Click here to visit our Web Site

Mimer SQL

Reference Manual

Version 8.2

Copyright © 2000 Mimer Information Technology AB

Mimer SQL version 8.2 Reference Manual
Second revised edition

December, 2000

Copyright © 2000 Mimer Information Technology AB.

Published by Mimer Information Technology AB,
P.O. Box 1713,
SE-751 47 Uppsala, Sweden.
Tel +46(0)18-18 50 00.
Fax +46(0)18-18 51 00.
Internet: http://www.mimer.com

Produced by Mimer Information Technology AB, Uppsala, Sweden.

All rights reserved under international copyright conventions.

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

FOREWORD

This manual provides a full reference description of the Mimer SQL language. It is a complement to the *Mimer SQL User's Manual* and to the *Mimer SQL Programmer's Manual.*

Intended audience

The manual is intended for all users of Mimer SQL in both interactive and embedded contexts.

Related Mimer SQL publications

- **Mimer SQL Programmer's Manual** contains a description of how Mimer SQL can be embedded within application programs, written in conventional programming languages.
- **Mimer SQL User's Manual** contains a description of the BSQL facilities. A user-oriented guide to the SQL statements is also included, which may provide help for less experienced users in formulating statements correctly (particularly the SELECT statement, which can be quite complex).
- **Mimer SQL System Management Handbook** describes system administration functions, including export/import, backup/restore, databank shadowing and the statistics functionality. The information in this manual is used primarily by the system administrator, and is not required by application program developers. The SQL statements which are part of the System Management API are described in the *Mimer SQL Reference Manual*.
- **Mimer SQL platform-specific documents** containing platform-specific information. A set of one or more documents is provided, where required, for each platform on which Mimer SQL is supplied.
- **Mimer SQL Release Notes** contain general and platform-specific information relating to the Mimer SQL release for which they are supplied.

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 following publication:

A Guide to the SQL Standard (Fourth Edition, 1997). ISBN: 0-201-96426-0. This work contains much constructive criticism and discussion of the SQL standard, including SQL99.

For JDBC users:

JDBC information can be found on the internet at the following web addresses: http://java.sun.com/products/jdbc/ and http://www.mimer.com/jdbc/.

For information on specific JDBC methods, please see the online documentation for the java.sql package. This documentation is normally included in the Java development environment.

JDBC™ API Tutorial and Reference, 2nd edition. ISBN: 0-201-43328-1. A useful book published by JavaSoft.

For ODBC users:

Microsoft ODBC 3.0 Programmer's Reference and SDK Guide for Microsoft Windows and Windows NT. ISBN: 1-57231-516-4. This manual contains information about the Microsoft Open Database Connectivity (ODBC) interface, including a complete API reference.

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

ISO/IEC 9075:1999(E) Information technology - Database languages - SQL. This document contains the standard referred to as SQL99.

ISO/IEC 9075:1992(E) Information technology - Database languages - SQL. This document contains the standard referred to as SQL92.

ISO/IEC 9075-4:1996(E) Database Language SQL - Part 4: Persistent Stored Modules (SQL/PSM). This document contains the standard which specifies the syntax and semantics of a database language for managing and using persistent database language routines.

CAE specification, Data Management: Structured Query Language (SQL), Version 2. X/Open document number: C449. ISBN: 1-85912-151-9. This document contains the X/Open-95 SQL specification.

Acronyms, terms and trademarks

IEC	International Electrotechnical Commission
ISO	International Standards Organization
NIST	National Institute of Standards and Technology
SQL	Structured Query Language
X/Open	X/Open is a trademark of the X/Open Company
BSQL	The Mimer facility for using SQL interactively or by running a command file

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

CONTENTS

1 USING THIS MANUAL

1.1 Reading syntax diagrams

The syntax of SQL statements is presented in the form of syntax diagrams, showing how the statements may be written. The diagrams are read from left to right. Valid statements are constructed by following the lines in the diagrams and “picking up” elements of the syntax on the way.

It is not practical to give the full, exhaustive syntax of each SQL statement in a single diagram. Instead, many of the syntax diagrams for statements in Chapter 6 refer to “language elements”, which are themselves expanded into syntax diagrams in Chapter 5. For each syntax diagram, references are given to where in the manual the expansion of the language elements may be found.

A sample diagram illustrating most of the features of the syntax diagrams is given at the end of this chapter, together with some valid and some invalid formulations.

Key to syntax diagrams

```
─── KEYWORD-1 ───
```

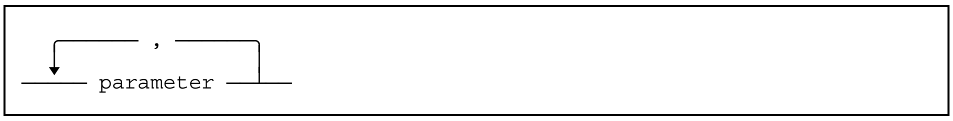
A word bounded by diagram lines must be separated from adjoining words by at least one separator. A separator is represented by a white-space character (ASCII HEX-values 09 to 0D, or 20, i.e. <TAB>, <LF>, <VT>, <FF>, <CR> or <SP>).

```
─── KEYWORD-2 string ───
```

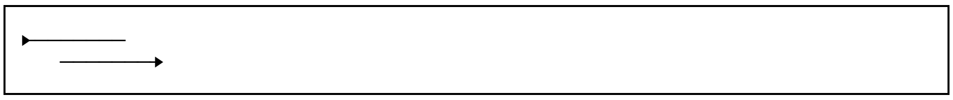
Words separated from each other by at least one space in the syntax diagram must also be separated from each other by at least one separator in the statement. Where the descriptive names for identifiers used in the diagrams consist of more than one word, these are bound together by hyphens.

```
────┬── option-1 ──┬────
    └── option-2 ──┘
```

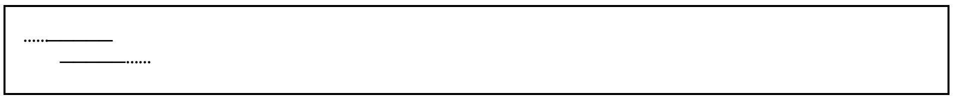
Branched lines indicate alternative constructions. Only one branch may be followed for any one passage along the line: in this example **either** option-1 **or** option-2 may be used, but not both.



This representation is used to show that a section of the syntax construction may be repeated. Any construction required between the repetitions is shown on the “repeat line”. In this example, the statement must contain at least one instance of “parameter”; if several instances are given, they must be separated from each other by a comma. If a comma or other separator is specified in a list, blank spaces need not be used between the elements of the list.



Arrows at the beginning and end of a statement show that the statement is complete.



Dots at the beginning or end of a line in a diagram show that the statement on the line is incomplete. The continuation may be in the same diagram or relate to a separate diagram (as in the language elements from Chapter 5). The dots are not part of the statement syntax.

KEYWORDS Keywords are words that are defined in the SQL language. Keywords are written in UPPERCASE in the diagrams. They must always be written in the statement exactly as shown, except that the case of letters is not significant. Examples of keywords are:

ALTER   CREATE SYNONYM   NULL

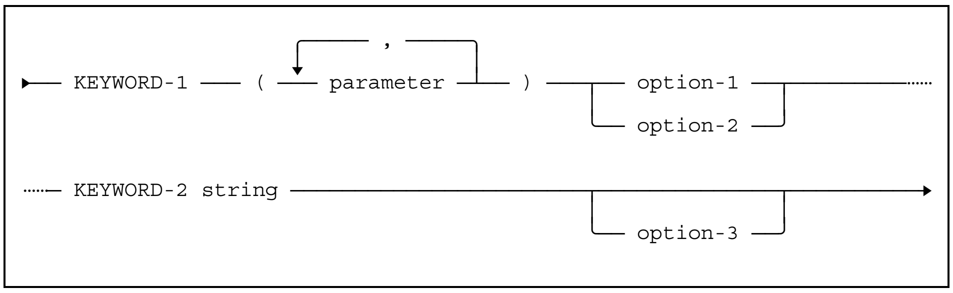
Spaces between keywords are significant. Thus the keywords CREATE SYNONYM in this example must be separated by at least one space.

parameters Parameters are indicated by words in lowercase in the diagrams, and replaced by the appropriate identifiers or constructions when statements are written. Examples of parameters are:

column-name   expression   data-type

The blank spaces in the diagrams are significant. Words bound together by hyphens (e.g. column-name, data-type) represent single parameters.

The following sample illustrates the use of the syntax diagrams.



