



MIMER/SQL

Interactive User's Manual

Version 7.3

Copyright © 1996 Sysdeco Mimer AB

MIMER/SQL version 7.3 Interactive User's Manual

November, 1996

Copyright © 1996 Sysdeco Mimer AB.

Published by Sysdeco Mimer AB,

P.O.Box 1713,

S-751 47 Uppsala, Sweden.

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

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

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

Produced by Sysdeco Mimer AB, Uppsala, Sweden.

All rights reserved under international copyright conventions.

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

FOREWORD

Documentation objectives

This manual is intended primarily for interactive SQL (Structured Query Language) users, and for users with little or no experience of the SQL language. It describes how to use SQL for creating and manipulating the database contents, and includes a detailed reference of the facilities within Interactive SQL (ISQL) and Batch SQL (BSQL).

This manual does not attempt give an exhaustive description of SQL. Refer to the MIMER/SQL Reference Manual for a complete syntax description of the SQL statements.

Prerequisites

There are no prerequisites for users of this manual. However, it is to the user's advantage to be familiar with the principles of relational database management when working with ISQL and BSQL.

Organization of this manual

This manual is divided into two main sections, dealing respectively with SQL database management facilities and the MIMER Interactive and Batch SQL interfaces.

Chapter 1 is a brief introduction to this manual.

Chapters 2-8 describe how to use SQL for database management, and may be used as a guide to SQL for users not familiar with the language:

Chapter 2 presents the general concepts of the MIMER database management system and MIMER/SQL. To a large extent, these concepts are common to other database management systems which support the SQL standards.

Chapter 3 describes how to manage connections (logging on) to a MIMER database.

Chapter 4 describes how to retrieve data from a database using SELECT statements.

Chapter 5 describes how to change the database contents using DELETE, INSERT and UPDATE statements.

Chapter 6 describes transaction handling in the MIMER database system.

Chapter 7 describes how to create database objects (databanks, idents, domains, tables etc).

Chapter 8 describes how to manage access rights and privileges in the database.

Chapters 9-13 describe MIMER Interactive SQL and Batch SQL (ISQL and BSQL) facilities:

Chapter 9 describes how to run ISQL.

Chapter 10 provides a reference to ISQL commands.

Chapter 11 describes the BSQL facility.

Chapter 12 describes how variables can be handled in ISQL and BSQL.

Chapter 13 describes error handling in ISQL and BSQL.

The manual also contains three appendices:

- Appendix A** contains machine dependent information.
- Appendix B** describes the MIMER editor.
- Appendix C** lists the structure and contents of an example database provided with the MIMER distribution and used in the examples in this manual.

Related MIMER publications

- **MIMER/SQL Reference Manual** contains a complete description of the syntax and usage of all statements in MIMER/SQL. The Reference Manual is a necessary complement to this manual.
- **MIMER/SQL Programmer's Manual** contains a description of how MIMER/SQL can be used within the context of application programs, written in conventional programming languages.
- **MIMER Shadowing Manual** describes Shadowing, which allows several concurrent copies of a databank to be updated at the same time. This provides extra resilience to disk crashes and allows “backup on the fly”.
- **MIMER System Management Handbook** describes system administration functions, including export/import, backup/restore, and the statistics functionality. The information in this manual is used primarily by the system administrator, and is not required by application program developers.
- **Machine-dependent information** comprises an Installation Guide and a Users Guide for each machine environment where MIMER is available. The Installation Guide is required only by the system administrator. The Users Guide contains machine-specific information relevant to the use of MIMER in the environment concerned, and should be available to all users.
- **Other MIMER manuals** are required only when the other MIMER modules are used.

Suggestions for further reading

We can recommend to users of MIMER/SQL the many works of C. J. Date. His insight into the potential and limitations of SQL, coupled with his pedagogical talents, makes his books invaluable sources of study material in the field of SQL theory and usage. In particular, we can mention:

The Guide to the SQL Standard (Third Edition, 1994). ISBN: 0-201-55822-X. This work contains much constructive criticism and discussion of the SQL standard including, “SQL2”.

Official documentation of the accepted SQL standards may be found in:

ISO/IEC 9075:1989 Database Language SQL with integrity enhancement. This document describes the standard referred to as SQL1.

ISO/IEC 9075:1992(E) Information technology - Database languages - SQL. This document contains the standard referred to as SQL2.

X/Open Portability Guide, issue 3, volume 5, Data Management. ISBN: 0-136858767. This document contains the specification referred to as XPG3 SQL.

CAE specification, Structured Query Language (SQL). X/Open document number: C201. ISBN: 1 872630 58 8. This document contains the X/Open-92-SQL specification.

CAE specification, Data Management: Structured Query Language (SQL), Version 2. X/Open document number: C449. ISBN: 1-85912-151-9.

Acronyms and trademarks

IEC	International Electrotechnical Commission
ISO	International Standards Organization
SQL	Structured Query Language
X/Open	X/Open is a trademark of the X/Open Company

CONTENTS

1	INTRODUCTION	
2	BASIC CONCEPTS OF MIMER/SQL	
2.1	The MIMER relational database.....	2-1
2.1.1	The data dictionary.....	2-1
2.1.2	Databanks.....	2-2
2.1.3	Idents.....	2-2
2.1.4	Tables.....	2-3
2.1.5	Base tables and views.....	2-5
2.1.6	Primary keys and indexes.....	2-6
2.1.7	Synonyms.....	2-7
2.1.8	Shadows.....	2-7
2.2	Data integrity.....	2-8
2.2.1	Domains.....	2-8
2.2.2	Foreign keys – Referential integrity.....	2-8
2.2.3	Check conditions.....	2-9
2.2.4	Check options in view definitions.....	2-10
2.3	Access rights and privileges.....	2-10
2.4	MIMER/SQL statements.....	2-11
3	MANAGING DATABASE CONNECTIONS	
3.1	Database connections.....	3-1
3.1.1	Connecting to a database.....	3-1
3.1.2	Changing connections.....	3-2
3.1.3	Disconnecting.....	3-2
3.2	Program idents – ENTER and LEAVE.....	3-3
4	RETRIEVING DATA FROM TABLES	
4.1	Retrieval from single tables.....	4-1
4.1.1	Simple retrieval.....	4-1
4.1.2	Setting column labels.....	4-2
4.1.3	Eliminating duplicate values.....	4-3
4.1.4	Selecting specific rows.....	4-4
4.1.5	Retrieving computed values.....	4-8
4.1.6	Using set functions.....	4-10
4.1.7	Grouped set functions: the GROUP BY clause.....	4-12
4.1.8	Selecting groups: the HAVING clause.....	4-13
4.1.9	Ordering the result table.....	4-13
4.1.10	Using scalar functions.....	4-15
4.1.11	Using CASE expression.....	4-16
4.1.12	Using CAST specification.....	4-17
4.1.13	Datetime arithmetic and functions.....	4-18
4.2	Retrieving data from more than one table.....	4-20
4.2.1	The join condition.....	4-20
4.2.2	Simple joins.....	4-22
4.2.3	Outer joins.....	4-24

4.2.4	Nested selects	4-25
4.2.5	Ordering nested queries	4-27
4.2.6	Correlation names	4-27
4.2.6.1	Simplifying complex queries	4-28
4.2.7	Retrieving with EXISTS, NOT EXISTS	4-30
4.2.8	Retrieval with ALL, ANY, SOME	4-31
4.2.9	Union queries	4-33
4.3	Handling NULL values.....	4-36
4.3.1	Searching for NULL	4-36
4.3.2	Null values in ALL, ANY, IN and EXISTS queries	4-37
4.4	Conceptual description of the selection process	4-39
5	CHANGING TABLE CONTENTS	
5.1	Inserting data.....	5-1
5.1.1	Inserting explicit values	5-2
5.1.2	Inserting with a subselect	5-3
5.1.3	Inserting NULL values.....	5-3
5.2	Updating tables	5-4
5.3	Deleting rows from tables.....	5-5
5.4	Updatable views	5-5
6	MANAGING TRANSACTIONS	
6.1	Transactions.....	6-1
6.2	Logging	6-1
6.3	Handling transactions.....	6-2
6.3.1	Transaction handling in BSQL and ISQL	6-2
6.3.2	Consistency within a transaction	6-3
6.3.2	Consistency within a transaction	6-3
7	DEFINING THE DATABASE	
7.1	Creating idents	7-1
7.2	Creating databanks	7-2
7.3	Creating domains	7-3
7.3.1	Domains with default values	7-3
7.3.2	Domains with check clauses.....	7-4
7.4	Creating tables	7-4
7.4.1	Column definitions	7-6
7.4.2	The primary key	7-6
7.4.3	Foreign keys - referential integrity	7-6
7.4.4	Check conditions.....	7-7
7.5	Creating views	7-8
7.5.1	Check options.....	7-10
7.6	Creating secondary indexes.....	7-10
7.7	Creating synonyms	7-11
7.8	Commenting objects.....	7-12
7.9	Altering databanks, tables and idents	7-12
7.9.1	Altering a databank	7-12
7.9.2	Altering tables.....	7-13
7.9.3	Altering idents.....	7-14
7.9.4	Objects which may not be altered	7-14
7.10	Dropping objects from the database.....	7-15
7.10.1	Dropping databanks and tables.....	7-15
7.10.2	Dropping domains	7-15
7.10.3	Dropping idents.....	7-16

8	DEFINING PRIVILEGES	
8.1	Ident hierarchy	8-2
8.2	Granting privileges	8-3
8.2.1	Granting system privileges	8-3
8.2.2	Granting object privileges	8-3
8.2.3	Granting access privileges	8-4
8.3	Revoking privileges	8-5
8.3.1	Revoking system privileges	8-5
8.3.2	Revoking object privileges	8-5
8.3.3	Revoking access privileges	8-5
8.3.4	Recursive effects of revoking privileges	8-6
9	USING ISQL	
9.1	Starting ISQL	9-1
9.2	The ISQL editor	9-2
9.2.1	Screen window setup	9-3
9.2.2	Viewing result sets	9-3
9.2.3	Menu displays in ISQL	9-4
9.3	Entering SQL statements	9-4
9.4	Leaving ISQL	9-5
10	ISQL COMMANDS	
	BACKWARD	10-3
	BOTTOM	10-3
	CANCEL	10-4
	COMMAND	10-5
	CONTINUE	10-5
	DEFAULT	10-6
	DESCRIBE	10-6
	DOWN	10-11
	EXECUTE	10-12
	EXIT	10-12
	FORWARD	10-13
	HELP	10-13
	LEFT	10-14
	LIST	10-14
	LOAD	10-17
	NEXT	10-20
	PERFORM	10-20
	PFK	10-20
	PREVIOUS	10-21
	PRINT	10-21
	PROFILE	10-22
	QUIT	10-23
	READ	10-23
	REMOVE	10-24
	RIGHT	10-24
	SET HEADER	10-24
	SET INITPAGE	10-25
	SET KEY	10-25
	SET LINESPACE	10-25
	SET MENUSTYLE	10-26
	SET PAGELENGTH	10-26
	SET PAGENUMBER	10-26
	SET PAGEWIDTH	10-27
	SET TOPLABEL	10-27
	SKIP	10-28
	TOGGLE	10-28

	TOP.....	10-28
	UNLOAD.....	10-29
	UP.....	10-30
	WDW.....	10-30
	WRITE.....	10-30
11	BSQL COMMANDS	
11.1	Running BSQL.....	11-1
11.1.1	Running BSQL from a batch job.....	11-1
11.1.2	Running BSQL via the terminal.....	11-2
11.2	BSQL commands.....	11-2
	CLOSE.....	11-3
	DESCRIBE.....	11-4
	EXIT.....	11-4
	HELP.....	11-5
	LIST.....	11-5
	LOAD.....	11-6
	LOG.....	11-7
	READ INPUT.....	11-7
	SET ECHO.....	11-8
	SET LINECOUNT.....	11-8
	SET LINESPACE.....	11-9
	SET LINEWIDTH.....	11-9
	SET LOG.....	11-9
	SET MESSAGE.....	11-10
	SET OUTPUT.....	11-10
	SET PAGELength.....	11-10
	SET PAGEWIDTH.....	11-11
	SHOW SETTINGS.....	11-11
	UNLOAD.....	11-12
	WHENEVER.....	11-13
12	VARIABLES IN ISQL AND BSQL	
12.1	Host variables in ISQL.....	12-2
12.2	Host variables in BSQL.....	12-2
13	ERROR HANDLING	
13.1	Message display.....	13-1
13.2	ISQL error messages.....	13-1
13.2.1	Semantic errors.....	13-2
13.2.2	Syntax errors.....	13-2
13.3	ISQL and BSQL error messages.....	13-4
A	MACHINE-DEPENDENT INFORMATION	
B	THE MIMER EDITOR	
B.1	General description.....	B-1
B.2	Text buffer and screen utilization.....	B-1
B.2.1	Text buffers.....	B-1
B.2.2	Screen utilization.....	B-2
B.2.3	The current line and the cursor.....	B-2
B.2.4	Standard menu types.....	B-3
B.2.5	Setting screen parameters (WDW command).....	B-4
B.3	Prefix commands.....	B-7
B.3.1	General command principles.....	B-7
B.3.2	The prefix commands.....	B-8
B.3.3	Command priorities.....	B-10

B.4	Command line commands.....	B-10
B.4.1	General command principles.....	B-10
B.4.2	Command syntax.....	B-10
B.4.3	Repeating commands.....	B-11
B.4.4	Moving the window	B-11
B.4.5	Line manipulation.....	B-13
B.4.6	String manipulation	B-14
B.4.7	Screen window setup	B-16
B.4.8	The scale line	B-17
B.4.9	Function key (PFK) assignments	B-18
C	EXAMPLE DATABASE	
C.1	Tables in the example database.....	C-1
C.2	Table descriptions	C-2
C.3	The tables.....	C-4
C.4	CREATE statements for example database.....	C-8

1 INTRODUCTION

MIMER is an advanced database management system developed by Sysdeco Mimer AB. The database management language MIMER/SQL (Structured Query Language) is compatible in all essential features with the currently accepted SQL standards (see the MIMER/SQL Reference Manual for details).

MIMER/SQL is available through four user interfaces:

- Interactive SQL (ISQL) is a full-screen interactive module offering the MIMER/SQL language for direct creation and manipulation of the database. No programming is required.
- Batch SQL (BSQL) is a line-oriented interface designed for use from command files and scripts. It may also be used from a printing terminal console.
- Embedded SQL (ESQL) is used through a host programming language - the programmer writes SQL statements as part of the source code for an application program, which is pre-processed and compiled with the appropriate language-specific facilities. The SQL statements are executed in the context of the application program.
- ODBC is a database independent interface specified by Microsoft. Through ODBC MIMER can support many of the tools available in the Microsoft Windows environment.

This manual describes the usage of SQL in the interactive and batch environments. ESQL is described in the MIMER/SQL Programmers Manual. A full description of the syntax and function of SQL statements is given in the MIMER/SQL Reference Manual.

New features in MIMER version 7

Version 7 of MIMER/SQL represents a major step towards implementation of the officially accepted SQL standards (see the MIMER/SQL Reference Manual for a discussion of the current standards). The syntax of SQL statements is documented in the MIMER/SQL Reference Manual. Version 7 of MIMER/SQL is fully backward-compatible with version 5.

In addition, version 7 of MIMER introduces support for connection to multiple databases (although only one connection may be active at any one time). Databases may be physically remote from the application program, accessed over networks through client/server support. Chapter 3 of this manual describes the statements available for managing database connections in SQL.

2 BASIC CONCEPTS OF MIMER/SQL

This chapter provides a general introduction to the basic concepts of MIMER databases and MIMER/SQL. It is an important introduction for users who have little or no previous knowledge of the MIMER system or SQL.

2.1 The MIMER relational database

A database is a collection of information organized so that storage, retrieval, and modification of the data is as efficient as possible. The MIMER database is a "relational" database, which means that the information in the database is presented to the user in the form of tables. These tables represent a *logical* description of the contents of the database. The actual physical storage format may well be something else, and is of no significance to the database user.

The term database refers to the entire collection of information in a MIMER system. In addition to the information itself, this term includes the *data dictionary*, which is a set of tables describing the organization of the database, used primarily by the database management system itself. The database, although located on a single physical system, may be accessed from several distinct systems, even at remote geographical locations (linked over a network through client/server support). Commands are available for managing the *connections* to different databases (see Chapter 3), so the actual database being accessed may change during the course of a SQL session. At any one time, however, the database may be regarded as one single organized collection of information.

2.1.1 The data dictionary

The data dictionary is part of a MIMER database, it contains information on all objects stored in the database and their inter-relationships to one another. The data dictionary stores information about:

- Databanks
- Access rights and privileges
- Views
- Indexes
- Shadows
- Idents
- Tables
- Domains
- Synonyms

These objects can be divided into two groups:

- System objects are global to the system. System object names must be unique for each object type since they are common to all users. System objects include databanks, shadows and idents.
- Private objects are 'owned' by their creator and have names that are local to the creator. Two different users may create two different objects with the same name as long as they are not both system objects. Private objects include tables, views, domains, indexes and synonyms.

Private objects are fully identified by the name of the creator and the name of the object: *creator.object*. Conflicts arising from the use of the same object name by two users are avoided in this way. In any context, a private object name without explicit reference to the creator is assumed to belong to the current user.

2.1.2 Databanks

The term databank refers to the physical file where a collection of tables is stored. A database may include any number of databanks. There are two types of databanks in the MIMER system:

- System databanks contain system information used by the database manager. These databanks are defined when the system is installed. The major system databanks are SYSDB (containing the data dictionary tables), TRANSDB (used for transaction handling), LOGDB (used for transaction logging) and SQLDB (used in transaction handling and for temporary storage of internal work tables).
- User databanks contain the user tables. These databanks are defined by the user(s) responsible for setting up the database.

The division of tables between different user databanks is a matter of file organization on the host computer, and does not affect the way database contents are presented to the user. Except in special situations (such as creating tables), databanks are completely invisible to the user.

2.1.3 Idents

An ident is an authorization-id used to identify users, programs and groups. There are four types of idents in a MIMER database:

- **User idents** identify individual users. User idents can connect to MIMER interactively, in ISQL or in other MIMER modules. Access to the database through user idents is protected by a password and is restricted by the specific privileges granted to the user. User idents are generally associated with specific physical individuals authorized to use the system.

- **OS_USER idents** are idents which allow the user currently logged in to the operating system to access the MIMER database without providing a username or password. For example, if the current operating system user is ALBERT, and ALBERT is defined as an OS_USER in MIMER, ALBERT may connect directly to MIMER simply by pressing <return> at the Username: prompt. If an OS_USER ident is defined with a password in MIMER, the ident may connect to MIMER in the same way as any other user ident (i.e. by providing a username and password). An OS_USER ident may not have the same name as another user ident in the database.
- **Program idents** may not connect to MIMER, but may be entered from within an application program or interactive environment by using the ENTER statement. The ENTER statement must be preceded by a successful CONNECT statement. Entering a program ident is analogous to connecting for 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.
- **Group idents** 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. All idents are automatically members of the global group ident PUBLIC.

2.1.4 Tables

Data in relational databases is logically organized in tables, which consist of horizontal rows and vertical columns. Columns are identified by a column-name. Each row in a table contains data pertaining to a specific entry in the database. Each field, defined by the intersection of a row and a column, contains one item of data. For example, a table containing information on the guests staying at a particular hotel may have columns for the guest's reservation number, guest name, address and check-in and check-out dates:

GUESTS				
RESERVATION	GUEST	ADDRESS	CHECKIN	CHECKOUT
1352	MARK FRANCIS	VIMPELGATAN 7, SKARA	1996-08-14	1996-08-15
1363	PAULE LE FEVRE	6 RUE PARISIEN, PARIS, FRA	1996-08-20	1996-08-26
1367	ERNST JOHNSON	DALGATAN 51, SALA	1996-09-06	1996-09-07
1382	JULIO PEREZ	CARLOTA 7, MADRID, SPAIN	1996-09-29	-
1384	SIGWARD PERSSON	GROPGATAN 43A, VADSTENA	1996-09-25	-
1385	RUNE NYQVIST	KARPV. 33, NYBROVIK	1996-09-25	-
1412	JOHAN TORP	GRANDV. 77, KRISTIANSTAD	1996-09-30	-

Each entry in the table must have the same set of data items, but not all the items need to be filled in. Thus, Julio Perez in the table above does not have a check-out date listed, but the table has space in the row for Julio in the CHECKOUT column. Similarly, if the table had been constructed with an additional column for telephone numbers, there would automatically be space for this item for every guest.

All fields in any one column contain the same type of information and are of the same physical length. This length and type of information is defined in a data type. The data types supported by MIMER are (see the MIMER/SQL Reference Manual for a detailed description of data types):

- CHARACTER(n) - character string of fixed length n, $1 \leq n \leq 15000$
- VARCHAR(n) - character string of variable length with maximum length n, $1 \leq n \leq 15000$
- INTEGER(p) - integer number of precision p digits, $1 \leq p \leq 45$
- SMALLINT - integer number of precision 5 digits (-32768 through 32767)
- INTEGER - integer number of precision 10 digits, (-2,147,483,648 through 2,147,483,647)
- DECIMAL(p,s) - decimal number of precision p and scale s, $1 \leq p \leq 45$, $0 \leq s \leq p$ (e.g. 12.345 has precision 5 and scale 3)
- FLOAT(p) - floating point number with mantissa precision p, $1 \leq p \leq 45$ (zero or absolute value 10^{-999} to 10^{+999})
- REAL - floating point number with mantissa precision 7 (zero or absolute value 10^{-38} to 10^{+38})
- FLOAT - floating point number with mantissa precision 15 (zero or absolute value 10^{-38} to 10^{+38})
- DOUBLE PRECISION - floating point number with mantissa precision 15 (zero or absolute value 10^{-38} to 10^{+38})
- DATETIME - composed of a number of integer fields, the value represents an absolute point in time. A variety of sub-types are supported - see the MIMER/SQL Reference Manual for details
- INTERVAL - composed of a number of integer fields, the value represents a period of time. A variety of interval types are supported - see the MIMER/SQL Reference Manual for details

Empty fields have a special NULL indicator in MIMER which means that the value is unknown. In this manual unknown (NULL) values in tables are shown as '-'.

A relational database is built up of several interdependent tables that can be joined through the values in one or more columns. Thus, the reservation number in the example here might reappear in a table of customer bills. Part of the flexibility of a relational database structure is the facility to add more tables to an existing database – for instance, a new table might be constructed for complaints from guests. The new table would use the already existing reservation number to identify guests, and no alterations of the existing data structure would be required.

2.1.5 Base tables and views

Data in a MIMER database may be regarded as being *stored* in *base tables*: these are the tables which hold the actual database contents (although the actual physical storage form may be something other than tables). Users can directly examine data in the base tables. In addition, data may be presented in *views*, which are specific parts of one or more tables. To the user, views may appear as regular tables, but operations on views are actually performed on the underlying tables. Access privileges on views and their underlying tables are completely independent of each other.

The essential difference between a table and a view is underlined by the action of the DROP command, which drops objects from the database. If a table is dropped, all data in the table is lost from the database and can only be recovered by redefining the table and re-entering data. If a view is dropped, however, the table or tables on which the view is defined remain in the database, and no data is lost. Data may, however, become inaccessible to a user who was allowed to access the view but who is not permitted to access the base table(s).

Note: Since views are logical representations of tables, all operations requested on a view are in fact performed on the underlying base table. For this reason, care must be taken when granting access privileges on views. These privileges may include inserting, updating and deleting information. As an example, deleting a row from a view removes the entire row from the underlying base table, including any columns which the user of the view was unable to access.

Views may be created to simplify presentation of data to the user (so-called *restriction views*). For example, a view may be created on the GUESTS table in the example above to include only names and dates for checkin and checkout:

GUESTS_VIEW		
GUEST	CHECKIN	CHECKOUT
MARK FRANCIS	1996-08-14	1996-08-15
PAULE LE FEVRE	1996-08-20	1996-08-26
ERNST JOHNSSON	1996-09-06	1996-09-07
JULIO PEREZ	1996-09-29	-
SIGWARD PERSSON	1996-09-25	-
RUNE NYQVIST	1996-09-25	-
JOHAN TORP	1996-09-30	-

Similarly, a view may be created to include only the rows in GUESTS where the CHECKIN column is filled and the CHECKOUT column is NULL (i.e. only guests who are currently staying at the hotel). Views of this kind are called *restriction views*.

Views may also be created to combine information from several tables (*join views*). Join views can be used to present data in more natural or useful combinations than the base tables themselves provide (the optimal design of base tables is governed by rules of relational database modelling). Join views may also contain restriction conditions.

For example, the join view below presents the names and amounts due (as separate items) for guests currently staying at the hotel (bill data is stored in a separate BILL table, linked to GUESTS through the RESERVATION column):

GUEST	AMOUNT
MARK FRANCIS	380.00
MARK FRANCIS	25.00
MARK FRANCIS	12.50
JULIO PEREZ	410.00
SIGWARD PERSSON	400.00
...	...
...	...

2.1.6 Primary keys and indexes

Rows in a base table are uniquely identified by the contents of one or more columns comprising the primary key. A table cannot contain two rows with the same primary key value. (If the primary key contains more than one column, the key value is the combined value of all columns in the key. Individual columns in the key may contain duplicate values as long as the whole key value is unique). Other columns may also be defined as UNIQUE; these columns may not contain duplicate values but are not necessarily part of the primary key.

Columns in the primary key may not contain NULL (this is one of the requirements of a strictly relational database). Values in primary key columns cannot be updated – they can only be changed by deleting the row and inserting new values.

Primary keys columns are automatically indexed to aid in effective information retrieval. Other columns or combinations of columns may be defined at any time as a *secondary index* to improve performance in data retrieval; for instance if a search is regularly performed on a non-keyed column in a table with many rows, defining an index on the column may speed up the search. The result of the search is unaffected by the index. Note however that indexes cause some extra overhead for update, delete and insert operations. In addition, the decision whether to make use of a secondary index is made internally at the time of query execution, and there is no guarantee that an index will in fact improve performance. (SQL queries are automatically *optimized*, meaning that the database manager finds the most effective way to execute the query. In some cases, the way a query is executed internally may avoid using the index altogether.)

Indexes in MIMER are maintained automatically by the system, and cannot be accessed directly by the user.

2.1.7 Synonyms

A synonym is an alternative name for a table, view or another synonym. Synonyms can be created or dropped at any time.

Using synonyms can be a convenient way to address tables created by another user. For example, if user SAMMY creates a view called ROOM_VIEW, the full name of the view is:

```
SAMMY.ROOM_VIEW
```

JIMMY may refer to this table by its full qualified name as given above. Alternatively, he may create a synonym for the view, e.g. RM_VIEW, and then simply refer to RM_VIEW. Note that the synonym RM_VIEW is owned by user JIMMY.

2.1.8 Shadows

MIMER Shadowing is a product that can create and maintain one or more copies of a databank on different disks. This provides extra protection from the consequences of disk crashes, etc. and allows backups to be taken without stopping the MIMER system. Shadowing requires a separate licence, and is described in the MIMER Shadowing Manual.

2.2 Data integrity

A vital aspect of a MIMER database is data integrity. Data integrity means that the data in the database is complete and consistent both at its creation and at all times during use.

MIMER/SQL has four built-in functions that uphold the data integrity in the database:

- Domains
- Foreign keys (also referred to as referential integrity)
- Check statements in table definitions
- Check options in view definitions

These features should be used whenever possible for protecting the integrity of the database, guaranteeing that undesirable data is not entered into the database. By assigning data integrity constraints to the database manager, the burden of ensuring the integrity of the database is shifted from the user to the database designer.

2.2.1 Domains

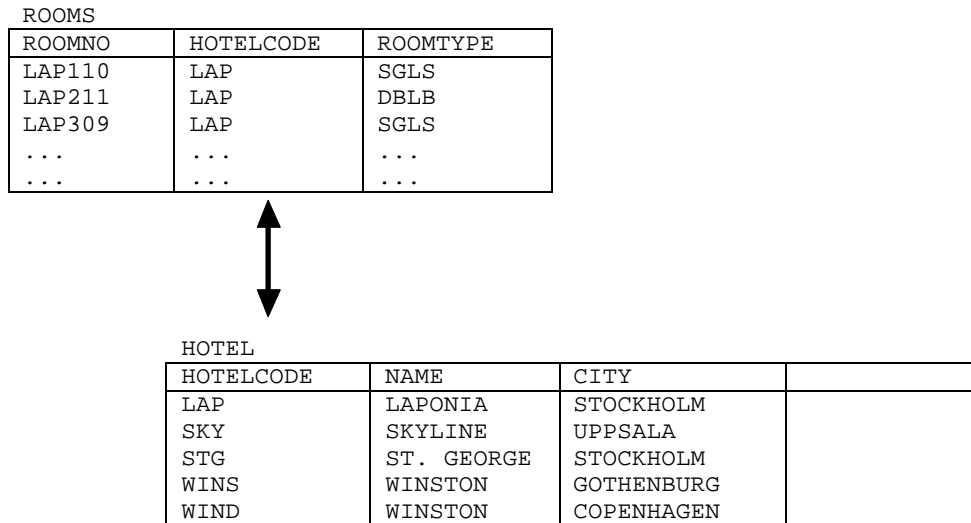
Each column in a table holds data of only one type and length, defined when the column is created. The type and length may be defined explicitly (e.g. CHARACTER(20) or INTEGER(5)), or by the use of *domains*, which can give a more precise framework for which data can be accepted in the column.

A domain definition consists of data type and length with optional restriction conditions and a default value. Data which falls outside the restriction conditions is not accepted in a column defined as belonging to the domain. A column belonging to a domain for which a default value is defined is automatically assigned that value if no value is explicitly specified.

2.2.2 Foreign keys – Referential integrity

A foreign key is one or more columns in a table defined as a cross-reference to the primary key or a unique key in another table. Data entered in the foreign key must either exist in the key of the referenced table or be NULL. This maintains *referential integrity* in the database, ensuring that references to non-existent data are not allowed. Conversely, a row in the referenced table cannot be deleted if the key value exists as a foreign key in another table. Foreign key relationships are defined when a table is created.

The following example illustrates the column HOTELCODE in table ROOMS as a foreign key referencing the primary key of table HOTEL.



In this example, there cannot be a room in a hotel that doesn't exist, and a hotel cannot be deleted if it has any rooms.

Note that the reference table must exist prior to the declaration of foreign keys on that table, unless the referenced and referencing tables are the same.

2.2.3 Check conditions

Check conditions may be specified in table and domain definitions to make sure that the values in a column conform to certain conditions. For example, the check condition in the definition of the BOOK_GUEST table (see Appendix C) specifies that a guest must be booked to arrive before he departs, and to checkout no earlier than he checks in.

Check conditions are discussed in detail in Section 7.4.4.

2.2.4 Check options in view definitions

You can maintain view integrity by including a check option in the view definition. This causes data entered through the view to be checked against the view definition. If the data conflicts with the conditions in the view definition, it is rejected.

For example, the restriction view HOTEL_STOCKHOLM is created with the following SQL statement:

```
CREATE VIEW HOTEL_STOCKHOLM
AS SELECT NAME, CITY
FROM HOTEL
WHERE CITY = 'STOCKHOLM'
WITH CHECK OPTION;
```

This means that the view HOTEL_STOCKHOLM contains NAME and CITY columns from the HOTEL table on the condition that the value in the CITY column is STOCKHOLM. Any attempts to change contents of the CITY column in the view or to insert data in the view where CITY does not contain STOCKHOLM is rejected.

2.3 Access rights and privileges

Access rights and privileges control users' access and operational privileges in the database.

User and program ident's are protected by a password, which must be given together with the correct ident name in order for a user to gain access to the database or to enter a program ident. Passwords are stored in encrypted form in the data dictionary and cannot be read by any ident including the system administrator. A password may only be changed by the ident to which it belongs or by the creator of the ident.

A set of access rights and privileges define the permitted operations for every ident in the system. There are three classes of privileges in a MIMER database:

- **System privileges**, which control the right to create new databanks and new ident's. System privileges are granted to the system administrator when the system is installed, and may be granted by the administrator to other ident's in the database. As a general rule, system privileges should be granted to a restricted group of users.
- **Object privileges**, which control membership in group ident's, the right to enter program ident's and the right to create new tables in a specified databank. The creator of an object is automatically granted full privileges on that object; thus the creator of a group is automatically a member of the group, the creator of a program may enter it, the creator of a databank may create tables in the databank, and the creator of a table has all access rights on the table. The creator of an object has the right to grant any of these privileges to other users.
- **Access privileges**, which define access to the contents of the database, i.e. the rights to retrieve data from tables or views, delete data, insert new rows, update data and use tables as foreign key references.

All privileges may be granted "with grant option" which means that the receiver has in turn the right to pass the privilege on to other users.

