer SQL module to which error belongs.
MESSAGEID	INTEGER	Identification number for message
LANGUAGE	INTEGER	Language number for message 1 = English.
LINENO	INTEGER	Line number for message.
MESSAGE	CHAR(80)	Message text.
Primary key: MODULEID, MESSAGEID, LANGUAGE, LINENO		

MODULES

Records the SQL-server modules in the database.

Column name	Data type	Description
MODULE_SYSID	INTEGER	System identifier for the module.
DEF_CHARSET_SYSID	INTEGER	System identifier for default character set for module.
DEF_SCHEMA_SYSID	INTEGER	System identifier for default schema for module.
MODULE_LENGTH	INTEGER	Length of module definition.
MODULE_DEFINITION	VARCHAR(2000)	Source text for module definition, if the length of the module definition exceeds 2000 characters this column is null.
SQL_PATH	VARCHAR(2000)	The path for the module.
Primary key: MODULE_SYSID		

OBJECTS

Records objects in the database.

Column name	Data type	Description
OBJECT_SYSID	INTEGER	System identifier for the database object.
OBJECT_TYPE	CHAR(20)	One of: "SCHEMA" "IDENT" "DATABANK" "BASE TABLE" "VIEW" "INDEX" "CONSTRAINT" "DOMAIN" "MODULE" "PROCEDURE" "FUNCTION" "SEQUENCE" "TRIGGER" "CHARACTER SET" "COLLATION" "TRANSLATION" "SHADOW" "SYNONYM".
OBJECT_SCHEMA	VARCHAR(128)	The name of the schema containing the object.
OBJECT_NAME	VARCHAR(128)	The name of the object.
OBJECT_CREATED	TIMESTAMP(2)	The date and time the object was created.
OBJECT_ALTERED	TIMESTAMP(2)	The date and time the object was last altered.
REMARKS	VARCHAR(254)	Comments defined on the object. If there is no comment defined the empty string ("") will appear. Remarks relating to columns will be stored in the COLUMNS table.
Primary key: OBJECT_SYSID		
Unique constraint: OBJECT_TYPE, OBJECT_SCHEMA, OBJECT_NAME		

OBJECT_COLUMN_USE

Records columns referenced by other database objects.

Column name	Data type	Description
TABLE_SYSID	INTEGER	System identifier for the table.
COLUMN_NAME	VARCHAR(128)	The name of the table column.
USED_BY_SYSID	INTEGER	The system identifier of the referencing object.
USED_BY_TYPE	CHAR(20)	One of: "VIEW" "PROCEDURE" "FUNCTION" "CHECK" "TRIGGER".
PRIVILEGE_TYPE	CHAR(20)	One of: "INSERT" "DELETE" "SELECT" "UPDATE" "REFERENCES".
Primary key: TABLE_SYSID, COLUMN_NAME, USED_BY_SYSID, USED_BY_TYPE, PRIVILEGE_TYPE		
Secondary index: USED_BY_SYSID, USED_BY_TYPE		

OBJECT_OBJECT_USE

Records objects that have a dependency on another object.

Column name	Data type	Description
OBJECT_SYSID	INTEGER	System identifier for the object.
OBJECT_TYPE	CHAR(20)	One of: "BASE TABLE" "VIEW" "PROCEDURE" "FUNCTION" "DOMAIN" "SEQUENCE" "CHARACTER SET" "COLLATION" "TRANSLATION".
USED_BY_SYSID	INTEGER	The system identifier of the referenced object.
USED_BY_TYPE	CHAR(20)	One of: "VIEW" "PROCEDURE" "FUNCTION" "DOMAIN" "TRIGGER" "CHARACTER SET" "COLLATION" "TRANSLATION".
Primary key: OBJECT_SYSID, OBJECT_TYPE, USED_BY_SYSID, USED_BY_TYPE		
Secondary index: USED_BY_SYSID, USED_BY_TYPE		

ONEROW

Dummy table containing one row.

Column name	Data type	Description
M	CHAR(1)	Contains the value "M".
Primary key: M		

PARAMETERS

Records parameters of routines in the database.