Some valid formulations are:

```
KEYWORD-1     (parameter) option-1 KEYWORD-2 string


KEYWORD-1     (parameter, parameter) option-1
              KEYWORD-2 string option-3


KEYWORD-1     (parameter, parameter, parameter) option-2
              KEYWORD-2 string
```

The following formulations are **not** valid:

```
KEYWORD-1     (parameter) KEYWORD-2 string
```

-- ‘option-1’ or ‘option-2’ missing

```
KEYWORD-1     parameter  option-1 KEYWORD-2 string
```

-- parentheses missing

```
KEYWORD-1     (parameter,) parameter option-2
              KEYWORD-2 string
```

-- closing parentheses wrongly placed

```
KEYWORD-1     (parameter, parameter)
              option-1KEYWORD-2 stringoption-3
```

-- separating blanks missing

```
KEYWORD-1     (parameter parameter parameter) option-2
              KEYWORD-2 string
```

-- no commas in parameter list

1.2 Reading standard compliance tables

For each language element and statement, the standards compliance is noted in a table, e.g. for GRANT ACCESS PRIVILEGE:

Standard	**Compliance**	**Comments**
X/Open-95 **SQL92**	YES	The use of the keyword PRIVILEGES in conjunction with the keyword ALL is not mandatory in Mimer SQL.

The table lists for each relevant standard:

- The name of the standard
- An indication of how Mimer SQL complies with the standard, i.e. “MIMER EXTENSION” means the entire SQL statement or concept is something specific to Mimer SQL and is not included in any standard; “EXTENDED” means that the SQL statement or concept is included in the named standard(s) and Mimer SQL implements extra features over and above those described in the standard(s); “YES” indicates that the SQL statement or concept is implemented in Mimer SQL in accordance with named standard(s)
- Comments that clarify, where required, details of Mimer SQL compliance with the standard. Typically used to describe extended Mimer SQL features or any applicable differences between the standard and the Mimer SQL implementation.

In writing standard-compliant SQL application programs, make sure that only features included in the standard are used where Mimer SQL offers extended features, and that features which are labeled “MIMER EXTENSION” are avoided as far as possible. When Mimer SQL extended features must be used, try to isolate the code containing these features and use comments to identify the non-standard statements. In this way, porting of application code to other standard environments is simplified.

2 INTRODUCTION TO SQL STANDARDS

This chapter compares important SQL standards.

2.1 Overview

SQL applications can be written in a vendor independent manner by using only those parts of SQL which are defined in one or more standards. This makes it possible to move applications between different database systems without the need to rewrite a substantial amount of code.

Mimer's policy is to develop Mimer SQL as far as possible in accordance with the established standards. This enables users to switch to and from Mimer SQL easily.

The following SQL standards are discussed in this manual:

- X/Open SQL 1995 (referred to here as X/Open-95)
- ISO/IEC 9075: (referred to here as SQL92)
- SQL/PSM (the section of the SQL92 standard covering Persistent Stored Modules)

SQL92 has been developed by ANSI/ISO. This standard is written in a very formal manner. It is therefore difficult to use as a programming guide.

Different standards vary in the detail of their specifications. While any one standard defines a workable version of SQL, there are few, if any, vendors who supply database management systems with a specification that falls entirely within the bounds of a single standard. Mimer SQL combines many of the advantages of the different standards.

The following sections describe the SQL standards in more detail.

2.2 X/Open-95 SQL

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

X/Open-95 SQL is based closely on the International Standard for the Database Language SQL, ISO 9075:1992, ('SQL92') and includes some extensions to the International Standard which reflect the capabilities of current implementations.

This version of the standard specifically incorporates the following features:

- New table combination operations (JOIN variants and use of UNION in CREATE VIEW).
- Referential ON DELETE actions for the CREATE TABLE statement, which define how database integrity should be maintained if row deletion causes violation of FOREIGN KEY constraints.
- New features and exceptions relating to the CAST function.
- Various operations on fixed and variable length character strings, including the concatenation operator and the scalar functions CHAR_LENGTH, SUBSTRING and TRIM.
- Inclusion of DATE, TIME, TIMESTAMP and INTERVAL named data types.
- New form of ALTER TABLE allowing for the dropping and adding of columns.
- Support for four isolation levels through the SET TRANSACTION statement.
- Support for the SET CATALOG and SET SCHEMA statements.
- Enhancements to PRIMARY KEY, allowing UNIQUE constraints on columns containing null values.
- New system views COLUMN_PRIVILEGES, INDEXES, SCHEMATA, TABLE_PRIVILEGES, USAGE_PRIVILEGES and VIEWS.
- The keyword AS is permitted in correlation names, the keyword TABLE in the GRANT and REVOKE statements, the keyword FROM in the FETCH statement and the keyword WORK becomes optional in COMMIT and ROLLBACK.
- Enhancements to GET DIAGNOSTICS allowing the type of the relevant SQL statement to be determined.
- Enhancements to the INSERT statement relating to how values are specified.
- Internationalization features relating to support for national character sets and collation sequences.
- Addition of the string functions LOWER, UPPER and POSITION.

2.3 SQL92

Source: ISO/IEC 9075:1992(E) Information technology - Database languages - SQL.

The SQL92 standard was accepted in 1992. The standard is different from the other three standards in that it contains much more functionality. At the time of writing, SQL92 is not fully implemented by any vendor, and serves the important function of defining a standard for implementation of new features.

The standard is divided into the following levels: entry, intermediate and full SQL92, along with a transitional level (defined by FIPS) covering some features of intermediate level.

Entry level contains the set of features that define basic SQL92 compliance.

When vendors claim they are SQL92 compliant it is important to know at which level.

Mimer SQL complies fully with **entry level** and **transitional level** SQL92 (verified by using the NIST test suite).

The standard features listed below in this section are marked with different bullets to clarify exactly which features are implemented in Mimer SQL and which are not. The bullet marks should be interpreted as follows:

- Denotes a feature in the SQL standard that is implemented in Mimer SQL

o Denotes a feature in the SQL standard that is not yet implemented in Mimer SQL.

Transitional level SQL92 includes the following additions:

- support for dynamic SQL
- requirement for an accessible INFORMATION_SCHEMA supporting the TABLES, VIEWS, COLUMNS, TABLE_PRIVILEGES, COLUMN_PRIVILEGES and USAGE PRIVILEGES views
- support for schema definition and manipulation statements
- support for various join features including all provisions for NATURAL JOIN, INNER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN
- support for data types DATE, TIME, TIMESTAMP and INTERVAL, including various datetime and interval features (excluding time zones)
- support for CHARACTER VARYING, CHAR VARYING and VARCHAR data types and various character data operations and functions
- the TRIM function as an alternative for a character value function in string value function
- support for specifying UNION in a view definition
- support for implicit numeric casting when an approximate numeric value is assigned to an exact numeric type
- support for implicit character casting when character string values are assigned to character string types
- support for general use of the SET TRANSACTION options for ISOLATION LEVEL, READ_ONLY, READ_WRITE and DIAGNOSTIC SIZE
- support for GET DIAGNOSTICS

- relaxation of restrictions concerning grouped operations
- support for the '*qualifier*.*' construction in select lists
- use of lowercase letters allowed in identifiers
- table columns involved in a UNIQUE or PRIMARY KEY constraint are no longer required to be explicitly declared as NOT NULL
- support for separation of schema name and authorization name in a schema definition, allowing multiple schemas per user
- support for multiple, separately compiled, modules
- support for delete actions in referential constraints
- provision for the use of the CAST function with all supported data types
- support for value expressions in INSERT statements
- support for the specification of explicit default values in INSERT and UPDATE statements and in a row value constructor
- relaxation of certain keyword restrictions defined in Entry level SQL92 (AS permitted before a correlation name in a table reference, TABLE permitted in a GRANT statement, FROM permitted in a FETCH statement, WORK becomes optional in COMMIT and ROLLBACK statements).

Intermediate level SQL92 includes the following additions:

- support for the definition and use of domains
- support for CASE expressions
- support for compound character literals
- support of the specification of a general character value expression for the match value in LIKE predicates

o support for the UNIQUE predicate

o support for table operations in query expressions

- support for the schema definition statement, including circular references in schema elements
- support for CURRENT_USER and SESSION_USER

o support for SYSTEM_USER

- inclusion of the following INFORMATION_SCHEMA views: TABLE_CONSTRAINTS, REFERENTIAL_CONSTRAINTS, DOMAINS, DOMAIN_CONSTRAINTS, CHECK_CONSTRAINTS, ASSERTIONS, KEY_COLUMN_USAGE, VIEW_TABLE_USAGE, SCHEMATA, VIEW_COLUMN_USAGE, CONSTRAINT_TABLE_USAGE, CONSTRAINT_COLUMN_USAGE, COLUMN_DOMAIN_USAGE, CHARACTER_SETS and SQL_LANGUAGES

o support for the accessible base table INFORMATION_ SCHEMA_CATALOG_NAME

- support for subprograms
- support for Entry and Intermediate SQL flagging
- support for schema manipulation including DROP SCHEMA and ALTER TABLE operations
- support for long identifiers (up to 128 characters)

o support for FULL OUTER JOIN

o support for time zones and timezone management

o support for NATIONAL CHARACTER

• support for the declaration of scrolled cursors and for fetch orientation in a FETCH statement

• removal of various restrictions concerning certain set function operations

o support for the definition and use of character sets

o support for the named character sets SQL_CHARACTER, ASCII_GRAPHIC, LATIN1, ASCII_FULL and SQL_TEXT

o support for the use of a scalar subquery in any value expression

• extending the NULL predicate to allow values other than a column reference

• support for user-defined names for constraints

• support for the documentation schema tables SQL_FEATURES and SQL_SIZING.