Privileges may be revoked at any time, but only by the original grantor. Grant options cannot be revoked without revoking the associated privilege. Section 8.3 describes revoking privileges in more detail.

2.4 MIMER/SQL statements

MIMER/SQL is a language made up of some twenty-five different statements, which may be divided into six different categories:

- data definition statements, used to create databases

CREATE	creates objects
ALTER	changes objects
DROP	drops objects
COMMENT	documents objects
- security control statements, used to manage access rights and privileges

GRANT	grants rights and privileges
REVOKE	revokes rights and privileges
- data manipulation statements, used to examine and change data in the database

SELECT	retrieves data
INSERT	adds new rows to tables
UPDATE	changes data in existing rows
DELETE	deletes data
- access control statements, used to connect and disconnect user and program users to or from the database

CONNECT	connects a user user to the database
DISCONNECT	disconnects a user user from the database
SET CONNECTION	changes the active database connection
ENTER	enters a program user
LEAVE	leaves a program user
- transaction control statements, used to control when database transactions begin and end, and when updates take effect

SET TRANSACTION	set transaction modes for subsequent transactions
START	starts a transaction build-up
COMMIT	commits the current transaction
ROLLBACK	abandons the current transaction

- database administration statements, used to manage backup/restore operations and the statistical information used to optimize transactions

CREATE BACKUP	creates a backup copy of a databank, with an optional incremental backup. Incremental backups may also be taken on their own with the statement CREATE INCREMENTAL BACKUP
ALTER DATABANK	the RESTORE variant of this statement recovers a databank from incremental backup information
SET DATABASE	sets the database ONLINE or OFFLINE
SET DATABANK	sets a databank ONLINE or OFFLINE
SET SHADOW	sets one or more shadows ONLINE or OFFLINE
UPDATE STATISTICS	updates the statistical information used for transaction optimization

The SQL statements are described in detail in subsequent chapters in this manual and in the MIMER/SQL Reference Manual.

In addition, there is a set of commands specific to the ISQL and BSQL environments, for managing screen display, printouts and so on (see Chapters 10 and 11).

Note: In ISQL and BSQL, statements are terminated by a semicolon (;). This is not strictly part of the SQL statement syntax, but is included in the examples in this manual.

3 MANAGING DATABASE CONNECTIONS

A *database* in MIMER version 7 refers to the complete collection of data which may be accessed from one MIMER system. An application *connects* to a database (referred to as *logging in* in previous versions of MIMER). A new feature in version 7 is the ability to switch between different connections (i.e. access different databases) from within the same application program. A program may have several database connections open simultaneously, although only one is active at any one time.

3.1 Database connections

3.1.1 Connecting to a database

Only idents of type USER and OS_USER are allowed to connect to MIMER. A connection is requested from ISQL with the CONNECT statement, with the general form (see the MIMER/SQL Reference Manual for details):

```
CONNECT TO 'database' [AS 'connection']  
        USER 'ident' USING 'password';
```

This statement establishes a connection between the user and a database. The database may be specified as a database name or as the keyword DEFAULT. Note that if the keyword DEFAULT is used, a user and password cannot be specified -see Section 5 of the MIMER/SQL Reference Manual. If you wish to connect to the default database and specify a user and password, specify an empty string (") for the database.

System databank locations corresponding to named databases and to the DEFAULT database are identified through the SQLHOSTS file in the current environment (see the MIMER System Management Handbook and machine-specific Users Guide for more details).

The database may be given an explicit connection name for use in DISCONNECT and SET CONNECTION statements. If no explicit name is given, the database name is used as the connection name.

3.1.2 Changing connections

A connection established by a successful `CONNECT` statement is automatically active. An application program may make multiple connections to the same or different databases using the same or different ids, provided that each connection is identified by a unique connection name. Only the most recent connection is active. Other connections are dormant, and may be made active by the `SET CONNECTION` statement.

```
SET CONNECTION connection;
```

3.1.3 Disconnecting

The `DISCONNECT` statement breaks the connection between the user and a database. The connection to be broken is specified as the connection name or as one of the keywords `ALL`, `CURRENT` or `DEFAULT`.

```
DISCONNECT connection;
```

A connection does not have to be active in order to be disconnected. If an inactive connection is broken, the application still has uninterrupted access to the database through the current (active) connection, but the broken connection is no longer available for activation with `SET CONNECTION`.

If the active connection is broken, the application program cannot access any database until a new `CONNECT` or `SET CONNECTION` statement is issued. Note the distinction between breaking a connection with `DISCONNECT` and making a connection inactive by issuing a `CONNECT` or `SET CONNECTION` for a different connection. A broken connection has no saved resources and cannot be reactivated by `SET CONNECTION`.

The table below summarizes the effect on the connection 'con1' of `CONNECT`, `DISCONNECT` and `SET CONNECTION` statements depending on the state of the connection

Statement	con1 non-existent	con1 current	con1 dormant
<code>CONNECT TO db1 AS con1</code>	con1 current	error- connection already exists	error- connection already exists
<code>DISCONNECT con1</code>	error- connection doesn't exist	con1 disconnected	con1 disconnected
<code>SET CONNECTION con1</code>	error- connection doesn't exist	ignored	con1 current
<code>CONNECT TO db2 AS con2</code>	-	con1 made dormant	con1 unaffected
<code>DISCONNECT con2</code>	-	con1 unaffected	con1 unaffected
<code>SET CONNECTION con2</code>	-	con1 made dormant	con1 unaffected

3.2 Program idents – ENTER and LEAVE

Program idents may be entered from within a SQL session by using the ENTER statement. The current user must have EXECUTE privilege on the program ident in order to perform an ENTER.

When a program ident is entered, any privileges granted to that ident become current and privileges belonging to the previous ident (i.e. the ident issuing the ENTER statement) are suspended.

Program idents are disconnected with the LEAVE statement.

The statements ENTER and LEAVE may not be issued within transactions (see Chapter 6).

4 RETRIEVING DATA FROM TABLES

This chapter describes how to retrieve information from the database. In a relational database, information retrieved from one or more *source tables* is returned in the form of a *result table* (sometimes called a *result set*). The statement used to retrieve information is SELECT.

The examples in this chapter are based on the example database included in the MIMER/SQL installation (see Appendix C).

4.1 Retrieval from single tables

4.1.1 Simple retrieval

The simplest retrievals fetch information from one table. The general form of the simple SELECT statement is

```
SELECT columns FROM table WHERE condition;
```

The column list specifies which columns to select, and the WHERE condition determines which rows to select. If no WHERE condition is specified, all rows are retrieved from the source table.

Find the name and city for all hotels.

```
SELECT NAME, CITY
FROM HOTEL;
```

NAME	CITY
LAPONIA	STOCKHOLM
SKYLINE	UPPSALA
ST. GEORGE	STOCKHOLM
WINSTON	COPENHAGEN
WINSTON	GOTHENBURG

Find the name and city for hotels in Stockholm.

```
SELECT NAME, CITY
FROM HOTEL
WHERE CITY= 'STOCKHOLM' ;
```

NAME	CITY
LAPONIA	STOCKHOLM
ST. GEORGE	STOCKHOLM

The formulation of selection conditions is described in detail in Section 4.1.4.

The columns in the result table are ordered as they are written in the `SELECT` statement, irrespective of the ordering in the source table:

```
SELECT CITY, NAME
FROM HOTEL;
```

CITY	NAME
STOCKHOLM	LAPONIA
UPPSALA	SKYLINE
STOCKHOLM	ST. GEORGE
COPENHAGEN	WINSTON
GOTHENBURG	WINSTON

A shorthand form for selecting all columns from a table is

```
SELECT * FROM table ...
```

In this case, the columns in the result table are ordered as they are defined in the source table.

Any table name in a `SELECT` statement may be qualified by the name of the creator in the form *creator.table*. Unqualified table names are implicitly qualified by the user's ident name. The table name must be written in the qualified form if it is not owned by the current user, unless it is replaced by a synonym.

Example

```
SELECT *
FROM BOOKADM.ROOMTYPES;
```

4.1.2 Setting column labels

Columns in the result table normally have the same name as the corresponding columns in the source table. By using an `AS` clause after the column name in the `SELECT` statement, the column in the result table can be given an alternative name. `AS` clauses can be used for as many columns as required. A label may be up to 18 characters long, and follows the same syntax rules as column names (see the MIMER/SQL Reference Manual).

```
SELECT NAME, CITY AS TOWN, OVERBOOK AS CAPACITY_FACTOR
FROM HOTEL;
```

NAME	TOWN	CAPACITY_FACTOR
LAPONIA	STOCKHOLM	1.10
SKYLINE	UPPSALA	1.10
ST. GEORGE	STOCKHOLM	1.10
WINSTON	COPENHAGEN	1.10
WINSTON	GOTHENBURG	1.10

Labels are particularly useful in queries that retrieve computed values, where the result table column is otherwise unnamed (see Section 4.1.5).

4.1.3 Eliminating duplicate values

The simple `SELECT` statement retrieves all rows which fulfil the selection conditions. The result table does not have a primary key, and may contain duplicate values.

```
SELECT RESERVATION, CHARGE_CODE
FROM BILL;
```

RESERVATION	CHARGE_CODE
1347	100
1347	120
1347	210
1347	100
1347	120
1348	100
1348	120
1348	200
1348	230
...	...

Adding the keyword `DISTINCT` before the column list eliminates all duplicate rows from the result table. The keyword `DISTINCT` may only be used once in a simple `SELECT` statement.

```
SELECT DISTINCT RESERVATION, CHARGE_CODE
FROM BILL;
```

RESERVATION	CHARGE_CODE
1347	100
1347	120
1347	210
1348	100
1348	120
1348	200
1348	230
...	...

`DISTINCT` also eliminates duplicate rows containing `NULL` values, although technically `NULL` is not regarded as equal to `NULL` (see Section 4.3).

If the selected columns include the whole primary key in the source table, the keyword `DISTINCT` is unnecessary, since all rows in the result table will be unique. Remember however that a view may contain duplicate rows, so that selecting all columns does not always guarantee that the result does not contain duplicate rows.

Certain select statements cannot be applied to views because the expansion of the statement to the corresponding base tables is illegal:

- If the view definition contains `DISTINCT`, you cannot select distinct values from that view because `DISTINCT` may only be used once in a simple `SELECT` statement or subselect (see Section 4.2.8 for a description of subselects). For example, the following selection is not allowed:

```
CREATE VIEW PRICE_CODES
AS SELECT DISTINCT RESERVATION, CHARGE_CODE
FROM BILL;

SELECT DISTINCT RESERVATION
FROM PRICE_CODES;
```

The query expands to:

```
SELECT DISTINCT RESERVATION
FROM (SELECT DISTINCT RESERVATION, CHARGE_CODE
      FROM BILL);
```

where the keyword **DISTINCT** appears twice.

- If the view is created with a set function (Section 4.1.6), the corresponding column in the view cannot be used in a **WHERE** clause:

```
CREATE VIEW BILL_TOTAL (CUSTOMER, TOTAL)
AS SELECT RESERVATION, SUM(AMOUNT)
FROM BILL
GROUP BY RESERVATION;

SELECT *
FROM BILL_TOTAL
WHERE TOTAL < 500;
```

Here the expansion is:

```
SELECT RESERVATION, SUM(AMOUNT)
FROM BILL
WHERE SUM(AMOUNT) < 500
GROUP BY RESERVATION;
```

with the illegal search condition 'WHERE SUM(AMOUNT) < 500'. (Note that in this case a **HAVING** clause (Section 4.1.8) may be used in place of the **WHERE** clause. The statement is then syntactically correct).

4.1.4 Selecting specific rows

Rows are selected in the **SELECT** statement according to the search condition in the **WHERE** clause. This condition relates column value(s) to expressions.

Comparison conditions

Comparison operators that may be used in the **WHERE** clause are:

- = equal to
- <> not equal to
- < less than
- <= less than or equal to
- > greater than
- >= greater than or equal to

Comparisons can be combined in the search condition using the logical operators **AND** and **OR**, and reversed using **NOT**. Each comparison must be expressed in full; for example

```
WHERE PRICE > 300 AND PRICE < 500
```

may not be expressed as

```
WHERE PRICE > 300 AND < 500
```

Character strings are compared character by character from left to right. If strings are of different lengths, the shorter is conceptually padded to the right with blanks before the comparison is made (i.e. character difference takes precedence over length difference). The collating sequence for characters is an extended ASCII character set as defined by ISO 8859-1 (see Appendix B of the MIMER/SQL Reference Manual for the exact sequence).

Retrieve the room type, price, and date from which the prices apply for all rooms with hotel code LAP and a cost of under 400.

```
SELECT ROOMTYPE, PRICE, FROM_DATE
FROM ROOM_PRICES
WHERE HOTELCODE = 'LAP' AND PRICE < 400;
```

ROOMTYPE	PRICE	FROM_DATE
SGLB	360	1996-10-01
SGLS	380	1996-06-01
SGLS	340	1996-10-01

When stating conditions on temporal data in tables, datetime and interval literals can be specified. There are also the pseudo literals `CURRENT_DATE`, `CURRENT_TIME` and `CURRENT_TIMESTAMP` which read the system clock and returns that value. If there are more than one occurrence of these in a statement the clock is only read once.

Retrieve guests who requested a wake up call at 6:00 clock today, the tenth of September

```
SELECT ROOMNO
FROM WAKE_UP
WHERE WAKE_DATE = DATE '1996-09-10'
AND WAKE_TIME = TIME '06:00:00';
```

ROOMNO
LAP112
SKY111
STG009
WIND401

Are there any guests scheduled for checkin today

```
SELECT RESERVED_FOR
FROM BOOK_GUEST
WHERE ARRIVE = CURRENT_DATE;
```

RESERVED_FOR
FREDRIK SELLIN

For an example of interval literals, see section 4.1.13 on datetime arithmetic.

Pattern conditions

LIKE and NOT LIKE are used to search for character strings that match or do not match a specified pattern.

Patterns in the LIKE condition can be written with "wildcard" characters (also called "meta-characters"):

- _ (underscore) stands for any single character
- % stands for any sequence of zero or more characters

Wildcards are only valid in LIKE statements.

Find all guests at the Hotel Laponia whose names include "HANSEN" .

```
SELECT GUEST
FROM   BOOK_GUEST
WHERE  GUEST LIKE '%HANSEN%' AND HOTELCODE = 'LAP';
```

GUEST
STEN JOHANSEN
STEFAN HANSEN

Find all guests at the Hotel Laponia whose last names do not include "HANSEN".

```
SELECT GUEST
FROM   BOOK_GUEST
WHERE  GUEST NOT LIKE '%HANSEN%' AND HOTELCODE = 'LAP';
```

GUEST
CHRISTOPHER DATE
GUNNAR ALVE
NILS KRISTOFERSEN
LARS HOLMER
KNUT KULLMER
JUDITH SMITH
ADOLF SCHMIDT
LAILA ZETTERBERG
MATS HANSSON

Remember that character strings in MIMER/SQL statements are always written within apostrophes ('). A LIKE predicate where the pattern string does not contain any wildcard characters is essentially equivalent to a basic predicate using the '=' operator, except that comparison strings in "=" comparison are conceptually padded with blanks whereas those in the LIKE comparison are not. Thus

```
'SKYLINE' = 'SKYLINE'           is true
'SKYLINE' LIKE 'SKYLINE'       is true
'SKYLINE' LIKE 'SKYLINE%'     is true
but 'SKYLINE' LIKE 'SKYLINE'   is false
```

The LIKE predicate may include an ESCAPE clause defining a character which is used to "escape" wildcard characters. A wildcard character immediately following an escape character is taken at face value. See the MIMER/SQL Reference Manual for more details.

Some other examples of searching for character strings are:

- LIKE '%P%' matches any string that contains an upper-case 'P'
- LIKE '_abc' matches any four letter character string ending with lower case 'abc'
- LIKE '%A\%' ESCAPE '\' matches any string ending with 'A%'
- LIKE 'D_d_' matches any four letter string with D and d in the first and third positions respectively. Examples of possible values: Dude, Dads.

Set conditions

The operator IN finds the values that are contained in a set of values. The set is given as a comma-separated list enclosed in parentheses. NOT IN finds values which are not contained in the specified set.

Which hotels are in Stockholm or Copenhagen?

```
SELECT NAME, CITY
FROM HOTEL
WHERE CITY IN ( 'STOCKHOLM', 'COPENHAGEN' );
```

NAME	CITY
LAPONIA	STOCKHOLM
ST. GEORGE	STOCKHOLM
WINSTON	COPENHAGEN

Which hotels are not in Stockholm or Copenhagen?

```
SELECT NAME, CITY
FROM HOTEL
WHERE CITY NOT IN ( 'STOCKHOLM', 'COPENHAGEN' );
```

NAME	CITY
SKYLINE	UPPSALA
WINSTON	GOTHENBURG

The operators BETWEEN and NOT BETWEEN are used to find values that are within or outside an interval. The interval includes the limits specified in the BETWEEN condition.

Find which room types that have prices in the range 500 to 600 at hotel LAPONIA.

```
SELECT ROOM_TYPE, PRICE
FROM ROOM_PRICES
WHERE PRICE BETWEEN 500 and 600
AND HOTELCODE = 'LAP'
```

ROOMTYPE	PRICE
DBLB	590
DBLB	530
DBLS	560
DBLS	510

Find the date, charge code and amount for items billed between 1996-08-30 and 1996-09-01 inclusive for reservation number 1371.

```
SELECT  ON_DATE, CHARGE_CODE, AMOUNT
FROM    BILL
WHERE   RESERVATION = 1371
AND     ON_DATE BETWEEN DATE '1996-08-30' AND
        DATE '1996-09-01' ;
```

ON_DATE	CHARGE_CODE	AMOUNT
1996-08-30	100	380.00
1996-08-31	100	380.00
1996-08-31	200	423.40
1996-08-31	230	93.40
1996-09-01	100	380.00
1996-09-01	270	255.60

Find the date, charge code and amount for items billed on dates outside the range 1996-08-30 and 1996-09-01 for the reservation number 1371.

```
SELECT  ON_DATE, CHARGE_CODE, AMOUNT
FROM    BILL
WHERE   RESERVATION = 1371
AND     ON_DATE NOT BETWEEN DATE '1996-08-30' AND
        DATE '1996-09-01' ;
```

ON_DATE	CHARGE_CODE	AMOUNT
1996-08-29	100	380.00
1996-08-29	230	79.00
1996-09-02	100	380.00
1996-09-02	330	150.00
1996-09-03	100	380.00
1996-09-03	200	143.00
1996-09-04	100	380.00

BETWEEN may also be used for character comparisons. Strings are compared character by character from left to right.

```
SELECT  NAME
FROM    HOTEL
WHERE   NAME BETWEEN 'SKYLINE' AND 'WINSTON' ;
```

NAME
SKYLINE
ST. GEORGE
WINSTON
WINSTON

4.1.5 Retrieving computed values

You can retrieve computed values by using arithmetic and string operators in the **SELECT** clause of the statement. The following computational operators may be used:

- + addition
- subtraction
- * multiplication
- / division
- || string concatenation

See the MIMER/SQL Reference Manual for information regarding the type and precision of the result of an arithmetic expression.

List room prices with a 12% reduction.

```
SELECT PRICE, PRICE*0.88
FROM ROOM_PRICES;
```

PRICE	
340	299.20
360	316.80
370	325.60
380	334.40
400	352.00
...	...

The computed column is unnamed by default in the result table. A label may be used to provide a name:

```
SELECT PRICE, PRICE*0.88 AS SPECIAL_RATE
FROM ROOM_PRICES;
```

PRICE	SPECIAL_RATE
340	299.20
360	316.80
370	325.60
380	334.40
400	352.00
...	...

A column may also be "computed" as a constant value, which adds an extra column to the result table:

```
SELECT PRICE, '12% reduction:', PRICE*0.88 AS SPECIAL_RATE
FROM ROOM_PRICES;
```

PRICE		SPECIAL_RATE
340	12% reduction:	299.20
360	12% reduction:	316.80
370	12% reduction:	325.60
380	12% reduction:	334.40
400	12% reduction:	352.00
...

You may also retrieve a value computed using the values in two or more columns, providing that the data types are compatible.

Retrieve hotel names prefixed with the word 'HOTEL ' and cities.

```
SELECT 'HOTEL ' || NAME, CITY
FROM HOTEL;
```

	CITY
HOTEL LAPONIA	STOCKHOLM
HOTEL SKYLINE	UPPSALA
HOTEL ST. GEORGE	STOCKHOLM
HOTEL WINSTON	COPENHAGEN
HOTEL WINSTON	GOTHENBURG

For string concatenation, column values are padded with trailing blanks to the length of the column definition. For example,

```
SELECT NAME || 'HOTEL', CITY
FROM HOTEL;
```

		CITY
LAPONIA	HOTEL	STOCKHOLM
SKYLINE	HOTEL	UPPSALA
ST. GEORGE	HOTEL	STOCKHOLM
WINSTON	HOTEL	COPENHAGEN
WINSTON	HOTEL	GOTHENBURG

When retrieving computed values, parentheses can be used to force the operation priority. Without parentheses, the normal precedence rules for arithmetic apply, i.e. multiplication and division are performed before addition and subtraction, and operators with the same precedence are evaluated from left to right.

4.1.6 Using set functions

The functions listed below can be used in the column list of the `SELECT` statement to retrieve the result of the function on a specified column. Set functions in `SELECT` statements are applied to data in the result table, not in the source table. Set functions return a single value for the whole table unless a `GROUP BY` clause is specified (see Section 4.1.7).

<code>AVG</code>	average of values (numerical columns only)
<code>COUNT</code>	number of rows
<code>MAX</code>	largest value
<code>MIN</code>	smallest value
<code>SUM</code>	sum of values (numerical columns only)

For all set functions, `NULL` values are eliminated from the column before the function is applied. The special form `COUNT(*)` counts the number of rows including `NULL` values.

The keywords `ALL` and `DISTINCT` may be used to qualify set functions. `ALL` gives a result based on all values including duplicates. `DISTINCT` eliminates duplicates before applying the function. If neither keyword is specified, duplicates are not removed.

Set functions may not be used together with direct column references in the `SELECT` list (unless the `SELECT` statement includes a `GROUP BY` clause, see Section 4.1.7). Thus

```
SELECT COUNT(HOTELCODE), NAME, CITY
FROM HOTEL;
```

is illegal.

The set functions are illustrated with results from the table

SAMPLE
1.0
2.0
2.0
2.0
3.0
3.0
4.0
5.0
-
-

(A hyphen '-' indicates NULL).

```

COUNT(SAMPLE)          8
COUNT(*)               10
COUNT(DISTINCT SAMPLE) 5
SUM(SAMPLE)             22.0
SUM(ALL SAMPLE)         22.0
SUM(DISTINCT SAMPLE)    15.0
AVG(SAMPLE)             2.75000000000
AVG(ALL SAMPLE)         2.75000000000
AVG(DISTINCT SAMPLE)    3.00000000000
MAX(SAMPLE)             5.0
MIN(SAMPLE)             1.0

```

Note that `AVG(column)` is equivalent to `SUM(column)/COUNT(column)`. However, the expression `SUM(column)/COUNT(*)` will give a different answer if the column includes NULL values. For the table above,

```

SUM(SAMPLE)/COUNT(SAMPLE)  2.75000000000  (22/8)
SUM(SAMPLE)/COUNT(*)      2.20000000000  (22/10)

```

Some further examples of set functions applied to the example database are given below.

How many rows are there in the BOOK_GUEST table?

```

SELECT COUNT(*)
FROM BOOK_GUEST;

```

How many guests have checked out (i.e. CHECKOUT is not NULL)?

```

SELECT COUNT(ALL CHECKOUT)
FROM BOOK_GUEST;

```

What is the total bill for reservation number 1359.

```

SELECT SUM(AMOUNT)
FROM BILL
WHERE RESERVATION = 1359;

```

Find the average price of single rooms in the hotel chain.

```

SELECT AVG(PRICE)
FROM ROOM_PRICES
WHERE ROOMTYPE IN ('SGLB', 'SGLS');

```

The `AVG` function returns an integer if the operand is an integer, and a decimal if the operand is decimal. To force decimal calculation of averages from an integer column, cast the column operand as decimal:

```

SELECT AVG(cast (column as decimal)) ...

```

4.1.7 Grouped set functions: the GROUP BY clause

Normally, set functions return a single value, calculated from the set of all values in the column or expression. If the SELECT statement includes a GROUP BY clause, set functions will be applied to groups of values. Columns used for GROUP BY do not have to be included in the SELECT list.

Find the most expensive single room in each hotel.

```
SELECT HOTELCODE, MAX(PRICE) AS EXPENSIVE
FROM ROOM_PRICES
WHERE ROOMTYPE = 'SGLB'
OR ROOMTYPE = 'SGLS'
GROUP BY HOTELCODE;
```

HOTELCODE	EXPENSIVE
LAP	400
SKY	370
STG	400
WIND	410
WINS	370

Using a GROUP BY clause places some restrictions on the SELECT statement:

- Only constants, columns used in the GROUP BY clause, and columns used in set functions may be included in the SELECT list
- A column used in the GROUP BY clause may not be used in a set function

How many hotels are there in each city?

```
SELECT CITY, COUNT(HOTELCODE)
FROM HOTEL
GROUP BY CITY;
```

CITY	
COPENHAGEN	1
GOTHENBURG	1
STOCKHOLM	2
UPPSALA	1

In a statement with column references in the SELECT list, all columns not used in set functions must be used as grouping columns.

For grouping purposes, NULL values are regarded as equivalent. Thus for the example table:

SAMPLE
1.0
2.0
2.0
2.0
3.0
3.0
4.0
5.0
-
-

```
SELECT SAMPLE, COUNT(*) AS NUMBER
...
GROUP BY SAMPLE;
```

SAMPLE	NUMBER
1.0	1
2.0	3
3.0	2
4.0	1
5.0	1
-	2

4.1.8 Selecting groups: the HAVING clause

The HAVING clause restricts the selection of groups in the same way that a WHERE clause restricts the selection of rows. However, in contrast to the WHERE clause, a HAVING clause may use a set function on the left-hand side of a comparison.

The HAVING clause is most often used together with a GROUP BY clause, but may also be used to impose selection conditions on a column derived from a set function.

Find the highest price for a single room in each hotel, but restrict the selection to prices over 399.

```
SELECT HOTELCODE, MAX(PRICE)
FROM ROOM_PRICES
WHERE ROOMTYPE = 'SGLB'
OR ROOMTYPE = 'SGLS'
GROUP BY HOTELCODE
HAVING MAX(PRICE) > 399;
```

HOTELCODE	
LAP	400
STG	400
WIND	410

4.1.9 Ordering the result table

Strictly, the order of rows in a result table is undefined unless an ORDER BY clause is included in the SELECT statement. Ascending or descending order may be specified; ascending order is the default. (A SELECT statement without an ORDER BY clause may *appear* always to give an ordered result in MIMER, but you should include an ORDER BY clause if the ordering is important. A change in the database contents may otherwise change the order, particularly for a complex query where the order of execution is determined by the SQL compiler).

Retrieve the hotel code, room type, from date and price for single rooms with showers with a cost of under 400 and order by the price in descending order.

```
SELECT *
FROM   ROOM_PRICES
WHERE  PRICE < 400
AND    ROOMTYPE = 'SGLS'
ORDER BY PRICE DESC;
```

HOTELCODE	ROOMTYPE	FROM_DATE	PRICE
STG	SGLS	1996-06-01	380
LAP	SGLS	1996-06-01	380
WIND	SGLS	1996-06-01	370
SKY	SGLS	1996-06-01	350
STG	SGLS	1996-10-01	340
LAP	SGLS	1996-10-01	340

More than one column may be specified in the ORDER BY clause:

```
SELECT *
FROM   ROOM_PRICES
WHERE  PRICE < 400
AND    ROOMTYPE = 'SGLS'
ORDER BY HOTELCODE, PRICE;
```

HOTELCODE	ROOMTYPE	FROM_DATE	PRICE
LAP	SGLS	1996-10-01	340
LAP	SGLS	1996-06-01	380
SKY	SGLS	1996-06-01	350
STG	SGLS	1996-10-01	340
STG	SGLS	1996-06-01	380
WIND	SGLS	1996-06-01	370

To order a result table by a set function or computed value, the column in the result table is given a label and the label is used in the ORDER BY clause:

```
SELECT ROOMTYPE, AVG(PRICE) AS AVERAGE_PRICE
FROM   ROOM_PRICES
GROUP BY ROOMTYPE
ORDER BY AVERAGE_PRICE;
```

ROOMTYPE	AVERAGE_PRICE
SGLS	360
SGLB	381
DBLS	543
DBLB	570

The following formulation is incorrect, since there is no PRICE column in the result table by which to perform the ordering:

```
SELECT ROOMTYPE, AVG(PRICE)
FROM   ROOM_PRICES
GROUP BY ROOMTYPE
ORDER BY PRICE;
```

4.1.10 Using scalar functions

These functions operate on expressions or on a single value received from a SELECT statement.

Some of the standard scalar functions available are:

CHAR_LENGTH	returns the length of a string
EXTRACT	returns a single field from a DATETIME or INTERVAL value
LOWER	converts all upper case letters in a character string to lower case
POSITION	returns the starting position of the first occurrence of a specified string expression, starting from the left, in the given character string
SUBSTRING	extracts a substring from a given string, according to specified start position and length of the substring
TRIM	removes leading and/or trailing instances of a specified character from a string
UPPER	converts all lower case letters in a character string to upper case

See the MIMER/SQL Reference Manual for the syntax rules and for information regarding the data type of the result of the scalar functions.

Here follows some examples in order to illustrate how the scalar functions may be used:

List all hotels with name Winston , spelled with either upper and lower case letters.

```
SELECT NAME , CITY
FROM HOTEL
WHERE UPPER (NAME) = 'WINSTON' ;
```

NAME	CITY
WINSTON	COPENHAGEN
WINSTON	GOTHENBURG
Winston	London

List all double rooms at hotel SKY.

```
SELECT ROOMNO , ROOMTYPE
FROM ROOMS
WHERE SUBSTRING (ROOMTYPE FROM 1 FOR 3) = 'DBL'
AND HOTELCODE = 'SKY' ;
```

ROOMNO	ROOMTYPE
SKY121	DBLS
SKY122	DBLS
SKY123	DBLS
SKY124	DBLB
SKY125	DBLB
SKY212	DBLB

Get name and address (without trailing blanks) of guest with reservation number 1348.

```
SELECT TRIM(TRAILING FROM GUEST) ||
      ', ' ||
      TRIM(TRAILING FROM ADDRESS)
FROM   BOOK_GUEST
WHERE  RESERVATION = 1348;
```

STEN JOHANSEN, MIMERGATAN 4, UPPSALA

Remove leading and trailing spaces and get length (no. of characters) of description and the description (in lower case) for all charges.

```
SELECT CHAR_LENGTH(TRIM(DESCRIPTION)), LOWER(TRIM(DESCRIPTION))
FROM   CHARGES;
```

	lodging
	telephone
	carpark
	restaurant
	minibar
	bar
	room service
	laundry
	extrabed
	miscellaneous

4.1.11 Using CASE expression

With a case expression it is possible to specify a conditional value. Depending on the result of one or more conditional expressions the case expression can return different values.

The rules for CASE expressions are described in the MIMER/SQL Reference Manual. The following select statements presents two examples of how CASE expressions can be used:

Translate the currency code in the exchange_rate table to descriptive names

```
SELECT CASE CURRENCY
      WHEN 'DDE' THEN 'German Marks'
      WHEN 'DKK' THEN 'Danish Crowns'
      WHEN 'FRF' THEN 'French Francs'
      WHEN 'GBP' THEN 'British Pounds'
      WHEN 'ITL' THEN 'Italian Lira'
      ELSE CURRENCY
      END AS CURRENCY, RATE
FROM   EXCHANGE_RATE;
```

CURRENCY	RATE
DEM	3.476
Danish Crowns	0.922
FIM	1.435
French Francs	1.041
British Pounds	10.335
Italian Lira	0.005
JPY	0.044
NOK	0.0939
SEK	1.000
USD	6.335

This form of a case expression is known as a simple case expression, in which an operand (CURRENCY in this case) is compared to a list of values. If there is a match in one of the when clauses, the result is the value to the right of the then clause. If none of these matches, the value in the else clause is returned. If there is no else clause in a case expression and no when clause matches, a null indicator is returned.

The other form of the case expression can be seen in the following example:

Divide room prices into different categories

```
SELECT CASE
      WHEN PRICE >= 600 then 'Expensive'
      WHEN PRICE <= 400 then 'Budget'
      ELSE 'Moderate'
    END AS CATEGORY, ROOMTYPE, PRICE
FROM   ROOM_PRICES;
```

CATEGORY	ROOMTYPE	PRICE
Moderate	DBLB	590
...		
Budget	SGLB	340
...		
Expensive	DBLB	600
...		