Column name	Data type	Description
ROUTINE_SYSID	INTEGER	System identifier for the function or procedure.
ORDINAL_POSITION	INTEGER	The ordinal position of the parameter in the routine. The first parameter in the routine is number 1.
IS_RESULT	CHAR(3)	One of: 'NO' = This parameter is not part of the result clause for a result set procedure 'YES' = This parameter is part of the result set.
PARAMETER_MODE	CHAR(5)	One of: "IN" "OUT" "INOUT".
PARAMETER_NAME	VARCHAR(128)	The name of the parameter.
DOMAIN_SYSID	INTEGER	System identifier of the domain that defines the data type of the parameter.
COLLATION_SYSID	INTEGER	System identifier of the collation associated with the parameter.
DATA_TYPE	CHAR(30)	The data type of the parameter.
INTERVAL_TYPE	CHAR(30)	For interval data types, this is a character string specifying the interval qualifier for the named interval data type (see Section 4.3.3.2 of the Mimer SQL Reference Manual). For other data types it is the NULL value.
CHAR_MAX_LENGTH	INTEGER	For a character data type, this shows the maximum length in characters.
CHAR_OCTET_LENGTH	INTEGER	For a character data type, this shows the maximum length in octets. (For single octet character sets, this is the same as CHARACTER_MAX_LENGTH).
NUMERIC_PRECISION	INTEGER	For numeric data types, this shows the total number of decimal digits allowed in the column. For all other data types it is the NULL value. NUMERIC_PREC_RADIX indicates the units of measurement.

NUMERIC_SCALE	INTEGER	This defines the total number of significant digits to the right of the decimal point. For INTEGER and SMALLINT, this is 0. For CHARACTER, CHARACTER VARYING, DATETIME, FLOAT, INTERVAL, REAL and DOUBLE PRECISION data types, it is the NULL value.
NUMERIC_PREC_RADIX	INTEGER	For numeric data types, the value 10 is shown because NUMERIC_PRECISION specifies a number of decimal digits. For all other data types it is the NULL value. NUMERIC_PRECISION and NUMERIC_PREC_RADIX can be combined to calculate the maximum number that the column can hold.
DATETIME_PRECISION	INTEGER	For DATE, TIME, TIMESTAMP and interval data types, this column contains the number of digits of precision for the fractional seconds component. For other data types it is the NULL value.
INTERVAL_PRECISION	INTEGER	For interval data types, this is the number of significant digits for the interval leading precision (see Section 4.3.3.2 of the Mimer SQL Reference Manual). For other data types it is the NULL value.
Primary key: ROUTINE_SYSID, ORDINAL_POSITION, IS_RESULT		

REFER_CONSTRAINTS

Records referential constraints in the database.

Column name	Data type	Description
TABLE_SYSID	INTEGER	System identifier for the table.
CONSTRAINT_SYSID	INTEGER	System identifier for the referential constraint.
UNIQUE_TABLE_SYSID	INTEGER	System identifier for the table that is referenced in the foreign key.
UNIQUE_CONST_SYSID	INTEGER	System identifier for the constraint that is (implicitly) referenced in the foreign key.
MATCH_OPTION	CHAR(20)	One of: "NONE" "PARTIAL" "FULL".
UPDATE_RULE	CHAR(20)	One of: "CASCADE" "SET NULL" "SET DEFAULT" "NO ACTION" "RESTRICT".
DELETE_RULE	CHAR(20)	One of: "CASCADE" "SET NULL" "SET DEFAULT" "NO ACTION" "RESTRICT".
Primary key: TABLE_SYSID, CONSTRAINT_SYSID		
Secondary index: UNIQUE_TABLE_SYSID, UNIQUE_CONST_SYSID		

ROUTINES

Records procedures and user-defined functions in the database.

Column name	Data type	Description
ROUTINE_SYSID	INTEGER	System identifier of the function or procedure.
MODULE_SYSID	INTEGER	System identifier of the module to which the routine belongs.
ROUTINE_TYPE	CHAR(20)	One of: "PROCEDURE" "FUNCTION".
SQL_DATA_ACCESS	CHAR(20)	One of: "NO SQL" "CONTAINS SQL" "READS SQL DATA" "MODIFIES SQL DATA".
ROUTINE_OFFSET	INTEGER	Offset for routine within module definition. This value is zero if the routine is not defined in a module.
ROUTINE_LENGTH	INTEGER	Length of routine definition
TOTAL_PARAMS	INTEGER	The total number of routine parameters.
RESULT_PARAMS	INTEGER	The number of result parameters.
INPUT_PARAMS	INTEGER	The number of input parameters.