Full SQL92 includes the following additions:

o support for the BIT data type

o support for ASSERTION constraints

o support for temporary tables

• support for full dynamic SQL

• support for specification of precision in TIME and TIMESTAMP data types

• full support for general use of value expressions

o support for truth value tests of TRUE, FALSE or UNKNOWN and their negations

• full support of the character functions POSITION, LOWER and UPPER

o support for the specification of a derived table in a FROM clause

• trailing underscore is permitted in an identifier

• removal of restrictions on the data types permitted for indicator parameters and variables

o removal of restrictions on the order of column names in a referential constraint definition

• support for complete Entry, Intermediate and Full SQL flagging

o support for the use of table value constructors and row value constructors in predicate and in the INSERT statement

o support for catalog name qualifiers

o support for simple or explicit table references in a query expression

o support for sub-queries in a CHECK constraint

o support for UNION JOIN and CROSS JOIN

o support for character set collations and translations

o support for update actions in referential constraints

o support for ALTER DOMAIN functionality

o support for deferrable constraints

• support for granting INSERT privilege on individual table columns

o support for MATCH FULL and MATCH PARTIAL in a referential constraint definition

o support for the CASCADED and LOCAL options in the WITH CHECK OPTION

o support for the session management statements for setting catalog, schema and names

- support for connection management, including CONNECT, SET CONNECTION and DISCONNECT
- support for self-referencing DELETE, INSERT and UPDATE statements
- o support for the INSENSITIVE option on a cursor declaration
- o support for full set function
- o support for the “Syntax Only” and “Catalog Lookup” checking options in SQL flagging
- o support for local table references qualified by module name
- o support for fully updatable cursors.

2.4 SQL/PSM

Source: ISO/IEC 9075-4:1996(E) Database Language SQL - Part 4: Persistent Stored Modules (SQL/PSM). This defines the portion of the SQL92 standard covering Stored Procedures.

2.5 SQL99

Source: ISO/IEC 9075:1999(E) Information technology - Database languages - SQL. This document is divided into the following parts:

- Part 1: Framework (SQL / Framework)
- Part 2: Foundation (SQL / Foundation)
- Part 3: Call-Level Interface (SQL / CLI)
- Part 4: Persistent Stored Modules (SQL / PSM)
- Part 5: Host Language Bindings (SQL / Bindings).

The SQL99 standard will replace SQL92 as the current SQL standard.

3 BASIC MIMER SQL CONCEPTS

This chapter provides a general introduction to the basic concepts of Mimer SQL databases and Mimer SQL objects.

3.1 The Mimer SQL relational database

Mimer SQL is a relational database system, which means that the information in the database is presented to the user in the form of tables. The tables represent a logical description of the contents of the database which is independent of, and insulates the user from, the physical storage format of the data.

The Mimer SQL database includes the **data dictionary** which is a set of tables describing the organization of the database and is used primarily by the database management system itself.

The database, although located on a single physical platform, may be accessed from many distinct platforms, 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 of the Mimer SQL User's Manual), so the actual database being accessed may change during the course of an SQL session. At any one time, however, the database may be regarded as one single organized collection of information.

3.1.1 The data dictionary

The data dictionary contains information on all the objects stored in a Mimer SQL database and how they relate to one another. The data dictionary stores information about:

- Databanks
- Access rights and privileges
- Views
- Indexes
- Shadows
- Sequences
- Modules
- Procedures
- Idents
- Schemas
- Tables
- Triggers
- Domains
- Synonyms
- Functions

Mimer SQL database objects can be divided into the following groups:

- **System objects** are global to the database. System object names must be unique for each object type since they are global and therefore common to all users. The system objects in a Mimer SQL database are: databanks, idents, schemas and shadows. A system object is owned by the ident that created it and only the creator of the object can drop it.
- **Private objects** belong to a schema. Private object names are local to a schema, so two different schemas may contain an object with the same name. The private objects in a Mimer SQL database are: domains, functions, indexes, modules, procedures, sequences, synonyms, tables, triggers and views.

  Private objects are fully identified by their qualified name, which is the name of the schema to which they belong and the name of the object in the following form: *schema.object* (see Section 4.2.3). Conflicts arising from the use of the same object name in two different schemas are avoided when the qualified name is used. If a private object name is specified without explicit reference to its schema, it is assumed to belong to a schema with the same name as the current ident.

3.1.2 Databanks

A databank is the physical file where a collection of tables is stored. A Mimer SQL database may include any number of databanks. There are two types of databank:

- **System databanks** contain system information used by the database manager. These databanks are defined when the system is created. The 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. (See Section 3.1.2.1 details concerning pathnames for user databank files.)

The division of tables between different user databanks is a physical file storage issue and does not affect the way the database contents are presented to the user. Except in special situations (such as when creating tables), databanks are completely invisible to the user.

Note: Backup and Restore in Mimer SQL can be performed on a per-databank basis rather than on the entire database file base (see Chapter 5 of the Mimer SQL System Management Handbook for more information).

3.1.2.1 Specifying the location of user databanks

The location for a user databank file can be specified completely (as an absolute pathname) or with some of the pathname components omitted (a relative pathname).

The default values used for omitted pathname components are taken from the pathname for the system databank file SYSDB, which is located in the database home directory.

Note: The databank location stored in the Mimer SQL data dictionary is the pathname as **explicitly** specified, i.e. without the addition of default values for any omitted pathname components (such additions are determined and added each time the file is accessed).

(Refer to the Mimer SQL System Management Handbook for recommendations concerning databank file management and for information on how the pathname for a databank file is determined).

3.1.3 Idents

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

- **User idents** identify individual users who can connect to a Mimer SQL database. A user ident's access to the database is protected by a password and is restricted by the specific privileges granted to the ident. User idents are generally associated with specific physical individuals who are authorized to use the system.
- **OS_USER idents** are idents which reflect a user id defined by the operating system. An OS_USER ident allows the user currently logged in to the operating system to access the Mimer SQL database without providing a username or password. For example: if the current operating system user is ALBERT and there is an OS_USER ident called ALBERT defined in Mimer SQL, ALBERT may start BSQL (for example) and connect directly to Mimer SQL simply by pressing <return> at the Username: prompt. If an OS_USER ident is defined with a password in Mimer SQL, the ident may also connect to Mimer SQL in the same way as a user ident (i.e. by providing the username and password). An OS_USER ident may not have the same name as a user ident in the database.
- **Program idents** do not strictly connect to Mimer SQL, but they may be entered from within an application program by using the ENTER statement. The ENTER statement may only be used by an ident who is already connected to a Mimer SQL database. An ident is granted the privilege to enter a program ident. A program ident is set up to have certain privileges and these apply after the ENTER statement has been used. Program idents are generally associated with specific functions within the system, rather than with physical individuals. The LEAVE statement is used to return to the state of privileges and database access that existed before ENTER was used.

- **Group idents** are collective identities used to define groups of user and/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 a 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 a logical group which is specified in Mimer SQL statements by using the keyword PUBLIC.

3.1.4 Schemas

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

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

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

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

3.1.5 Tables

Data in a relational database 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 a single item of data.

Each row in a table must have the same set of data items (one for each column in the table), but not all the items need to be filled in. A column can have a default value defined (either as part of the column specification itself or by using a domain with a default value) and this is stored in a field where an explicit value for the data item has not been specified.

If no default value been defined for a column, the NULL value is stored when no data value is supplied (the way the NULL value is displayed depends on the application, in BSQL the minus sign is used).

A relational database is built up of several inter-dependent tables which can be joined together. Tables are joined by using related values that appear in one or more columns in each of the tables. Part of the flexibility of a relational database structure is the ability to add more tables to an existing database. A new table can relate to an existing database structure by having columns with data that relates to the data in columns of the existing tables. No alterations to the existing data structure are required.

All the 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 by a data type (see Section 4.3 for a detailed description of data types).

3.1.6 Base tables and views

The logical representation of data in a Mimer SQL database is stored in tables (this is what the user sees, as distinct from the physical storage format which is transparent to the user). The tables which store the data are referred to as *base tables*. Users can directly examine data in the base tables. In addition, data may be presented in *views*, which are created from specific parts of one or more base tables. To the user, views may look the same as tables, but operations on views are actually performed on the underlying base tables. Access privileges on views and their underlying base tables are completely independent of each other, so views provide a mechanism for setting up specific access to tables.

The essential difference between a table and a view is underlined by the action of the DROP command, which removes 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 the 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 underlying base table(s).

Note: Since views are logical representations of tables, all operations requested on a view are actually performed on the underlying base table, so care must be taken when granting access privileges on views. Such privileges may include the right to insert, update and delete information. As an example, deleting a row from a view will remove the **entire** row from the underlying base table and this may include table columns the user of the view had no privilege to access.

Views may be created to simplify presentation of data to the user by including only some of the base table columns in the view or only by including selected rows from the base table. 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 the base tables will have been governed by rules of relational database modeling). Join views may also contain restriction conditions.