In this form it is possible that more than one of the when clauses evaluates to true, in which case the value in the first (from left) of the matching clauses is returned.

4.1.12 Using CAST specification

The cast specification explicitly converts data of one data type to another data type. Conversion between data types are allowed if the rules for assignment to the target data type are not violated. See MIMER/SQL Reference Manual for conversion rules.

List the billed charges for reservation number 1347. Convert the charged amounts to US-dollars to decimal with scale 4. Convert the date of charges (in format YYYY-MM-DD) to character in format DD/MM/YY

```
SELECT CAST(CHARGE_CODE AS SMALLINT) AS CODE,
      CAST(AMOUNT/6.335 AS DECIMAL(10,4)) AS USD,
      SUBSTRING(CAST(ON_DATE AS CHAR(10)) FROM 9 FOR 2) || '/' ||
      SUBSTRING(CAST(ON_DATE AS CHAR(10)) FROM 6 FOR 2) || '/' ||
      SUBSTRING(CAST(ON_DATE AS CHAR(10)) FROM 3 FOR 2) AS DATE
FROM   BILL
WHERE  RESERVATION = 1347
ORDER BY CODE;
```

CODE	USD	DATE
100	59.9842	04/09/96
100	59.9842	05/09/96
120	2.0047	04/09/96
120	5.1302	05/09/96
210	2.8413	04/09/96

4.1.13 Datetime arithmetic and functions

It is possible to use datetime and interval values in expressions to calculate new datetime and interval values.

Valid operations are:

- addition or subtraction between an interval value and a datetime value
- subtracting a datetime from another datetime value
- adding or subtracting two interval values
- multiplying or dividing an interval by a numerical value

The first of these operations yields a datetime value while the others result in an interval value.

How many days have the guests at hotel LAPONIA stayed?

```
SELECT  GUEST ,
        (COALESCE(CHECKOUT, CURRENT_DATE)-CHECKIN) DAY(2) AS DAYS
FROM    BOOK_GUEST
WHERE   HOTELCODE = 'LAP'
AND     CHECKIN IS NOT NULL;
```

GUEST	DAYS
CHRISTOPHER DATE	2
STEN JOHANSEN	1
STEFAN HANSEN	1
GUNNAR ALVE	2
NILS KRISTOFFERSEN	20
LARS HOLMER	4
KNUT KULLMER	3
JUDITH SCHMIDT	20
ADOLF SCHMIDT	3
LAILA ZETTERBERG	3
MATS HANSSON	6

When taking the difference between two datetime values it is necessary to specify the type of the resulting interval. It is also possible to specify the precision of the interval as shown in the example above. In that example the precision is actually superfluous as the default precision for day is 2.

Which hotel rooms have requested a wake up call within the next hour and a half

```
SELECT  ROOMNO
FROM    WAKE_UP
WHERE   WAKE_DATE = CURRENT_DATE
AND     WAKE_TIME BETWEEN CURRENT_TIME AND
        CURRENT_TIME + INTERVAL '01:30' HOUR TO MINUTE;
```

ROOMNO
WINS120

SQL distinguishes between YEAR-MONTH (long) intervals and DAY-TIME (short) intervals.

YEAR-MONTH intervals are: YEAR, MONTH and YEAR TO MONTH.

DAY-TIME intervals are: DAY, HOUR, MINUTE, SECOND, HOUR TO MINUTE, HOUR TO SECOND, MINUTE TO SECOND, DAY TO HOUR, DAY TO MINUTE and DAY TO SECOND.

It is possible to extract part of a datetime value with the EXTRACT function. The function returns a numeric value.

Which month did FREDRIK SELLIN stay at any of the hotels?

```
SELECT CASE EXTRACT (MONTH FROM ARRIVE)
        WHEN 1 THEN 'JANUARY'
        WHEN 2 THEN 'FEBRUARY'
        WHEN 3 THEN 'MARCH'
        WHEN 4 THEN 'APRIL'
        WHEN 5 THEN 'MAY'
        WHEN 6 THEN 'JUNE'
        WHEN 7 THEN 'JULY'
        WHEN 8 THEN 'AUGUST'
        WHEN 9 THEN 'SEPTEMBER'
        WHEN 10 THEN 'OCTOBER'
        WHEN 11 THEN 'NOVEMBER'
        WHEN 12 THEN 'DECEMBER'
        END AS MONTH
FROM   BOOK_GUEST
WHERE  GUEST = 'FREDRIK SELLIN' ;
```

MONTH
SEPTEMBER

Another useful function is DAYOFWEEK which returns the day number within a week. MONDAY has the value 1 and SUNDAY has the value 7.

Which day did FREDRIK SELLIN arrive at any of the hotels?

```
SELECT CASE DAYOFWEEK (ARRIVE)
        WHEN 1 THEN 'MONDAY'
        WHEN 2 THEN 'TUESDAY'
        WHEN 3 THEN 'WEDNESDAY'
        WHEN 4 THEN 'THURSDAY'
        WHEN 5 THEN 'FRIDAY'
        WHEN 6 THEN 'SATURDAY'
        WHEN 7 THEN 'SUNDAY'
        END AS DAY
FROM   BOOK_GUEST
WHERE  GUEST = 'FREDRIK SELLIN' ;
```

DAY
THURSDAY

4.2 Retrieving data from more than one table

The examples so far presented in this chapter have illustrated the essential features of simple SELECT statements with data retrieval from single tables. However, much of the power of SQL lies in the ability to perform *joins* through a single statement, i.e. to select data from two or more tables, using the search condition to link the tables in a meaningful way.

4.2.1 The join condition

In retrieving data from more than one table, the search condition or *join condition* specifies the way the tables are to be linked.

List the billed charges for reservation number 1349.

```
SELECT DESCRIPTION, AMOUNT
FROM   CHARGES, BILL
WHERE  RESERVATION = 1349
AND    BILL.CHARGE_CODE = CHARGES.CHARGE_CODE;
```

The join condition here is `BILL.CHARGE_CODE=CHARGES.CHARGE_CODE`, which relates the charge code in table `BILL` (where amounts are listed) to the charge code in table `CHARGES` (where the text description of the charge code is listed). The result is:

DESCRIPTION	AMOUNT
LODGING	380.00
CAR PARK	25.00
MISCELLANEOUS	12.50

Conceptually, the join first establishes a table containing all combinations of the rows in `CHARGES` with the rows in `BILL`, then selects those rows in which the two `CHARGE_CODE` values are equal (see Section 4.4 for a fuller description of the conceptual SELECT process). This does not necessarily represent the order in which the operations are actually performed; the order of evaluation of a complex SELECT statement is determined by the SQL optimizer, regardless of the order in which the component clauses are written.

Without the join condition, the result is a *cross product* of the columns in the tables in question, containing all possible combinations of the selected columns:

```
SELECT DESCRIPTION, AMOUNT
FROM   CHARGES, BILL
WHERE  RESERVATION = 1349;
```

DESCRIPTION	AMOUNT
LODGING	380.00
TELEPHONE	380.00
CAR PARK	380.00
RESTAURANT	380.00
MINIBAR	380.00
BAR	380.00
ROOM SERVICE	380.00
LAUNDRY	380.00
EXTRA BED	380.00
MISCELLANEOUS	380.00
LODGING	25.00
TELEPHONE	25.00
CAR PARK	25.00
RESTAURANT	25.00
MINIBAR	25.00
BAR	25.00
ROOM SERVICE	25.00
LAUNDRY	25.00
EXTRA BED	25.00
MISCELLANEOUS	25.00
LODGING	12.50
TELEPHONE	12.50
CAR PARK	12.50
RESTAURANT	12.50
MINIBAR	12.50
BAR	12.50
ROOM SERVICE	12.50
LAUNDRY	12.50
EXTRA BED	12.50
MISCELLANEOUS	12.50

It is easy to see that a carelessly formulated join query can produce a very large result table. Two tables of 100 rows each, for instance, give a cross product with 10,000 rows; three tables of 100 rows each give a cross product with 1,000,000 rows! The risk of generating large (erroneous) result tables is particularly high in ISQL, where queries are so easily written and submitted.

4.2.2 Simple joins

In simple joins, all tables used in the join are listed in the FROM clause of the SELECT statement. This is in distinction to nested joins, where the search condition for one SELECT is expressed in terms of another SELECT (see Section 4.2.3).

An example of a simple join is the query described above:

```
SELECT DESCRIPTION, AMOUNT
FROM   CHARGES, BILL
WHERE  BILL.CHARGE_CODE = CHARGES.CHARGE_CODE
AND    RESERVATION = 1349;
```

DESCRIPTION	AMOUNT
LODGING	380.00
CAR PARK	25.00
MISCELLANEOUS	12.50

The form SELECT * may be used in a join query, but since this selects all columns in the result set, at least one column is usually duplicated:

```
SELECT *
FROM   CHARGES, BILL
...;
```

<i>(From CHARGES)</i>			<i>(From BILL)</i>		
CHARGE_CODE	DESCRIPTION	RESERVATION	ON_DATE	CHARGE_CODE	AMOUNT
...

Columns in the join query that are uniquely identified by the column name may be specified by name alone. Columns that have the same name in the joined tables must be qualified by their respective table names.

There is an alternative formulation of the query above:

```
SELECT DESCRIPTION, AMOUNT
FROM   CHARGES JOIN BILL
ON     CHARGES.CHARGE_CODE = BILL.CHARGE_CODE
AND    RESERVATION = 1349;
```

All predicates that can be used in a where clause, except sub-selects, can be used in an on-clause. The join clause can be used as a statement on it's own:

```
CHARGES JOIN BILL ON CHARGES.CHARGE_CODE = BILL.CHARGE_CODE;
```

or

```
CHARGES NATURAL JOIN BILL;
```

A natural join, joins the table on the condition of equality between any columns with the same name, in the two tables. In the first example, all columns from the two tables are present in the result. In the second example the join columns will only occur once. Thus, in the first case, the CHARGE_CODE column appears twice in the result, while there is only one occurrence of this column in the second result.

It is possible to nest join-clauses

Select the status of all rooms at hotel LAPONIA.

```
SELECT ROOMNO, ROOMSTATUS.STATUS
FROM ROOMSTATUS NATURAL JOIN ROOMS
JOIN HOTEL
ON HOTEL.HOTELCODE = ROOMS.HOTELCODE
AND HOTEL.NAME = 'LAPONIA';
```

ROOMNO	STATUS
LAP112	KEY OUT
LAP200	MAINT
LAP201	MAINT
LAP205	KEY OUT

A join query can join any number of tables, using complex search conditions to select the relevant information from each table:

Select the total bill for guest Sten Johansen and list it in both Swedish and Danish crowns (SEK and DKK respectively).

```
SELECT GUEST, SUM(AMOUNT)/RATE AS TOTAL_BILL, CURRENCY
FROM BOOK_GUEST, BILL, EXCHANGE_RATE
WHERE GUEST = 'STEN JOHANSEN'
AND (CURRENCY = 'DKK'
OR CURRENCY = 'SEK')
AND BOOK_GUEST.RESERVATION = BILL.RESERVATION
GROUP BY GUEST, CURRENCY, RATE;
```

NAME	TOTAL_BILL	CURRENCY
STEN JOHANSEN	628.52484	DKK
STEN JOHANSEN	579.50000	SEK

In formulating a search condition for a join query, it can help to write out the columns that would appear in a complete cross-product of the tables. The search condition is then formulated as though the query was a simple SELECT from the cross-product table.

4.2.3 Outer joins

The joins in the previous chapter were all *inner* joins. In an inner join between two tables, only rows that fulfil the join condition are present in the result. An outer join, on the contrary, contains non-matching rows as well based on one of the tables in the join.

```
SELECT DESCRIPTION, AMOUNT
FROM   CHARGES LEFT OUTER JOIN BILL
ON     CHARGES.CHARGE_CODE = BILL.CHARGE_CODE
AND    RESERVATION = 1349;
```

DESCRIPTION	AMOUNT
LODGING	380.00
TELEPHONE	-
CAR PARK	25.00
RESTAURANT	-
MINIBAR	-
ROOM SERVICE	-
LAUNDRY	-
EXTRA BED	-
MISCELLANEOUS	12.50

In this example, all rows from the table to the left in the join clause, i.e. CHARGES, are present in the result. Non-matching rows from the BILL table are filled with null values in the result. A right outer join will take all records from the table to the right in the join-clause.

Note the difference in result against this statement

```
SELECT DESCRIPTION, AMOUNT
FROM   CHARGES LEFT OUTER JOIN BILL
ON     CHARGES.CHARGE_CODE = BILL.CHARGE_CODE
WHERE  RESERVATION = 1349;
```

DESCRIPTION	AMOUNT
LODGING	380.00
CAR PARK	25.00
MISCELLANEOUS	12.50

and the previous one. The reason is that conditions in the where clause are applied to the result of the join-clause and not to the joined tables as is the case with the conditions in the on-clause.

As with inner joins, it is possible to nest join-clauses. These can be of different types, i.e. both inner and outer joins. The result of nested outer joins can be somewhat unexpected though, as it is the result of the first join-clause that is the left table in the next join, and not the right table in the first join-clause.

4.2.4 Nested selects

A form of SELECT, called a *subselect*, can be used in the search condition of a SELECT statement to form a nested query. The main SELECT statement is then referred to as the *outer select*. For example

Select the names of hotels which have rooms with a price under 350.

```
SELECT  NAME
FROM    HOTEL
WHERE   HOTELCODE IN (SELECT  HOTELCODE
                       FROM    ROOM_PRICES
                       WHERE   PRICE < 350 );
```

NAME
LAPONIA
ST.GEORGE

To see how this works, evaluate the subselect first:

```
SELECT  HOTELCODE
FROM    ROOM_PRICES
WHERE   PRICE < 350;
```

HOTELCODE
LAP
STG

Then use the result of the subselect in the search condition of the outer select:

```
SELECT  NAME
FROM    HOTEL
WHERE   HOTELCODE IN ('LAP', 'STG');
```

A subselect can be used in a search condition wherever the result of the subselect can provide the correct form of the data for the search condition. Thus a subselect used with '=' must give a single value as a result, a subselect used with IN, ALL or ANY must give a set of single values and a subselect used with EXISTS may give any result (see Section 4.2.8).

```
WHERE column = (subselect)
WHERE column IN (subselect)
WHERE column = ALL (subselect)
WHERE column = ANY (subselect)
WHERE EXISTS (subselect)
```

Subselects cannot include ORDER BY clauses. The UNION operator can be used to combine two or more subselects in more complex statements (see Section 4.2.9).

Many nested queries can equally well be written as simple joins. For example:

Select the names of hotels which have rooms with a price under 350.

```
SELECT NAME
FROM HOTEL
WHERE HOTELCODE IN (SELECT HOTELCODE
                     FROM ROOM_PRICES
                     WHERE PRICE < 350 );
```

or

```
SELECT NAME
FROM HOTEL, ROOM_PRICES
WHERE HOTEL.HOTELCODE = ROOM_PRICES.HOTELCODE
AND ROOM_PRICES.PRICE < 350;
```

Both these queries give exactly the same result and are equivalent in performance. In most cases, the choice of which form to use is a matter of personal preference. Choose the form which you can understand most easily; the clearest formulation is least likely to cause problems.

Queries may contain any number of subselects, for example:

List hotels which have rooms that are more expensive than any of the rooms at the Hotel Laponia.

```
SELECT NAME
FROM HOTEL
WHERE HOTELCODE IN
      (SELECT HOTELCODE
       FROM ROOM_PRICES
       WHERE PRICE >
            (SELECT MAX(PRICE)
             FROM ROOM_PRICES
             WHERE HOTELCODE =
                  (SELECT HOTELCODE
                   FROM HOTEL
                   WHERE NAME = 'LAPONIA')));
```

(Note the balanced parentheses for the nested levels).

It is particularly important at this level of complication to think carefully through the query to make sure that it is correctly formulated. Often, writing some of the levels as simple joins can simplify the structure. The example above may also be written

```
SELECT NAME
FROM HOTEL, ROOM_PRICES
WHERE HOTEL.HOTELCODE = ROOM_PRICES.HOTELCODE
AND PRICE > (SELECT MAX(PRICE)
             FROM ROOM_PRICES, HOTEL
             WHERE ROOM_PRICES.HOTELCODE = HOTEL.HOTELCODE
             AND NAME = 'LAPONIA' );
```

Another example illustrates that the simplest queries can be written as nested joins:

Select the names of the guests in the GUEST column of the BOOK_GUEST table.

```
SELECT GUEST
FROM BOOK_GUEST
WHERE GUEST IN (SELECT GUEST
                FROM BOOK_GUEST );
```

is equivalent to

```
SELECT GUEST
FROM BOOK_GUEST;
```

4.2.5 Ordering nested queries

The ORDER BY clause may only be used in outer SELECT statements and not in subselects.

The following example is correct:

```
SELECT NAME, ROOMTYPE, FROM_DATE, PRICE
FROM HOTEL, ROOM_PRICES
WHERE HOTEL.HOTELCODE IN
      (SELECT HOTELCODE
       FROM ROOM_PRICES
       WHERE ROOMTYPE IN ('SGLS', 'SGLB'))
ORDER BY NAME;
```

The following example is incorrect:

```
SELECT NAME, ROOMTYPE, FROM_DATE, PRICE
FROM HOTEL, ROOM_PRICES
WHERE HOTEL.HOTELCODE IN (SELECT HOTELCODE
                          FROM ROOM_PRICES
                          WHERE ROOMTYPE IN ('SGLS', 'SGLB')
                          ORDER BY HOTELCODE);
```

4.2.6 Correlation names

A correlation name is a temporary name given to a table to represent a logical copy of the table within a query. Correlation names can be up to a maximum of 18 characters long.

There are three uses for correlation names:

- simplifying complex queries
- joining a table to itself
- outer references in subselects

4.2.6.1 Simplifying complex queries

Using short correlation names into complicated queries can make the query easier to write and understand, particularly when qualified table names are used:

```
SELECT BOOKADM.BOOK_GUEST.GUEST,
       BOOKADM.HOTEL.NAME, SUM(AMOUNT)
FROM   BOOKADM.BOOK_GUEST, BOOKADM.HOTEL, BOOKADM.BILL
WHERE  BOOKADM.BILL.RESERVATION = BOOKADM.BOOK_GUEST.RESERVATION
AND    BOOKADM.HOTEL.HOTELCODE = 'WINS'
GROUP BY BOOKADM.BOOK_GUEST.GUEST, BOOKADM.HOTEL.NAME;
```

may be rewritten

```
SELECT  G.GUEST, H.NAME, SUM(AMOUNT)
FROM    BOOKADM.BOOK_GUEST AS G,
        BOOKADM.HOTEL      AS H,
        BOOKADM.BILL       AS B
WHERE   B.RESERVATION = G.RESERVATION
AND     H.HOTELCODE = 'WINS'
GROUP BY G.GUEST, H.NAME;
```

The keyword AS in the FROM clause may be omitted, but is recommended for clarity. Do not confuse AS in the FROM clause (defining a correlation name) with AS in the select list (see Section 4.1.2, defining a label).

Correlation names are local to the query in which they are defined.

When a correlation name is introduced for a table name, all subsequent references to the table in the same query must use the correlation name. The following expression is not accepted:

```
...
FROM   BOOKADM.BOOK_GUEST AS G,
...
WHERE  H.RESERVATION = BOOKADM.BOOK_GUEST.RESERVATION
```

4.2.6.2 Joining a table with itself

Joining a table with itself allows you to compare information in a table with other information in the same table. This can be done with a correlation name.

Select all pairs of hotels located in the same city.

```
SELECT HOTEL.NAME, HOTEL.CITY
FROM   HOTEL, HOTEL AS COPY
WHERE  HOTEL.CITY = COPY.CITY
AND    HOTEL.NAME <> COPY.NAME;
```

NAME	CITY
LAPONIA	STOCKHOLM
ST. GEORGE	STOCKHOLM

Here, the table HOTEL is joined to a logical copy of itself called COPY. The first search condition finds pairs of hotels in the same city, and the second eliminates 'pairs' with the same name. (Without the second condition in the search condition, all hotel names would be selected!)

Without correlation names, this kind of query cannot be formulated. The following query would select all the hotel names from the table:

```
SELECT HOTEL.NAME, HOTEL.CITY
FROM HOTEL
WHERE HOTEL.CITY = HOTEL.CITY;
```

4.2.6.3 Outer references in subselects

In some constructions using subselects, a subselect at a lower level may refer to a value in a table addressed at a higher level. This kind of reference is called an *outer reference*.

```
SELECT NAME
FROM HOTEL
WHERE EXISTS (SELECT *
              FROM BOOK_GUEST
              WHERE HOTELCODE = HOTEL.HOTELCODE);
```

This kind of query processes the subselect for every row in the outer select, and the outer reference represents the value in the current outer select row. In descriptive terms, the query says "For each row in HOTEL, select the NAME column if there are rows in BOOK_GUEST containing the current HOTELCODE value".

If the qualifying name in an outer reference is not unambiguous in the context of the subselect, a correlation name must be defined in the outer select. A correlation name *may* always be used for clarity, as in the following example:

```
SELECT NAME
FROM HOTEL AS H
WHERE EXISTS (SELECT *
              FROM BOOK_GUEST
              WHERE HOTELCODE = H.HOTELCODE);
```

4.2.7 Retrieving with EXISTS, NOT EXISTS

EXISTS is used to check for the existence of some row or rows which satisfy a specified condition. EXISTS differs from the other operators in that it does not compare specific values; instead, it tests whether a set of values is empty or not. The set of values is specified as a subselect.

The subselect following the EXISTS clause most often uses of "SELECT *" as opposed to "SELECT column-list" since EXISTS only searches to see if the set of values addressed by the subselect is empty or not - a specified column is seldom relevant in the subquery.

EXISTS (subselect) is true if the result set of the subselect is not empty

NOT EXISTS (subselect) is true if the result set of the subselect is empty

SELECT statements with EXISTS almost always include an outer reference linking the subselect to the outer select.

Find the names of hotels for which guests exist in the BOOK_GUEST table.

```
SELECT  NAME
FROM    HOTEL AS H
WHERE   EXISTS (SELECT  *
                FROM    BOOK_GUEST
                WHERE   HOTELCODE = H.HOTELCODE);
```

Without the outer reference, the select becomes a conditional "all-or-nothing" statement: perform the outer select if the subselect result is not empty, otherwise select nothing.

List all reservation numbers if anybody has checked out without paying.

```
SELECT  DISTINCT RESERVATION
FROM    BILL
WHERE   EXISTS (SELECT  *
                FROM    BOOK_GUEST
                WHERE   CHECKOUT IS NOT NULL
                AND     PAYMENT IS NULL);
```

The next example illustrates NOT EXISTS:

Which hotels do not have double rooms with showers?

```
SELECT  NAME, HOTELCODE
FROM    HOTEL AS H
WHERE   NOT EXISTS (SELECT  *
                    FROM    ROOMS
                    WHERE   HOTELCODE = H.HOTELCODE
                    AND     ROOMTYPE = 'DBLS');
```

NAME	HOTELCODE
WINSTON	WINS

Negated EXISTS clauses must be handled with care. There are two semantic 'opposites' to EXISTS, with very different meanings:

```
WHERE EXISTS (SELECT *
              FROM   GUESTS
              WHERE  GUEST = 'CODD')
```

is true if at least one guest is called CODD.

```
WHERE NOT EXISTS (SELECT *
                 FROM   GUESTS
                 WHERE  GUEST = 'CODD')
```

is true if no guest is called CODD.

But

```
WHERE EXISTS (SELECT *
             FROM   GUESTS
             WHERE  GUEST <> 'CODD')
```

is true if at least one guest is not called CODD.

```
WHERE NOT EXISTS (SELECT *
                FROM   GUESTS
                WHERE  GUEST <> 'CODD')
```

is true if no guest is not called CODD, that is, if every guest is called CODD.

The double negative in the last example above is an SQL implementation of the universal quantifier FORALL (see "A Guide to DB2" by C. J. Date for more information on EXISTS and FORALL).

4.2.8 Retrieval with ALL, ANY, SOME

Subselects that return a set of values may be used in the quantified predicates ALL, ANY or SOME. Thus

```
WHERE PRICE < ALL (subselect)
```

selects rows where the price is less than all of the values in the subselect

```
WHERE PRICE < ANY (subselect)
```

selects rows where the price is less than at least one of the values in the subselect's results

Unfortunately, English language usage of "all" and "any" can cause confusion with SQL usage. In particular, the SQL condition WHERE PRICE < ALL(values) is often verbalized as "where the price is less than any value in the set". The SQL usage of ANY is a little clearer if it is replaced by the alternative keyword SOME.

Select room types and hotel codes for rooms that do not cost the same as any of the rooms at Hotel Skyline.

```
SELECT ROOMTYPE, HOTELCODE
FROM   ROOM_PRICES
WHERE  PRICE <> ALL (SELECT PRICE
                   FROM   ROOM_PRICES
                   WHERE  HOTELCODE = 'SKY');
```

If the result of the subselect is an empty set, ALL evaluates to true, while ANY or SOME evaluates to false.

Queries using ALL, ANY or SOME can always be written with EXISTS or NOT EXISTS. It is generally preferable to use EXISTS to avoid the linguistic confusion between ALL and ANY. For example:

Select the room type, price and hotel code for all rooms that cost the same as any room at the hotel Skyline.

```
SELECT ROOMTYPE, PRICE, HOTELCODE
FROM ROOM_PRICES
WHERE PRICE = ANY (SELECT PRICE
                   FROM ROOM_PRICES
                   WHERE HOTELCODE = 'SKY');
```

is equivalent to

```
SELECT ROOMTYPE, PRICE, HOTELCODE
FROM ROOM_PRICES RP
WHERE EXISTS (SELECT *
             FROM ROOM_PRICES
             WHERE HOTELCODE = 'SKY'
             AND RP.PRICE = PRICE);
```

4.2.9 Union queries

The UNION operator combines the results of two or more subselect clauses. UNION first merges the result tables specified by the separate subselects and then eliminates duplicate rows from the merged set.

Select the codes for hotels which are in Stockholm or have single rooms with showers.

```
SELECT HOTELCODE
FROM HOTEL
WHERE CITY = 'STOCKHOLM'

UNION

SELECT DISTINCT HOTELCODE
FROM ROOMS
WHERE ROOMTYPE = 'SGLS' ;
```

The result is obtained by merging the results of the two subselects and eliminating duplicates:

```
SELECT HOTELCODE          SELECT DISTINCT HOTELCODE
FROM HOTEL                FROM ROOMS
WHERE CITY = 'STOCKHOLM' ; WHERE ROOMTYPE = 'SGLS' ;
```

HOTELCODE
LAP
STG

HOTELCODE
LAP
SKY
STG
WIND

giving the result table

HOTELCODE
LAP
SKY
STG
WIND

To retain duplicates in the result table, use UNION ALL in place of UNION (see the MIMER/SQL Reference Manual for details).

Columns which are merged by UNION must have compatible data types (numerical with numerical, character with character). Subselects addressing more than one result column are merged column by column in the order of selection. The number of columns addressed in each subselect must be the same.

The column names in the result of a UNION are taken from the names in the first subselect. Use labels in the first subselect to assign different column names to the result table:

Merge the codes and names of hotels in Stockholm with the hotel codes and room type for rooms which are more expensive than any room at the St. George hotel.

```
SELECT HOTELCODE AS CODE, NAME AS NAME_OR_TYPE
FROM HOTEL
WHERE CITY = 'STOCKHOLM'

UNION

SELECT HOTELCODE, ROOMTYPE
FROM ROOM_PRICES
WHERE PRICE > (SELECT MAX(PRICE)
                FROM ROOM_PRICES
                WHERE HOTELCODE = 'STG');
```

CODE	NAME_OR_TYPE
LAP	LAPONIA
STG	ST. GEORGE
WIND	DBLB

Subselects merged by UNION may not include an ORDER BY clause. However, the result of the UNION query may be ordered with an ORDER BY clause placed after the last query in the UNION.

UNION may not be used within a nested subselect. However, the results of nested queries may be joined by UNION.

Unions can also be used to combine information from the same table:

Find the highest and lowest prices for rooms at the Hotel Skyline.

```
SELECT 'HIGHEST' AS PRICE, MAX(PRICE) AS AMOUNT
FROM ROOM_PRICES
WHERE HOTELCODE = 'SKY'

UNION

SELECT 'LOWEST', MIN(PRICE)
FROM ROOM_PRICES
WHERE HOTELCODE = 'SKY'
ORDER BY AMOUNT;
```

PRICE	AMOUNT
LOWEST	350
HIGHEST	580

Unions can also be used to perform *outer joins*, joining information in a table or tables with information not listed in those tables (i.e. information that is null). For example:

List the room types available for each hotel code. Include a row for hotel codes which do not have a given room type with a shower.

```
SELECT DISTINCT H.HOTELCODE, ROOMTYPE
FROM   ROOMS R, HOTEL H
WHERE  R.HOTELCODE = H.HOTELCODE

UNION

SELECT DISTINCT H.HOTELCODE, 'NO ' || ROOMTYPE AS ROOMTYPE
FROM   HOTEL H, ROOMS
WHERE  H.HOTELCODE = ROOMS.HOTELCODE
AND    NOT EXISTS (SELECT *
                   FROM   ROOMS R
                   WHERE  R.HOTELCODE = H.HOTELCODE
                   AND    ROOMTYPE LIKE '%S')

ORDER BY HOTELCODE;
```

HOTELCODE	ROOMTYPE
LAP	DBLB
LAP	DBLS
LAP	SGLB
LAP	SGLB
SKY	DBLB
SKY	DBLS
SKY	SGLB
SKY	SGLS
STG	DBLB
STG	DBLS
STG	SGLB
STG	SGLS
WIND	DBLB
WIND	DBLS
WIND	SGLB
WIND	SGLS
WINS	DBLB
WINS	NO DBLS
WINS	SGLB
WINS	NO SGLS

Note: UNION statements including DISTINCT treat NULL values as duplicates.

In UNION queries, the keyword NULL can be included in the column list of one or both of the queries, so that columns not represented in all of the queries in the statement are retained in the result set.

4.3 Handling NULL values

NULL values require special handling in SQL queries. NULL represents an unknown value, and strictly speaking NULL is never equal to NULL. (NULL values are however treated as equal for the purposes of GROUP BY, DISTINCT and UNION).

4.3.1 Searching for NULL

The search condition

```
WHERE column = NULL
```

will not retrieve any rows since NULL is not equal to anything. The condition for selecting NULL values is

```
WHERE column IS NULL
```

The negated form (WHERE column IS NOT NULL) selects values which are not NULL (i.e. values which are known).

Find the names of the persons who made the reservations for those customers who have not yet checked in to the Hotel Skyline.

'Not checked in' is represented by NULL in the CHECKIN column.

```
SELECT RESERVED_BY
FROM BOOK_GUEST
WHERE CHECKIN IS NULL
AND HOTELCODE = (SELECT HOTELCODE
                  FROM HOTEL
                  WHERE NAME = 'SKYLINE');
```

RESERVED_BY
OMAR CHAFIR
AGNETA ERIKSSON
SVEN LINDHOLM
HENRIK PIHL
URBAN FRANSSON

Find the names of the guests who have checked in to the Hotel Laponia.

```
SELECT GUEST
FROM BOOK_GUEST
WHERE CHECKIN IS NOT NULL
AND HOTELCODE = (SELECT HOTELCODE
                  FROM HOTEL
                  WHERE NAME = 'LAPONIA');
```

GUEST
CHRISTOPHER DATE
STEN JOHANSEN
STEFAN HANSEN
GUNNAR ALVE
NILS KRISTOFERSEN
LARS HOLMER
KNUT KULLMER
JUDITH SMITH
ADOLF SCHMIDT
LAILA ZETTERBERG
MATS HANSSON

4.3.2 Null values in ALL, ANY, IN and EXISTS queries

Null values should be treated cautiously, particularly in ALL, ANY, IN and EXISTS queries.

The result of a comparison involving NULL is unknown, which is generally treated as false. This can lead to unexpected results. For example, neither of the following conditions are true:

```
<null>      IN (... ,null,...)
<null> NOT IN (... ,null,...)
```

The first result is almost intuitive: since NULL is not equal to NULL, NULL is not a member of a set containing NULL. But if NULL is not a member of a set containing NULL, the second result is intuitively true. In fact, neither result is true or false: both are unknown. If NULL values are involved on either side of the comparison, IN and NOT IN are not complementary. Similar arguments apply to queries containing ALL or ANY:

Where are hotels with rooms that are more expensive than those at the hotel Skyline (hotel code SKY)?

```
SELECT NAME, CITY
FROM   HOTEL AS H, ROOM_PRICES AS RP
WHERE  H.HOTELCODE = RP.HOTELCODE
AND    PRICE > ALL (SELECT PRICE
                   FROM   ROOM_PRICES
                   WHERE  HOTELCODE = 'SKY');
```

This query works as long as there are no NULL values in the PRICE column. But introduce a new room type at Skyline with an unknown price, and the query results in an empty set. Moreover, the reverse query (hotels that are cheaper than all rooms at Skyline) also results in an empty set. (A justification for this is that as long as one price at Skyline is unknown, it is impossible to say whether rooms at other hotels are more or less expensive than those at Skyline).