OUTPUT_PARAMS	INTEGER	The number of output parameters.
ROUTINE_DEFINITION	VARCHAR(2000)	Routine definition. If the length of the routine definition exceeds 2000 characters this value is null.
SQL_PATH	VARCHAR(2000)	The search path for the routine.
IS_DETERMINISTIC	CHAR(3)	One of: "NO" = The routine is not deterministic. i.e. invoking the routine with the same input values is not guaranteed to return the same result "YES" = The routine is deterministic, i.e. when invoked with same input values it will always return the same result.
ROUTINE_BODY	CHAR(20)	One of: "SQL" = The routine is a SQL routine "EXTERNAL" = The routine is external
EXTERNAL_NAME	VARCHAR(128)	The name of the program that implements the routine if it is an external routine otherwise null.
EXTERNAL_LANGUAGE	CHAR(20)	The language for the routine if it is external otherwise null.
PARAMETER_STYLE	CHAR(20)	The parameter passing style for the routine if it is an external routine, otherwise null.
SCHEMA_LEVEL_ROU	CHAR(3)	One of: "NO" = The routine is defined within a module "YES" = The routine is defined in a module.
MX_DYN_RESULT_SETS	INTEGER	The maximal number of result sets that the routine may return.
Primary key: ROUTINE_SYSID		
Secondary index: MODULE_SYSID		

SCHEMATA

Records schemas in the database.

Column name	Data type	Description
SCHEMA_SYSID	INTEGER	System identifier of the schema.
DEF_CHARSET_SYSID	INTEGER	System identifier of the default character set.
SQL_PATH	VARCHAR(2000)	The default search path for objects belonging to the schema.
Primary key: SCHEMA_SYSID		

SEQUENCES

Records sequences in the database.

Column name	Data type	Description
SEQUENCE_SYSID	INTEGER	System identifier of the sequence.
IS_UNIQUE	CHAR(3)	One of: "YES" = the sequence will yield unique values "NO" = the sequence may repeat itself.
START_VALUE	INTEGER	The initial value of the sequence.
CURRENT_VALUE	INTEGER	The current value of the sequence.
MAX_VALUE	INTEGER	The maximum value of the sequence.
INCREMENT_VALUE	INTEGER	The increment value of sequence.
IS_EXHAUSTED	CHAR(3)	One of: "YES" = the series of values defined by the sequence is exhausted "NO" = the series of values defined by the sequence is not exhausted.
CYCLES	INTEGER	Number of times the maximum value of the sequence is exceeded.
Primary key: SEQUENCE_SYSID		

SERVER_INFO

Records attributes of the current database system or server.

Column name	Data type	Description
SERVER_ATTRIBUTE	VARCHAR(254)	One of: "CATALOG_NAME" "COLLATION_SEQ" "IDENTIFIER_LENGTH" "INTERVAL_FRACTIONAL_PRECISION" "INTERVAL_LEADING_PRECISION" "ROW_LENGTH" "TIME_PRECISION" "TIMESTAMP_PRECISION" "TXN_ISOLATION" "USERID_LENGTH".
ATTRIBUTE_VALUE	VARCHAR(254)	The value for the attribute.
Primary key: SERVER_ATTRIBUTE		

SEVERITY

Records severity levels and optional module for error codes.

Column name	Data type	Description
MODULEID	INTEGER	Identification number for the Mimer SQL module to which the message belongs.
MESSAGEID	INTEGER	Identification number for message.
SEVERITY	INTEGER	Severity for message 1 = Message 2 = Warning 3 = Error 4 = Fatal error 5 = Internal error 6 = Code.
MODULE	CHAR(20)	Name of Mimer SQL module to which message belongs.
SQLSTATE_VALUE	CHAR(5)	Corresponding SQLSTATE value for message.
CLASS_ORIGIN	CHAR(20)	Class origin for the SQLSTATE value.
SUBCLASS_ORIGIN	CHAR(20)	Sub class origin for the SQLSTATE value.
Primary key: MODULEID, MESSAGEID		

SOURCE_DEFINITION

Records source definitions for defaults, check constraints, views, modules, procedures, functions and triggers.