3.1.7 Primary keys and indexes

Rows in a base table are uniquely identified by the value of the primary key defined for the table. The primary key for a table is composed of the values of one or more columns. 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 the 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. A unique column is also a key, because it may not contain duplicate values, and need not necessarily be part of the primary key.

The columns of the primary key may not contain NULL (this is one of the requirements of a strictly relational database).

Values in primary key columns can be updated if the table involved is stored in a databank with the TRANS or LOG option.

Primary key and unique columns are automatically indexed to facilitate effective information retrieval.

Other columns or combinations of columns may be defined as a **secondary index** to improve performance in data retrieval. Secondary indexes are defined on a table after it has been created (using the CREATE INDEX statement).

A secondary index may be useful when, for example, a search is regularly performed on a non-keyed column in a table with many rows, then defining an index on the column may speed up the search. The search result is not affected by the index but the speed of the search is optimized.

It should be noted, however, that indexes create an overhead for update, delete and insert operations because the index must also be updated. Indexes are internal structures which cannot be explicitly accessed by the user once created.

There is no guarantee that the presence of an index will actually improve performance because the decision to use it or not is made by the internal query optimization process.

SQL queries are automatically **optimized** when they are internally prepared for execution. The optimization process determines the most effective way to execute the query and in some cases optimal query execution may not actually involve using an index.

3.1.8 Routines (functions and procedures)

In Mimer SQL it is possible to define SQL routines that are stored in the data dictionary and which may be invoked when needed. The term “routine” is a collective term for **functions** and **procedures**. The acronym PSM, Persistent Stored Modules, is also used to denote stored routines.

For a complete and detailed discussion of functions, procedures and the Stored Procedures functionality supported in Mimer SQL see Chapter 8 of the Mimer SQL Programmer’s Manual.

Functions are distinguished from procedures in that they return a single value and the parameters of a function are used for input only. A function is invoked by using it where a value expression would normally be used.

Mimer SQL supports standard procedures and also result set procedures, which are procedures capable of returning the row value(s) of a result-set.

Standard procedures are invoked directly by using the CALL statement and can pass values back to the calling environment through the procedure parameters.

In embedded SQL, result set procedures are invoked by declaring a cursor which includes the procedure call specification and by then using the FETCH statement to execute the procedure and return the row(s) of the result-set.

In interactive SQL, a result set procedure is invoked by using the CALL statement directly and the result-set values are presented in the same way as for a select returning more than one row.

The ident invoking a routine must have EXECUTE rights on it.

The creator of a routine must hold the appropriate access rights on any database objects referenced from within the routine. These access rights must be held for the life of the routine.

Routine names, like those of other private objects in the database, are qualified with the name of the schema to which they belong.

The PSM constructs available in Mimer SQL allow powerful functionality to be defined and used through the creation and execution of routines. The use of routines also makes it possible to move application logic from the client to the server, thereby reducing network traffic.

3.1.9 Modules

A module is simply a collection of routines. All the routines in a module are created when the module is created and belong to the same schema.

EXECUTE rights on the routines contained in a module are held on a per-routine basis, **not** on the module.

If a module is dropped, all the routines contained in the module are dropped.

Under certain circumstances a routine may be dropped because of the cascade effect of dropping some other database object (see Section 7.12.5 of the Mimer SQL User’s Manual for details). If such a routine is contained in a module, it is implicitly removed from the module and dropped. The other routines contained in the module remain unaffected.

In general, care should be taken when using DROP or REVOKE in connection with routines, modules or objects referenced from within routines because the cascade effects can often affect many other objects (see Sections 7.12 and 8.3.4 of the *Mimer SQL User’s Manual*).

3.1.10 Synonyms

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

A synonym cannot be created for a function, procedure or a module.

Using synonyms can be a convenient way to address tables that are contained in another schema. For example, if a view called ROOM_VIEW is contained in the schema called SAMMY, the full name of the view is SAMMY.ROOM_VIEW.

This view may be referenced from the schema called JIMMY by its fully qualified name as given above.

Alternatively, a synonym may be created for the view in schema JIMMY, e.g. RM_VIEW. Then the name RM_VIEW can simply be used to refer to the view SAMMY.ROOM_VIEW.

Note: The name RM_VIEW is contained in schema JIMMY and can only be used in that context.

3.1.11 Shadows

Mimer SQL 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. Shadowing requires a separate license.

3.1.12 Triggers

A trigger defines a number of procedural SQL statements that are executed whenever a specified data manipulation statement is executed on the table or view on which the trigger has been created.

The trigger can be set up to execute AFTER, BEFORE or INSTEAD OF the data manipulation statement. Trigger execution can also be made conditional on a search condition specified as part of the trigger.

Triggers are described in detail in Chapter 9 of the Mimer SQL Programmer's Manual.

3.1.13 Sequences

A sequence is a private database object that can provide a series of integer values. A sequence can be defined as unique or non-unique.

A unique sequence generate unique values. If all values between the initial value and the maximum value has been used, the sequence becomes exhausted and can not be used any more.

A non-uniquec sequence will generate its series of values repeatedly.

A sequence has an initial value, an increment step value and a maximum value defined when it is created (by using the CREATE SEQUENCE statement).

A sequence is created with an **undefined** value initially.

It is possible to generate the next value in the integer series of a sequence by using the NEXT_VALUE function (see Section 5.5.19). When this function is used for the first time after the sequence has been created, it establishes the initial value for the sequence. Subsequent uses will establish the next value in the series of integer values of the sequence as the current value of the sequence.

It is possible to get the current value of a sequence that has been initialized by using the CURRENT_VALUE function (see Section 5.5.10). This function cannot be used until the initial value has been established for the sequence (by using NEXT_VALUE for the first time).

An ident must hold USAGE privilege on the sequence in order to use it.

If a sequence is dropped, with the CASCADE option in effect, all objects referencing the sequence will also be dropped.

3.2 Data integrity

A vital aspect of a Mimer SQL 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 facilities that ensure 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 to protect the integrity of the database, guaranteeing that incorrect or inconsistent data is not entered into it. By applying data integrity constraints through the database management system, the responsibility of ensuring the data integrity of the database is moved from the users of the database to the database designer.

3.2.1 Domains

Each column in a table holds data of a single data type and length, specified when the column is created or altered. The data type and length may be specified explicitly (e.g. CHARACTER(20) or INTEGER(5)) or through the use of **domains**, which can give more precise control over the data that will be accepted in the column.

A domain definition consists of a data type and length specification with optional check conditions and a default value. Data which falls outside the constraints defined by the check conditions is not accepted in a column which is defined using the domain.

A column defined using a domain for which a default value is defined will automatically receive that value if row data is entered without a value being explicitly specified for the column.

In order for an ident to create a table containing columns whose data type is defined through the use of a domain, the ident must first have been granted USAGE rights on the domain (see Section 8.2.2 of the Mimer SQL User's Manual).

3.2.2 Foreign keys - referential integrity

A foreign key is one or more columns in a table defined as cross-referencing the primary key or a unique key of another table. Data entered into the foreign key must either exist in the key that it cross-references or be NULL. This maintains **referential integrity** in the database, ensuring that a table can only contain data that already exists in the selected key of the referenced table.

As a consequence of this, a key value that is cross-referenced by a foreign key of another table must not be removed from the table to which it belongs by an update or delete operation if this ultimately violates the referential constraint.

The DELETE rule defined for the referential constraint provides a mechanism for adjusting the values in a foreign key in a way that may permit a cross-referenced key value to effectively be removed.

Note: The referential integrity constraints are effectively checked at the end of an INSERT, DELETE or UPDATE statement.

Foreign key relationships are defined when a table is created using the CREATE TABLE statement and can be added to an existing table by using the ALTER TABLE statement.

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

If foreign key relationships are defined for tables in a CREATE SCHEMA statement, it is possible to reference a table that will not be created until later in the CREATE SCHEMA statement.

Note: Both the table containing the foreign key and the cross-referenced table must be stored in a databank with either the TRANS or LOG option.

3.2.3 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.

Check conditions are discussed in detail in Section 7.5.5 of the Mimer SQL User's Manual.

3.2.4 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.

3.3 Access rights and privileges

Access rights and privileges control users' access to database objects and the operations they can perform in the database.

User and program idents 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 operations each ident is permitted to perform. There are three classes of privileges in a Mimer SQL database:

- **System privileges**, which control the right to perform backup and restore operations, the right to execute the UPDATE STATISTICS statement as well as the right to create new databanks, idents, schemas and to manage shadows. System privileges are granted to the system administrator when the system is installed and may be granted by the administrator to other idents in the database. As a general rule, system privileges should be granted to a restricted group of users.

  Note: An ident who is given the privilege to create new idents is also able to create new schemas.