It is always possible to rephrase a query using ALL, ANY or IN in terms of one using EXISTS (with an outer reference between the selection and the EXISTS condition). This is to be recommended if the NULL indicator is to be permitted in the comparison sets, since NULL handling is then written out explicitly in the query. Thus, the query above can also be written as follows:

```
SELECT NAME, CITY
FROM   HOTEL AS H, ROOM_PRICES AS RP
WHERE  H.HOTELCODE = RP.HOTELCODE
AND    NOT EXISTS (SELECT *
                  FROM   ROOM_PRICES
                  WHERE  HOTELCODE = 'SKY'
                  AND    ( PRICE <= RP.PRICE
                        OR PRICE IS NULL
                        OR RP.PRICE IS NULL ));
```

This formulation may be read as "Find hotels where no room at Skyline is cheaper than or the same price as any room in the hotel in question, as long as no prices are unknown". The explicit PRICE IS NULL clause tests that if either of the components of the comparison is NULL, then the subselect is not empty, NOT EXISTS is false, and no row is returned.

In general, a query of the form (\$ stands for any comparison operator):

```
SELECT column-list
FROM table1
WHERE column1 $ ALL (SELECT column2
                     FROM table2
                     WHERE condition)
```

is equivalent to

```
SELECT column-list
FROM table1
WHERE NOT EXISTS (SELECT *
                  FROM table2
                  WHERE condition
                  AND ( NOT table1.column1 $ table2.column2
                      OR table1.column1 IS NULL
                      OR table2.column2 IS NULL ));
```

A similar example is:

Where are hotels with rooms that have unknown prices or that are more expensive than rooms with known prices at hotel Skyline?

```
SELECT NAME, CITY
FROM HOTEL H, ROOM_PRICES RP
WHERE H.HOTELCODE = RP.HOTELCODE
AND NOT EXISTS (SELECT *
                FROM ROOM_PRICES
                WHERE HOTELCODE = 'SKY'
                AND PRICE <= RP.PRICE);
```

This query does not exclude the occurrence of the NULL indicator from the comparisons. If there is an unknown price, then the hotel concerned will be included in the result set - even if the unknown price is at Skyline itself. (Skyline might have a room that is more expensive than all rooms with known prices at Skyline).

Formulated with ALL, this query would be:

```
SELECT NAME, CITY
FROM HOTEL H, ROOM_PRICES RP
WHERE H.HOTELCODE = RP.HOTELCODE
AND PRICE > ALL (SELECT PRICE
                 FROM ROOM_PRICES
                 WHERE HOTELCODE = 'SKY'
                 AND PRICE IS NOT NULL);
```

It is clear from the examples above that distinctions between queries involving NULL comparisons are subtle and are easily overlooked. It is essential that the aim of a query is stringently defined before the query is formulated in SQL, and that the possible effects of NULL values in the search condition are considered. There are many real-life examples where the presence of NULL has resulted in unforeseen and sometimes misleading data retrievals. It is advisable to define all columns in the database tables as NOT NULL except those where unknown values have a specific meaning (such as the CHECKIN and CHECKOUT columns in the BOOK_GUEST table). In this way the risks of confusion with NULL handling are minimized.

4.4 Conceptual description of the selection process

This section presents a conceptual step-by-step analysis of the evaluation of a SELECT statement. It is intended as an aid in formulating complex SELECT statements, and can also help you in understanding details of the statement syntax.

Note: The description here is purely conceptual. It does not represent the actual sequence of events performed by the database manager. In particular, the computer resource requirements implied by the intermediate result set defined in a FROM clause do not necessarily reflect actual requirements.

The query used in the analysis is

List the total amount due for reservations above number 1347. Sort the result by guest name.

```
SELECT  G.RESERVATION, G.GUEST, SUM(B.AMOUNT)
FROM    GUEST_VIEW G, BILL_VIEW B
WHERE   G.RESERVATION = B.RESERVATION
GROUP BY G.RESERVATION, G.GUEST
HAVING  G.RESERVATION > 1347
ORDER BY GUEST;
```

RESERVATION	GUEST	
1349	STEFAN HANSEN	380.00
1348	STEN JOHANSEN	524.50

To simplify the presentation, the query is based on the views GUEST_VIEW and BILL_VIEW with the following contents:

GUEST_VIEW	
RESERVATION	GUEST
1347	CHRISTOPHER DATE
1348	STEN JOHANSEN
1349	STEFAN HANSEN

BILL_VIEW	
RESERVATION	AMOUNT
1347	380.00
1347	12.70
1347	18.00
1348	400.00
1348	12.00
1348	112.50
1349	380.00

1. Subselects at the lowest nesting level are evaluated first

The first step in evaluating a select is to resolve subselects from the lowest level up, and conceptually replace the subselect with the result set. (The example here does not use a nested select). When all subselects are resolved, a (possible complicated) single-level SELECT statement remains.

2. The FROM clause defines an intermediate result set

Tables addressed in the FROM clause are combined to form an intermediate result set which is the full cross product of the tables. The cross product is a table with one column for each column in each of the table, and one row for every combination of rows from the different tables. The columns in the result set are identified by the qualified column names from the table from which they are derived.

FROM GUEST_VIEW G, BILL_VIEW B

G.RESERVATION	G.GUEST	B.RESERVATION	B.AMOUNT
1347	CHRISTOPHER DATE	1347	380.00
1347	CHRISTOPHER DATE	1347	12.70
1347	CHRISTOPHER DATE	1347	18.00
1347	CHRISTOPHER DATE	1348	400.00
1347	CHRISTOPHER DATE	1348	12.00
1347	CHRISTOPHER DATE	1348	112.50
1347	CHRISTOPHER DATE	1349	380.00
1348	STEN JOHANSEN	1347	380.00
1348	STEN JOHANSEN	1347	12.70
1348	STEN JOHANSEN	1347	18.00
1348	STEN JOHANSEN	1348	400.00
1348	STEN JOHANSEN	1348	12.00
1348	STEN JOHANSEN	1348	112.50
1348	STEN JOHANSEN	1349	380.00
1349	STEFAN HANSEN	1347	380.00
1349	STEFAN HANSEN	1347	12.70
1349	STEFAN HANSEN	1347	18.00
1349	STEFAN HANSEN	1348	400.00
1349	STEFAN HANSEN	1348	12.00
1349	STEFAN HANSEN	1348	112.50
1349	STEFAN HANSEN	1349	380.00

3. The WHERE clause selects rows from the intermediate set

WHERE G.RESERVATION = B.RESERVATION

G.RESERVATION	G.GUEST	B.RESERVATION	B.AMOUNT
1347	CHRISTOPHER DATE	1347	380.00
1347	CHRISTOPHER DATE	1347	12.70
1347	CHRISTOPHER DATE	1347	18.00
1348	STEN JOHANSEN	1348	400.00
1348	STEN JOHANSEN	1348	12.00
1348	STEN JOHANSEN	1348	112.50
1349	STEFAN HANSEN	1349	380.00

4. The GROUP BY clause groups the remaining result set

GROUP BY G.RESERVATION, G.GUEST

G.RESERVATION	G.GUEST	B.RESERVATION	B.AMOUNT
1347	CHRISTOPHER DATE	1347	380.00
1347	CHRISTOPHER DATE	1347	12.70
1347	CHRISTOPHER DATE	1347	18.00
1348	STEN JOHANSEN	1348	400.00
1348	STEN JOHANSEN	1348	12.00
1348	STEN JOHANSEN	1348	112.50
1349	STEFAN HANSEN	1349	380.00

5. The HAVING clause selects groups

```
HAVING G.RESERVATION > 1347
```

G.RESERVATION	G.GUEST	B.RESERVATION	B.AMOUNT
1348	STEN JOHANSEN	1348	400.00
1348	STEN JOHANSEN	1348	12.00
1348	STEN JOHANSEN	1348	112.50
1349	STEFAN HANSEN	1349	380.00

6. The SELECT list selects columns, evaluates any expressions in the SELECT list, and reduces groups to single rows if set functions are used

```
SELECT G.RESERVATION,  
       G.GUEST,  
       SUM(B.AMOUNT)
```

G.RESERVATION	G.GUEST	
1348	STEN JOHANSEN	524.50
1349	STEFAN HANSEN	380.00

7. The results of subselects joined by UNION are merged

This example does not include a UNION.

8. The final result is sorted according to the ORDER BY clause

```
ORDER BY GUEST;
```

RESERVATION	GUEST	
1349	STEFAN HANSEN	380.00
1348	STEN JOHANSEN	524.50

5 CHANGING TABLE CONTENTS

The previous chapter described how to retrieve data from tables with `SELECT`. This chapter deals with changing the data in tables with the statements:

- `INSERT` for inserting new rows into tables
- `UPDATE` for updating rows
- `DELETE` for deleting rows from tables

You must have the appropriate access privileges on the relevant table(s) in order to use `INSERT`, `UPDATE` or `DELETE`. See Chapter 8 Defining privileges. In addition, the table itself must be updatable. All base tables are updatable, but some views are not (see Section 5.4).

5.1 Inserting data

The `INSERT` statement is used to insert new rows into existing tables.

Values to be inserted may be specified explicitly (as constants or expressions) or in the form of a subselect (see below). The data to be inserted must be of a type compatible with the corresponding column definition. If the length of the inserted data differs from that of the column definition, the data is handled as follows:

character strings	<p>If the inserted data is longer than the column definition, an error is reported and the <code>INSERT</code> operation fails (trailing spaces are truncated without error).</p> <p>If the inserted data is shorter than the column definition, the inserted data is padded to the right with spaces to the required length.</p>
decimal values	<p>Decimal values which are longer than the column definition are truncated (not rounded) from the right to meet the column definition. Thus 12.3456 is inserted into <code>DEC(6,3)</code> as 12.345.</p> <p>Decimal values which are shorter than the column definition are padded to the right of the decimal point with zeros. Thus 12.3 is inserted into <code>DEC(6,3)</code> as 12.300.</p>

integer values	If the inserted data has more digits than the column definition or is outside the range of the definition, an error is reported and the INSERT operation fails.
floating point values	Floating point values are converted to decimal by truncating the fractional part of the value as required by the scale of the decimal target. An error occurs if the scale of the target cannot accommodate the integral part of the value.
datetime values	Date values are converted to timestamp by setting the hour, minute and second fields to zero. Time values are converted to timestamp by taking values for the year, month and day fields from CURRENT_DATE. Timestamp values are converted to date or time by discarding the field values that do not appear in the target.
interval values	Single field interval values are converted to exact numeric by truncating decimal digits or by padding decimal digits with zeros. If any loss of leading precision occurs, or if overflow occurs, an error is raised.

5.1.1 Inserting explicit values

The explicit INSERT statement has the general form

```
INSERT INTO table (column-list)
VALUES (value-list);
```

Values in the value list are inserted into columns in the column list in the order specified. The order of columns in the column list need not be the same as the order in the table definition. Any columns in the table definition which are not included in the column list are assigned NULL values (or the column default value if one is defined).

An explicit INSERT statement can only insert a single row.

Insert the values 'SUTB' and 'SUITE WITH BATH' into the ROOMTYPE and DESCRIPTION columns respectively into the ROOMTYPES table.

```
INSERT INTO ROOMTYPES (ROOMTYPE,DESCRIPTION)
VALUES ('SUTB', 'SUITE WITH BATH');
```

inserts the row

ROOMTYPE	DESCRIPTION
SUTB	SUITE WITH BATH

If you insert explicit values into all of the columns in a table, the column list can be omitted from the INSERT statement. The values specified are then inserted into the table in the order that the columns are defined in the table. Thus the example above could also be written:

```
INSERT INTO ROOMTYPES
VALUES ('SUTB', 'SUITE WITH BATH');
```

You can also insert the result of an expression into a table:

```
INSERT INTO ROOM_PRICES
VALUES ( 'LAP', 'SUTB', DATE '1996-06-01', 500 + 40 );
```

HOTELCODE	ROOMTYPE	FROM_DATE	PRICE
LAP	SUTB	1996-06-01	540

5.1.2 Inserting with a subselect

Values to be inserted can also be specified in the form of a subselect, i.e. fetched from another table in the database.

```
INSERT INTO ROOMSTATUS
      SELECT DISTINCT ROOMNO, 'KEY OUT'
      FROM          BOOK_GUEST
      WHERE         CHECKIN IS NOT NULL
      AND          CHECKOUT IS NULL;
```

The same table cannot be listed in the subselect's FROM clause that is listed in the INSERT INTO clause - data cannot be selected from a table for insertion into the same table.

Inserting the result of a subselect can insert several rows into a table. If any of the rows are rejected (e.g. because of a duplicate primary or unique key), the whole INSERT statement fails and no rows are inserted.

5.1.3 Inserting NULL values

The keyword NULL may be used to insert the NULL value into a column (provided that the column is not defined as NOT NULL):

```
INSERT INTO EXCHANGE_RATE ( CURRENCY, RATE )
VALUES ( 'XYZ', NULL );
```

The NULL indicator is implicitly inserted into columns when no value is given for that column and the column definition does not include a default value. Thus, the following INSERT statement will give the same results as the example above:

```
INSERT INTO EXCHANGE_RATE ( CURRENCY )
VALUES ( 'XYZ' );
```

5.2 Updating tables

Data in existing table rows can be changed with the UPDATE statement. This statement has the general form:

```
UPDATE table
SET column = value
[WHERE search-condition];
```

The search condition specifies which rows in the table are to be updated. If no search condition is specified, all rows will be updated.

Update the exchange rate for US dollars to 7.25.

```
UPDATE EXCHANGE_RATE
SET    RATE = 7.25
WHERE CURRENCY = 'USD';
```

Add 1 to the count of free rooms for Hotel Laponia on October 15, 1996.

```
UPDATE FREEROOMS
SET    FREECOUNT = FREECOUNT + 1
WHERE ROOMTYPE = 'SGLS'
AND    ON_DATE   = DATE '1996-10-15'
AND    HOTELCODE = (SELECT HOTELCODE
                    FROM    HOTEL
                    WHERE   NAME = 'LAPONIA');
```

When a subselect is used in the search condition, the table being updated may not be used in the subselect.

Primary key columns cannot be updated.

5.3 Deleting rows from tables

The DELETE statement removes rows from a table, and has the general form:

```
DELETE FROM table
[WHERE search-condition];
```

The search condition specifies which rows in the table are to be deleted. If no search condition is specified, all rows will be deleted (the table is emptied but not dropped).

Delete all hotels in STOCKHOLM from the HOTEL table.

```
DELETE FROM HOTEL
WHERE CITY = 'STOCKHOLM';
```

Delete all rows from the HOTEL table.

```
DELETE FROM HOTEL;
```

Delete information for guests with the last name SVENSON from the BILL table.

```
DELETE FROM BILL
WHERE RESERVATION IN (SELECT RESERVATION
                      FROM BOOK_GUEST
                      WHERE TRIM(GUEST) LIKE '% SVENSON');
```

When a subselect is used in the search condition, the table from which rows are deleted may not be used in the subselect.

5.4 Updatable views

INSERT, UPDATE and DELETE statements may be used on views: the operation is then performed on the base table on which the view is defined. However, certain views may not be updated (for example a view containing DISTINCT values, where a single row in the view may represent several rows in the base table). A view is not updatable if any of the following conditions are true:

- the keyword DISTINCT is used in the view definition
- the select list contains components other than column specifications, or contains more than one specification of the same column
- the FROM clause specifies more than one table reference or refers to a non-updatable view
- the WHERE clause contains a subselect whose FROM clause refers to the same table or view specified in the outer select
- the GROUP BY clause is used in the view definition
- the HAVING clause is used in the view definition

6 MANAGING TRANSACTIONS

6.1 Transactions

Transactions are defined as "atomic operations", meaning groups of operations which are to be executed together (i.e. either all statements in the transaction are executed, or none are executed).

A transaction is divided into two phases. During *build-up*, the user requests the database operations which will form the transaction. Once the build-up is complete, the transaction is *committed*, i.e. the user signals that the changes requested during transaction build-up are to be made permanent.

During build-up, changes requested in the contents of the database are not visible to other users of the system, and the database remains fully accessible to all users. Changes requested during the transaction build-up only become visible when the transaction is successfully committed.

A major function of transaction handling in MIMER multi-user systems is concurrency control. This means protecting the database from corruption which might arise when two users attempt to change the same information at the same time.

See the MIMER/SQL Programmers Manual for a more detailed discussion of transaction handling and database security.

6.2 Logging

Transaction control also provides the basis for protection of the database against hardware failure.

All changes made to the database within transactions may be recorded in the system logging databank, LOGDB. This contains a record of all transactions executed since the latest back-up copy of the databank was made. In the event of a system crash, the contents of LOGDB together with the latest back-up copy of the database may be used to restore the system.

It is important to note that only operations included in transactions can be recorded in LOGDB.

Transaction and logging are determined on a databank basis by options set when the databank is defined. The options are:

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. Sets of operations (DELETE, UPDATE and INSERT on several rows) which are interrupted will not be rolled back.

All important databanks should be defined with LOG option, so that valuable data is not lost by system failure.

6.3 Handling transactions

Transaction control statements in MIMER/SQL are:

```
COMMIT;
ROLLBACK;
SET TRANSACTION CHANGES INVISIBLE;
SET TRANSACTION CHANGES VISIBLE;
SET TRANSACTION START EXPLICIT;
SET TRANSACTION START IMPLICIT;
START TRANSACTION;
```

The default settings for ISQL and BSQL are:

```
SET TRANSACTION START EXPLICIT;
SET TRANSACTION CHANGES VISIBLE;
```

The following SQL statements may not be used inside a transaction:

```
ALTER          COMMENT          CREATE          DISCONNECT
DROP           ENTER            GRANT          LEAVE
REVOKE        SET DATABASE  SET DATABANK  SET SHADOW
UPDATE STATISTICS
```

In addition, the following ISQL and BSQL commands (see Chapter 10) may not be used inside a transaction:

```
EXIT          LOAD          UNLOAD
```

6.3.1 Transaction handling in BSQL and ISQL

The transaction handling behavior in BSQL and ISQL has been designed so that, by default, no attention needs to be paid to transaction handling at all. Transactions are automatically started whenever they are required and they are automatically committed after each statement.

The START and COMMIT (or ROLLBACK) statements may be used together to group a number of statements into a single transaction, rather than having each occur within its own transaction (the default).

The default **transaction start** setting for BSQL and ISQL is EXPLICIT.

It should not be necessary to alter the transaction start setting in BSQL and ISQL, the START and COMMIT (or ROLLBACK) statements can be used to explicitly control transactions when this is required.

Note that the transaction handling behavior described here is a specific implementation for BSQL and ISQL and it does not precisely match the general effects of the transaction handling options.

For a description of general transaction handling as it normally applies to embedded MIMER/SQL - refer to Section 6.2 of the MIMER/SQL Programmer's Manual.

6.3.2 Consistency within a transaction

Since transaction changes are not executed on the database until the transaction is committed, inconsistencies can in principle arise during transaction build-up. For example:

```
START ;
UPDATE HOTEL
SET     NAME = 'LAPLAND'
WHERE  HOTECD = 'LAP' ;
SELECT CITY FROM HOTEL
WHERE  NAME = 'LAPLAND' ;
COMMIT ;
```

Here, the hotel name is not changed to LAPLAND until the transaction is actually committed, so that the SELECT statement will not find any rows.

To avoid inconsistencies of this kind, by default, a transaction checks write operations within the transaction to see whether data accessed at one point in the transaction build-up is modified by an update request earlier in the same transaction ('read-through-write-set'). This can be relatively time-consuming for large transactions, and facilities are provided for turning this function off.

To turn read-through-write-set off, issue the statement

```
SET TRANSACTION CHANGES INVISIBLE ;
```

To restore the read-through-write-set function, issue the statement

```
SET TRANSACTION CHANGES VISIBLE ;
```

The default setting is VISIBLE. CHANGES should be set to INVISIBLE only for transactions where it is clear that no operations are affected by previous updates in the same transaction. Once the setting is changed, the new setting applies until another SET TRANSACTION CHANGES statement is issued.

7 DEFINING THE DATABASE

SQL includes statements for creating and modifying the database structure:

- create idents, databanks, domains, tables, views, indexes and synonyms
- saving documentary comments on objects
- altering the definition of idents, databanks and tables
- dropping objects from the database

All information describing the database structure is stored in the data dictionary.

Before the database is defined, it is extremely important to design a database model. Well-functioning and efficient databases cannot be created without a model as the foundation. Without careful design, much of the flexibility and efficiency inherent in a relational database structure may be lost.

This chapter describes the SQL statements for creating and managing the database structure. Examples are based on the database listed in Appendix C. In addition, ISQL provides commands for listing and describing database objects in an interactive environment (see Chapter 10).

7.1 Creating idents

Idents are authorized users of the system or groups of users defined for easier ident management (see Chapter 2).

Ident names and passwords can have a maximum length of 18 characters. The case of letters is insignificant for ident names but passwords must be entered exactly as they are defined.

The statement for creating idents has the general form

```
CREATE IDENT username
AS ident-type
[IDENTIFIED BY 'password'];
```

Passwords are required for user and program idents but are not used for group idents. Passwords are optional for OS_USER idents: an OS_USER with a password may connect to MIMER in the same way as any other user ident.

Create a user ident BOOKADM with the password "Bookadm".

```
CREATE IDENT BOOKADM
AS USER
IDENTIFIED BY 'Bookadm';
```

Create a program ident AUDIT with the password "economy".

```
CREATE IDENT AUDIT
AS PROGRAM
IDENTIFIED BY 'economy';
```

Create a group ident for the group ECONOMY_DEPT.

```
CREATE IDENT ECONOMY_DEPT
AS GROUP;
```

7.2 Creating databanks

The statement for creating a databank has the general form

```
CREATE DATABANK databank-name
      OF initial-size PAGES
      IN 'filename'
      WITH transaction-control OPTION;
```

- The CREATE DATABANK clause defines the databank name (which may be up to 18 characters long).
- The OF clause allocates a specified number of MIMER pages.
- The IN clause defines the file where the databank is to be stored (the form of this specification is machine-specific).
- The WITH clause defines the transaction handling and logging option (see Section 6.2).

Create the ROOMSDB databank with TRANS option, allocate 10 MIMER pages for it, and store it in the specified file.

```
CREATE DATABANK ROOMSDB
      OF 10 PAGES
      IN 'ROOMSDB'
      WITH TRANS OPTION;
```

At this point, the databank is empty.

7.3 Creating domains

Domains are used as data types in column definitions when creating tables

- to assist in keeping the database consistent
- to limit the data (particular values or data type) accepted in the columns
- to define default values for columns

The statement for creating domains has the general form:

```
CREATE DOMAIN domain-name
      AS data-type
      [DEFAULT default-value]
      [CHECK (check-condition)];
```

Domain names can be up to 18 characters long.

It is a good practice for upholding the integrity of the database to define domains for as many columns as possible.

7.3.1 Domains with default values

The default clause defines values that are inserted into the column when an explicit value is not specified in an INSERT statement.

Define the default value '-ND-' ('not defined') for the domain ROOMTYPE.

```
CREATE DOMAIN ROOMTYPE
      AS CHAR(4)
      DEFAULT '-ND-';
```

Define the current user's name as the default value for the domain NAME.

```
CREATE DOMAIN NAME AS CHAR(18)
      DEFAULT USER;
```

Domains defining default values can also include check clauses. You could define the ROOMTYPE domain as:

```
CREATE DOMAIN ROOMTYPE
      AS CHAR(4)
      DEFAULT '-ND-'
      CHECK (VALUE IS NOT NULL);
```

This means that the NULL indicator will not be accepted into columns belonging to this domain.

If the default value is outside the check limits, an explicit value must always be inserted into the column.

7.3.2 Domains with check clauses

The check clause in domain definitions relates the domain to a constant value and specifies the limitations of the data to be placed into columns belonging to the domain.

The domain CALENDAR uses a check clause to limit the range of accepted values:

```
CREATE DOMAIN CALENDAR
AS DATE
CHECK (VALUE BETWEEN DATE '1996-01-01' AND
DATE '2099-12-31');
```

- The CREATE DOMAIN clause defines the domain name.
- The AS clause defines the domain data type.
- The CHECK clause defines the domain limits.

7.4 Creating tables

After the physical file space has been allocated on a disk for the databank, (CREATE DATABANK), you can create the tables. The basic CREATE TABLE statement defines the columns in the table, the primary key, any unique or foreign keys and which databank the table is to be stored in. Table names and column names may be up to 18 characters long.

As a convention, we have defined primary key column(s) as the first column(s) in the table definition. However, this is not a necessity; primary key columns may be defined anywhere in the column list. Primary keys are always NOT NULL, so there is no need to explicitly state that in the table definition (they are included in the examples here for clarity).

Create the table EXCHANGE_RATE with two columns. Name the first column CURRENCY, make it of the CHARACTER data type with a maximum of three characters. Name the second column RATE and make it of the data type DECIMAL with a total of six digits, three of which can be decimal values. Declare the CURRENCY column as the primary key and place this table in the BOOKDB databank.

```
CREATE TABLE EXCHANGE_RATE (CURRENCY CHAR(3) NOT NULL,
                             RATE      DECIMAL(6,3),
                             PRIMARY KEY (CURRENCY))
IN BOOKDB;
```

This CREATE TABLE clause defines the name of the table followed by a column list, which includes the names of the columns in the table, their data type, if they should allow the NULL indicator and the primary key declaration. Each item in the column-list is separated from the next by a comma, and the entire list is enclosed in parentheses.

A table definition may only include one primary key clause. The primary key can be made up of more than one column.

The IN clause states which databank the table is to be stored in. This clause may be omitted; if the IN clause is not specified, MIMER will select the "best" databank in which to place the table (see the MIMER/SQL Reference Manual for details of how the best databank is chosen).

The empty table now exists in the databank. Data is inserted into the table with the INSERT statement (Chapter 5).

The preceding example shows the simplest form of column list. The following variants may also be used:

- columns belonging to domains
- columns not belonging to the primary key defined as NOT NULL
- unique columns (in addition to the primary key)
- default values (overriding any domain default for the column)
- foreign key references
- check conditions

The BOOK_GUEST table in the example database is defined with many of the options that can be used in creating tables. See the MIMER/SQL Reference Manual for a full description of table creation facilities.

```
CREATE TABLE BOOK_GUEST (RESERVATION    INTEGER(5) ,
                          BOOKING_DATE  DATE          NOT NULL ,
                          HOTELCODE     HOTELCODE    NOT NULL ,
                          ROOMTYPE      ROOMTYPE     NOT NULL ,
                          RESERVED_BY   PERSONNAME   NOT NULL ,
                          TELEPHONE     CHAR(15) ,
                          RESERVED_FOR  PERSONNAME ,
                          ARRIVE        DATE          NOT NULL ,
                          DEPART        DATE          NOT NULL ,
                          GUEST         PERSONNAME ,
                          ADDRESS       CHAR(30) ,
                          CHECKIN       DATE ,
                          CHECKOUT      DATE ,
                          ROOMNO        ROOMNO ,
                          PAYMENT       CHAR(10) ,
                          PRIMARY KEY (RESERVATION) ,
                          FOREIGN KEY (HOTELCODE) REFERENCES HOTEL ,
                          FOREIGN KEY (ROOMTYPE) REFERENCES ROOMTYPES ,
                          FOREIGN KEY (ROOMNO)   REFERENCES ROOMS ,
                          CHECK (ARRIVE < DEPART AND CHECKIN <= CHECKOUT) )
IN ROOMSDB;
```

The ordering of column specifications, key clauses and check conditions is not fixed. If desired, the key and check clauses can be written in association with the respective column specifications:

```
CREATE TABLE BOOK_GUEST
  (RESERVATION INTEGER(5) ,
   PRIMARY KEY (RESERVATION) ,
   BOOKING_DATE DATE          NOT NULL ,
   HOTELCODE   HOTELCODE    NOT NULL ,
   FOREIGN KEY (HOTELCODE) REFERENCES HOTEL ,
   ROOMTYPE   ROOMTYPE     NOT NULL ,
   FOREIGN KEY (ROOMTYPE) REFERENCES ROOMTYPES ,
   ...
```

7.4.1 Column definitions

Domains are used for many columns in the example database to help in maintaining database integrity. By using the same domain for columns in different tables, the column data types are guaranteed to be the same.

Columns should in general be defined as NOT NULL unless there is a specific reason for using the NULL value in the column (e.g. CHECKIN and CHECKOUT in the table BOOK_GUEST, where NULL indicates that the reservation has not checked in or out). The presence of NULL values can often complicate the formulation of queries (see Section 4.3). Take particular care to exclude NULL from numerical columns which are to be used for mathematical operations.

7.4.2 The primary key

The primary key can consist of more than one column in the table. The choice of columns to use as the primary key is determined by the relational model for the database, which is outside the scope of this manual.

7.4.3 Foreign keys - referential integrity

Use foreign keys to maintain integrity between the contents of related tables.

Note that the tables referenced in a foreign key clause of a table definition must exist prior to the definition of the foreign key (unless the key is in the reference table itself, to ensure referential integrity within a table).

The number of columns listed as FOREIGN KEY must be the same as the number of columns in the primary key of the REFERENCES table, unless unique key columns are referenced explicitly in a column list (see the CREATE TABLE syntax in the MIMER/SQL Reference Manual for details). The nth FOREIGN KEY column corresponds to the nth column in the primary key of the REFERENCES table, and the data types and lengths of corresponding columns must be identical. Columns may not be used more than once in the same FOREIGN KEY clause.

If the NULL indicator is permitted in a foreign key, then either at least one of the columns in the foreign key is NULL or the values in the foreign key columns must be present in the corresponding primary key columns of the reference table.

A table definition may contain as many FOREIGN KEY references as required. The same column in the table may be used in separate FOREIGN KEY clauses referring to different REFERENCES tables.

Note! Neither foreign keys nor tables referenced in foreign keys may be included in databanks that are defined with the NULL option.

The BOOK_GUEST table has three foreign key references:

```
CREATE TABLE BOOK_GUEST (RESERVATION    INTEGER(5),
                          BOOKING_DATE   DATE        NOT NULL,
                          HOTELCODE      HOTELCODE  NOT NULL,
                          ROOMTYPE       ROOMTYPE    NOT NULL,
                          .
                          .
                          ROOMNO         ROOMNO,
                          .
                          FOREIGN KEY (HOTELCODE) REFERENCES HOTEL,
                          FOREIGN KEY (ROOMTYPE) REFERENCES ROOMTYPES,
                          FOREIGN KEY (ROOMNO)  REFERENCES ROOMS
                          )
                          .
```

These maintain referential integrity as follows:

- FOREIGN KEY (HOTELCODE) REFERENCES HOTEL
Data that is not present in the HOTELCODE column of the HOTEL table will not be accepted in the HOTELCODE column in the BOOK_GUEST table.
- FOREIGN KEY (ROOMTYPE) REFERENCES ROOMTYPES
Data that is not present in the ROOMTYPE column of the ROOMTYPES table will not be accepted in the ROOMTYPE column in the BOOK_GUEST table.
- FOREIGN KEY (ROOMNO) REFERENCES ROOMS
Data that is not present in the ROOMNO column of the ROOMS table will not be accepted in the ROOMNO column in the BOOK_GUEST table.

7.4.4 Check conditions

Check conditions in table definitions are used to make sure that data in a column in the table fits certain conditions. This section gives three different examples of check conditions.

Note that the first two examples below are not used in the example database.

Limit the city for hotels to Stockholm or Gothenburg.

```
CREATE TABLE HOTEL (HOTELCODE HOTELCODE,
                     NAME        CHAR(15) NOT NULL,
                     CITY        CHAR(15) NOT NULL,
                     OVERBOOK    BOOK_RATE NOT NULL,
                     PRIMARY KEY (HOTELCODE),
                     CHECK (CITY IN ('STOCKHOLM', 'GOTHENBURG')))
IN ROOMSDB;
```

Prevent blank entries in the HOTELCODE column.

```
CREATE TABLE HOTEL (HOTELCODE HOTELCODE,
                     NAME        CHAR(15) NOT NULL,
                     CITY        CHAR(15) NOT NULL,
                     OVERBOOK    BOOK_RATE NOT NULL,
                     PRIMARY KEY (HOTELCODE),
                     CHECK (HOTELCODE <> ' '))
IN ROOMSDB;
```


The example database does not contain any view definitions. Two examples are given below:

Create a restriction view of the BOOK_GUEST table called RECEPTION containing limited information for the hotel reception (reservation number, customer name, check-in date and room number).

```
CREATE VIEW RECEPTION (RESERVATION, NAME, DATE, ROOM)
AS SELECT RESERVATION, GUEST, CHECKIN, ROOMNO
FROM BOOK_GUEST;
```

RESERVATION	NAME	DATE	ROOM
1348	STEN JOHANSEN	1996-08-23	LAP205
1349	STEFAN HANSEN	1996-08-23	LAP206
1350	SALLY WEBERT	1996-08-06	SKY124
1351	ANNA ALBERTSON	1996-08-06	SKY125
1352	MARK FRANCIS	1996-08-14	WINS103
1353	ALFRED FIMPLEY	1996-09-03	SKY110
...
...