Column name	Data type	Description
SOURCE_SYSID	INTEGER	System identifier of the source definition.
COLUMN_NAME	VARCHAR(128)	Column name if the source types is a default value for a column.
SOURCE_TYPE	CHAR(20)	One of: "DEFAULT" "CHECK" "VIEW" "MODULE" "PROCEDURE" "FUNCTION" "TRIGGER".
SEQUENCE_NO	INTEGER	Sequence number in source definition.
SOURCE_LENGTH	INTEGER	The length of the stored source text.
SOURCE_DEFINITION	VARCHAR(5000)	The stored source text.
Primary key: SOURCE_SYSID, COLUMN_NAME, SEQUENCE_NO		

SPECIFIC_NAMES

Records specific names of the routines in the database.

Column name	Data type	Description
ROUTINE_SYSID	INTEGER	System identifier of the routine.
SPECIFIC_SCHEMA	VARCHAR(128)	The name of the schema containing the routine.
SPECIFIC_NAME	VARCHAR(128)	The specific name for the routine.
Primary key: ROUTINE_SYSID		

SQL_LANGUAGES

Records the SQL standards and SQL dialects supported.

Column name	Data type	Description
SOURCE	VARCHAR(254)	Body that has defined standard.
SOURCE_YEAR	VARCHAR(254)	Which year the standard was approved.
CONFORMANCE	VARCHAR(254)	Level of conformance within standard.
INTEGRITY	VARCHAR(254)	If the supported standard is 'ISO 9075' the value "YES" in this column means that the integrity enhancement feature of this standard is supported, otherwise the value of this column is null.
IMPLEMENTATION	VARCHAR(254)	Implementation style, always null in Mimer SQL.
BINDING_STYLE	VARCHAR(254)	One of: "DIRECT" = Support for interactive (ad hoc) access to the database "EMBEDDED" = Support for access through an application programming interface.
PROGRAMMING_LANG	VARCHAR(254)	Name of programming languages that are supported if the binding style is EMBEDDED.
Primary key: SOURCE, SOURCE_YEAR		

SYNONYMS

Records synonyms and shadows in the database.

Column name	Data type	Description
SYNONYM_SYSID	INTEGER	System identifier of the synonym.
ORDINARY_SYSID	INTEGER	System identifier of the original object.
Primary key: SYNONYM_SYSID		

TABLES

Records tables and views in the database.

Column name	Data type	Description
TABLE_SYSID	INTEGER	System identifier of the table or view.
TABLE_TYPE	CHAR(20)	One of: "BASE TABLE" "VIEW".
TABLE_NOCOLS	INTEGER	Number of columns in table.
TABLE_RECLLEN	INTEGER	Record length for table.
TABLE_CARD	BIGINT	Number of records in table.
STATISTIC_GATHERED	TIMESTAMP(2)	Date and time when statistics for the table or view were last updated.
IS_LEVEL2_APPROVED	CHAR(3)	One of: "NO" = The table can not be used with level 2 DB interface.
IS_PERSISTENT	CHAR(3)	One of: "NO" = The table is a temporary table "YES" = The table is not temporary.
DATABANK_SYSID	INTEGER	System identifier of the databank in which the table is stored.
Primary key: TABLE_SYSID		
Secondary index: DATABANK_SYSID		

TABLE_CONSTRAINTS

Records table constraints in the database.

Column name	Data type	Description
TABLE_SYSID	INTEGER	System identifier of the table.
CONSTRAINT_TYPE	CHAR(20)	One of: "PRIMARY KEY" "UNIQUE" "FOREIGN KEY" "INDEX" "INDEX UNIQUE" "INTERNAL KEY" "CHECK".
CONSTRAINT_SYSID	INTEGER	System identifier of the constraint.
CONSTRAINT_KEYCOLS	INTEGER	Number of key columns in the constraint.
CONSTRAINT_KEYLEN	INTEGER	Record length for key columns in the constraint.
CONSTRAINT_RECCOLS	INTEGER	Number of columns in constraint.
CONSTRAINT_RECLEN	INTEGER	Record length for constraint.
CONSTRAINT_CARD	BIGINT	Number of records in constraint.
DATABANK_SYSID	INTEGER	System identifier of the databank in which the table is stored.
IS_CONSISTENT	CHAR(3)	One of: "NO" = The index may be inconsistent and should be used for index lookup only "YES" = The index may be used for index lookup only.
IS_DEFERRABLE	CHAR(3)	One of: "NO" = The constraint is not deferrable "YES" = The constraint is deferrable.
INITIALLY_DEFERRED	CHAR(3)	One of: "NO" = The constraint is immediate "YES" = The constraint is not immediate.
Primary key: TABLE_SYSID, CONSTRAINT_TYPE, CONSTRAINT_SYSID		
Unique constraint: CONSTRAINT_SYSID		