- **Object privileges**, which control membership in group idents, the right to invoke functions and procedures, the right to enter program idents, the right to create new tables in a specified databank and the right to use a domain or sequence. 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 function or procedure may execute it, the creator of a program ident may enter it, the creator of a schema may create objects in and drop objects from it, the creator of a databank may create tables in the databank, the creator of a table has all access rights on the table, the creator of a domain may use that domain and the creator of a sequence may use that sequence. The creator of an object generally has the right to grant any of these privileges to other users, in the case of functions and procedures this actually depends on the creator's access rights on objects referenced from within the routine.

- **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 to refer to table columns as foreign key references.

Granted privileges can be regarded as instances of grantor/privilege stored for an ident. An ident will hold **more than one** instance of a privilege if **different** grantors grant it.

A privilege will be held as long as at least one instance of that privilege is stored for the ident. All privileges may be granted with the WITH GRANT OPTION which means that the receiver has, in turn, the right to grant the privilege to other idents. An ident will hold a privilege with the WITH GRANT OPTION as long as at least one of the instances stored for the ident was granted with this option.

If the **same** grantor grants a privilege to an ident more than once, this will **not** result in more than one instance of the privilege being recorded for the ident. If a particular grantor grants a privilege without the WITH GRANT OPTION and subsequently grants the privilege again with the WITH GRANT OPTION, the WITH GRANT OPTION will be added to the existing instance of the privilege.

Each instance of a privilege held by an ident is revoked separately by the appropriate grantor. It is possible to revoke the WITH GRANT OPTION without revoking the associated privilege completely. Section 8.3 of the Mimer SQL User's Manual describes revoking privileges in more detail.

4 BASIC SQL SYNTAX RULES

This chapter presents the basic units of the SQL language and the simplest relationships between them. The following units are considered:

Syntax unit	Summary	Section
Characters	Basic language symbols: letters, digits, special characters.	4.1
Identifiers	SQL identifiers, host variable names.	4.2
Data types	Character, exact numeric, approximate numeric, date, time, timestamp, interval. Datetime and interval arithmetic.	4.3
Literals	Character, integer, decimal, floating point, date, time, timestamp, interval.	4.4
Assignments	Character, numeric, datetime and interval assignments. Data type conversions.	4.5
Comparisons	Character, numeric, datetime and interval comparison rules.	4.6

4.1 Characters

The character set used by Mimer SQL is ISO 8859-1 (LATIN1) - see Appendix B.

For the purposes of the syntax rules in SQL, characters may be divided into the following classes:

Character class	Description
Letters	A ... Z, a ... z
Digits	0 1 2 3 4 5 6 7 8 9
Alphanumeric	Letters and digits
Separators	White-space characters (ASCII HEX-values 09 to 0D, or 20, i.e. <TAB>, <LF>, <VT>, <FF>, <CR> or <SP>)
Special	All characters except letters and digits

Note: The definition of letters is restricted to the English standard alphabet. National characters are treated as special characters.

Certain special characters have particular meanings in SQL statements, for example delimiters (quotation marks " or apostrophes ') and arithmetic and comparative operators (+ - / * = < > etc.). The special characters $ and # may in some circumstances be used in the same contexts as letters (see Section 4.2).

A separator is used to separate keywords, identifiers and literals from each other.

4.2 Identifiers

An identifier is defined as a sequence of one or more characters forming a unique name. Identifiers are constructed according to certain fixed rules. It is useful to distinguish between SQL identifiers, which are local to SQL statements, and host identifiers, which relate to the host programming language. Rules for constructing the latter may vary between host languages.

4.2.1 SQL identifiers

SQL identifiers consist of a sequence of one or more characters. The maximum length of an SQL identifier is 128 characters.

SQL identifiers (except for delimited identifiers) must begin with a letter or one of the special characters $ or #, and may only contain letters, digits and the special characters $, # and _.

The case of letters in identifiers is not significant, unless it is a delimited identifier. All non-delimited letters are treated as uppercase.

Delimited identifiers means identifiers enclosed in quotation marks ("). Such identifiers are special in three aspects:

- They can contain characters normally not supported in SQL identifiers.
- They can be identical to a reserved word.
- They are treated in a case-sensitive manner.

The following examples illustrate the general rules for forming SQL identifiers:

Valid	**Invalid**	**Explanation**
COLUMN_1	COLUMN+1	COLUMN+1 is an expression
#14	14	14 is an integer literal
“TABLE NAME”	TABLE NAME	TABLE NAME contains a blank
“SELECT”	SELECT	SELECT is a reserved word

Note: Leading blanks are significant in delimited identifiers.

4.2.2 Rules for naming objects

Objects in the database may be divided into two classes:

- System objects (databanks, idents, schemas and shadows) are global to the system. System object names must be unique within each object class since they are common to all users. System objects are uniquely identified by their name alone.

- Private objects (domains, functions, indexes, modules, procedures, tables, triggers, sequences, synonyms and views) belong to a schema and have names that are local to that schema. In a given schema, the names used for tables, synonyms and views must be unique within that group of objects (i.e. a table cannot have a name that is already being used by a synonym or view, etc.). The names of all other objects (domains, indexes, functions, modules, procedures, sequences) in the schema must be unique within their respective object-type. Two different schemas may contain objects of the same type with the same name. Private objects are uniquely identified by their qualified name (see below).

4.2.3 Qualified object names

Names of private objects in the database may always be qualified by the name of the schema to which they belong. The schema name is separated from the object name by a period, with the general syntax **schema.object**.

If a qualified object name is specified when an object is created, it will be created in the named schema. If an object name is unqualified, a schema name with the same name as the current ident is assumed.

It is recommended that object names are always qualified with the schema name in embedded SQL statements, to avoid confusion if the same program is run by different Mimer SQL idents.

Names of columns in tables or views are used in SQL statements both as an explicit indication of the column itself and as an indication of the values stored in the column.

When the name of a column is expressed in its **unqualified** form it is syntactically referred to as a ***column-name***.

When the name of a column must be expressed **unambiguously** it is generally expressed in its fully qualified form (i.e. *schema.table.column* or *table.column*) and this is syntactically referred to as a ***column-reference***.

It is possible for a *column-reference* to be the unqualified name of a column in contexts where this is sufficient to unambiguously identify the column.

When the name of a column is used to indicate the column itself (e.g. in CREATE TABLE statements), a *column-name* **must** be used (i.e. the name of the column cannot be qualified).

The exception to this is in the COMMENT ON COLUMN statement where a *column-reference* is required because the name of the column **must** be qualified by the name of the table or view to which it belongs.

The contexts where the name of a column refers to the values stored in the column are:

- in expressions
- in set functions
- in search conditions
- in GROUP BY clauses.

In these contexts a *column-reference* must be used to identify the column.

The column name qualifiers which may be used in a particular SQL statement are determined by the way the table is identified in the FROM clause of the SELECT statement.

Alternative names (correlation names) may be introduced in the FROM clause, and the table reference used to qualify column names must conform to the following rules:

- If no correlation names are introduced, the column name qualifier is the table name exactly as it appears in the FROM clause. For example:

  ```
  SELECT  BOOKADM.HOTEL.NAME, ROOMS.ROOMNO
  FROM    BOOKADM.HOTEL,      ROOMS ...
  ```

  but not

  ```
  SELECT  BOOKADM.HOTEL.NAME, ROOMS.ROOMNO
  FROM    HOTEL,              BOOKADM.ROOMS ...
  ```

- If a correlation name is introduced, the correlation name and not the original table reference, may be used to qualify a column name. The correlation name may not itself be qualified. For example:

  ```
  SELECT  H.NAME,  ROOMS.ROOMNO
  FROM    HOTEL H, ROOMS ...
  ```

  but not

  ```
  SELECT  HOTEL.NAME, ROOMS.ROOMNO
  FROM    HOTEL H,    ROOMS ...
  ```

4.2.4 Outer references

In some constructions where subselects are used in search conditions (see Section 5.11), it may be necessary to refer in the lower level subselect to a value in the current row of a table addressed at the higher level. A reference to a column of a table identified at a higher level is called an outer reference. The following example shows the outer reference in bold type:

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

The lower-level subselect is evaluated for every row in the higher level result table. The example selects the name of every hotel with at least one entry in the BOOK_GUEST table.

A qualified column name is an outer reference if, and only if, the following conditions are met:

- The qualified column name is used in a search condition of a subselect.
- The qualifying name is not introduced in the FROM clause of that subselect.
- The qualifying name is introduced at some higher level.

4.2.5 Host identifiers

Host identifiers are used in SQL statements to identify objects associated with the host language (variables, declared areas, program statement labels).

Host identifiers are formed in accordance with the rules for forming variable names in the particular host language (see Appendix A of the Mimer SQL Programmer's Manual). Host identifiers are never enclosed in delimiters and may coincide with SQL reserved words. The length of host identifiers used in SQL statements may not exceed 128 characters (even if the host language accepts longer names).