Create a join view listing the billing details for each reservation.

```
CREATE VIEW CHARGE_DESCRIPTION
AS SELECT RESERVATION, AMOUNT, DESCRIPTION
FROM BILL, CHARGES
WHERE BILL.CHARGE_CODE = CHARGES.CHARGE_CODE;
```

If the view definition does not include a list of column names, the columns in the view will be named after the columns listed in the SELECT clause.

RESERVATION	AMOUNT	DESCRIPTION
1348	400.00	LODGING
1348	12.00	TELEPHONE
1348	112.50	RESTAURANT
1348	55.00	BAR
1350	580.00	LODGING
1350	60.00	MINIBAR
1350	580.00	LODGING
1350	265.00	RESTAURANT
1350	175.00	BAR
1350	120.00	LAUNDRY
1350	580.00	LODGING
...
...

7.5.1 Check options

Check options can be used in updatable view definitions to limit the data that can be inserted into the view. If a check option is specified, data which does not fulfil the definition of the view cannot be inserted in to the view.

```
CREATE VIEW GUEST_VIEW
AS SELECT RESERVATION, HOTELCODE, GUEST, CHECKIN, ROOMNO
FROM BOOK_GUEST
WHERE HOTELCODE = 'STG' OR HOTELCODE = 'WINS'
WITH CHECK OPTION;
```

RESERVATION	HOTELCODE	GUEST	CHECKIN	ROOMNO
1355	STG	INGER SVENSON	1996-09-01	STG111
1363	WINS	PAULE LE FEVRE	1996-08-20	WINS117
1364	STG	LARS HOLLSTEN	1996-09-01	STG116
1367	WINS	EARNST JOHNSSON	1996-09-06	WINS109
1371	STG	MARY TENMAR	1996-08-29	STG010
1382	WINS	JULIO PEREZ	1996-09-29	WINS119
1383	STG	ROBERT LIND	1996-08-31	STG142
1384	WINS	SIGWARD PERSSON	1996-09-25	WINS120
1385	WINS	RUNE NYQVIST	1996-09-25	WINS121
1398	STG	LENNART RYDELL	1996-09-30	STG1421
1401	STG	JAN BLOM	1996-09-23	STG001
1408	STG	EINAR SUNDMAN	1996-09-20	STG117
1412	WINS	JOHAN TORP	1996-09-30	WINS119

The check option in the view definition (WITH CHECK OPTION) means that no new rows may be inserted into the view if the value for the HOTELCODE column is not STG or WINS.

Creating views based on other views

Views can be based on other views. When a view is created based upon another view or views, the original view's limitations are carried over to the new view.

```
CREATE VIEW NEW_VIEW
AS SELECT RESERVATION, HOTELCODE, GUEST
FROM GUEST_VIEW
WHERE RESERVATION > 1385;
```

7.6 Creating secondary indexes

Secondary indexes are maintained by the system and are invisible to the user. The index is automatically used during searching when it improves the efficiency of the search.

Any column(s) may be specified as a secondary index. Note however that columns in the primary key and columns addressed through a FOREIGN KEY reference from another table are automatically indexed, (in the order in which they are defined in the key), and creation of an index on these columns will not improve performance.

Secondary index tables are purely for MIMER/SQL's internal use - you create the index, and MIMER/SQL handles the rest. Index names can be made up of a maximum of 18 characters.

If, for instance, you want to know which room a certain person is staying in at a hotel, MIMER/SQL would have to search successively through the customer reference numbers and the names corresponding to each in order to find the information you want. If, however, you create a secondary index on guest names, MIMER/SQL would search for the name of that person directly in the secondary index, which would save time.

Create a secondary index called NAME on the GUEST column in the BOOK_GUEST table.

```
CREATE INDEX NAME
ON BOOK_GUEST (GUEST);
```

Primary key columns may also be included in a secondary index. If a table has the primary key 'A,B,C', the primary index would cover all three columns in the primary key. The following combinations of the columns in the primary key are automatically indexed: 'A', 'A,B', and 'A,B,C'. In addition, you could create secondary indexes on columns B, C, BC, AC, AD etc.

An index may also be defined as UNIQUE, which means that the index value may only occur once in the table. (For this purpose, NULL is treated as equal to NULL).

The sorting order for indexes may be defined as ascending or descending. However, this makes no difference to the efficiency of the index, since MIMER searches indexes forwards or backwards depending on the circumstances.

Secondary indexes can improve the efficiency of data retrieval; but introduce overhead for write operations (UPDATE, INSERT, DELETE). In general, you should create indexes only for columns that are frequently searched.

Indexes cannot be created directly on columns in views. However, since searching in a view is actually implemented as searching in the base table, an index on the base table will also be used in view operations.

7.7 Creating synonyms

Synonyms, or alternative names can be created for tables, views or other synonyms. You can create synonyms to personalize tables or just for your own convenience. Synonym names can be made up of a maximum of 18 characters.

Table names are "qualified" by the name of their creator. The qualified form of the table name is the creator name followed by the table name and the two are separated by a period. Thus the table ROOMS with the creator BOOKADM has the qualified name:

```
BOOKADM.ROOMS
```

The table's creator need only refer to it as:

```
ROOMS
```

Its qualifier is implicit since the creator is using the table. However, should another user wish to use this table, he must refer to it by its full qualified name since he is not its creator.

If a user named James wishes to refer to the ROOMS table belonging to the user BOOKADM as simply ROOMS, he can create a synonym:

```
CREATE SYNONYM ROOMS
FOR BOOKADM.ROOMS ;
```

Another user can then create his own synonym for James' ROOMS synonym, which has the the full name:

```
JAMES.ROOMS
```

Synonyms are particularly useful when several users refer to a common table, such as BOOKADM.ROOMS, BOOKADM.HOTEL, etc. With synonyms, several users can work in the same apparent environment without needing to refer to the tables by their qualified names.

7.8 Commenting objects

Comments may be stored against any of the following objects:

COLUMN	IDENT	SYNONYM
DATABANK	INDEX	TABLE
DOMAIN	SHADOW	VIEW

Store the comment "MIMER Hotels Databank" on the BOOKDB databank.

```
COMMENT ON DATABANK BOOKDB IS 'MIMER Hotels Databank';
```

Comments cannot be deleted - they can only be replaced by a new comment (a blank string may be provided as a comment if you want to suppress an existing comment).

Comments are for information only and do not affect data retrieval or manipulation in any way. Comments may be read with the DESCRIBE command (Chapter 10) or by retrieving the appropriate columns from the data dictionary tables.

7.9 Altering databanks, tables and idents

7.9.1 Altering a databank

Databanks can only be altered by their creator. There are three uses for the ALTER statement:

- to change the physical file location for a databank
- to change the transaction and logging options on the databank
- to increase the size allocated for the databank

Change which file the BOOKDB is stored in from its previous file to file "SQLDB:NEWDB.DBF" (the file specification is in VAX/VMS format).

```
ALTER DATABANK BOOKDB
      INTO 'SQLDB:NEWDB.DBF' ;
```

Note: This statement changes the file name stored for the databank in the data dictionary. It does not actually move the databank to the new location. To move a databank, copy or rename the file in the operating system, then use ALTER DATABANK ... INTO to change the file specification in the data dictionary.

Change the option on the BOOKDB databank from TRANS to LOG.

```
ALTER DATABANK BOOKDB
      TO LOG OPTION;
```

Increase the size of the BOOKDB database by 20 MIMER pages.

```
ALTER DATABANK BOOKDB
      ADD 20 PAGES;
```

Note: On platforms which support dynamic file expansion, the ALTER DATABANK ... ADD statement is not strictly necessary. However, increasing the file allocation by a relatively large figure can help to minimize file fragmentation and improve response times.

7.9.2 Altering tables

The ALTER TABLE statement changes the definition of the specified table and has no effect on the table's existing contents.

There are four uses for the ALTER TABLE statement:

- to add a new column to an existing table
- to drop a column from an existing table
- to change the default value for a column in an existing table
- to drop the default value for a column in an existing table

A new column created with the ALTER TABLE .. ADD statement is appended to end of the existing column list. The new column will include the default value defined for the column or, if no default value exists, the NULL indicator.

A column added to an existing table has the following limitations:

- it cannot be defined as a primary, unique or foreign key
- it cannot include a CHECK clause
- it cannot include a default value (unless column defined as a domain)
- it can only be NOT NULL if it belongs to a domain with a default value

Add a column called NOSMOKE with a data type of CHAR(1) to the BOOK_GUEST table.

```
ALTER TABLE BOOK_GUEST
ADD NOSMOKE CHAR(1);
```

This creates a column containing the NULL indicator for each row in the table.

When dropping a column from a table, the CASCADE and RESTRICT keywords can be used to specify the action that will be taken on objects that are dependent on the dropped column. If CASCADE is specified, depending objects are dropped. For instance if a dropped column is part of a secondary index, the index will also be dropped. If RESTRICT is specified and there are other objects affected, the statement will be aborted, with an error condition.

Drop the column TELEPHONE from the table BOOK_GUEST, subject to the condition that there are no other objects dependant on this column.

```
ALTER TABLE BOOK_GUEST DROP TELEPHONE RESTRICT;
```

Drop the column TELEPHONE from the table BOOK_GUEST, if dependant objects exist, these are dropped as well.

```
ALTER TABLE BOOK_GUEST DROP TELEPHONE CASCADE;
```

Change the default value for the column BOOKING_DATE, the new default value is current date

```
ALTER TABLE BOOK_GUEST ALTER BOOKING_DATE SET DEFAULT CURRENT_DATE;
```

Drop the default value for the column BOOKING_DATE,

```
ALTER TABLE BOOK_GUEST ALTER BOOKING_DATE DROP DEFAULT;
```

7.9.3 Altering idents

Only passwords can be altered with the ALTER IDENT statement - ident names cannot be altered. User and program idents can change their own password if they so wish. Passwords can also be changed by the creator of the ident.

Change the user SAMMY's password to "SamJo".

```
ALTER IDENT SAMMY
IDENTIFIED BY 'SamJo';
```

7.9.4 Objects which may not be altered

Domains, views and indexes cannot be altered. It is therefore important that you think through your domains and views thoroughly and carefully before you create them to make sure that they suit the needs of your database.

The next section will discuss dropping objects and the results of this on the database.

7.10 Dropping objects from the database

The DROP statement is used to drop the following objects from the database:

DATABANK	SHADOW
DOMAIN	SYNONYM
IDENT	TABLE
INDEX	VIEW

The CASCADE or RESTRICT keywords may be used to specify the action to be taken if other objects exist that are dependent on the object being dropped. If RESTRICT is specified, an error is returned if other objects are affected, and the drop operation is aborted. If CASCADE (the default) is specified, dependant objects are dropped as well.

Therefore use caution when using the DROP statement with CASCADE, as the operation may have a recursive effect on all objects relating to it. For example, when a table is dropped, all views, synonyms, and indexes based on that table are also dropped.

The DROP statement removes whole objects from the database. It cannot be used to remove columns from tables.

7.10.1 Dropping databanks and tables

Drop the HOTEL table.

```
DROP TABLE HOTEL;
```

Any views, synonyms and indexes based on HOTEL are also dropped.

Drop the BOOKDB databank.

```
DROP DATABANK BOOKDB;
```

All tables in the BOOKDB databank are also dropped and any views, synonyms and indexes based on those tables are also dropped.

On most platforms, the physical databank file is deleted when a databank is dropped.

7.10.2 Dropping domains

When a domain is dropped, existing columns assigned the domain retain all the properties of the domain. No new columns may however be assigned the domain.

Note that if you re-create a domain that has been dropped, the domain will be seen as being a completely new domain and it will not be tied to any columns that previously belonged to the old domain.

To change the restrictions on those columns that were defined with a domain that has been dropped, use the `LOAD` and `UNLOAD` utilities described in Chapter 10. The procedure to be followed is listed below:

1. Unload the data in the table into a sequential file with the `UNLOAD` utility.
2. Drop the table from the databank.
3. Re-create the table.
4. Load the data with the `LOAD` utility from the sequential file containing the old table into the new table.

7.10.3 Dropping idents

When an ident is dropped, everything that the ident has created (including idents and everything created by those idents) as well as all privileges granted by the ident are dropped. For this reason, physical users should never own objects except for synonyms and personal views.

8 DEFINING PRIVILEGES

Privileges and access rights control the operations which users are allowed to perform in the database. Well-structured privileges and access rights are essential for maintaining data security.

There are three types of privileges:

- System privileges, which give the right to create global objects within the database.
- Object privileges, which give rights over certain specified objects in the system.
- Access privileges, which give rights of access to the contents of a specified table.

System privileges are granted to the system administrator upon installation, and may be passed on to other users. Objects and access privileges are initially granted only to the creator of an object. The creator may however pass the privileges on to other users.

Privileges are granted to users with the GRANT statement and revoked from users with the REVOKE statement.

All privileges may be granted "with grant option", which means that the receiver of the privilege in turn has the right to grant that privilege to other users.

The creator of an object is automatically granted full privileges on that object with grant option. Thus the creator of a group is automatically a member of that group, the creator of a program may enter it, and the creator of a table has all access privileges with grant option, etc.

When privileges that were granted "with grant option" are revoked, the right to grant those privileges to other users is also revoked. Grant option cannot be revoked without revoking the privilege. Users may only grant privileges that they themselves possess to other users, that is, users cannot grant privileges to themselves. Likewise, privileges may only be revoked by the grantor - users cannot revoke privileges from themselves.

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 USER, which may be performed by either the user himself or by the creator of the user), DROP, and COMMENT.

8.1 Ident hierarchy

In the initial installation, one user ident, the system administrator with ident name SYSADM, is automatically created. The system administrator has DATABANK and IDENT privileges with GRANT OPTION, and SELECT access on all tables in the data dictionary, also with GRANT OPTION. The system administrator is ultimately responsible for the structure of the whole system.

Certain system utilities may only be run by the system administrator (see System Management Handbook). In other respects, however, the system administrator is an ordinary user ident in the system. There is no ident in MIMER with automatic right of access to all objects within the system. It is quite possible (and may be advisable especially in large systems) that the system administrator is prevented from accessing the actual contents of the database; the administrator's job is concerned with objects in the system, not with the data.

The initial installation of MIMER also includes a general group ident called PUBLIC. All idents in the system are automatically members of the group PUBLIC, which may thus be used for granting global privileges.

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

- Roles within the system should be assigned to program idents. These are not coupled to any physical individual or group of individuals, and thus have a lifetime independent of turnover of personnel. (The system administrator is just such a function, but is coupled to a user ident rather than a program ident for practical purposes).
- Physical users of the system are user or OS_USER idents. They may be dropped if the person concerned leaves the company. User idents should not be granted privileges directly, other than membership in groups. OS_USER idents are allowed access to the database on the authorization of a valid log-in to the operating system. For added maximum protection, do not use OS_USER idents.
- Group idents are used to represent logical users of the system. Privileges are granted to groups rather than to individual programs or users. The individual idents are granted membership in the group to which they belong, and thereby gain the correct access to the system.
- Physical user idents should not in general be allowed to create objects (i.e. granted DATABANK, IDENT and TABLE privileges). In this way, individual user idents may be dropped with no cascading effects except loss of views created by the user.
- GRANT OPTION should be used sparingly and the ident hierarchy kept shallow. This minimizes the risk of cascading revocation of privileges.

If these recommendations are followed, maintenance of the ident structure in the system is simplified. Access to the contents of the database is granted to relatively few group idents instead of many individual programs or users, and when a physical individual leaves the company, his user ident can be dropped with no cascading consequences.

8.2 Granting privileges

8.2.1 Granting system privileges

System privileges are granted to the system administrator at the time of installation of the system. System privileges refer to global information, that affects the database as a whole. The two system privileges are:

DATABANK the right to create databanks

IDENT the right to create idents

Give the ident BOOKADM the privilege to create new databanks.

```
GRANT DATABANK
      TO BOOKADM;
```

Give the idents AUDIT and ECONOMY_DEPT the privilege to create new idents with grant option.

```
GRANT IDENT
      TO AUDIT, ECONOMY_DEPT
      WITH GRANT OPTION;
```

8.2.2 Granting object privileges

Object privileges refer to access rights for idents in relation to objects (programs, groups, tables). The three object privileges are:

EXECUTE the right to enter (become) a specified program ident

MEMBER membership in a specified group ident

TABLE the right to create tables in a specified databank

Give ECONOMY_DEPT the privilege to enter the AUDIT program ident.

```
GRANT EXECUTE ON AUDIT
      TO ECONOMY_DEPT;
```

Make STEVE, MARIANNE and JAMES members of the ECONOMY_DEPT group with grant option.

```
GRANT MEMBER ON ECONOMY_DEPT
      TO STEVE, MARIANNE, JAMES
      WITH GRANT OPTION;
```

Give ECONOMY_DEPT the privilege to create new tables in the BOOKDB databank.

```
GRANT TABLE ON BOOKDB
      TO ECONOMY_DEPT;
```

8.2.3 Granting access privileges

Access privileges define what data the users are allowed to manipulate in tables. There are five access privileges:

SELECT	the right to read the table contents
INSERT	the right to add new rows to the table
DELETE	the right to remove rows from the table
UPDATE	the right to change the contents of existing rows in the table (this privilege may be limited to specified columns within the table)
REFERENCES	the right to use the primary or unique key of the table as a foreign key reference (this privilege may be limited to specified columns within the table)

The keyword ALL may be used as shorthand for all of privileges that the grantor holds with grant option (ALL may be followed by the optional keyword PRIVILEGES).

Give BOOKADM the privilege to read, insert, and delete rows from the BOOK_GUEST table and give the user the right to pass these privileges on to other users.

```
GRANT SELECT, INSERT, DELETE
      ON BOOK_GUEST
      TO BOOKADM
      WITH GRANT OPTION;
```

Give ECONOMY_DEPT and AUDIT all privileges that you hold on the table CHARGES but do not give them the right to pass these privileges on to other users.

```
GRANT ALL ON CHARGES
      TO ECONOMY_DEPT, AUDIT;
```

Give ECONOMY_DEPT the privilege to update all columns not belonging to the primary key in the BOOK_GUEST table.

```
GRANT UPDATE ON BOOK_GUEST
      TO ECONOMY_DEPT;
```

Give RECEPTION the privilege to update only the GUEST, ADDRESS, and ROOMNO columns in the BOOK_GUEST table.

```
GRANT UPDATE (GUEST, ADDRESS, ROOMNO)
      ON BOOK_GUEST
      TO RECEPTION;
```

Give ECONOMY_DEPT the right to use the ROOMS table as a foreign key.

```
GRANT REFERENCES
      ON BOOKADM.ROOMS
      TO ECONOMY_DEPT;
```

8.3 Revoking privileges

Privileges can only be revoked by the grantor. Care must be taken when revoking privileges, especially when those privileges were granted "with grant option". Revoking such privileges from an ident can have a cascading effect on all idents who have been granted privileges by that ident (see Section 8.3.4).

Privileges granted to a group cannot be revoked separately from individual members of the group. To revoke a group privilege from an individual, either revoke the privilege from the group or revoke the individual's membership in the group.

8.3.1 Revoking system privileges

Take away the privilege to create new databanks from the ident BOOKADM.

```
REVOKE DATABANK
FROM BOOKADM;
```

Take away the privilege to create new idents from the idents AUDIT and ECONOMY_DEPT.

```
REVOKE IDENT
FROM AUDIT, ECONOMY_DEPT;
```

Revoking system privileges does not affect objects already created under the authorization of the privilege.

8.3.2 Revoking object privileges

Take away the privilege to execute the AUDIT program from the ident ECONOMY_DEPT.

```
REVOKE EXECUTE ON AUDIT
FROM ECONOMY_DEPT;
```

Take away the idents' STEVE, MARIANNE and JAMES memberships in the group ECONOMY_DEPT.

```
REVOKE MEMBER ON ECONOMY_DEPT
FROM STEVE, MARIANNE, JAMES;
```

8.3.3 Revoking access privileges

Revoke the privileges to delete and insert rows and retrieve data to and from the BOOK_GUEST table from the ident MARIANNE.

```
REVOKE SELECT, DELETE, INSERT ON BOOK_GUEST
FROM MARIANNE;
```

When the REFERENCES privilege is taken away from an ident for a table, all foreign keys links referencing that table are removed.

Revoke the right to use columns in ROOMS as foreign keys from ECONOMY_DEPT.

```
REVOKE REFERENCES
      ON ROOMS
      FROM ECONOMY_DEPT;
```

The keyword ALL may be used as a shorthand for all the privileges that may be revoked in the current context.

8.3.4 Recursive effects of revoking privileges

Revoking a privilege from an ident may have cascading effects on other objects, depending on the way the database is organized. The keywords CASCADE and RESTRICT can be used in the REVOKE statements to control whether the recursive effects should be allowed or not. If RESTRICT is specified and any recursive effect are identified the whole operation is inhibited. If CASCADE (the default) is specified the following recursive effects may occur:

- If a privilege WITH GRANT OPTION is revoked from an ident, all instances of that privilege granted to other idents under the authorization of that GRANT OPTION are also revoked.
- 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 REFERENCE privilege on a table is revoked from an ident, any FOREIGN KEY constraints in tables created by that ident under the authorization of that REFERENCE privilege are removed.

The recursive effects of revoking privileges depend on the temporal order in which the privileges are granted and revoked. An ident grants privileges, creates views and so on under the authorization of the most recent valid instance he has received of the GRANT OPTION, SELECT or appropriate privilege. The data dictionary keeps a record of the authorization under which privileges are granted. The recursive effects apply only to privileges granted or objects created under the authorization of the particular instance being revoked. This is illustrated on the next page.

CASE 1

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

CASE 2

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

CASE 3

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

CASE 4

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

As a consequence of the cascading effects of revoking privileges, careful advance planning of the hierarchical structure of idents in a system can be essential to the viability of the system. An unplanned ident structure can easily become impossible to overview and control after a relatively short period of system use.

9 USING ISQL

ISQL provides a terminal environment for using interactive SQL facilities. The environment offers:

- a text editor where SQL statements are written: statements issued during a session are saved in the editor buffer and may be modified and/or re-executed.
- scrolling facilities for examining the results of SQL queries.
- a set of interactive SQL commands for examining the database structure and managing the ISQL environment.

This chapter describes how to start and use ISQL. Chapter 10 describes the ISQL commands.

9.1 Starting ISQL

The operating system command for starting ISQL is machine-dependent. See the machine-specific Users Guide for your system.

Once ISQL is started, a login screen is displayed:

```

MMMM      MMMMM  MMMMM  MMMMM      MMMMM  MMMMMMMMMMMM  MMMMMMMMM
MMMMMM    MMMMMM  MMMMM  MMMMMM    MMMMMM  MMMMMMMMMMMM  MMMMMMMMM
  MMMMMM  MMMMMM    MMM  MMMMMM  MMMMMM    MMM  MMM  MMM  MMM
MMMMMMMMMMMMMMMM  MMM  MMMMMMMMMMMMMMM  MMMMM      MMMMMMM
MMM  MMMMM  MMM  MMM  MMM  MMMMM  MMM  MMM  MMMM  MMM  MMM  MMM
MMMM  MMM  MMMM  MMMMM  MMMM  MMM  MMMM  MMMMM  MMMMM  MMMM  MMMM
MMMMM  M  MMMM  MMMMM  MMMM  M  MMMM  MMMMMMMMMMMM  MMMM  MMMM

```

(C) Copyright SYSDECO MIMER AB 1987-1996. All rights reserved.

MIMER/ISQL

Version x.x.x

Username: _____
 Password:

Enter the username and password for a MIMER user ident. Press ENTER or RETURN after each entry.

The case of letters in the username is not significant, but the password must be entered exactly as defined. The password is not echoed to screen as it is entered.

If your current username in the operating system is defined as an OS_USER ident in MIMER, simply press ENTER or RETURN (without entering any username) at the username prompt to connect.

Note that ENTER is marked SEND on some keyboards.

9.2 The ISQL editor

The ISQL editor screen is divided into an editor window and a command line:

```

----- M I M E R / I S Q L -----
                                     E d i t o r   w i n d o w
                                     (PF2=Help) Command ==> Enter commands here

```

Enter SQL statements in the editor window. Enter ISQL commands (Chapter 10) on the command line at the bottom of the screen. To move from the editor to the command line, press ENTER or use the arrow keys. To move from the command line to the editor window, use the arrow keys.

System messages are shown in the upper right-hand corner of the screen. SQL error messages are shown in the editor window. For further information on messages in ISQL, see Chapter 13.

The default programmable function (PF) key for Help is machine-specific. You may change the help key assignment with the PFK command (Chapter 10).

Default values for the programmable function keys are machine dependent. By entering the command PFK on the command line, you can see what the values are for your machine. To change these values, simply replace the command specified for the particular PF key by writing a new command over the current one. PF keys may be defined as any of the ISQL commands listed in Chapter 10.

For more information on the MIMER editor, see Appendix B.

9.2.1 Screen window setup

The screen parameters may be set by the command WDW. The default values are shown below:

```

----- MIMER/ISQL Window Setup Definitions -----

                Window Pos. & Size
Row.....: 3_   (first row on screen)
Column..: 1_   (first col on screen)
Rows....: 21   (no. of rows      )
Columns.: 80   (no. of columns   )

                Prefix Area
Length..: 0    ( 0 or in <3,6>  )
Position: R    ( Left ! Right )
Type....: N    ( * or N<umeric> )

                Current Line
Row ....: 5_   (relative to first )
Attrib..: B    (Bright ! Reverse )

                Scale Line
Row ....: -1   (relative to first )

                Standard Menu Type
Type....: 1_   ( 1 or 2 )

(PF2=Help) - Change settings as required, then press ENTER

```

In ISQL, the maximum number of columns (characters) in the window is 80. If you read a file that includes statements with a record length greater than 80, the record will be truncated.

9.2.2 Viewing result sets

Result sets are the results of SELECT statements and DESCRIBE and LIST commands. A number on the bottom line of the screen shows how many rows of the result set have been displayed. If there are more lines below the screen, the text "more" is shown. Otherwise, the text "bottom" is shown.

Scroll down to display more lines by writing the DOWN command followed by the number of rows you wish to move down (example, DOWN 15). If the number is omitted, a value of 1 is used. Alternatively, use the function key defined as DOWN (use the PFK command to list function key assignments).

Scroll up by writing the UP command followed by the number of rows you wish to move up. If the number is omitted, a value of 1 is used. Alternatively, use the function key defined as UP.

The result set may sometimes be wider than the screen. The symbol => or <= on the bottom line of the screen indicates that there are more characters to the right or left respectively.

Scroll to the right or left with the RIGHT or LEFT command respectively followed by the number of characters to scroll. If the number is omitted, the display will be moved to the beginning of the next column (or 75 characters if the column is wider than the screen). Alternatively, use the function key defined as RIGHT or LEFT.

9.2.3 Menu displays in ISQL

The following commands have menu displays:

```
DESCRIBE
HELP
LIST
```

There are two menu styles available. Use the PROFILE command (Chapter 10) to change or display the current menu style.

- In menu style 1, choose an option by moving to the appropriate keyword with the arrow keys. The chosen option is highlighted. Press ENTER to accept the option.
- In menu style 2, choose an option by entering 'X' in the field directly to the left of the option. Press ENTER to accept the option.

9.3 Entering SQL statements

SQL statements are entered in the editor window using MIMER editor functions.

SQL statements may be written across several lines on the screen. The end of a statement is defined by a semicolon (;). The beginning of a statement is defined by the end of the previous statement or the beginning of the editor buffer. Any characters between a double hyphen (--) and the end of the line are treated as comments.

It is advisable to break complex SQL statements into clauses, with each clause on a separate line. The clauses may be indented if desired. This has several advantages:

- it is easier to see the statement structure
- it is often easier to detect errors
- individual clauses can easily be commented out

In the example below, the statement to the left would be typed as explained in the comments to the right.

```
SELECT *          -- type SELECT * and press <RETURN>
FROM   HOTEL;    -- type FROM HOTEL; and press <EXECUTE>
```

To execute a statement, move the cursor to any position within the statement and press the EXECUTE key. See the EXECUTE command in Chapter 10 for more details.

9.4 Leaving ISQL

To leave ISQL and return to your computer's operating system, enter the command EXIT or QUIT on the command line at the bottom of the screen or press the function key that has been defined as this command (see PFK command in Chapter 10 for the definition of function keys). You can only leave the system while in the editor - if the display shows a result set or a help screen, EXIT will return you to the editor screen.

EXIT prompts for confirmation before leaving ISQL:

```
EXIT <N> :
```

The default answer is N. Enter Y and press ENTER to leave ISQL.

10 ISQL COMMANDS

In addition to SQL statements, ISQL provides a number of commands for managing the database and its environment.

ISQL commands are entered on the command line at the bottom of the ISQL editor screen, and executed by pressing the SEND/ENTER key (Appendix B for more information on the editor environment). ISQL commands cannot be entered in the ISQL editor buffer.

The table below summarizes the ISQL commands. The remainder of this chapter describes the command functions in alphabetical order.

Command	Function
BACKWARD	Moves back to the previous screen display
BOTTOM	Moves to the bottom of a buffer
CANCEL	Cancels an operation
COMMAND	Executes computer-dependent functions
CONTINUE	Re-tries the execution of a statement from a sequential file
DEFAULT	Sets the default values for the start and end positions for loading data to a table and unloading from a table
DESCRIBE	Describes a specified object
DOWN	Scrolls downward
EXECUTE	Executes the statement on the current line in the editor window
EXIT	Leaves the current environment
FORWARD	Moves forward one screen display
HELP	Provides help text on a specified subject
LEFT	Moves to the left
LIST	Lists information on a specified object in the database
LOAD	Loads tables from a sequential file
NEXT	Moves to the next menu in LIST and DESCRIBE functions
PERFORM	Sends a file to the SQL compiler for execution
PFK	Examines and changes the function key definitions

PREVIOUS	Moves to the previous menu in LIST and DESCRIBE functions
PRINT	Prints statement results into the current printer file
PROFILE	Changes the printer file settings, menu type, generation of SELECT statement with LIST, list current ident and version number
QUIT	Leaves the current environment
READ	Reads a file into the editor buffer
REMOVE	Removes compiler error messages from the editor window
RIGHT	Moves to the right
SET	Specifies printer file characteristics and other miscellaneous settings
SKIP	Skips over an erroneous statement in a sequential file
TOGGLE	Switches between the current environment and the result of the latest LIST, DESCRIBE or SELECT statement
TOP	Moves to the top of a buffer
UNLOAD	Unloads tables into a sequential file
UP	Scrolls upwards
WDW	Displays and changes screen settings
WRITE	Writes the editor buffer contents to a sequential file

Note: CONNECT, DISCONNECT, ENTER and LEAVE may also be entered on the command line. However this is a deprecated feature.

ISQL commands are not case-sensitive.

Note on syntax descriptions

In the syntax descriptions, items in square brackets ([]) are optional. Items separated by a vertical bar (|) are alternatives. For example

```
READ [COMMAND|ALL] [INPUT FROM] 'filename';
```

allows the following forms

```
READ COMMAND INPUT FROM 'filename';
```

```
READ ALL INPUT FROM 'filename';
```

```
READ INPUT FROM 'filename';
```

```
READ 'filename';
```

BACKWARD

Moves back one screen display in the results of DESCRIBE or LIST functions, in the results of a SELECT statement or in the editor environment.

Syntax

BACKWARD [value]

Description

This command scrolls backwards in the text buffer in the DESCRIBE and LIST display, in the results of a SELECT statement or in the editor environment. The parameter specifies the number of screens to scroll (default=1). See also FORWARD.

BOTTOM

Moves to the bottom of the text buffer.

Syntax

BOTTOM

Description

This command moves to the bottom of the text buffer in the DESCRIBE and LIST display, in the results of a SELECT statement or in the editor environment. See also TOP.

CANCEL

Interrupts an operation.

Syntax

CANCEL

Description

This command stops the execution of the current function and returns the user to the previous environment.

If the CANCEL command is given while the user is in the PROFILE form, any changes that the user has made in the menu will be ignored.

The CANCEL command can also be used to stop an active PERFORM. The current PERFORM is active until it is completed or cancelled. Thus, should an error occur during the execution of a sequential file, you must first CANCEL the current PERFORM before you can run a new PERFORM with an edited version of the file.