TABLE_PRIVILEGES

Records instances of privileges granted on a table.

Column name	Data type	Description
TABLE_SYSID	INTEGER	System identifier of the table.
PRIVILEGE_TYPE	CHAR(20)	One of: "DELETE" "INSERT" "LOAD" "REFERENCES" "SELECT" "UPDATE" "TRIGGER".
GRANTEE	VARCHAR(128)	The name of the ident to whom the table privilege was granted.
GRANTOR	VARCHAR(128)	The name of the ident granting the table privilege.
IS_GRANTABLE	CHAR(3)	One of: "YES" = WITH GRANT OPTION is held with the privilege "NO" = WITH GRANT OPTION is not held with the privilege.
IS_INSTEAD_OF	CHAR(3)	One of: "NO" = The privilege was granted explicitly "YES" = The privilege was granted implicitly when an instead of trigger was created.
IS_ON_TABLE	CHAR(3)	One of: "NO" = The privilege was granted on individual columns "YES" = The privilege was granted on the complete table.
ALL_COLUMNS	CHAR(3)	One of: "YES" = the privilege was granted on the table and therefore applies to all table columns, including new ones added "NO" = the privilege only applies to the table columns explicitly specified when the privilege was granted.
Primary key: TABLE_SYSID, PRIVILEGE_TYPE, GRANTEE, GRANTOR		
Secondary index: GRANTEE		
Secondary index: GRANTOR		

TABLE_TYPES

Records the types of table supported.

Column name	Data type	Description
TABLE_TYPE	CHAR(20)	One of: "SYNONYM" "SYSTEM TABLE" "TABLE" "VIEW".
Primary key: TABLE_TYPE		

TRANSLATIONS

Records character translations in the database.

Column name	Data type	Description
TRANSLATION_SYSID	INTEGER	System identifier of the character set translation.
SRC_CHARSET_SYSID	INTEGER	System identifier of the source character set for the translation.
TGT_CHARSET_SYSID	INTEGER	System identifier of the target character set for the translation.
Primary key: TRANSLATION_SYSID		

TRIGGERED_COLUMNS

Records table columns explicitly specified in an UPDATE trigger event.

Column name	Data type	Description
TRIGGER_SYSID	INTEGER	System identifier of the trigger.
EVENT_TABLE_SYSID	INTEGER	System identifier of the table on which the trigger is defined.
EVENT_COLUMN_NAME	VARCHAR(128)	The name of the column in which updates will cause the trigger to execute.
Primary key: TRIGGER_SYSID, EVENT_TABLE_SYSID, EVENT_COLUMN_NAME		

TRIGGERS

Records triggers in the database.

Column name	Data type	Description
TRIGGER_SYSID	INTEGER	System identifier of the trigger.
EVENT_TABLE_SYSID	INTEGER	System identifier of the table on which the trigger is defined.
EVENT_MANIPULATION	CHAR(20)	One of: "DELETE" "INSERT" "UPDATE".
ACTION_ORDER	INTEGER	Ordinal number for trigger execution. This number will define the execution order of triggers on the same table and with the same value for EVENT_MANIPULATION, ACTION_CONDITION, CONDITION_TIMING and ACTION_ORIENTATION. The trigger with 1 in this column will be executed first, followed by the trigger with 2 etc.
ACTION_CONDITION	VARCHAR(2000)	The text of the search condition of the trigger action WHEN clause.
ACTION_STATEMENT	VARCHAR(2000)	The character representation of the body of the trigger. If the length of the text exceeds 2000 characters, the NULL value will be shown.
ACTION_ORIENTATION	CHAR(20)	One of: "ROW" "STATEMENT".
CONDITION_TIMING	CHAR(20)	One of: "BEFORE" "AFTER" "INSTEAD OF".
COND_REF_NEW_TABLE	VARCHAR(128)	Name of new table/row alias.
COND_REF_OLD_TABLE	VARCHAR(128)	Name of old table/row alias.
REFERENCE_NEW	CHAR(3)	One of: "NO" = The trigger has no new table/row alias "YES" = The trigger has new table/row alias.
REFERENCE_OLD	CHAR(3)	One of: "NO" = The trigger has no new table/row alias "YES" = The trigger has new table/row alias.
COLS_IS_IMPLICIT	CHAR(3)	One of: "NO" = The trigger is invoked on update on any column (if the event is update) "YES" = The trigger will only be invoked if a column in the for update of list is updated.
REF_SYSID	INTEGER	System identifier for foreign key constraint, if the trigger is defined for checking of a delete rule.
Primary key: TRIGGER_SYSID		
Secondary index: EVENT_TABLE_SYSID		