Whenever the term "host-variable" appears in the syntax diagrams, one of the three following constructions must be used:

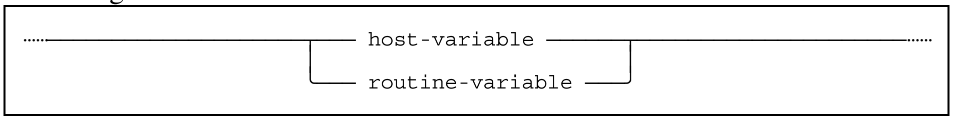
```
    :host-identifier1
or
    :host-identifier1 :host-identifier2
or
    :host-identifier1 INDICATOR :host-identifier2
```

Host-identifier1 is the name of the main host variable. Host-identifier2 is the name of the indicator variable, used to signal the assignment of a NULL value to the host variable. See Section 3.1.3 of the Mimer SQL Programmer's Manual for a description of the use of indicator variables. The colon preceding the host identifier serves to identify the variable to the SQL compiler and is not part of the variable name in the host language.

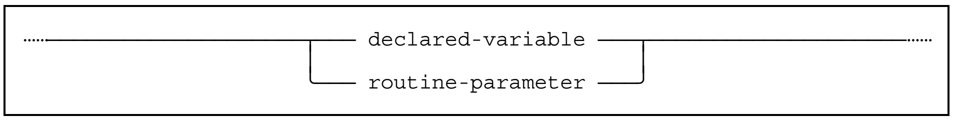
4.2.6 Target variables

A target variable is an item that may be specified as the object receiving the result of an assignment or a SELECT INTO. The objects that may be specified where a target variable is expected differ depending on whether the context is Procedural usage or Embedded usage.

In the syntax diagrams the term *target-variable* should be replaced by the following construction:



where *routine-variable is:*



See DECLARE VARIABLE for a description of how a *declared-variable* is defined, and CREATE FUNCTION or CREATE PROCEDURE for a description of how a *routine-parameter* is defined.

Note: A *routine-variable* may only be specified in a Procedural usage context and a *host-variable* may only be specified in Embedded usage.

4.2.7 Reserved words

Appendix A gives a list of keywords reserved in SQL statements. These words must be enclosed in quotation marks (") if they are used as SQL identifiers.

Example:

```
SELECT "MODULE" FROM ...
```

4.2.8 Standard compliance

This section summarizes standard compliance concerning identifiers.

Standard	Compliance	Comments
X/Open-95 SQL92	EXTENDED	The use of the special characters $ and # in identifiers is a Mimer SQL extension.

4.3 Data types

4.3.1 Data types in SQL statements

Explicit data type references are made in SQL statements in the creation of domains and base tables and in the alteration of table definitions. The permissible data types and their ranges are:

Data type	Description	Range
CHARACTER(n)	Character string, fixed length n.	1 ≤ n ≤ 15000
CHARACTER VARYING(n) or VARCHAR(n)	Variable length character string, maximum length n.	1 ≤ n ≤ 15000
BINARY(n)	Fixed length binary string, maximum length n.	1 ≤ n ≤ 15000
BINARY VARYING(n) or VARBINARY(n)	Variable length binary string, maximum length n.	1 ≤ n ≤ 15000
INTEGER(p)	Integer numerical, precision p.	1 ≤ p ≤ 45
SMALLINT	Integer numerical precision 5.	-32768 through 32767
INTEGER	Integer numerical, precision 10.	-2,147,483,648 through 2,147,483,647
BIGINT	Integer numerical, precision 19.	-9,223,372,036,854,775,808 through 9,223,372,036,854,775,807
DECIMAL(p, s)	Exact numerical, precision p, scale s.	1 ≤ p ≤ 45 0 ≤ s ≤ p
NUMERIC(p, s)	Exact numerical, precision p, scale s. (Same as DECIMAL - see below).	1 ≤ p ≤ 45 0 ≤ s ≤ p
FLOAT(p)	Approximate numerical, mantissa precision p.	1 ≤ p ≤ 45 Zero or absolute value 10^{-999} to 10^{+999}
REAL	Approximate numerical mantissa precision 7.	Zero or absolute value 10^{-38} to 10^{+38}
FLOAT	Approximate numerical mantissa precision 16.	Zero or absolute value 10^{-308} to 10^{+308}
DOUBLE PRECISION	Approximate numerical mantissa precision 16.	Zero or absolute value 10^{-308} to 10^{+308}
DATE TIME TIMESTAMP	Composed of a number of integer fields, represents an absolute point in time, depending on sub-type.	* Refer to Section 4.3.2 for a complete explanation of this data type.

Data type	**Description**	**Range**
INTERVAL	Composed of a number of integer fields, represents a period of time, depending on the type of interval.	* Refer to Section 4.3.3 for a complete explanation of this data type.

There is an additional pseudo data type supported by Mimer SQL, called the ROW data type. A ROW data type definition can be specified where one of the above data types would normally be used in a variable declaration in a compound statement (see Section 8.2.7 of the Mimer SQL Programmer's Manual for details).

Binary data may only be stored in the database and retrieved again, it cannot be used in arithmetical operations. If binary data is retrieved into a character data type, the length of the character data type must be twice that of the binary data type to accommodate the resulting hexadecimal character string. In interactive SQL, the way binary data is displayed depends on how the interactive tool is configured (in BSQL binary data is displayed as its hexadecimal value).

All numerical data may be signed.

For all numerical data, the precision "p" indicates the maximum number of decimal digits the number may contain, excluding any sign or decimal point.

For decimal data, the scale "s" indicates the fixed number of digits following the decimal point. Note that decimal data with scale zero (DECIMAL(p,0)) is not the same as integer (INTEGER(p)).

Floating point (approximate numerical) data is stored in exponential form. The precision is specified for the mantissa only. The permissible range of the exponent is -999 to +999.

Note: In Mimer SQL the NUMERIC data type is exactly equivalent to DECIMAL.

In the following cases, the omission of scale, or the omission of **both** precision and scale, is allowed (scale may not be specified without precision):

CHARACTER	is equivalent to CHARACTER(1)
DECIMAL	is equivalent to DECIMAL(15,0)
DECIMAL(5)	is equivalent to DECIMAL(5,0)
NUMERIC	is equivalent to NUMERIC(15,0)
NUMERIC(5)	is equivalent to NUMERIC(5,0)

Note: The data type INTEGER is distinct from INTEGER(10).

The following abbreviations are accepted for data type definitions:

CHAR(n)	for	CHARACTER(n)
CHAR	for	CHARACTER
CHAR VARYING(n)	for	CHARACTER VARYING(n)
VARCHAR(n)	for	CHARACTER VARYING(n)
VARBINARY(n)	for	BINARY VARYING(n)
INT(p)	for	INTEGER(p)
INT	for	INTEGER
DEC(p, s)	for	DECIMAL(p, s)
DEC	for	DECIMAL
NUM(p, s)	for	NUMERIC(p, s)
NUM	for	NUMERIC

4.3.2 DATE, TIME and TIMESTAMP

The DATE or TIMESTAMP data type represents an absolute position on the timeline and the TIME data type represents an absolute time of day.

“DATETIME” is a term used to collectively refer to the data types DATE, TIME and TIMESTAMP.

A DATETIME contains some or all of the fields YEAR, MONTH, DAY, HOUR, MINUTE and SECOND. These fields always occur in the order listed, which is from the most significant to least significant. Each of the fields is an integer value, except that the SECOND field may have an additional integer component to represent the fractional seconds.

For a DATETIME value with a SECOND component, it is possible to specify an optional **seconds precision** which is the number of significant digits in the fractional part of the SECOND value. This must be a value between 0 and 9. If a seconds precision is not specified, the default is 0 for TIME and 6 for TIMESTAMP.

DATE values are represented according to the Gregorian calendar and TIME values are represented according to the 24 hour clock. The **inclusive** value limits for the DATETIME fields are as follows:

YEAR	0001 to 9999
MONTH	01 to 12
DAY	01 to 31 (upper limit further constrained by MONTH and YEAR)
HOUR	00 to 23
MINUTE	00 to 59
SECOND	00 to 59.999999999

The three DATETIME data types are:

DATE	This describes a date using the fields YEAR, MONTH and DAY in the format YYYY-MM-DD. The length is 10.
TIME(s)	This describes a time in an unspecified day, with seconds precision *s,* using the fields HOUR, MINUTE and SECOND in the format HH:MM:SS[.*s*F] where F is the fractional part of the SECOND value. If a seconds precision is not specified, *s* defaults to 0. The length is 8 (or 9+*s*, if *s* > 0).
TIMESTAMP(s)	This describes both a date and time, with seconds precision *s*, using the fields YEAR, MONTH, DAY, HOUR, MINUTE and SECOND in the format YYYY-MM-DD HH:MM:SS[.*s*F] where F is the fractional part of the SECOND value. If a seconds precision is not specified, *s* defaults to 6. The length is 26 (or 19, if s = 0 or 20+*s*, if *s* > 0).

4.3.3 Interval

An INTERVAL is a period of time, such as “3 years” or “90 days” or “5 minutes and 45 seconds”. There are effectively two kinds of INTERVAL:

“YEAR-MONTH”	containing one or both of the fields YEAR and MONTH.
“DAY-TIME”	containing one or more **consecutive** fields from the set DAY, HOUR, MINUTE and SECOND.