COMMAND

Executes an operating system command without leaving ISQL.

Syntax

COMMAND [operating-system-command]

Description

This command can be used to perform operating system commands without leaving ISQL. See the MIMER User's Guide for restrictions on which operating system commands can be performed on your machine.

As an example, you could write COMMAND EDIT... in a VAX/VMS environment to suspend an ISQL session and then edit a file. When you are finished editing, you can leave the edit function and continue the ISQL session.

Example

```
(PF2=Help) Command ==>  COMMAND EDIT SOMEFILE.DAT
                        (suspends ISQL and runs the system editor)

[Editing on SOMEFILE.DAT]
      .
      .
      .
      *Exit
      (Exit editor, return to ISQL)
```

CONTINUE

Re-tries an execution of a statement in a sequential file that wasn't executed for some reason in a PERFORM.

Syntax

CONTINUE

Description

When a statement read from a PERFORM file is not executed due to an error, the statement will be placed in the editor buffer. The statement may be edited and then re-executed by typing the CONTINUE command on the command line. If the re-execution is successful, the execution will continue with the next statement in the file.

DEFAULT

Sets the default column names and start and end positions for LOAD and UNLOAD.

Syntax

DEFAULT

Description

The DEFAULT command sets the default column names and start/end positions for the LOAD and UNLOAD functions. The command can only be used in the LOAD and UNLOAD forms after the table name has been entered (see LOAD, UNLOAD).

DESCRIBE

Provides a description of a selected object in the database.

Syntax

DESCRIBE [object-type [object-name]]

Description

When the DESCRIBE command is issued, the following menu is shown:

----- M I M E R / I S Q L D E S C R I B E -----			
DATABANK	DOMAIN	IDENT	INDEX
SYNONYM	TABLE	VIEW	
Choose the type of item you wish to DESCRIBE by using the arrow keys and press ENTER			
(PF2=Help) Command ==>			

Use the arrow keys to move to the item you wish to describe. For menu type 1 (see Appendix B), press ENTER when the item you want is highlighted. For menu type 2, enter 'X' in the field directly to the left of the item and press ENTER. See the PROFILE command for more information on menu types.

When you have chosen the object to describe, a new menu lists the different options available for the object to be described. The options for each object are listed in the table below. Only one option may be chosen for one operation.

Giving an object type and/or object name on the command line skips one or both of the menus above.

When you have chosen the option, enter the object name and press ENTER. The description is displayed on the screen, and may be printed with the PRINT command.

DESCRIBE	OPTION	RESULT
DATABANK	DATABANK	Lists the following information on the specified databank: creator file space used allocated size physical file name option tables
	BY TABLE PRIVILEGE	Lists the idents with table privilege on the databank.
	FULL	Lists the following information on the specified databank: creator file space used allocated size physical file name option tables comments creation date a list of all the idents with table privilege on the databank.

DESCRIBE	OPTION	RESULT
DOMAIN	DOMAIN	Lists the following information on the specified domain: creator data type default value check clause
	TABLES USING	Lists the following information on the specified domain: tables including domain columns in those tables that use the domain in their definition
	FULL	Lists the following information on the specified domain: creator data type default value check clause tables including domain columns in those tables that use the domain in their definition comments creation date

DESCRIBE	OPTION	RESULT
IDENT	IDENT	Lists the following information on the specified ident: creator type of ident (user, os_user, program or group) privileges held by ident (DATABANK or IDENT)
	BY ACCESS	Lists the following information on the specified ident: tables and columns to which the ident has access access privileges that the ident has on the tables and columns listed above
	BY EXECUTE PRIVILEGE	Lists programs on which the ident has the execute privilege.
	BY MEMBERSHIP	Lists groups of which the ident is a member.
	BY OWNERSHIP	Lists objects that the ident has created: object type object name
	BY TABLE PRIVILEGE	Lists databanks in which the ident has the right to create tables.
	FULL	Lists the following information on the specified ident: creator ident type privileges objects to which the ident has access program names (on which the ident has execute privilege) group names (of which the ident is a member) objects which the ident has created databanks on which the ident has table privileges comments creation date
INDEX	NONE	Lists the following information on the index: whether index is unique creator index name table name columns on which the index is defined whether sort order is descending comments creation date

SYNONYM	SYNONYM	Lists the following information on the specified synonym: creator original name of the table/view creator of the table/view
	BY ACCESS	Describes the table with the specified synonym name and which idents have access to that synonym.
	FULL	Lists the following information on the specified synonym: creator table/view name table/view creator idents with access to synonym comments creation date
TABLE	SHORT	Lists the column names and data types in a table, view or synonym.
	TABLE	Lists the following information on the table/view/synonym: creator column names (data type, size, scale) primary key default values unique constraint domains (used in column definitions) indexes
	BY ACCESS	Lists which idents have access to the table/view/synonym: ident names access privileges
	BY REFERENCES	Describes which foreign key references the table/view/synonym contains: referencing table referencing columns referenced table referenced columns
	BY VIEWS	Lists views defined on the table/view/synonym.
	FULL	Lists the following information on the specified table/view/synonym: creator column names (data type, size, scale) primary key default values unique constraint domains used indexes idents with access to table foreign key references views defined on table comments creation date date when statistics were generated

DESCRIBE	OPTION	RESULT
VIEW	VIEW	Lists the following information on the specified view: creator view definition
	BY ACCESS	Lists the idents that have access privileges on the view.
	FULL	Lists the following information on the specified view: creator view definition ident names with access to the view comments creation date

DOWN

Scrolls downwards a specified number of rows in the results of DESCRIBE or LIST functions, in the results of a SELECT statement or in the editor environment.

Syntax

DOWN [value]

Description

This command scrolls downwards by *value* rows in the DESCRIBE and LIST display, in the results of a SELECT statement or in the editor environment. See also UP. The default for *value* is 1.

EXECUTE

Executes the statement on the current line in the editor window. (A statement is one or more lines terminated by a semicolon.)

Syntax

EXECUTE

Description

If the cursor is on the command line, the command executes the statement on the current line (the line shown in bright or reverse video on the screen). Otherwise, the line that the cursor is on in the editor window is used as the current line. The following rules determine which statement in the editor window is executed:

- The current line is placed on a line of text which is part of an SQL statement. That statement will be executed.
- The current line lies on a blank line between two statements or below the last statement in the window. The statement immediately preceding the current line will be executed.
- The current line is above the first statement in the window. The first statement in the window will be executed.

Unless the window is empty, some statement will always be executed with the EXECUTE command. If the statement is a SELECT, the result table will be shown in the editor window. If the statement is another SQL statement, a verification of the completion of the operation (x rows updated, etc.) will be shown in the message area.

EXIT

Leaves the current environment and returns to the previous environment.

Syntax

EXIT

Description

If you are in the editor window, the EXIT command will result in your leaving ISQL. If the screen shows the result of some statement, the EXIT command will return you to the editor window.

EXIT is equivalent to QUIT.

FORWARD

Moves forward one screen display in the results of DESCRIBE or LIST functions, in the results of a SELECT statement or in the editor environment.

Syntax

FORWARD [value]

Description

This command scrolls forwards in the text buffer in the DESCRIBE and LIST display, in the results of a SELECT statement or in the editor environment. The parameter specifies the number of screens to scroll (default=1). See also BACKWARD.

HELP

Provides help text on specified information.

Syntax

HELP [topic]

Description

The HELP command provides on-line help for the ISQL environment. If a valid topic is specified as a parameter, help is displayed for that topic. If no parameter is given, help is displayed for the HELP function itself.

The HELP display screen is split in two parts. The upper part is a text window displaying the help. You can move up and down in the help text with the TEXT ABOVE and TEXT BELOW menu options. The lower part is a menu window showing keywords for additional information on the subject shown. You can move up and down in the additional information menu with the arrow keys or the UP, DOWN, BACKWARD, and FORWARD commands

A number of standard alternatives are included in every additional information menu:

TEXT ABOVE	shows the text above the screen window, if any
TEXT BELOW	shows the text below the screen window, if any
HELP CONTENTS	shows a menu of contents for the entire help system. Move up and down in the contents with the arrow keys or the UP, DOWN, BACKWARD, and FORWARD commands. Select the item marked by the cursor by pressing ENTER.
QUIT HELP	leaves the help function and returns to the previous environment.

LEFT

Moves left the specified number of positions.

Syntax

LEFT [value]

Description

This command moves to the left by *value* positions in the results of LIST functions, in the results of a SELECT statement or in the editor environment. If no value is given, the display moves 10 positions in the editor and one column width or 75 characters whichever is the smaller in LIST and SELECT result sets.

LIST

Lists information about database objects.

Syntax

LIST [object-type]

Description

When the LIST command is issued, the following menu is shown:

```

----- M I M E R / I S Q L   L I S T   -----
      DATABANKS   DOMAINS   IDENTS   INDEXES
      OBJECTS     SYNONYMS  TABLES  VIEWS

      Choose the type of item you wish to LIST
      by using the arrow keys and press ENTER

(PF2=Help) Command ===>

```

Use the arrow keys to move to the item you wish to list. For menu type 1 (see Appendix B), press ENTER when the item you want is highlighted. For menu type 2, enter 'X' in the field directly to the left of the item and press ENTER. See the PROFILE command for more information on menu types.

Entering an object type on the command line skips the first menu.

When you have chosen the object to list, a new menu is shown listing the different options available for the object. The options for each object are listed in the table below. Only one option may be chosen for one operation.

When you have chosen the option, enter the object name and press ENTER. Some options restrict the listing by a second parameter (e.g. list databanks created by a specific ident). You will be prompted for the second parameter.

Only objects to which you have access are listed.

The result of the LIST command is displayed on the screen, and may be printed by issuing a PRINT command directly after the LIST.

LIST	OPTION	RESULT
DATABANKS	DATABANKS	Lists all databanks in the database.
	CREATED BY	Lists databanks created by a specified ident.
	WITH OPTION	Lists databanks with a specified option (LOG, TRANS or NULL).
	SHADOWS	Lists all shadows in the database.
	FOR DATABANK	Lists all shadows for a specified databank.
DOMAINS	DOMAINS	Lists all domains in the database.
	CREATED BY	Lists domains created by a specified ident.
IDENTS	IDENTS	Lists all idents in the database.
	CREATED BY	Lists idents created by a specified ident.
	BY TYPE	Lists idents of a specified type (PROGRAM, USER, OS_USER or GROUP).
	WITH ACCESS TO TABLE	Lists idents that have access to a specified table.
	WITH EXECUTE ON PROGRAM	Lists idents that have EXECUTE privilege on a specified program.
	WITH MEMBER ON GROUP	Lists idents that are members of a specified group.
	WITH TABLE ON DATABANK	Lists idents that have TABLE privilege on a specified databank.
	WITH DATABANK PRIVILEGE	Lists idents with DATABANK privilege.
	WITH IDENT PRIVILEGE	Lists idents with IDENT privilege.
	WITH RESTRICTED UPDATE	Lists idents with restricted UPDATE privilege.

LIST	OPTION	RESULT
INDEXES	INDEXES	Lists the secondary indexes in the database.
	CREATED BY	Lists secondary indexes created by a specified ident.
	DEFINED ON TABLE	Lists secondary indexes defined on a specified table.
OBJECTS	OBJECTS	Lists objects in the database.
	CREATED BY	Lists objects created by a specified ident.
	BY TYPE	Lists objects of a specified type.
SYNONYMS	SYNONYMS	Lists synonyms in the database.
	CREATED BY	Lists synonyms created by a specified ident.
	DEFINED ON TABLE	Lists synonyms defined for a specified table, view or synonym.
TABLES	TABLES	Lists tables in the database.
	CREATED BY	Lists tables created by a specified ident.
	USING DOMAIN	Lists tables that use a specified domain in the table definition.
	IN DATABANK	Lists tables in a specified databank.
VIEWS	VIEWS	Lists views in the database.
	BY CREATOR	Lists views created by a specified ident.
	DEFINED ON TABLE	Lists views defined on a specified table.

Logfile

Specifies a sequential file where duplicate rows are logged if Report duplicates is set to 'Yes'. The sequential file is created when the first duplicate record is encountered.

The Logfile setting is ignored if Report duplicates is set to 'No'.

Columnname, Start, End

Specifies which columns to load and the sequential position of the first and last byte of the column value in the file. The length of the value in the file depends on the data type of the column as follows:

Datatype	Length in file (bytes) Nullbyte = No	Length in file (bytes) Nullbyte = Yes
CHAR(n)	n	n + 1
INT(p)	p + 1	p + 2
DEC(p,s)	p + 2	p + 3
FLOAT(p)	p + 7	p + 8
DATE	10	11
TIME(0)	8	9
TIME(p)	p+9	p+10
TIMESTAMP(0)	19	20
TIMESTAMP(p)	20+p	21+p
YEAR(p)	p+1	p+2
MONTH(p)	p+1	p+2
YEAR(p) to MONTH	p+4	p+5
DAY(p)	p+1	p+2
HOUR(p)	p+1	p+2
MINUTE(p)	p+1	p+2
SECOND(p,0)	p+1	p+2
SECOND(p,s)	p+s+2	p+s+3
DAY(p) to HOUR	p+4	p+5
DAY(p) to MINUTE	p+7	p+8
DAY(p) to SECOND(0)	p+10	p+11
DAY(p) to SECOND(s)	p+s+11	p+s+12

Datatype	Length in file (bytes) Nullbyte = No	Length in file (bytes) Nullbyte = Yes
HOUR(p) to MINUTE	p+4	p+5
HOUR(p) to SECOND(0)	p+7	p+8
HOUR(p) to SECOND(s)	p+s+8	p+s+9
MINUTE(p) to SECOND(0)	p+4	p+5
MINUTE(p) to SECOND(s)	p+s+5	p+s+6

If no columns are specified, data will be loaded into all columns in the order defined in the table. If no start/end positions are specified, data will be read sequentially from the file as required by the column data length. Issue the command DEFAULT to set default values according to the table definition.

For example:

Sequential file

```
line in file   S K Y L I N E       S T O C K H O L M
position       1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
```

LOAD form

```
Nullbyte N
Columnname      Start      End
HOTELCODE_____  __1      __3
NAME_____       __1      __7
CITY_____       __9      __20
```

Loaded table

HOTELCODE	NAME	CITY
SKY	SKYLINE	STOCKHOLM

The LOAD form does not handle table rows that extend over more than one line in the sequential file. Each line in the file is loaded as one row in the table.

The LOAD command may not be used if a transaction is active. For further information on transactions, see Chapter 6.

NEXT

Moves to the next menu in the DESCRIBE or LIST functions.

Syntax

NEXT

Description

This command may only be used while in the DESCRIBE and LIST functions. The command is used to go to the next menu display. See also PREVIOUS.

PERFORM

Sends the specified sequential file to the SQL compiler for execution.

Syntax

PERFORM file_name

Description

The specified sequential file will be read and sent to the SQL compiler for execution. Each statement in the file should be terminated with a semicolon (;).

In the event of an error during the execution of the file, the erroneous statement will be placed in the editor buffer. The user then has three alternative choices in handling the error which are the SKIP, CONTINUE or CANCEL commands described in this chapter.

PFK

Assigns commands to function keys. See Appendix B for details.

PREVIOUS

Moves back to the previous menu in the DESCRIBE or LIST functions.

Syntax

PREVIOUS

Description

This command may only be used in the DESCRIBE and LIST functions. The command is used to return to the previous menu display. See also NEXT.

PRINT

Prints the results of a SELECT statement or a LIST or DESCRIBE in the current printer file.

Syntax

PRINT

Description

This command can be used with SELECT statements or with the LIST and DESCRIBE commands.

When PRINT is requested for a SELECT statement, the result set of that statement is printed on the current printer file in accordance to the PROFILE settings (see the command PROFILE for a list of printer file settings). This is equivalent to requesting EXECUTE then printing the result set.

PROFILE

Lists and/or changes the current environment settings.

Syntax

PROFILE

Description

When the PROFILE command is issued, the following menu is displayed.

```

----- M I M E R / I S Q L   P R O F I L E -----

                Printer file characteristics

File name   PRINTER.LIS
Page width  132                               Page length : 45
Initial page : Y                               Column header : Y
Page number  : Y                               Printer line spacing : 0

                Miscellaneous

Menu style : 2                               Toplabel : Y

Server : ...

Connection : ...

Current user: ...

ISQL version ...                             DB version ...

Transaction start Explicit                    Transaction changes Visible

(PF2=Help) Command ===>

```

The settings are described in the table below.

Printer file characteristics	
File name	The name of the file or device where the result of a PRINT command will be written. Changing the printer file name will result in closing any previously opened file. The default name is machine-dependent.
Page width	The record length for the printer file or device. If a single column value exceeds the page width, it will be divided into two or more lines. Changing the page width value does not affect the record length for an already opened printer file. The default is machine-dependent.
Page length	The page length for the printer file. Column headings and page numbers are printed at the top of each page. The default is machine-dependent.
Initial page	A header page with information about who requested the printer file and when. Values Yes/No (default Yes).
Column header	Column headers are included at the top of each page in the printout. Values Yes/No (default Yes).
Page number	Page numbers are included at the top of each page in the printout. Values Yes/No (default Yes).

Printer line spacing	Number of blank lines between each record in the printout. Default 0.
Miscellaneous settings	
Menu style	Style for the HELP, LIST and DESCRIBE commands. Valid values are 1 and 2. The default setting is machine-dependent. See Appendix B for a description of menu styles.
Toplabel	Type of picture for displaying results. Values Yes (for several hits), No (for one hit).
Current server	Display-only field for information.
Current connection	Display-only field for information (only if specified)
Current user	Display-only field for information.
ISQL version	Display-only field for information.
DB version	Display-only field for information.
Transaction start	Display-only field for information (values Explicit/Implicit).
Transaction changes	Display-only field for information (values Visible/Invisible).

QUIT

Leaves the current environment and return to previous environment.

Syntax

QUIT

Description

If you are in the editor buffer, the QUIT command will result in your leaving ISQL. If the screen shows the result of some statement, the QUIT command will return you to the editor window.

QUIT is equivalent to EXIT.

READ

Reads a specified file into the editor buffer.

Syntax

READ file_name

Description

The specified sequential file is inserted into the editor window immediately after the current line.

REMOVE

Removes error messages produced by the SQL compiler from the editor window.

Syntax

REMOVE

Description

This command removes error messages produced by the SQL compiler from the editor window. Error messages from the SQL compiler appear within the editing area in the editor window. System errors, which appear on the top of the screen, are not affected by the REMOVE command.

Compiler error messages are automatically removed when an EXECUTE or PRINT command is given.

RIGHT

Moves right the specified number of positions.

Syntax

RIGHT [value]

Description

This command moves to the right by *value* positions in the results of LIST and SELECT and in the editor environment. If no value is given, the display moves 10 characters in the editor and one column width or 75 characters whichever is the smaller in LIST and SELECT result sets.

SET HEADER

Specifies whether column headers are to be included at the top of each page in the printer file.

Syntax

SET HEADER ON|OFF

Description

The SET HEADER command defines if column headers should be included at the top of each page in the printer file. ON gives a column header. OFF excludes the column header.

The default is ON.

SET INITPAGE

Specifies whether an initial page should be written to the printer file.

Syntax

```
SET INITPAGE ON|OFF
```

Description

The SET INITPAGE command defines whether a header page, with information about who requested the printout and when, should be included in the printer file.

The default is ON.

SET KEY

To assign commands to function keys.

Syntax

```
SET KEY keyname command
```

Description

The SET KEY command gives the possibility to change the assignment of commands defined for function keys. See Appendix B for details.

Example:

```
SET KEY F7 UNDO
```

SET LINESPACE

Sets the number of blank lines between each output record in the printer file.

Syntax

```
SET LINESPACE|LS value
```

Description

The LINESPACE value defines the number of blank lines to be written between each output record.

The maximum value for LINESPACE is 9. The default value is 0.

SET MENUSTYLE

Sets menu style for HELP, LIST and DESCRIBE commands.

Syntax

```
SET MENUSTYLE 1|2
```

Description

The SET MENUSTYLE command specifies the style for HELP, LIST and DESCRIBE commands. Valid values are 1 and 2. The default setting is machine-dependent. See Appendix B for a description of menu styles.

SET PAGELENGTH

Specifies the page length for the printer file.

Syntax

```
SET PAGELENGTH|PL value
```

Description

The PAGELENGTH value defines the page length of the printer file, i.e. at what interval a page break will be performed. Column headers and page numbers may be printed at the top of each page. A value of zero will result in no page breaks.

The PAGELENGTH value can either be set to zero or ≥ 10 . The default value is machine-dependent.

SET PAGENUMBER

Specifies whether page numbers should be included.

Syntax

```
SET PAGENUMBER ON|OFF
```

Description

The SET PAGENUMBER command defines if page numbers are to be included at the top of each page in the printer file.

The default is ON.

SET PAGEWIDTH

Specifies the page width for the printer file or device.

Syntax

```
SET PAGEWIDTH|PW value
```

Description

The SET PAGEWIDTH command controls the page size for the printer file. If a single column value exceeds the page width, it will be divided into two or more lines. Changing the page width value does not affect the value for an already opened printer file.

The value should be ≥ 20 . The default value is machine-dependent.

SET TOPLABEL

Specifies type of picture for results.

Syntax

```
SET TOPLABEL ON|OFF
```

Description

The SET TOPLABEL command defines the type of picture for displaying results. TOPLABEL ON gives a picture for several hits and TOPLABEL OFF gives a picture for one hit.

The default is ON.

SKIP

Skips over an invalid statement in a sequential file during a PERFORM.

Syntax

SKIP

Description

When an error occurs in the execution of a sequential file, the command SKIP can be chosen to ignore the invalid statement in the file and continue execution with the next statement in the file. Other options are CONTINUE to re-try the execution of the statement or CANCEL to stop the PERFORM.

TOGGLE

Switches between the current environment and the result of the latest LIST or DESCRIBE command or SELECT statement.

Syntax

TOGGLE

Description

This command switches the display between the current environment and a buffer containing the result of the latest LIST, DESCRIBE or SELECT statement. This buffer will only be kept until the next successful execution of a LIST, DESCRIBE or SELECT statement. This command cannot be used in the HELP system or when changing values in the PROFILE form.

TOP

Moves to the top of the text buffer.

Syntax

TOP

Description

This command moves to the top of the text buffer in the DESCRIBE and LIST display, in the results of a SELECT statement or in the editor environment.

UP

Scrolls upwards the specified number of rows in the results of DESCRIBE or LIST functions, in the results of a SELECT statement or in the editor environment.

Syntax

UP [value]

Description

This command scrolls upwards by *value* rows in the DESCRIBE and LIST display, in the results of a SELECT statement or in the editor environment. The default for *value* is 1.

WDW

Sets display parameters for the editor window. See Appendix B for details.

WRITE

Writes the contents of the editor window to a sequential file.

Syntax

WRITE [file_name]

Description

The contents of the editor window are written to the specified sequential file. Any blank lines at the end of the window are not written.

If no file name is given, ISQL will prompt for one.

11 BSQL COMMANDS

BSQL is a facility for executing SQL statements in batch jobs. All SQL statements may be used in BSQL. The ISQL commands (Chapter 10) are replaced in BSQL by a set of batch-oriented commands, documented in this chapter.

11.1 Running BSQL

BSQL can be run from a batch job or from a terminal. Operation from a terminal can be used to execute statements entered directly or written in sequential files.

11.1.1 Running BSQL from a batch job

To run BSQL unattended from a batch job, create a batch file with the following contents:

- command to start BSQL
- username
- password
- SQL statements and BSQL commands
- EXIT command (or end of file)

Note that for unattended operation, a batch file must either include the MIMER ident username and password in explicit form or connect as OS_USER. For security reasons, make sure that your batch files are well protected and/or remove your password from the file after execution. Alternatively, SQL statements and BSQL commands may be written in a sequential file without username and password, and executed with the READ command from a BSQL terminal session.

11.1.2 Running BSQL via the terminal

See your machine-specific MIMER User's Guide for how to start BSQL for your system. Starting BSQL displays the following screen:

```

MMMMM      MMMMM  MMMMM  MMMMM      MMMMM  MMMMMMMMMMMM  MMMMMMMM
MMMMMMM    MMMMMM  MMMMM  MMMMMM    MMMMMM  MMMMMMMMMMMM  MMMMMMMMMM
MMMMMMM  MMMMMM    MMM    MMMMMM  MMMMMM    MMM    MMM    MMM  MMM
MMMMMMMMMMMMMMMM    MMM    MMMMMMMMMMMMMMMM    MMMMM    MMMMMMMM
MMM  MMMMM  MMM    MMM    MMM  MMMMM  MMM    MMM    MMM    MMM  MMM
MMMM  MMM  MMMM  MMMMM  MMMM  MMM  MMMM  MMMMMMMMMMMM  MMMM  MMMM
MMMMM  M  MMMM  MMMMM  MMMM  M  MMMM  MMMMMMMMMMMM  MMMM  MMMM

```

(C) Copyright SYSDECO MIMER AB 1987-1996. All rights reserved

MIMER/BSQL

Version x.x.x

Username:

Password:

When the username and correct password are entered, the BSQL prompt will be shown:

SQL>

BSQL commands and SQL statements can now be entered. Output will be echoed on the terminal.

11.2 BSQL commands

Command	Function
CLOSE	Close active log files
DESCRIBE	Describes a specified object
EXIT/QUIT	Leaves BSQL
HELP	Provides on-line help
LIST	Lists information on a specified object
LOAD	Loads data into a table
LOG	Logs input, output or both on a sequential file
READ INPUT	Reads commands from a sequential file
SET ECHO	Specifies whether lines are echoed to the terminal during READ INPUT
SET LINECOUNT	Sets the terminal page size
SET LINESPACE	Sets the number of blank lines between each output record
SET LINEWIDTH	Sets the terminal page width
SET LOG	Stops or resumes logging input, output or both

SET MESSAGE	Specifies whether messages are displayed on the terminal
SET OUTPUT	Specifies whether output should be written to the terminal
SET PAGELength	Defines the page length of output file
SET PAGEWIDTH	Defines the page width of output file
SHOW SETTINGS	Displays current values of all set options
UNLOAD	Unloads data from a table
WHENEVER	Sets action to be taken in response to error or warning

BSQL commands are not case sensitive.

Note on syntax descriptions

In the syntax descriptions, items in square brackets ([]) are optional. Items separated by a vertical bar (|) are alternatives. For example

```
READ [COMMAND|ALL] [INPUT FROM] 'filename';
```

allows the following forms

```
READ COMMAND INPUT FROM 'filename';
```

```
READ ALL INPUT FROM 'filename';
```

```
READ INPUT FROM 'filename';
```

```
READ 'filename';
```

CLOSE

Closes log files.

Syntax

```
CLOSE [INPUT|OUTPUT|INPUT,OUTPUT] log;
```

Description

The command closes the specified log file. If no log file is specified, all activated log files are closed.

DESCRIBE

Describes a specified object.

Syntax

DESCRIBE [object-type [object-name]];

Description

The DESCRIBE command presents the following menu:

```

-- Menu for describe --
1. Databank      4. Index      7. View
2. Domain       5. Synonym   0. Exit
3. Ident        6. Table
Select :_

```

Choosing an item presents a submenu for choosing between different DESCRIBE functions. See the ISQL command DESCRIBE for details (Chapter 10). Entering an exclamation mark (!) in the Select field returns to the previous menu level. Entering a double exclamation mark (!!) terminates the DESCRIBE session.

Giving an object type and name in the command executes the first menu choice for that object. If no object name is given, the user is prompted for a name.

Selection numbers can be provided in a batch file for unattended operation. However, DESCRIBE is most useful in interactive mode from a terminal.

EXIT

Leave BSQL.

Syntax

EXIT;

Description

Leave BSQL and return to the operating system. EXIT may also be written as QUIT.

HELP

Provides help text on the specified argument.

Syntax

HELP [argument];

Description

Provides a general help text or help text on the specified argument. The help text is displayed one screen at a time according to the screen length set by the SET LINECOUNT command. Press RETURN to display the next screen. If LINECOUNT is set to zero, the help text will be shown without interruption.

At the end of the help text for a specified argument, a menu lists related topics and offers the two additional choices CONTENTS and QUIT. CONTENTS lists all topics available in the help system. QUIT leaves the help system.

Note that the QUIT command issued at the SQL> prompt terminates the BSQL session.

LIST

Lists information on a specified object.

Syntax

LIST [object-type];

Description

The LIST command presents the following menu:

```

-- Menu for List --
1. Databanks      4. Indexes      7. Tables
2. Domains        5. Objects      8. Views
3. Idents         6. Synonyms    0. Exit

```

```
Select :_
```

Choosing an item presents a submenu for choosing between different LIST functions. See the ISQL command LIST for details (Chapter 10). Entering an exclamation mark (!) in the Select field returns to the previous menu level. Entering a double exclamation mark (!!) returns two levels.

Giving an object type in the command executes the first menu choice for that type.

Selection numbers can be provided in a batch file for unattended operation. However, LIST is most useful in interactive mode from a terminal.

LOAD

The LOAD command can be used to load data from a sequential file into a target table.

Syntax

```
LOAD FROM 'file-name' INTO table-name <NULL|NONNULL,>
      <DUPLICATES|NODUPLICATES> <LOGFILE 'file-name'>
      < (column-name POS(s:e), ..., column-name POS(s:e) ) >;
```

Description

NULL (default) specifies that the first byte for each column value in the input file is used to indicate whether the value is NULL or not. An ampersand (&) in this byte indicates NULL, all other values indicate NOT NULL.

NONULL specifies that the values in the input file are entered into the columns exactly as read (i.e. NULL values can not be entered).

DUPLICATE (default) specifies that the number of duplicates found during the load operation will be reported. If a LOGFILE is specified, any duplicate row will also be logged there. NODUPLICATES means that number of duplicates will not be reported or logged.

LOGFILE specifies a sequential file, where duplicate rows may be logged.

If column-specifications are given, only values for the columns which are given will be read from the input file. For each column, the sequential position for the start and the end byte of the value to assign should be specified in POS(s:e).

If column-specifications are not given, default values for positions to read from are determined from the table definition. All columns will be given values.

The LOAD command may not be used if a transaction is active. For further information on transactions, see Chapter 6.

Example:

```
LOAD FROM 'rooms.dat' INTO rooms NULL,DUPLICATES
LOGFILE 'rooms.dup';
```

```
LOAD FROM 'rooms2' INTO rooms NONULL (roomno POS(1:5),
roomtype POS(8:18));
```

LOG

Logs input, output or both to a specified sequential file.

Syntax

```
LOG INPUT|OUTPUT| INPUT,OUTPUT ON|APPEND 'filename';
```

Description

All input, output or both will be logged in the specified sequential file. If ON is specified a new file will always be created, otherwise the log data is appended to the file.

Logging is stopped with the SET LOG OFF command and is resumed with the SET LOG ON command.

READ INPUT

Reads commands from a sequential file.

Syntax

```
READ [COMMAND|ALL] [INPUT FROM] 'filename';
```

Description

Commands and SQL statements are read from the specified file.

When READ COMMAND INPUT is specified, commands are read from the file while prompt answers are taken from the terminal (batch job, command procedure).

When READ ALL INPUT or READ INPUT is specified, both commands and prompt answers are read from the sequential file.

SET ECHO

Controls whether or not lines read during READ INPUT are echoed.

Syntax

```
SET ECHO ON|OFF;
```

Description

When echo is set to ON, lines read during READ INPUT are echoed to the terminal or batch log file. When echo is set to OFF, these lines are not echoed. The default value is ON.

The setting has no effect on the output of responses to BSQL commands and statements.

SET LINECOUNT

Sets the length of the terminal page.

Syntax

```
SET LINECOUNT|LC value;
```

Description

The LINECOUNT value defines the length of the terminal page.

If LINECOUNT has a value greater than zero, terminal output will temporarily be stopped after the number of lines defined for the value. After the "Continue"-prompt, the user will have the choice of either continuing with the display or terminating the output. Answering 'Y' (default) implies that the output will continue until the number of lines are reached again. Answering 'N' terminates the output. Answering 'G' will ignore the linecount and the output will continue until all data are displayed.

If LINECOUNT is zero, the output will continue until all data is displayed.

The value of LINECOUNT must either be zero or ≥ 10 .

Default

If BSQL is run from a batch job, LINECOUNT is zero by default. For interactive operation, the default value is machine- and terminal-dependent.

SET LINESPACE

Sets the number of blank lines between each output record.

Syntax

```
SET LINESPACE|LS value;
```

Description

The LINESPACE value defines the number of blank lines to be written between each output record. This value is only used when printing the result of a SELECT statement.

The maximum value for LINESPACE is 9. The default value is 0.

SET LINEWIDTH

Specifies the width of the output.

Syntax

```
SET LINEWIDTH|LW value;
```

Description

The LINEWIDTH value defines the maximum line width for output to the terminal or batch log file.