TYPE_INFO

Records information about data types supported.

Column name	Data type	Description
DATA_TYPE	SMALLINT	Identification number for data type.
TYPE_NAME	CHAR(30)	Name of data type.
COLUMN_SIZE	INTEGER	Length of data type name.
LITERAL_PREFIX	CHAR(30)	Optional prefix for a literal of this type.
LITERAL_SUFFIX	CHAR(30)	Optional suffix for a literal of this type.
CREATE_PARAMS	CHAR(30)	A description of how to specify length attributes for the data type.
NULLABLE	SMALLINT	One of: 1 = A not null constraint can be used with this type in a domain or column definition 0 = A not null constraint can not be used with this type in a domain or column definition.
CASE_SENSITIVE	SMALLINT	One of: 1 = The data type name is case sensitive 0 = The data type name is not case sensitive.
SEARCHABLE	SMALLINT	One of: 3 = The data type can be used with any comparison operator 2 = The data type can be used with any comparison operator except like 1 = The data type can only be used with the LIKE operator 0 = The data type can not be used with any comparison operator.
UNSIGNED_ATTRIBUTE	SMALLINT	One of: 1 = The data type is unsigned 0 = The data type is signed The value is null if the data type is non-numeric.
FIXED_PREC_SCALE	SMALLINT	One of: 1 = The data type has a determined precision and scale 0 = The data type does not have a determined precision and scale.
AUTO_UNIQUE_VALUE	SMALLINT	One of: 1 = The data type will generate unique values 0 = The data type will not generate unique values.
LOCAL_TYPE_NAME	CHAR(30)	Local name for data type.
MINIMUM_SCALE	SMALLINT	Minimum value for scale for a numeric data type.
MAXIMUM_SCALE	SMALLINT	Maximum value for scale for a numeric data type.
SQL_DATA_TYPE	SMALLINT	A numeric value representing the data type.
SQL_DATETIME_SUB	SMALLINT	Sub type for an interval data type.
NUM_PREC_RADIX	SMALLINT	Radix for numeric data type.
INTERVAL_PRECISION	SMALLINT	Precision for an interval data type.

PRECISION	INTEGER	Precision for data type.
INTERNAL_LEN_DIFF	SMALLINT	Internal length difference.
INTERNAL_TYPENAME	CHAR(30)	Internal type name.
INTERNAL_VERSION	SMALLINT	Internal version number.
INTERNAL_TYPEORDER	SMALLINT	Internal type ordering number.
Primary key: DATA_TYPE, TYPE_NAME		

USAGE_PRIVILEGES

Records instances of privileges which grant the right to use a database object.

Column name	Data type	Description
OBJECT_SYSID	INTEGER	System identifier of the database object.
OBJECT_PRIVILEGE	CHAR(20)	One of: "DATABANK" "BACKUP" "SHADOW" "IDENT" "MEMBER" "PROGRAM" "TABLE" "DOMAIN" "MODULE" "PROCEDURE" "FUNCTION" "SCHEMA" "SEQUENCE" "CHARACTER SET" "COLLATION" "TRANSLATION".
GRANTEE	VARCHAR(128)	The name of the ident to whom the usage privilege was granted.
GRANTOR	VARCHAR(128)	The name of the ident granting the usage privilege.
IS_GRANTABLE	CHAR(3)	One of: "YES" = WITH GRANT OPTION is held with the privilege "NO" = WITH GRANT OPTION is not held with the privilege.
Primary key: OBJECT_SYSID, OBJECT_PRIVILEGE, GRANTEE, GRANTOR		
Secondary index: GRANTEE		
Secondary index: GRANTOR		

USERS

Records idents (group, OS_USER, program or user) in the database.