The distinction is made between the two interval types in order to avoid the ambiguity that would arise if a MONTH value was combined with a field of lower significance, e.g. DAY, given that different months contain differing numbers of days. For example, the hypothetical interval “2 months and 10 days” could vary between 69 and 72 days in length, depending on the months involved. Therefore, to avoid unwanted variations in the downstream arithmetic etc., the variable length MONTH component may only exist at the lowest significance level in an INTERVAL.

The SECOND field may also only exist at the lowest significance level in an INTERVAL, simply because it is the least significant of all the fields.

An INTERVAL data type is a signed numeric quantity (i.e. negative INTERVALs are allowed) comprising a specific set of fields. The list of fields in an INTERVAL is called the **interval precision.**

The fields in an INTERVAL are exactly the same as those previously described for DATETIME except that the value constraints imposed on the **most significant field** are determined by the **leading precision** (***p*** in Section 4.3.3.2) for the INTERVAL type and **not** by the Gregorian calendar and 24 hour clock. A leading precision value between 1 and the maximum allowed for the field type may be specified for an INTERVAL. If none is specified, the default is 2.

4.3.3.1 Value Constraints for fields in an Interval

The table below shows the maximum permitted leading precision values for each field type in an INTERVAL:

Field	**Maximum leading precision**
YEAR	7
MONTH	7
DAY	7
HOUR	8
MINUTE	10
SECOND	12

The value of a MONTH field, which is **not** in the leading field position, is constrained between 0 and 11, inclusive, in an INTERVAL (and not between 1 and 12 as in a DATETIME).

Where the SECOND field is involved, **seconds precision** (*s* in Section 4.3.3.2) can be specified for it in the same way as for DATETIME. Note that in the INTERVAL consisting only of a SECOND field ("INTERVAL SECOND"), the SECOND field will have both a leading precision and a seconds precision, specified together. A seconds precision value between 0 and 9 may be specified for an INTERVAL. If the seconds precision is not specified, a default value of 6 is implied.

4.3.3.2 Named Interval Data Types

The syntactic element that is used to specify the interval precision, leading precision and (where appropriate) the seconds precision is the *interval qualifier*. This follows the keyword INTERVAL when specifying an INTERVAL data type.

The following table lists the valid interval qualifiers for YEAR-MONTH intervals:

Interval Qualifier	**Description**
YEAR(*p*)	An interval class describing a number of years, with a leading precision *p*. It contains a YEAR field in the format: *p*Y.
MONTH(*p*)	An interval class describing a number of months, with leading precision *p*. It contains a MONTH field in the format: *p*M.
YEAR(*p*) TO MONTH	An interval class describing a number of years and months, with leading precision *p*. The format is: *p*Y-MM.

The following table lists the valid interval qualifiers for DAY-TIME intervals:

Interval Qualifier	**Description**
DAY(*p*)	An interval class describing a number of days, with a leading precision *p*. It contains a DAY field in the format: *p*D.
HOUR(*p*)	An interval class describing a number of hours, with leading precision *p*. It contains an HOUR field in the format: *p*H.
MINUTE(*p*)	An interval class describing a number of minutes, with leading precision *p*. It contains a MINUTE field in the format: *p*M.
SECOND(*p*, *s*)	An interval class describing a number of seconds, with leading precision *p* and seconds precision *s*. It contains a SECOND field in the format: *p*S[.*s*F]. (F is the fractional part of the seconds value.)
DAY(*p*) TO HOUR	An interval class describing a number of days and hours, with leading precision *p*. The format is: *p*D HH.
DAY(*p*) TO MINUTE	An interval class describing a number of days, hours and minutes, with leading precision *p*. The format is: *p*D HH:MM.
DAY(*p*) TO SECOND(*s*)	An interval class describing a number of days, hours, minutes and seconds, with leading precision *p*. The format is: *p*D HH:MM:SS[.*s*F].
HOUR(*p*) TO MINUTE	An interval class describing a number of hours and minutes, with leading precision *p*. The format is: *p*H:MM.
HOUR(*p*) TO SECOND(*s*)	An interval class describing a number of hours, minutes and seconds, with leading precision *p* and seconds precision *s*. The format is: *p*H:MM:SS[.*s*F].
MINUTE(*p*) TO SECOND(*s*)	An interval class describing a number of minutes and seconds, with leading precision *p* and seconds precision *s*. The format is: *p*M:SS[.*s*F].

4.3.3.3 Length of an Interval

The length of an INTERVAL is the same as the number of characters required to represent it as a string and is determined by the interval precision, leading precision and the seconds precision (where it applies).

The maximum length of an INTERVAL can be computed according to the following rules:

- The length of the most significant field is the leading precision value (*p*).

- Allow a length of 2 for **each** field following the most significant field.
- Allow a length of 1 for each separator between fields. Separators occur between YEAR and MONTH, DAY and HOUR, HOUR and MINUTE, and MINUTE and SECOND.
- If seconds precision applies, and is non-zero, allow a length equal to the seconds precision value, plus 1 for the decimal point preceding the fractional part of the seconds value.

4.3.4 The NULL value

Columns which contain an undefined value are assigned a NULL value. Depending on the context, this is represented in SQL statements either by the keyword NULL or by a host variable associated with an indicator variable whose value is minus one (see Section 3.1.3 of the Mimer SQL Programmer's Manual).

The NULL value is generally never equal to any value, not even to itself. All comparisons involving NULL evaluate to "unknown" (see Section 4.6).

Note: NULL values are treated as **equal** to each other for the purposes of DISTINCT, GROUP BY, and ORDER BY.

NULL is sorted at the end of ascending sequences and at the beginning of descending sequences.

4.3.5 Data type compatibility

Assignment and comparison operations generally require that the data types of the items involved (literals, variables or column values) are compatible but not necessarily exactly equivalent. Any exceptions to this rule are specified in the detailed syntax descriptions (Chapters 5 and 6).

All character data is compatible with all other character data.

Numerical data is compatible with other numerical data regardless of specific data type (integer, decimal or float). Rules for operations involving mixed numerical data types are described in Section 4.6.

Datetime and interval data types can be combined in arithmetic operations, for details see Section 4.3.6.

Values stored in host variables (but not literals or column values) may be converted between character and numerical data types if required by the operation using the variable. (The declared type of the variable itself is not altered). Similarly, character columns may be assigned to numerical variables and vice versa. The rules for data type conversion are given below.

Variables may be converted between different data types by using the CAST function.

4.3.6 Datetime and Interval arithmetic

The following table lists the arithmetic operations that are permitted involving DATE, TIME, TIMESTAMP (“DATETIME”) or INTERVAL values:

Operand 1	Operator	Operand 2	Result Type
DATETIME	-	DATETIME	*See discussion below.
DATETIME	+ or -	INTERVAL	DATETIME
INTERVAL	+	DATETIME	DATETIME
INTERVAL	+ or -	INTERVAL	INTERVAL
INTERVAL	* or /	Numeric	INTERVAL
Numeric	*	INTERVAL	INTERVAL

Operands cannot be combined arithmetically unless their data types are **comparable** (see Section 4.6). If either operand is the NULL value, then the result will always be the NULL value.

If an arithmetic operation involves two DATETIME or INTERVAL values with a defined scale, the scale of the result will be the larger of the scales of the two operands.

When an INTERVAL value is multiplied by a numeric value, the scale of the result is equal to that of the INTERVAL and the precision of the result is the leading precision of the INTERVAL increased by 1. In case of the division, the same is true except that the precision of the result is equal to the leading precision of the INTERVAL (i.e. it is not increased by 1).

When two INTERVAL values are added or subtracted, the scale (s) and precision (p) of the result are described by the following rule:

p = min(MLP, max(p'-s', p"-s") + max(s', s") + 1)
s = max(s', s")

where MLP is the maximum permitted leading precision for the INTERVAL type of the result (refer to the table in Section 4.3.3 for these values).

The interval precision (see Section 4.3.3) of the result is the combined interval precision of the two operands, e.g. DAY TO HOUR + MINUTE TO SECOND will produce a DAY TO SECOND result.

One DATETIME value may be subtracted from another to produce an INTERVAL that is the signed difference between the stated dates or times. The application must, however, specify an INTERVAL date type for the result by using an *interval-qualifier*. Thus, the syntax is:

(DATETIME1 - DATETIME2) interval-qualifier

Example:

(DATE '1996-01-09' - DATE '1996-01-01') DAY

This, therefore, evaluates to INTERVAL '8' DAY.

4.3.7 Data types for parameter markers

Parameter markers in statements submitted to dynamic SQL are assigned data types appropriate to their usage. (See Chapter 7 of the Mimer SQL Programmer's Manual for a discussion of dynamic SQL). For parameter markers used to represent numerical data in arithmetic or comparison expressions, precision 45 is used.

For parameter markers used to represent data assigned to columns, the precision is assigned in accordance with the column definition.

4.3.8 Host variable data type conversion

When a host variable is used in assignments, comparisons or expressions where the data type of the variable is different from the data type of literals or column declarations, an attempt is made internally to convert the value of the variable to the appropriate type.