The value for LINEWIDTH cannot be set to a value less than 20.

SET LOG

Stops or resumes logging input, output or both.

Syntax

```
SET [INPUT|OUTPUT|INPUT, OUTPUT] LOG OFF|ON;
```

Description

When SET LOG is set to OFF, logging of input, output or both in a sequential file is temporarily stopped.

Resume logging with the SET LOG ON command.

If no input/output log is specified, all activated logs are stopped or resumed.

SET MESSAGE

Specifies whether or not messages should be displayed.

Syntax

```
SET MESSAGE|MSG ON|OFF;
```

Description

Specifies whether or not result messages such as "One row found" etc. are written to the terminal screen or batch log file.

The default setting is ON.

SET OUTPUT

Specifies whether or not output should be displayed.

Syntax

```
SET OUTPUT ON|OFF;
```

Description

When OUTPUT is set to ON, the output from BSQL is written to the terminal or batch log file. When it is set to OFF, the output does not appear. The default value is ON.

SET PAGELENGTH

Specifies the page size of the output log file.

Syntax

```
SET PAGELENGTH|PL value;
```

Description

The PAGELENGTH value defines the page size of the file on which output is logged, i.e. at what interval a page break will be performed. A value of zero will result in no page breaks.

The PAGELENGTH value can either be set to zero or ≥ 10 . The default value is machine-dependent.

SET PAGEWIDTH

Specifies the page width of the output log file.

Syntax

```
SET PAGEWIDTH|PW value;
```

Description

The PAGEWIDTH value defines the page width of the output file. The value should be ≥ 20 . The default value is machine-dependent.

SHOW SETTINGS

Displays the current values of all set options.

Syntax

```
SHOW SETTINGS;
```

Description

Display the current values for all set options, i.e. ECHO, LINECOUNT, LINESPACE, LINEWIDTH, LOG, MESSAGE, OUTPUT, PAGELENGTH, PAGEWIDTH, TRANSACTION START and TRANSACTION CHANGES.

Current server and connection names are also displayed.

UNLOAD

The UNLOAD command can be used to unload data from a table into a sequential file.

Syntax

```
UNLOAD TO 'file-name' FROM table-name <NULL|NONULL>
      <(column-name POS(s:e), ..., column-name POS(s:e)) >;
```

Description

NULL (default) specifies that the first byte for each column value in the output file is used to indicate whether the value is NULL or not. This byte is assigned an ampersand (&) if the column from which the field is derived contains NULL, the rest of the field is filled with periods (...). Otherwise the byte is blank.

NONULL specifies that the first byte for each column value in the output file is the first data byte of the value.

If column-specifications are given, the output file will only hold values for the columns which are given. For each column the sequential position of the start and the end byte of the column value should be specified in POS(s.e). Overlapping is not controlled.

If column-specifications are not given, default values for positions are determined by the table definition. All columns will be included.

The UNLOAD command may not be used if a transaction is active. For further information on transactions, see Chapter 6.

Example:

```
UNLOAD TO 'rooms.dat' FROM rooms;

UNLOAD TO 'rooms2' FROM rooms NONULL
      (roomno POS(1:5), roomtype POS(8:18));
```

WHENEVER

Determines which actions should be taken in the event of an error or warning.

Syntax

```
WHENEVER ERROR|WARNING action<,>action>;
```

Description

If an error or warning should occur in a file being run in batch, there are several "action" options that may be chosen to determine what should happen.

The actions can be broken down into two groups:

Execution flow

EXIT leaves BSQL in batch mode. Returns to prompt if interactive mode. I.e if interactive mode and file input mode, the remaining file input is ignored and a new prompt is received.

CONTINUE continues execution.

Transaction control

ROLLBACK abandons the transaction; no changes are made to the database.

COMMIT requests that the operations are executed against the database, and the changes in the database are made permanent.

Default

The default value for warning is CONTINUE.

The default values for errors are EXIT, ROLLBACK in batch mode or file input mode and CONTINUE in interactive mode.

12 VARIABLES IN ISQL AND BSQL

Host variables are used in embedded SQL statements to pass values between the database and an application program (see the MIMER/SQL Programmer's Manual). Host variables are also supported in ISQL and BSQL, to facilitate interactive design and testing of SQL statements intended for use in embedded SQL application programs. In ISQL and BSQL, the host variables serve as parameter markers, and the user is prompted for parameter values when the statement is executed.

Host variables may be used to assign values to columns in the database (UPDATE and INSERT statements), to manipulate information taken from the database or contained in other variables (in expressions), and to provide values for comparison predicates. In all these contexts, the data type and length of the host variable must be compatible with that of any database values within the same syntax unit.

Host variables are written in SQL as

```
                :host-identifier  
or                :host-identifier:indicator-identifier
```

In the first construction, the host identifier is the name of the main host variable. In the second construction, the main variable host-identifier is associated with an indicator variable indicator-identifier, used to signal the assignment of a NULL value to the main variable. See the MIMER/SQL Programmers Manual for a description of the use of indicator variables.

The scope of host variables in ISQL and BSQL is restricted to the individual usage instance in each statement. Variables may not be used to pass values between separate statements, and the same variable name used more than once in a statement represents separate, independent variables.

12.1 Host variables in ISQL

When host variables are used in ISQL, a form is displayed showing the statement where the variables are used and one input field for each variable. To execute the query, fill in values for each host variable and press ENTER. When the statement has been executed, the form will be shown again, and you can then enter new values for the host variables and execute the statement again. If the indicator variable is associated with a host variable, there will be an extra field on the form in which you indicate whether the NULL indicator is to be set for the column. When this field is filled in (any character may be used to fill it in) the NULL indicator will be set for the column. If the field is left blank, the NULL indicator will not be set.

Enter the EXIT command on the command line to leave the form and return to the editor window.

For more information on host variables in embedded SQL statements, see the MIMER/SQL Programmer's Manual.

12.2 Host variables in BSQL

When host variables are used in BSQL, MIMER prompts for the variable values, for example:

```
SQL>SELECT * FROM HOTEL WHERE CITY = :CITY;
CITY: STOCKHOLM
```

This statement is then executed as

```
SQL>SELECT * FROM HOTEL WHERE CITY = 'STOCKHOLM';
```

Note that the entered variable is *not* enclosed between apostrophes, in contrast to the corresponding string value. Variables enclosed in apostrophes will be interpreted as literal strings.

If an indicator variable is included, you will be prompted for whether to use a NULL value. If you answer the prompt with No, you will then be prompted for a value. If you answer Yes, the NULL value will be used. For example:

```
SQL>UPDATE BOOK_GUEST SET ARRIVE = :ARRIVE:NULL,
SQL> DEPART = :DEPART:NULL
SQL> WHERE RESERVATION = 1348;
Null:N
ARRIVE: 1997-04-23
Null:Y
```

Note that the prompts appear in the order in which the variables are used in the statement. In the example above, the ARRIVE value will be updated to 1997-04-23 and the DEPART value will be set to NULL.

The value prompting makes host variables limited to 80 characters in BSQL.

13 ERROR HANDLING

13.1 Message display

There are two types of messages in ISQL: system messages (including error messages, prompts, and result messages) and ISQL language specific error messages.

System messages appear in the upper right-hand corner of the screen and ISQL error messages appear in the editor window:

```
----- M I M E R / I S Q L -----
System messages appear here

ISQL error messages appear here

(PF2=Help) Command ==>
```

System messages are removed (or replaced) when a statement is executed. ISQL error messages may be removed with the command REMOVE (see Chapter 10).

13.2 ISQL error messages

ISQL error messages are shown when you attempt to execute an erroneous SQL statement. ISQL error messages are written in the editor window. There are two types of ISQL errors: semantic errors and syntax errors.

13.2.1 Semantic errors

Semantic errors arise when SQL statements are formulated with correct syntax, but do not reflect the user's intentions. For example, suppose that a user wishes to select the string constant 'Hotel:' and the actual hotel name from the table HOTEL, but uses quotation marks instead of apostrophes around the string constant:

```
SELECT  "Hotel:",NAME
FROM    HOTEL;
```

Quotation marks are used to delimit identifiers containing special characters, so that the statement is interpreted as a request to select two columns, called "Hotel:" and NAME, from the table. The first 'column' does not exist.

This example will in fact lead to an execution error, and is easily detected. Other semantic mistakes can be more difficult to find, when the statement is executed but gives the 'wrong' answer. An example is the incorrect use of NULL in a search condition:

```
SELECT  RESERVATION FROM BOOK_GUEST
WHERE   CHECKOUT = NULL;
```

This will always give an empty result set, since NULL is not equal to anything. (The correct formulation would read WHERE CHECKOUT IS NULL).

Always check the result of an ISQL query for reasonableness, in particular if the query is complicated.

13.2.2 Syntax errors

Syntax errors are constructions which break the rules for formulating SQL statements. For example:

- spelling errors in keywords
SLEECT (for SELECT)
- incorrect or missing delimiters
DELETEFROM (for DELETE FROM)
SELECT column1 column2 (for SELECT column1,column2)
- incorrect clause ordering
UPDATE table WHERE condition SET values
(for UPDATE table SET values WHERE condition)

Syntactically incorrect statements are not accepted and an appropriate error message is displayed. The error must be corrected before the statement can be executed.

For syntax errors, ISQL analyzes the statement and makes an intelligent guess as to where the error lies. This guess is based upon the most likely syntax or appearance of the statement in question. The system then points out the error and lists an error message based on this analysis. The appearance of this pointer on your screen is machine dependent. In the examples shown in this chapter, the pointer appears as '^'. The messages are self-explanatory.

The statement analysis is however not completely foolproof and misleading error messages may arise. If the message seems to be inaccurate, check the statement construction against the syntax diagram in the MIMER/SQL Reference Manual.

Some examples of errors and resulting error messages are listed below.

```
SELECT  AVG(NAME)
FROM    HOTEL;
```

Error message:

```
SELECT  AVG(NAME)
        ^
Invalid operand type. Expected type is NUMERIC or INTERVAL.
FROM HOTEL;
```

```
SELECT  NAME FROM HOTEL
WHERE   CITY ON ('STOCKHOLM', 'UPPSALA');
```

Error message:

```
SELECT  NAME FROM HOTEL
WHERE   CITY ON ('STOCKHOLM', 'UPPSALA');
        ^
Syntax error, 'ON' assumed to mean 'IN'
```

In the following example, the error analysis is incorrect:

```
SELECT  NAME FROM HOTEL
WJERE   HOTELCODE = 'LAP';
```

Error message:

```
SELECT  NAME FROM HOTEL
WJERE   HOTELCODE = 'LAP';
        ^
Syntax error, END-OF-QUERY assumed missing.
```

The misspelled word WJERE is not recognized as an attempt to write WHERE, so that the second line is not interpreted as a selection condition.

13.3 ISQL and BSQL error messages

Error messages from ISQL/BSQL are shown when you enter an illegal ISQL/BSQL command or attempt to execute an erroneous SQL statement. The error messages for erroneous SQL statements are the same as the return codes found in MIMER/SQL Programmer's Manual. Error messages that can be received for illegal ISQL/BSQL commands are:

-1500	Illegal value for <%>
-1400	Invalid numerical argument
-1300	Only select statements can be used with PRINT
-1200	Previous perform file is not finished
-1101	Disk space exhausted
-1009	Unspecified file open error
-1008	** Installation dependent **
-1007	** Installation dependent **
-1006	Disk space exhausted
-1005	Maximum number of opened files exceeded
-1004	File locked
-1003	File protection violation
-1002	File not found
-1001	Syntax error in file name
-999	Too long statement
-900	No buffer saved
-801	Pending transaction, Commit or Rollback
-800	Load/unload is not allowed within a transaction
-777	Maximum header length exceeded
-776	Maximum record length <%> exceeded
-701	Help topic not found
-700	Help databank not installed or inaccessible
-666	Space area exhausted
-600	The number of host variables cannot exceed 20
-400	Record too large for one page (<%> lines required) Increase value of LC/PL or set them to zero.
-300	Failed to read dictionary
-207	Too many parameters
-206	Unexpected end of command
-205	Invalid numerical literal
-204	Filenames must be enclosed in apostrophes
-203	String expected
-202	Undefined keyword
-201	Syntax error
-103	Missing semicolon
-102	<%> command not valid in this context
-101	Ambiguous command <%>
-100	Undefined command <%>

- 5 Conflict. One of COMMIT or ROLLBACK and EXIT or CONTINUE
- 3 Too many files have been opened
- 2 File could not be opened
- 1 String exceeds 256 characters which is not allowed

A MACHINE-DEPENDENT INFORMATION

The information in this manual is as far as possible independent of the computer system on which MIMER/SQL is used. While the MIMER product is designed to be identical on different machines, there are some inevitable differences in the user interface in different implementations. Machine-dependent information is collected in a separate publication, the MIMER User's Guide for each specific machine.

If you do not have a copy of the machine-dependent information for Interactive SQL, contact your system administrator or equivalent person.

B THE MIMER EDITOR

B.1 General description

The MIMER Editor provides facilities for simple text editing in full-screen mode. The editor is embedded in several MIMER modules where text-editing is necessary or desirable, for instance the interactive input in ISQL, and the editing facility for defining screen forms in FM. The core of editing functions is common to all implementations, so that the user has access to the same text-editing environment in many different contexts.

This documentation describes the central common functions of the editor. The interface to ISQL is described in Chapter 10.

B.2 Text buffer and screen utilization

B.2.1 Text buffers

The text on which the editor works is held in a freely formatted buffer. This means that in principle the editor can handle text lines of any length. The line length may in practice be limited by settings imposed by the environment in which the editor is used. The exact number of screen lines which the buffer can hold varies with the content of the buffer, since the text is stored with a certain amount of compression. For example in program source code, "average" text lines tend to occupy about 30-50 effective characters.

The text buffer may be visualized as a single piece of paper containing the text. The buffer is never divided into pages. The directions up (backward), down (forward), right and left in the buffer are used in their normal sense as for text on paper.

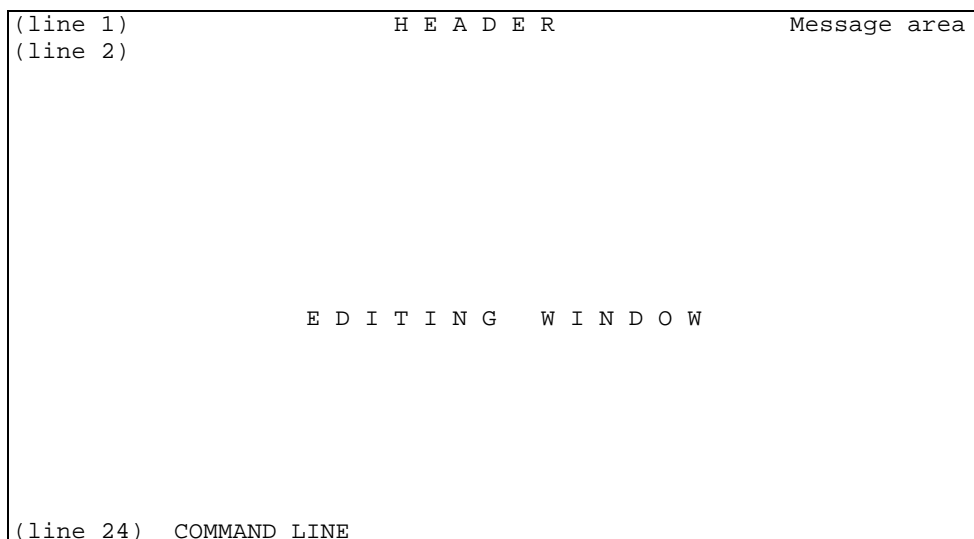
Editing is performed within a window corresponding to the size of the screen superimposed on the buffer. The window can be moved freely within the buffer, allowing editing of different parts of the text.

B.2.2 Screen utilization

The size of the editing window (i.e. the screen) may be determined to some degree by the user. By default, the top two lines of the screen are used as header and message area and the 5 rightmost positions are a prefix command area. The bottom line is a command line (see figure below).

The user can reset the top and bottom lines and the line length of the editing window within the limits of the physical screen, and with the restriction that the bottom line is always reserved. (The maximum number of lines on the screen is 24). The prefix command area may be placed to the right or left, or removed entirely. The length of the area may be set between 3 and 6 characters. When a prefix command area is used, the blank position immediately preceding the area on each line is reserved.

Commands for setting the screen utilization parameters are described in Section B.2.5.



B.2.3 The current line and the cursor

One line in the editing window is always marked as the current line (displayed with either bright or reverse video - see Section B.2.4). Several line and string manipulation commands operate in relation to the current line. The current line is moved within the text buffer by window movement commands (Section B.4.4).

The position of the current line within the editing window may be set by the user (see Section B.2.5). Changing the current line position in the editing window also moves the current line relative to the text buffer (the window is *not* moved relative to the buffer).

The position where character manipulation (insert, delete and overwrite) occurs is marked by the cursor, independent of the current line. The cursor may be moved independently of character manipulation using the arrow keys, tab, and backspace. Tab and backspace move the cursor to the next field forwards and backwards respectively. The text area in the editing window and the prefix command area on each line are separate fields. There is a single field on the command line.

Note: Some users may be accustomed to using the tab key to move a preset number of positions to the right. In the MIMER editor, the tab key always moves the cursor to the next field - the only way to tabulate text with the editor is to enter the appropriate number of blanks.

B.2.4 Standard menu types

Menu choices are offered in a number of situations in the different editor implementations. Two standard menu types are available, referred to as 'pseudocursor' and 'input field' menus. These differ in the way in which options are selected. The type of menu to be used may be set at any time by the user.

Pseudocursor menu type

In a 'pseudocursor' menu, the selected item is marked by a field attribute (either reverse or bright video). This marking is moved between the menu items with the arrow keys or the commands RIGHT, LEFT, UP, DOWN, entered via function keys or on the command line. If the display area is too small to hold all the menu items at once, the commands BACKWARD and FORWARD move the pseudocursor (i.e. the item selection) a page at a time.

The true cursor remains on the command line, where commands can be entered. The selected item is activated by pressing ENTER with no command on the command line. The command QUIT or EXIT leaves the menu without activating any item.

Input field menu type

In an 'input field' menu, every menu item is preceded by a one-character input field. The cursor may be moved between these fields in the normal way. An item is selected by typing any non-blank character in the corresponding field. Pressing the ENTER key activates the first selected item, or the item at the cursor position if no other item has been selected.

The cursor is moved with the TAB or arrow keys, or with the RIGHT, LEFT, UP, DOWN, BACKWARD and FORWARD commands as described above for the pseudocursor. In addition, UNDO clears all selected items and leaves the cursor position unchanged. RESET clears all selected items and sets the cursor on the command line. The command QUIT or EXIT leaves the menu without activating any item.

B.2.5 Setting screen parameters (WDW command)

Screen parameters may be set at any time by giving the command WDW on the command line. The following possibilities are offered (default values are shown):

```

----- MIMER Editor - Window Setup Assignments -----
                                Window Pos. & Size
Row.....: 3_ (first row on screen)
Column..: 1_ (first col on screen)
Rows....: 21 (no. of rows      )
Columns.: 80 (no. of columns   )

                                Prefix Area
Length..: 0 ( 0 or in <3,6> )
Position: R ( Left ! Right  )
Type....: N ( * or N<umeric> )

                                Current Line
Row ....: 5_ (relative to first )
Attrib..: B (Bright ! Reverse  )

                                Scale Line
Row ....: -1 (relative to first )

                                Standard Menu Type
Type....: 1_ ( 1 or 2 )

(PF2=Help) - Change settings as required, then press ENTER

```

Window position

Row First row on screen.

Permissible values between 1 and a maximum determined by the size of the physical screen and the number of rows set. The sum of the first row value and number of rows may not exceed the number of physical rows on the screen less one (the window may not overlap the command line at the bottom of the screen), and may not be more than 29.

If values of 1 or 2 are given the editing window will overwrite the header and message areas. Any system messages displayed in an editing window on line 1 are removed at the next screen rewrite, and do not affect the contents of the text buffer.

Column First column on screen.

Permissible values between 1 and a maximum determined by the size of the physical screen and the number of columns set. The sum of the first column value and number of columns may not exceed the number of physical columns on the screen (the window may not extend beyond the limits of of the screen). Any prefix command area is included within the number of columns set.

Rows	<p>Number of rows.</p> <p>Permissible values between 1 and the number of physical rows on the screen less one, with the restriction as described under 'Row' above.</p>
Columns	<p>Number of columns.</p> <p>Permissible values between 1 and the number of physical columns on the screen, with the restriction as described under 'Column' above.</p>
Prefix area	
Length	<p>The length of the prefix area may be set to between 3 and 6 characters. The prefix area overwrites the corresponding area of the editing window without affecting the contents of the text buffer.</p> <p>If a length of 0 is given, the prefix area is eliminated. Prefix commands cannot then be issued until the area is restored by re-setting the length.</p> <p>A minimum length of 4 is recommended for the prefix area, so that the two-letter prefix commands for block manipulation may be conveniently entered in pairs.</p>
Position	<p>The prefix area may be placed either on the right (R) or left (L) side of the editing window.</p>
Type	<p>Two display types are available for the prefix area. If an asterisk (*) is entered, the area is marked with asterisks enclosing the appropriate number of equals signs (e.g. *===*). If N (numeric) is entered, the prefix area is marked by line numbers with an appropriate number of leading zeros (e.g. 00015). There is no functional difference between the two display types.</p>

Current line

Row Row number (relative to first).

The current line (see Section B.2.5) may be placed anywhere within the editing window. A value of 1 places the current line on the first line of the window. The current line may not be placed outside the editing window. If the current line is set, for example, on the first line of the window, it will always be on the first line. That is, if you move the current line down in the window, the text in the window will move in respect to the current line, which remains the first line in the window.

Attrib The current line may be marked either by bright (B) or reverse (R) display. Entering a blank removes the marking, i.e. the current line still exists but is not marked in any way.

Scale line

Row Row number (relative to first).

A scale line may be displayed in the editing window by giving a positive value for the row number. See Section B.4.8 for a description of the scale line.

Zero or a negative value for the scale line position eliminates the scale line.

Menu type

Determines the type of menu to be used in automatic menu applications (see Section B.2.4). The types are:

- 1 The active menu item is marked by a pseudocursor (field attribute), and the cursor remains on the command line.
- 2 Each menu item is preceded by a single-character input field. Menu items may be selected by typing any character in the field. Pressing ENTER activates the first selected item in the menu list, or the item on which the cursor stands if no items are selected.

B.3 Prefix commands

B.3.1 General command principles

Prefix commands are written in the prefix command area and manipulate whole lines in the text buffer. The commands are executed when the ENTER key or any other function key is pressed (but not when RETURN is pressed). Commands are cleared when they are executed. Several prefix commands may be given at the same time. The execution priority is documented below (Section B.3.3).

Prefix commands operate on the text buffer, not just on the editing window. Thus lines in the buffer which are longer than the window are handled as single entities.

Prefix commands consists of either one letter followed by an optional number or a pair of letters. Letters may be either upper- or lower-case. The commands may be given in any position within the prefix command area. The commands are parsed as follows by the editor:

1. Characters in the prefix area which are unchanged since the last screen update are stripped from both the beginning and the end of the area. The remaining characters are taken as the command. Thus if the prefix area is marked by line numbers, the command '0D415' written on line 415 will delete line 415 only, whereas the command '0D4 5' will delete lines 415-418 inclusive. Confusion in this context may be avoided if the prefix command area is set to the *====* display type, or if all numerical qualifiers are followed by a space.
2. The remaining command string is read from right to left. The first valid command is executed: any other characters are ignored. Thus the command 'MC' will perform a move operation, not a copy.
3. Any characters other than those defined as prefix commands are ignored.

B.3.2 The prefix commands

- Delete** Single lines may be deleted with the command "D".
- Several consecutive lines may be deleted by entering the command "Dn", where "n" is the number of lines to be deleted, on the first line of the group. Alternatively, a block of lines to be deleted can be marked with "DD" on the first and last lines.
- Insert** Empty lines may be inserted with the command "I" for one line or "In" for n consecutive lines (up to 20). The lines are inserted immediately after the command. If a number larger than 20 is entered, only 20 lines are created.
- Move** Lines are marked for moving with the "M" command for one line or "Mn" for n consecutive lines. Block moves may also be requested by marking the first and last lines of the block with "MM".
- The target is marked "A" (after) or "B" (before). When the SEND key is pressed, the line or lines marked for moving are transferred to immediately after a target marked "A" or before a target marked "B". Several targets may be marked for the same move operation: one copy of the moved line or block is then inserted at each target.
- Note: if a target is set within a block to be moved, the result is unpredictable.
- Copy** Lines may be copied to a specified target in the same way as they are moved, replacing the "M" or "MM...MM" command with "C" (or "CC...CC" for a block).
- The target for copy operations is handled in the same way as that for move operations.
- Repeat** A single line may be copied to the position immediately below with the repeat command, issued by marking the line with R (for one copy) or Rn (for n copies). There is no block repeat command.

Set current Placing a slash (/) in the prefix command area causes the current line to be set to that position the next time a PF-key is pressed.

The set current command can only be issued within the editing window.

If several lines are marked for the set current command, the command closest to the bottom of the editing window is executed and the others are ignored.

A set current command placed inside a block to be moved is ignored. A set current command placed inside a block to be copied is valid for the original, not for the copied line.

Label Lines may be labelled in the prefix command area by entering a period (.) immediately followed by a label. The label may be as long as the prefix command area permits. Labels are an aid in locating specified lines in the text buffer.

A label is not a true command, and is not annulled by pressing a PF-key, in contrast to all other prefix commands. Labels remain as placed until they are specifically overwritten, or the labelled lines are moved or deleted, or the editor session is terminated. Labels are lost from lines that are moved, and also from the copy, but not the original, of lines that are copied.

The cursor will move to the command line if the prefix command affects the line that the cursor is located on. For example, when the Delete command is used, the cursor will move to the command line if it is located on the line or lines being removed.

Summary of prefix commands:

D[n]	Delete 1 or n lines
DD...DD	Delete marked block
I[n]	Insert 1 or n lines
M[n]	Move 1 or n lines
MM...MM	Move marked block
C[n]	Copy 1 or n lines
CC...CC	Copy marked block
A B	Mark target line after before
R[n]	Repeat line 1 or n times
/	Set current line
.P	Set label '.P'

B.3.3 Command priorities

When simultaneous execution of several prefix commands is requested, the commands are executed in the following order:

1. Label
2. Delete
3. Insert
4. Move
5. Copy
6. Repeat
7. Set current (with restrictions as described above)

B.4 Command line commands

B.4.1 General command principles

A number of commands may be entered on the command line (the bottom line on the screen). These commands cover scrolling, help functions, exiting, and several line and string manipulation functions. Commands on the command line are executed whenever ENTER or any function key is pressed. On asynchronous terminals only, pressing RETURN when the cursor is on the command line also initiates command execution. Commands on the command line are executed after any prefix commands which may have been entered since the last execution.

Any command line command may be assigned to any one of 14 function keys (see B.4.9). Pressing a function key executes first any pending prefix commands and then the command associated with the key. Avoid using function keys if there is a command on the command line: the two command strings are concatenated (with a separating blank), and the result may be unpredictable.

In most implementations of the editor there are environment-specific commands available on the command line. These commands are treated in the same way as any other command line command, and may be assigned to function keys in the same way. Commands specific to a given implementation of the editor are described in the documentation of the module concerned.

B.4.2 Command syntax

All commands on the command line may be abbreviated to a certain unambiguous minimum length. This is indicated in the descriptions below (and in the on-line help functions) by writing the minimum requirement in upper-case and the rest of the command in lower-case. For example, the command 'LEft' may be abbreviated to 'LE' but not to 'L'. The case used when the commands are entered at the terminal is irrelevant. Any characters in the command following the minimum requirement are in fact ignored; therefore the wrongly typed command 'RIGTH' will still move to the right.

In the descriptions below, pointed brackets (<>) are used to indicate optional parts of the command. They are not to be written when the command is entered on the command line.

B.4.3 Repeating commands

Normally, the command field on the command line is blanked out after every execution. Commands may be repeated by prefixing the command with an ampersand (&), which results in retention of the command on the command line after execution. This is most useful for functions such as search-replace: it is meaningless in connection with certain commands such as exit.

B.4.4 Moving the window

The editor window may be moved up, down, left and right in relation to the text buffer. Moving the window 5 characters to the right brings the next 5 characters to the right of the present screen position into view.

Note: We deliberately do not use the word 'scrolling', to avoid any ambiguity in the direction associated with window movement (does 'window movement up' move the text up and display text farther down, or move the window up and display text farther up?) Directional command words in the MIMER Editor give the direction in which you want to look in the text buffer - down for text below the last line of the screen, right for text off the right-hand edge, and so on.

In general, the window moves the requested distance as long as this does not exceed the limits of the text buffer. For movement left and right, the left and right edges of the screen are taken as reference points respectively, and the longest line in the text buffer determines the width of the buffer. Thus if the longest line is 100 characters and the screen width is 80, the maximum movement right or left is 20 positions regardless of how much movement is requested.

Moving up and down moves the current line in relation to the text buffer, and moves the screen window as well if necessary. (The current line may be regarded as a marked position in the screen window). The position of the current line on the screen is determined by the window settings (see Section B.2.5). At the top of the text buffer, the current line moves up from its preset position when up is requested, rather than moving the whole screen window up and leaving blank lines on the screen above the beginning of the text buffer. (This behaviour is best appreciated if the current line position is set near the bottom of the screen - say line 20. Moving up within the first 20 lines does not then change the displayed text).

Note: Some users may be accustomed to editors where pressing the down arrow when the cursor is on the bottom line moves the window down one line. In the MIMER editor, the arrows move the cursor only within the limits of the screen window. Pressing the down arrow when the cursor is on the bottom line moves the cursor to the top line on the screen.

The commands for moving the window are as follows (a vertical bar (|) is used to separate different alternatives for the function - it is not a part of the command):

- UP <n>** Move upwards n lines. If n is omitted, a value of 1 is used.
- Down <n>** Move downwards n lines. If n is omitted, a value of 1 is used. Pressing the ENTER key also moves down one line if the cursor is positioned on the command line and the command field is empty.
- Backward <n>** Move upwards n pages (one page is one screen).
- Forward <n>** Move downwards n pages (one page is one screen).
- Top** Move to top of text buffer.
- Bottom** Move to bottom of text buffer.
- LEft <n>** Move left n positions.
- RIght <n>** Move right n positions.
- pos n** Move to line number n.
- EOL** Move the cursor to just after the end of the text line (i.e. to the first empty screen position to the right of the last non-blank character on the line). If this command is entered on the command line, the cursor must be moved with the arrow keys to the desired line before the command is activated.

Note: EOL only moves the cursor, not the window. If the text line extends outside the right hand edge of the screen, the command EOL will move the cursor to the rightmost column on the screen, not to the end of the text line.

B.4.5 Line manipulation

Four commands are available for line manipulation from the command line. The Insert and DElete commands are functionally equivalent to the corresponding prefix commands (see Section B.3).

Insert <n> Insert n lines after the current line. The cursor is placed at the beginning of the first inserted line.

DElete <n|:l|*> Delete n lines from the current line and forwards, or delete forwards to line number l, or delete forwards to the end of the buffer.

Split Split a line into two. The position of the split is determined by the argument following the Split command. The second part of a split line is placed so that the first non-blank character on the new line lies under the first non-blank character on the original line.

No argument Split before the last word on the current line. Words are separated by spaces.

n Split at position n on the current line. See Section B.4.8 for scale line as an aid in counting position.

C Split at the cursor. For this command, the cursor must be moved to the desired position before the Split C command is activated. The split position does not have to be on the current line.

<A|B>/s/ Split after (A) or before (B) the first occurrence of the string s in the current line. The slashes delimiting the string s may be replaced by any pair of non-blank characters except ampersand not contained in the string itself. If the control letter A or B is omitted, A is assumed. The delimiters may be omitted if the control letter is omitted and the string s does not contain any spaces.

Join Join the current line with the line below. The position of the join is determined by the argument following the Join command. Any leading blanks on the second line are eliminated in the join.

No argument Join at the end of the current line.

n Join at position n on the current line. If the position n is beyond the end of the current line, intervening space is filled with blanks. If the position n is before the end of the current line, the current line from the position n to the end is overwritten.

C Join at the cursor. For this command, the cursor must be moved to the desired position before the Join C command is activated. The line following that on which the cursor is placed (which does not have to be the current line) is appended at the cursor position. If the cursor is placed beyond the end of the line, intervening space is filled with blanks. If the cursor is placed before the end of the line, any text to the right of the cursor is overwritten.

B.4.6 String manipulation

String manipulation commands allow search and replace functions in the text buffer. All commands operate in a forward direction from the current line. Search functions are not sensitive to the case of letters (i.e. searching for 'a' will find both 'a' and 'A').