Column name	Data type	Description
USER_NAME	VARCHAR(128)	The ident name.
USER_SYSID	INTEGER	System identifier for the ident.
USER_TYPE	CHAR(20)	One of: "USER" "OS_USER" "PROGRAM" "GROUP".
USER_PASSWORD	BINARY(18)	The encoded password for the ident.
USER_PASSWORD_MINIMUM_LENGTH	INTEGER	The minimum length for a password.
Primary key: USER_NAME		
Unique constraint: USER_SYSID		

VIEWS

Records views in the database.

Column name	Data type	Description
VIEW_SYSID	INTEGER	System identifier of the view.
CHECK_OPTION	CHAR(20)	One of: "CASCADED" "LOCAL" "NONE".
IS_UPDATABLE	CHAR(3)	One of: "NO" = The view can not be updated "YES" = The view can be updated.
IS_INSTEAD_OF	CHAR(3)	One of: "NO" = The view can be updated per se "YES" = The view can be updated due to the existence of an instead of trigger.
VIEW_DEFINITION	VARCHAR(2000)	The text of the view definition.
Primary key: VIEW_SYSID		

INDEX

A

- access privileges 2-13
- authorization
 - database statistics 10-1
 - DBC program 6-2
 - DBOPEN functionality 7-2
 - export functions 11-2
 - import functions 11-7
 - load and unload 11-10

B

- background threads 4-4
- backup and restore 5-1
 - SQL statements 5-8, 5-9
- backups of databanks 5-5
- balanced I/O 2-8
- BSQL command-line arguments 3-13
- bufferpool 4-2

C

- cascade effects with drop and revoke 2-14
- CHECK constraints in tables 2-17
- CHECK OPTION in views 2-17
- command-line arguments
 - BSQL 3-13
 - MIMCONTROL 4-6
 - MIMINFO 4-11
 - MIMLICENSE 3-6
 - SDBGEN 3-8, 3-9
 - UTIL 3-13
- concurrency control 2-10
- connecting to a database 3-10
- creating Mimer SQL system databanks 3-7

D

- data dictionary 2-1
- data dictionary tables E-1
- data integrity 2-16
- databank check (DBC) functionality 6-1
 - authorization 6-2
 - error messages 6-5
 - error status 6-2
 - syntax 6-1
- databank open (DBOPEN) functionality 7-1
 - authorization 7-2
 - error status 7-2
 - syntax 7-1

- DATABANK privilege 2-13
- databanks
 - allocating disk space 2-6
 - backups 5-5
 - changing location 2-9
 - data security 2-7
 - disk I/O 2-8
 - file access 2-9
 - file deletion 2-10
 - locating files 2-5
 - moving between systems 11-1
 - options 2-4
 - organizing 2-6
 - physical integrity check 6-1
 - recreating the system databanks 5-11
 - restoring from backup 5-10
 - system 2-3
 - user 2-4
- database
 - administration 1-1
 - client-server interface 3-3
 - connection 3-10
 - creating “HOTEL” example 3-14
 - default (node-specific) 3-11
 - default (user-specific) 3-11
 - ident and data structure 3-9
 - listing connected users 4-12
 - local 3-2
 - names 3-10
 - remote 3-3
 - removal 3-14
 - security 2-11
 - server control 4-5
 - server log 4-18
 - server memory requirements 4-1
 - single-user mode A-1
 - statistics 10-1
 - system information 4-10
 - system performance parameters 4-1
 - system resource requirements 4-4
 - troubleshooting connect failures 3-12
- database performance parameters 4-1
 - bufferpool 4-2
 - number of background threads 4-4
 - number of request threads 4-4
 - SQLPOOL 4-3
- DBC 6-1
- DBC functionality 6-1
 - authorization 6-2
 - error messages 6-5
 - error status 6-2
 - syntax 6-1
- DBOPEN functionality 7-1
 - authorization 7-2
 - error status 7-2
 - syntax 7-1
- DEFAULT section B-3
- DELETE privilege 2-13