LOcate /s/|.label

Find a specified string or label. If the operation is successful, the line found becomes the current line and the cursor is placed at the beginning of the string in question or at the beginning of the labelled line.

A string to be located is enclosed in a pair of delimiters, which may be any non-alphanumeric character except ampersand not contained in the string itself. String delimiters may be omitted if the string contains only alphanumeric characters and does not contain any blanks.

If both the keyword 'LOcate' and string delimiters are omitted, care must be taken that the first characters in the string do not coincide with the shortest form of any recognized command. For instance, entering the string 'delicious' without delimiters will DELEte the current line instead of searching for the word 'delicious'.

Change /<s1>/<s2>/ <n|:|*>

Change string s1 to string s2.

The string delimiter (slash in the syntax here) may be replaced by any non-alphanumeric character except ampersand which is not contained in either s1 or s2.

If string s2 is omitted (with syntax /s1//), string s1 is deleted (replaced by nothing).

If string s1 is omitted (with syntax //s2/), string s2 is inserted at the beginning of the current line.

If none of the repetition arguments n, :l or * is given, the change is made once.

The Change command followed by an argument n changes n occurrences of s1 to s2 from the beginning of the current line and forwards. The command followed by argument :l changes all occurrences forwards to but not including line number l. The command followed by an asterisk changes all occurrences from the current line to the end of the buffer. In all cases, the cursor and current line are moved to the position of the last change made.

Note that multiple Change commands requested with an argument do not allow user intervention during the operation. Contrast the repeat formulation &Change, which may be used together with UNdo to change selected occurrences of string s1 to s2.

FIX /<s1>/<s2>/ <n|:|*>

The FIX command operates in the same way as the Change command except that the first occurrence of s1 on the line is changed to s2 regardless of the cursor position. Multiple FIX commands change the first occurrence on each line, not multiple occurrences on a single line.

UNdo Reverse the most recent Change or FIX operation in a repeated sequence requested with &Change or &FIX. Note that UNdo has no effect on

- a) changes requested without the ampersand prefix
- b) changes requested with a repetition argument, regardless of the ampersand prefix (e.g. '&FIX /a/b/ 10').

The UNdo command must be issued immediately after the change to be reversed, with no intervening function key. Only the last single change may be reversed.

The UNdo command is used for controlled search-replace operations. Enter the required &Change or &FIX command. Each time ENTER is pressed, the next replacement is performed. Enter UNdo (suitably assigned to a function key, see Section B.4.9) to reverse any changes not required. The &Change or &FIX command remains on the command line for continued execution.

RESet Reverse any changes made in the text buffer since the last command execution (ENTER, function key, or command line command) and place the cursor on the command line.

The RESet command is most useful for terminating a repeated command sequence requested with the ampersand prefix. For instance, the &Change command leaves the cursor on the replaced string after each operation. The cursor may be rapidly relocated to the command line with RESet, provided that the RESet command is assigned to a function key (see Section B.4.9).

EXIT|QUIT Save and leave the current text buffer.

B.4.7 Screen window setup

The screen window parameters are set by entering the command 'Wdw' on the command line. See B.2.5 for a description of these parameters.

B.4.8 The scale line

The scale line is an aid in determining positions within a line (particularly useful for the Split *n* and Join *n* commands, see Section B.4.5). A scale line is displayed in a chosen position by setting a value greater than zero and less than the size of the window in the 'Scale line' field of the screen parameters (command WDW, see Section B.2.5). For instance, the following screen has a scale line on row 10:

```

----- MIMER Editor -----
*** TOP OF FILE ***
*****

00001

00002

00003
Summary of prefix commands:
00004

00005          D[n]          Delete 1 or n lines
00006          DD...DD      Delete marked block
00007          I[n]          Insert 1 or n lines
00008          .....1.....2.....3.....4.....5.....6.....7.....
          M[n]          Move 1 or n lines
00009          MM...MM      Move marked block
00010          C[n]          Copy 1 or n lines
00011          CC...CC      Copy marked block
00012          A|B          Mark target line after|before
00013          R[n]          Repeat line 1 or n times
00014          /            Set current line
00015          .P           Set label '.P'
00016

00017
*** END OF FILE ***
*****
(PF2=Help) Command ==>

```

The scale line does not 'cover' a line in the text buffer. Instead, the editing window is divided into two parts, one above and one below the scale line (see the line numbers in the prefix area in the example above). Window movementing a divided display is slower than window movementing a display with an undivided editing window, since the two parts of the display are treated separately. If a scale line is required in situations where window movementing is to be performed, placing the scale line on the first or last lines of the screen gives the best results.

B.4.9 Function key (PFK) assignments

A number of function keys may be programmed at any time by the user. The actual number may depend on the terminal. Any command line command (but not prefix commands) may be assigned to any function key. Pressing the function key is then equivalent to entering the command on the command line and pressing ENTER. Commands assigned to function keys may be preceded by an ampersand, in which case the command repeats as described in Section B.4.3.

The function keys are programmed by entering the command 'PFk' on the command line. This displays the current settings of the function keys: new assignments are simply written in on the display.

Example display:

```

----- MIMER Editor - Function Key Assignments -----

      KP1 ==> UP 1
      KP2 ==> DOWN 15
      KP3 ==> EOL
      KP4 ==> LEFT
      KP5 ==> UNDO
      KP6 ==> RIGHT
      KP7 ==> SPLIT C
      KP8 ==> UP 15
      KP9 ==> JOIN C
      KP0 ==> DOWN 1
      PF1 ==>
      PF2 ==> HELP
      PF3 ==> RESET
      PF4 ==> EXIT

(PF2=Help) - Change settings as required, then press ENTER

```

The names shown for the function keys are terminal-dependent (e.g. for Digital Equipment terminals, KP0-KP9 are the numerical keypad, PF1-PF4 are the keypad function keys, and the F series are the extra function keys on VT200-series terminals. For IBM terminals, F1-F24 are used).

Press ENTER to display further function keys.

If text is entered on the command line and a function key is then pressed, the command line text is appended to the command assigned to the function key (with an intervening blank) before the command is executed. For example, if CHANGE is assigned to PF5, then writing the string /OLD/NEW/ on the command line and pressing PF5 will execute the command CHANGE /OLD/NEW/.

Most command assignments to function keys are made for the user's convenience. It is however recommended that certain commands are *always* assigned to a function key, since their use on the command line is limited or awkward. These commands are as follows:

RESET	to interrupt repeated command sequences and return the cursor to the command line
UNDO	to reverse single instances in a repeated search/replace operation without moving to the command line
SPLIT C	to split a line at the cursor without moving to the command line and then back to the split position
JOIN C	to join a line at the cursor without moving to the command line and then back to the join position.
EOL	This command is useful only when assigned to a function key, since if the command is written on the command line, the cursor must be moved manually to the relevant line before the command is activated.

C EXAMPLE DATABASE

A simple example database is used throughout this manual to illustrate the use of ISQL and BSQL. It is based upon an imaginary company that owns a chain of hotels.

The database consists of two databanks, BOOKDB and ROOMSDB. The BOOKDB databank contains information needed when booking guests: information on guests, available rooms and room status. The ROOMSDB databank contains information on the hotels: hotel locations, room types, number of rooms per hotel, prices, etc.

All tables in the example database are created by the ident BOOKADM.

This example database is provided with the MIMER/SQL installation so that you may try out the examples yourself (if you do not have the example database, ask your MIMER system administrator to generate it). The tables shown here provide an easy reference for the examples in the manual. The statements used to create this database are also shown in this appendix.

C.1 Tables in the example database

Tables in the example database are described in this section.

The table descriptions are set up as follows:

- The first column lists the table name and the column names.
- The second column shows which columns which make up the primary key (*).
- The third column shows the columns that are foreign keys (*f*). Refer to the CREATE statements later in this section for a full definition of foreign keys in the database.
- The fourth column shows the column data type. CHAR(*n*) is a character string of length *n* bytes. INT(*p*) specifies an integer of up to *p* digits long. DEC(*p,s*) specifies numbers of up to *p* digits long, of which *s* follow the decimal point. DATE is a date in the Gregorian calendar in the form YYYY-MM-DD. TIME(*s*) is a time on an unspecified day, in the form HH:MM:SS, with *s* digits following the decimal point in the seconds value.
- The fifth column explains the column contents.

C.2 Table descriptions

HOTEL				
HOTELCODE	*		CHAR(4)	Hotel identity code
NAME			CHAR(15)	Hotel name
CITY			CHAR(15)	Location
OVERBOOK			DEC(3,2)	Booking status

ROOMTYPES				
ROOMTYPE	*		CHAR(4)	Room type
DESCRIPTION			CHAR(25)	Room description

ROOMS				
ROOMNO	*		CHAR(7)	Room number
HOTELCODE		<i>f</i>	CHAR(4)	Hotel identity code
ROOMTYPE		<i>f</i>	CHAR(4)	Room type

ROOM_PRICES				
HOTELCODE	*	<i>f</i>	CHAR(4)	Hotel identity code
ROOMTYPE	*	<i>f</i>	CHAR(4)	Room type
FROM_DATE	*		DATE	Date when price is valid
PRICE			INT(4)	Cost of room per day

CHARGES				
CHARGE_CODE	*		CHAR(3)	Charge code
DESCRIPTION			CHAR(25)	Cost description

FREEROOMS				
HOTELCODE	*	<i>f</i>	CHAR(4)	Hotel identity code
ROOMTYPE	*	<i>f</i>	CHAR(4)	Room type
ON_DATE	*		DATE	Data valid for this date
FREECOUNT			INT(3)	Number of bookable rooms

ROOMSTATUS				
ROOMNO	*	<i>f</i>	CHAR(7)	Room number
STATUS			CHAR(10)	Room status

BOOK_GUEST				
RESERVATION	*		INT(5)	Guest reference number
BOOKING_DATE			DATE	Date of booking
HOTELCODE		<i>f</i>	CHAR(4)	Hotel identity code
ROOMTYPE		<i>f</i>	CHAR(4)	Room type
RESERVED_BY			CHAR(25)	Name of reservation maker
TELEPHONE			CHAR(15)	Telephone number of above person
RESERVED_FOR			CHAR(25)	Name of expected guest
ARRIVE			DATE	Expected check-in date
DEPART			DATE	Expected check-out date
GUEST			CHAR(25)	Guest name
ADDRESS			CHAR(30)	Guest address
CHECKIN			DATE	Actual check-in date
CHECKOUT			DATE	Actual check-out date
ROOMNO		<i>f</i>	CHAR(7)	Room number
PAYMENT			CHAR(10)	Payment type

BILL				
RESERVATION	*	<i>f</i>	INT(5)	Guest reference number
ON_DATE	*		DATE	Billing date
CHARGE_CODE	*	<i>f</i>	CHAR(3)	Charge code
AMOUNT			DEC(8,2)	Price

EXCHANGE_RATE				
CURRENCY	*		CHAR(3)	Currency
RATE			DEC(6,3)	Exchange rate

WAKE_UP				
ROOMNO	*	<i>f</i>	CHAR(7)	Room number
WAKE_DATE	*		DATE	Wake up date
WAKE_TIME			TIME	Wake up time

C.3 The tables

This section illustrates the contents of the tables in the example database. Only partial data is shown for some tables.

HOTEL			
HOTELCODE	NAME	CITY	OVERBOOK
LAP	LAPONIA	STOCKHOLM	1.10
SKY	SKYLINE	UPPSALA	1.10
STG	ST. GEORGE	STOCKHOLM	1.10
WIND	WINSTON	GOTHENBURG	1.10
WINS	WINSTON	COPENHAGEN	1.10

ROOMTYPES	
ROOMTYPE	DESCRIPTION
DBLB	DOUBLE WITH BATH
DBLS	DOUBLE WITH SHOWER
SGLB	SINGLE WITH BATH
SGLS	SINGLE WITH SHOWER

ROOMS		
ROOMNO	HOTELCODE	ROOMTYPE
LAP110	LAP	SGLS
LAP211	LAP	DBLB
LAP309	LAP	SGLS
...
SKY117	SKY	SGLS
SKY121	SKY	DBLS
...
STG111	STG	DBLS
STG114	STG	DBLB
...
WIND308	WIND	DBLS
WIND524	WIND	DBLB
...
WINS108	WINS	DBLB
WINS109	WINS	SGLB
WINS116	WINS	DBLB

ROOM_PRICES			
HOTELCODE	ROOMTYPE	FROM_DATE	PRICE
LAP	SGLS	1996-06-01	380
LAP	SGLS	1996-10-01	340
...
SKY	SGLB	1996-06-01	370
SKY	DBLS	1996-10-01	550
...
STG	SGLB	1996-06-01	400
STG	SGLB	1996-10-01	360
STG	DBLS	1996-10-01	510
...
WIND	SGLS	1996-06-01	370
...
WINS	DBLB	1996-06-01	570

CHARGES	
CHARGE_CODE	DESCRIPTION
100	LODGING
120	TELEPHONE
170	CAR PARK
200	RESTAURANT
210	MINIBAR
230	BAR
270	ROOM SERVICE
330	LAUNDRY
720	EXTRA BED
900	MISCELLANEOUS

FREEROOMS			
HOTELCODE	ROOMTYPE	ON_DATE	FREECOUNT
LAP	DBLB	1996-09-24	2
LAP	DBLB	1996-09-25	2
...
SKY	DBLS	1996-10-05	3
SKY	SGLB	1996-08-13	7
SKY	SGLB	1996-09-02	7
...
STG	DBLB	1996-10-17	4
STG	DBLS	1996-09-01	4
...
WIND	DBLS	1996-10-04	3
WIND	DBLS	1996-10-10	5
WIND	SGLS	1996-08-15	0
...
WINS	SGLS	1996-10-24	-1

ROOMSTATUS	
ROOMNO	STATUS
LAP205	KEY OUT
LAP206	KEY OUT
...	...
...	...
SKY125	KEY OUT
SKY204	MAINT
...	...
...	...
WINS103	KEY OUT

BOOK_GUEST					
RESERVATION	BOOKING_DATE	HOTELCODE	ROOMTYPE	RESERVED_BY	
1348	1996-08-05	LAP	SGLB	MIMER AB	
1349	1996-08-05	LAP	SGLS	MIMER AB	
1350	1996-08-06	SKY	DBLB	SALLY WEBERT	
1351	1996-08-06	SKY	DBLB	SALLY WEBERT	
1352	1996-08-06	WINS	DBLB	MARK FRANCIS	
1353	1996-08-06	SKY	SGLB	ASATRON AB	
...	
...	

TELEPHONE	RESERVED_FOR	ARRIVE	DEPART	GUEST
018-185210	STEN JOHANSEN	1996-08-23	1996-08-24	STEN JOHANSEN
018-185210	MATS LINDBLOM	1996-08-23	1996-08-24	STEFAN HANSEN
0760-57609	SALLY WEBERT	1996-08-06	1996-08-10	SALLY WEBERT
0760-57609	JOHN ALBERTSON	1996-08-06	1996-08-10	ANNA ALBERTSON
08-320668	MARK FRANCIS	1996-08-14	1996-08-15	MARK FRANCIS
08-135709	BASIL FAWCETT	1996-09-02	1996-09-09	ALFRED FIMPLEY
...
...

ADDRESS	CHECKIN	CHECKOUT	ROOMNO	PAYMENT
MIMERGATAN 4, UPPSALA	1996-08-23	1996-08-24	LAP205	EUROCARD
IDUNGATAN 24, UPPSALA	1996-08-23	1996-08-24	LAP206	EUROCARD
KRONPARKEN 44, JOKKMOKK	1996-08-06	1996-08-09	SKY124	CASH
32 SPRING DRIVE, DENVER, USA	1996-08-06	1996-08-10	SKY125	AM.EXPR
VIMPELGATAN 7, SKARA	1996-08-14	1996-08-15	WINS103	EUROCARD
23 BACK NELLY VIEW, ACKWORTH	1996-09-03	1996-09-08	SKY110	CASH
...
...

BILL			
RESERVATION	ON_DATE	CHARGE_CODE	AMOUNT
1347	1996-09-04	100	380.00
1347	1996-09-04	120	12.70
1347	1996-09-04	210	18.00
1347	1996-09-05	100	380.00
1347	1996-09-05	120	32.50
1348	1996-08-23	100	400.00
1348	1996-08-23	120	12.00
1348	1996-08-23	200	112.50
1348	1996-08-23	230	55.00
1349	1996-08-23	100	380.00
1349	1996-08-23	170	25.00
1349	1996-08-23	900	12.50
1350	1996-08-06	100	580.00
1350	1996-08-06	210	60.00
1350	1996-08-07	100	580.00
1350	1996-08-07	200	265.00
1350	1996-08-07	230	175.00
1350	1996-08-07	330	120.00
1350	1996-08-08	100	580.00
1350	1996-08-08	120	30.00
1350	1996-08-08	270	85.00
...

WAKE_UP		
ROOMNO	WAKE_DATE	WAKE_TIME
LAP112	1996-09-10	06:00:00
LAP112	1996-09-11	07:00:00
LAP201	1996-09-10	06:45:00
LAP205	1996-09-10	08:00:00
SKY101	1996-09-10	09:00:00
SKY110	1996-09-10	07:30:00
SKY111	1996-09-10	06:00:00
SKY124	1996-09-10	06:15:00
SKY124	1996-09-11	06:15:00
SKY124	1996-09-12	06:15:00
SKY201	1996-09-10	10:00:00
SKY212	1996-09-10	04:30:00
STG009	1996-09-10	06:00:00
STG117	1996-09-10	07:00:00
STG142	1996-09-10	08:30:00
WIND401	1996-09-10	06:00:00
WIND402	1996-09-10	06:20:00
WIND514	1996-09-10	07:00:00
WINS119	1996-09-10	08:00:00
WINS120	1996-09-10	07:30:00
WINS121	1996-09-10	06:20:00

EXCHANGE_RATE	
CURRENCY	RATE
DEM	3.476
DKK	0.922
FIM	1.435
FRF	1.041
GBP	10.335
ITL	0.005
JPY	0.044
NOK	0.939
SEK	1.00
USD	6.335

C.4 CREATE statements for example database

The following statements were used to create the tables in the example database. Only the CREATE statements are listed here.

```
CREATE DATABANK BOOKDB
  OF 10 PAGES
  IN 'BOOKDB'
  WITH TRANS OPTION;
```

```
CREATE DATABANK ROOMSDB
  OF 10 PAGES
  IN 'ROOMSDB'
  WITH TRANS OPTION;
```

```
CREATE DOMAIN HOTELCODE
  AS CHAR(4);
```

```
CREATE DOMAIN ROOMTYPE
  AS CHAR(4)
  DEFAULT '-ND-';
```

```
CREATE DOMAIN ROOMNO
  AS CHAR(7);
```

```
CREATE DOMAIN PERSONNAME
  AS CHAR(25);
```

```
CREATE DOMAIN STATUS
  AS CHAR(10)
  DEFAULT 'UNKNOWN';
```

```
CREATE DOMAIN NUMBER
  AS INT(3)
  DEFAULT 0;
```

```
CREATE DOMAIN BOOK_RATE
  AS DEC(3,2)
  DEFAULT 1.10;
```

```

CREATE TABLE HOTEL (HOTELCODE HOTELCODE,
                    NAME          CHAR(15) NOT NULL,
                    CITY          CHAR(15) NOT NULL,
                    OVERBOOK     BOOK_RATE NOT NULL,
                    PRIMARY KEY (HOTELCODE))
IN ROOMSDB;

CREATE TABLE ROOMTYPES (ROOMTYPE ROOMTYPE,
                        DESCRIPTION CHAR(25) NOT NULL,
                        PRIMARY KEY (ROOMTYPE))
IN ROOMSDB;

CREATE TABLE ROOMS (ROOMNO ROOMNO,
                   HOTELCODE HOTELCODE NOT NULL,
                   ROOMTYPE ROOMTYPE NOT NULL,
                   PRIMARY KEY (ROOMNO),
                   FOREIGN KEY (HOTELCODE) REFERENCES HOTEL,
                   FOREIGN KEY (ROOMTYPE) REFERENCES ROOMTYPES)
IN ROOMSDB;

CREATE TABLE ROOM_PRICES (HOTELCODE HOTELCODE,
                          ROOMTYPE ROOMTYPE,
                          FROM_DATE DATE,
                          PRICE INT(4),
                          PRIMARY KEY (HOTELCODE,ROOMTYPE,FROM_DATE),
                          FOREIGN KEY (HOTELCODE) REFERENCES HOTEL,
                          FOREIGN KEY (ROOMTYPE) REFERENCES ROOMTYPES)
IN ROOMSDB;

CREATE TABLE CHARGES (CHARGE_CODE CHAR(3),
                      DESCRIPTION CHAR(25) NOT NULL,
                      PRIMARY KEY (CHARGE_CODE))
IN ROOMSDB;

CREATE TABLE FREEROOMS (HOTELCODE HOTELCODE,
                       ROOMTYPE ROOMTYPE,
                       ON_DATE DATE,
                       FREECOUNT NUMBER NOT NULL,
                       PRIMARY KEY (HOTELCODE,ROOMTYPE,ON_DATE),
                       FOREIGN KEY (HOTELCODE) REFERENCES HOTEL,
                       FOREIGN KEY (ROOMTYPE) REFERENCES ROOMTYPES)
IN BOOKDB;

CREATE TABLE ROOMSTATUS (ROOMNO ROOMNO,
                        STATUS STATUS NOT NULL,
                        PRIMARY KEY (ROOMNO),
                        FOREIGN KEY (ROOMNO) REFERENCES ROOMS)
IN BOOKDB;

```

```

CREATE TABLE BOOK_GUEST (RESERVATION    INTEGER(5),
                          BOOKING_DATE  DATE          NOT NULL,
                          HOTELCODE     HOTELCODE    NOT NULL,
                          ROOMTYPE     ROOMTYPE     NOT NULL,
                          RESERVED_BY   PERSONNAME   NOT NULL,
                          TELEPHONE     CHAR(15),
                          RESERVED_FOR  PERSONNAME,
                          ARRIVE        DATE          NOT NULL,
                          DEPART        DATE          NOT NULL,
                          GUEST         PERSONNAME,
                          ADDRESS       CHAR(30),
                          CHECKIN       DATE,
                          CHECKOUT      DATE,
                          ROOMNO        ROOMNO,
                          PAYMENT       CHAR(10),
                          PRIMARY KEY (RESERVATION),
                          FOREIGN KEY (HOTELCODE) REFERENCES HOTEL,
                          FOREIGN KEY (ROOMTYPE) REFERENCES ROOMTYPES,
                          FOREIGN KEY (ROOMNO) REFERENCES ROOMS,
                          CHECK (ARRIVE < DEPART AND CHECKIN <= CHECKOUT))
IN BOOKDB;

```

```

CREATE TABLE BILL (RESERVATION    INT(5),
                   ON_DATE        DATE,
                   CHARGE_CODE    CHAR(3),
                   AMOUNT         DEC(8,2) NOT NULL,
                   PRIMARY KEY (RESERVATION,ON_DATE,CHARGE_CODE),
                   FOREIGN KEY (RESERVATION) REFERENCES BOOK_GUEST,
                   FOREIGN KEY (CHARGE_CODE) REFERENCES CHARGES)
IN BOOKDB;

```

```

CREATE TABLE EXCHANGE_RATE (CURRENCY CHAR(3),
                             RATE     DECIMAL(6,3),
                             PRIMARY KEY (CURRENCY))
IN BOOKDB;

```

```

CREATE TABLE WAKE_UP (ROOMNO    ROOMNO    NOT NULL,
                      WAKE_DATE  DATE      NOT NULL,
                      WAKE_TIME  TIME      NOT NULL,
                      PRIMARY KEY (ROOMNO,WAKE_DATE),
                      FOREIGN KEY (ROOMNO) REFERENCES ROOMS)
IN BOOKDB;

```

INDEX

A

- access rights 2-10, 8-1
- active connection 3-2
- ALL 4-31
- ALTER DATABANK 7-12
- ALTER IDENT 7-14
- ALTER TABLE 7-13
- ANY 4-31
- arithmetic operations 4-8
- AS
 - for column labels 4-2
 - for connection name 3-1
- AVG 4-10

B

- BACKWARD 10-3
- base tables 2-5
- batch operation 11-1
- BETWEEN condition 4-7
- BOTTOM 10-3
- BSQL 11-1

C

- CANCEL 10-4
- CASCADE 7-15, 8-6
- CASE 4-16
- CAST 4-17
- changing connections 3-2
- changing passwords 7-14
- changing table contents 5-1
- CHAR_LENGTH 4-15
- character set 4-5
- character string comparison 4-5
- check clause in domains 7-4
- check conditions 2-9
- check conditions in tables 7-7
- check option in views 2-10, 7-10
- client/server 2-1
- CLOSE
 - BSQL 11-3
- column labels 4-2
- column names in UNION 4-34
- COMMAND 10-5
- command line commands B-10
- comments 7-12
- COMMIT 11-13
- comparison 4-4

- computed values 4-8
- CONNECT 3-1
- connections 3-1
- CONTINUE 10-5, 11-13
- correlation names 4-27
- COUNT 4-10
- creating
 - databanks 7-2
 - domains 7-3
 - idents 7-1
 - secondary indexes 7-10
 - synonyms 7-11
 - tables 7-4
 - views 7-8
- cross product 4-21
- current line 10-12
- D**
- data consistency 6-3
- data dictionary 2-1
- data integrity 2-8
- data types 2-4
- databank 2-2
- databank options 6-2
- DATABANK privilege 8-3
- databank shadows 2-7
- databanks
 - altering 7-12
 - creating 7-2
 - dropping 7-15
- database connections 3-1
- database definition statements 7-1
- database design 7-1
- database name 3-1
- database organization 2-1
- DEFAULT command 10-6
- default database 3-1
- default values in domains 7-3
- DELETE 5-5
- DELETE access 8-4
- DESCRIBE
 - BSQL 11-4
 - ISQL 10-6
- DISCONNECT 3-2
- DISTINCT 4-3
 - in set functions 4-10
- domains 2-8
 - check clause 7-4
 - creating 7-3
 - default values 7-3
 - dropping 7-15
- DOWN 9-3, 10-11
- dropping objects 7-15
- duplicate values 4-3
- E**
- ECHO 11-8
- editor B-1
- editor command syntax B-10

- editor commands B-7
- embedded SQL 1-1
- ENTER 3-3
- error messages 10-24, 13-1
- ESCAPE in LIKE conditions 4-6
- ESQL 1-1
- EXECUTE 10-12
- EXECUTE privilege 8-3
- EXISTS
 - NULL values 4-37
- EXISTS condition 4-30
- EXIT
 - BSQL 11-4, 11-13
 - ISQL 10-12

F

- FORALL 4-31
- foreign key 7-6
- foreign keys 2-8
- FORWARD 10-13
- function keys 9-2, B-18

G

- grant option 2-11, 8-1
- granting privileges 8-3
- GROUP BY 4-12
- group idents 2-3

H

- HAVING 4-13
- HEADER 10-24
- help 9-2
 - BSQL 11-5
 - ISQL 10-13
- host variables 12-1

I

- IDENT privilege 8-3
- idents 2-2
 - altering 7-14
 - creating 7-1
 - dropping 7-16
 - organization 8-2
- IN condition 4-7
- indexes 2-6
- indicator variables 12-1
- INITPAGE 10-25
- INSERT 5-1
- INSERT access 8-4
- inserting NULL values 5-3
- inserting with a subselect 5-3
- Interactive SQL commands 10-1
- invalid statements 10-28
- ISQL 9-1
- ISQL commands 9-2, 10-1
- ISQL editor 9-2

J

- join condition 4-20
- join views 2-5

joining a table with itself 4-28

K

KEY 10-25

keys 2-6

L

LEAVE 3-3

leaving ISQL 9-5

LEFT 9-3, 10-14

LIKE 4-6

LINECOUNT 11-8

LINESPACE 10-25, 11-9

LINEWIDTH 11-9

LIST

 BSQL 11-5

 ISQL 10-14

LOAD

 BSQL 11-6

 ISQL 10-17

LOG 11-9

 BSQL 11-7

LOGDB 2-2, 6-1

logging 6-1

logical operators 4-4

LOWER 4-15

M

MAX 4-10

MEMBER privilege 8-3

menu types B-3

menus 9-4

MENUSTYLE 10-26

merging with UNION 4-34

MESSAGE 11-10

MIMER version 7 1-1

MIN 4-10

N

nested selects 4-25

NEXT 10-20

NULL values 2-4

 in EXISTS 4-37

 in SELECT 4-36

 in set functions 4-10

 in variables 12-1

 inserting 5-3

 treated as equal 4-3

O

object names 2-2

object privileges 2-10

objects 2-1

operating system commands 10-5

optimization 2-6

ORDER BY 4-13

 in subselects 4-27

OS_USER 2-3

outer references 4-29

OUTPUT 11-10

P

PAGELength 10-26, 11-10
PAGENumber 10-26
PAGEWidth 10-27, 11-11
passwords 7-1
pattern conditions 4-6
PERFORM 10-20
PFK 9-2, 10-20, B-18
prefix commands B-7
PREVIOUS 10-21
primary key 2-6, 7-6
PRINT 10-21
private objects 2-2
privileges 2-10, 8-1
PROFILE 10-22
program idents 2-3, 3-3
PUBLIC group ident 8-2

Q

quantified predicates 4-31
QUIT 10-23

R

READ 10-23
READ INPUT 11-7
read-through-write-set 6-3
recursive effects of revoking privileges 8-6
REFERENCES 7-6
REFERENCES access 8-4
referential integrity 2-8, 7-6
REMOVE 10-24
RESTRICT 7-15, 8-6
restriction views 2-5
result table 4-1
retrieving data
 from multiple tables 4-20
 from single tables 4-1
revoking privileges 8-5
RIGHT 9-3, 10-24
ROLLBACK 11-13

S

scalar functions 4-15
scale line B-17
scrolling 9-3
secondary indexes 2-6
 creating 7-10
SELECT
 computed values 4-8
 creating views 7-8
 DISTINCT 4-3
 EXISTS 4-30
 GROUP BY 4-12
 HAVING 4-13
 illegal statements on views 4-3
 NULL values 4-36
 ordering the result 4-13
 quantified predicate 4-31
 simple form 4-1

- UNION 4-34
 - WHERE 4-4
 - SELECT access 8-4
 - selecting groups 4-13
 - selection process 4-39
 - semantic errors 13-2
 - sequential command files 10-20, 10-23, 10-30, 11-7
 - sequential data file 10-29
 - sequential data files 10-17
 - SET
 - CONNECTION 3-2
 - ECHO 11-8
 - HEADER 10-24
 - INITPAGE 10-25
 - KEY 10-25
 - LINECOUNT 11-8
 - LINESPACE 10-25, 11-9
 - LINEWIDTH 11-9
 - LOG 11-9
 - MENUSTYLEE 10-26
 - MESSAGE 11-10
 - OUTPUT 11-10
 - PAGELength 10-26, 11-10
 - PAGENUMBER 10-26
 - PAGEWIDTH 10-27, 11-11
 - TOPLABEL 10-27
 - set conditions 4-7
 - set functions 4-10
 - SET TRANSACTION CHANGES 6-3
 - SETTINGS 11-11
 - shadowing 2-7
 - SHOW SETTINGS 11-11
 - simple joins 4-22
 - SKIP 10-28
 - SOME 4-31
 - source table 4-1
 - SQL statements 2-11, 9-4
 - sqlhosts 3-1
 - SQLstandards 1-1
 - standards 1-1
 - starting ISQL 9-1
 - string concatenation 4-8
 - subselects 4-25
 - in INSERT 5-3
 - SUBSTRING 4-15
 - SUM 4-10
 - synonyms 2-7
 - creating 7-11
 - syntax errors 13- 13-2
 - SYSADM 8-2
 - SYSDB 2-2
 - system databanks 2-2
 - system messages 9-2, 13-1
 - system objects 2-2
 - system privileges 2-10
 - system utilities 8-2
- T**
- TABLE privilege 8-3

- tables 2-3
 - altering 7-13
 - check conditions 7-7
 - column definitions 7-6
 - creating 7-4
 - dropping 7-15
- terminal environment 9-1
- text buffer B-1
- TOGGLE 10-28
- TOP 10-28
- TOPLABEL 10-27
- transactions 6-1
- TRANSDB 2-2
- TRIM 4-15

U

- UNION 4-33
- UNLOAD
 - BSQL 11-12
 - ISQL 10-29
- UP 9-3, 10-30
- updatable views 5-5
- UPDATE 5-4
- UPDATE access 8-4
- UPPER 4-15
- user databanks 2-2
- user idents 2-2

V

- variables 12-1
- version 7 1-1
- views 2-5
 - check option 7-10
 - check options 2-10
 - creating 7-8
 - updatable 5-5

W

- WDW 9-3, 10-30
- WHENEVER
 - BSQL 11-13
- WHERE condition 4-4
- wildcard characters 4-6
- window setup 9-3
- WRITE 10-30