- domains 2-16
 - default value 2-16
- drop log 5-9
- dropping objects - cascade effects 2-14
- duplicate rows - handling in import load 11-6

E

- entity integrity 2-16
- EXECUTE privilege 2-13
- export 11-2
 - authorization 11-2
 - file contents 11-3

F

- foreign keys in import 11-5

G

- GRANT OPTION 2-11, 2-14
- group idents 2-2

I

- IDENT privilege 2-13
- idents 2-2
 - recommended organization 2-12
- import 11-3
 - authorization 11-7
 - data loading 11-6
 - naming conflicts 11-4
 - object creation 11-3
 - referential conflicts 11-5
- INSERT privilege 2-13
- integrity
 - domains 2-16
 - foreign key 2-17
 - in view definitions 2-17
 - of databank files 6-1
 - primary key 2-16
 - within tables 2-17

L

- license key 3-4
- load and unload 11-7
 - authorization 11-10
 - data files 11-7
 - duplicate rows 11-9
 - NULL indicators in data files 11-8
- loading data into tables 11-8
- local database 3-2
- LOCAL section B-3
- locating databank files 2-5
- LOG databank option 2-4
- LOGDB 2-4, 5-2
 - information contained in 5-3
 - initial creation 3-7
 - reading contents of 9-1

M

- MEMBER privilege 2-13
- MIMCONTROL 4-5
 - syntax 4-6

- mimdump 4-18
- mimer log 4-18
- Mimer SQL license key 3-4
- Mimer SQL page size 3-7
- Mimer SQL system databanks 2-3
 - recreating after loss 5-11
- MIMER_MODE variable A-2
- MIMINFO 4-10
 - bufferpool report 4-18
 - mimserv report 4-12
 - SQLPOOL report 4-18
 - syntax 4-11
 - users list 4-12
- MIMLICENSE
 - syntax 3-6
- MULTIDEFS file C-1
- N**
- NULL databank option 2-5
- NULL indicators in load and unload data files 11-8
- O**
- object privileges 2-13
- optimistic concurrency control 2-10
- OS_USER idents 2-2
- P**
- page size 3-7
- performance
 - database system parameters 4-1
- privileges
 - access 2-13
 - object 2-13
 - system 2-13
- program idents 2-2
 - using in import/export 11-7
- PUBLIC logical group 2-3
- R**
- READLOG functionality 9-1
 - listing restrictions 9-2
 - output destination 9-2
 - output format 9-4
- REFERENCES privilege 2-14
- referential integrity 2-17
- relocating databanks
 - system 2-9
 - user 2-9
- remote database 3-3
- REMOTE section B-3
- request threads 4-4
- reset log 5-9
- restore 5-10
- restriction views 2-15
- revoking privileges - cascade effects 2-14
- S**
- SDBGEN 3-7
 - syntax 3-8, 3-9

- SELECT privilege 2-13
- shadowing in backup and restore 5-1
- 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 10-1
- SQL statements for managing databank backups 5-6
- SQLDB 2-4
 - backups of 5-5
 - initial creation 3-7
- SQLHOSTS file B-1
- SQLPOOL 4-3
- statistics on data access 10-1
 - authorization 10-1
- SYSADM 1-1, 2-3
 - creation 3-7
 - privileges 1-2
- SYSDB 2-4
 - initial creation 3-7
- system (machine) administration 1-1
- system databanks 2-3
 - recreating after loss 5-11
- system information - MIMINFO 4-10
- system privileges 2-13

T

- table integrity 2-17
- TABLE privilege 2-13
- threads
 - background 4-4
 - request 4-4
- TRANS databank option 2-5
- transaction control 2-11
- transactions 2-10
 - build-up 2-10
 - read-set 2-11
 - write-set 2-10
- TRANSDB 2-4
 - backups of 5-4
 - initial creation 3-7

U

- unloading data from tables 11-9
- UPDATE privilege 2-14
- USAGE ON DOMAIN privilege 2-13
- user databanks 2-4
- users list - MIMINFO 4-12
- UTIL command-line arguments 3-13

V

- view integrity 2-17
- views - use in database security 2-15