Conversion between a fixed length character variable and a VARCHAR value is always allowed. The conversion follows these rules:

- When assigning a VARCHAR value to a character variable, where the variable is longer than the VARCHAR value, the variable is padded with trailing blanks.
- When assigning a VARCHAR value to a character variable, where the value is longer than the variable, the value is truncated and a warning status returned. If only blanks are truncated, no warning is returned.
- When assigning a VARCHAR column from a character variable the column is padded with blanks up to the length of the character variable, if the VARCHAR column is longer than the variable.
- When assigning a VARCHAR column from a character variable, where the VARCHAR column is shorter than the variable (except for trailing spaces), the assignment will fail and an error status is issued.

Numerical values may always be converted to character strings, provided that the character string variable is sufficiently long enough. The resulting string format is illustrated below, using "n" to represent the appropriate number of digits and "s" to represent the sign position (blank for positive values and a minus sign for negative values).

Three digits are always used for the exponent derived from floating point numbers, regardless of the value of the exponent. The sign of the exponent is always given explicitly (+ or -).

Numerical data	**String length**	**String format**
Integer numerical precision p	p+1	'sn'
Exact numerical precision p, scale s	p+2	'sn.n'
Approximate numerical precision p	p+7	'sn.nEsn'

<u>Note:</u> Decimal values with scale 0 are converted to strings with the format 'sn.'. Decimal values where the scale is equal to the precision result in strings with the format 's.n'.

Examples of assignment results:

Value	**Type**	**Character value**
1342	INTEGER(6)	'1342'
-15	INTEGER(2)	'-15'
13.42	DECIMAL(6,4)	'13.4200'
-13.	DECIMAL(5,0)	'-13.'
.13	DECIMAL(2,2)	'.13'
-1.3E56	FLOAT(2)	'-1.3E+056'

Only numerical character strings can be converted to numerical data. Numerical strings are defined as follows:

integer	One optional sign character (+ or -) followed by at least one digit (0-9). Leading and trailing blanks are ignored. No other character is allowed.
decimal	As integer, but with one decimal point (.) placed immediately before or after a digit.
float	As decimal, but followed directly by an uppercase or lowercase letter “E” and an exponent written as a signed integer.

The precision and scale of a number derived from a numerical character string follows the format of the string. Leading and trailing zeros are significant for assigning precision.

Thus:

“3”	gives	INTEGER(1)
“003”	gives	INTEGER(3)
“0.3”	gives	DECIMAL(2,1)
“00.30”	gives	DECIMAL(4,2)
“.3”	gives	DECIMAL(1,1)
“-33”	gives	INTEGER(2)
“-33.”	gives	DECIMAL(2,0)
“003.3E14”	gives	FLOAT(4)

4.3.9 Standard compliance

This section summarizes standard compliance concerning data types.

Standard	Compliance	Comments
X/Open-95 **SQL92**	EXTENDED	Conversion between character and numeric when storing values from or retrieving values into host variables is a Mimer SQL extension. Support for the abbreviation NUM is a Mimer SQL extension. Specifying a precision for INTEGER is a Mimer SQL extension. Support for BINARY, BINARY VARYING and BIGINT is a Mimer SQL extension.

4.4 Literals

Literal values may be given for any of the data types supported in SQL statements, wherever the term “literal” appears in the syntax diagrams.

4.4.1 String literals

String literals may be represented in two ways, as character strings or hexadecimal strings.

A **character string literal** consists of a sequence of characters enclosed in string delimiters. The standard string delimiter is the apostrophe ('). Two consecutive apostrophes within a string are interpreted as a single apostrophe.

Note: An empty string (i.e. '') is a defined value. It is **not** NULL.

A **hexadecimal string literal** is a string specified as a sequence of hexadecimal values, enclosed in apostrophes and preceded by the letter X. The sequence of values must contain an even number of positions (every character in the string literal is represented by a two-position value), and may not contain any characters other than the digits 0-9 and the letters A-F. The case of letters (and of the preceding X) is irrelevant. The code values for characters are those which apply in the host system.

For character and hexadecimal string literals, a separator may be used within the literal to join two or more substrings. Separators are described in Section 4.1. This is particularly useful when a string literal extends over more than one physical line, or when control codes are to be combined with character sequences.

Examples (ASCII codes are used for the hexadecimal literals):

String	**Value**
'ABCD'	ABCD
'Mimer''s'	Mimer's
'data'<LF>'base'	database
X'0D0A09'	<CR><LF><TAB>
X'0D0A'<LF>'09'	<CR><LF><TAB>

Note: Since the SQL92 standard states that a hexadecimal string is a bit-string and Mimer SQL currently does not support a BIT data type, it is advisable to explicitly type cast hexadecimal strings to the CHARACTER data type to assure forward compatibility. This is done with the CAST specification described in Section 4.5.

4.4.2 Numerical integer literals

A numerical integer literal is a signed or unsigned number that does not include a decimal point. The sign is a plus (+) or minus (-) sign immediately preceding the first digit. In determining the precision of an integer literal, leading zeros are significant (i.e. the literal 007 has precision 3).

Examples:

```
  47
-125
+006
   0
```

4.4.3 Numerical decimal literals

A numerical decimal literal is a signed or unsigned number containing exactly one decimal point. In determining the precision and scale of a decimal literal, both leading and trailing zeros are significant (i.e. the literal 003.1400 has precision 7, scale 4).

Examples:

```
   4.7
  -3.
+012.067
   0.0
    .370
```

4.4.4 Numerical floating point literals

Floating point literals are represented in exponential notation, with a signed or unsigned integer or decimal mantissa, followed by an letter E, followed in turn by a signed or unsigned integer exponent. The base for the exponent is always 10. The exponent zero may be used. The case of the letter E is irrelevant. In determining the precision of a floating point literal, leading zeros in the mantissa are significant (i.e. the literal 007E4 has precision 3).

Examples:

1.3E5	value	130000
-4e-2		-0.04
+03.3E2		330
0E+45		0
1.53E00		1.53

4.4.5 DATE, TIME and TIMESTAMP literals

A literal that represents a DATE, TIME or TIMESTAMP value consists of the corresponding keyword shown below, followed by text enclosed in single quotes ('').

The following formats are allowed:

DATE '*date-value*'

TIME '*time-value*'

TIMESTAMP '*date-value* <space> *time-value*'

A *date-value* has the following format:

year-value – *month-value* – *day-value*

A *time-value* has the following format:

hour-value **:** *minute-value* **:** *second-value*

where *second-value* has the following format:

whole-seconds-value [**.** *fractional-seconds-value*]

The *year-value, month-value, day-value, hour-value*, *minute-value*, *whole-seconds-value* and *fractional-seconds-value* are all unsigned integers.

A *year-value* contains **exactly** 4 digits, a *fractional-seconds-value* may contain **up to** 9 digits and all the other components each contain **exactly** 2 digits.

Examples:

DATE '1997-02-19'

TIME '10:59:23'

TIMESTAMP '1997-02-14 10:59:23.4567'

TIMESTAMP '1928-12-25 23:59:30'

4.4.6 Interval literals

An INTERVAL literal represents an INTERVAL value and consists of the keyword INTERVAL followed by text enclosed in single quotes, in the following format:

INTERVAL '[+ | -] *interval-value*' *interval-qualifier*

The *interval-value* text must be a valid representation of a value compatible with the interval data type specified by the *interval-qualifier* (see Section 4.3.3.2).

- If the interval precision includes the YEAR and MONTH fields, the values of these fields should be separated by a minus sign.
- If the interval precision includes the DAY and HOUR fields, the values of these fields should be separated by a space.
- If the interval precision includes the HOUR fields and another field of lower significance (MINUTE and/or SECOND), the values of these fields should be separated by a colon.

All fields may contain up to 2 digits except that:

- The number of digits in the most significant field must not exceed the leading precision defined by the *interval-qualifier*. If a leading precision is not explicitly specified in the *interval*-qualifier, the default (2) applies.
- The SECOND field may have a fractional part, whose maximum length is defined by the *interval-qualifier*.

Examples:

INTERVAL '1:30' HOUR TO MINUTE

INTERVAL '1-6' YEAR TO MONTH

INTERVAL '1000 10:20:30.123' DAY(4) TO SECOND(3)

INTERVAL '-199' YEAR(3) ****evaluates to -199**

INTERVAL '199' YEAR ****Invalid : default leading precision is 2**

INTERVAL '5.555' SECOND(1,2) ****evaluates to 5.55**

INTERVAL '-5.555' SECOND(1,2) ****evaluates to -5.55**

INTERVAL '19 23' DAY TO MINUTE ****Invalid : no minutes in literal**

4.4.7 Standard compliance

This section summarizes standard compliance concerning literals.

Standard	Compliance	Comments
X/Open-95 SQL92	EXTENDED	The presence of a *newline* character (<LF>) between substrings in a character or hexadecimal string literal is not mandatory in Mimer SQL. Hexadecimal string literals are of type BINARY because Mimer SQL does not support the BIT data type.

4.5 Assignments

The following rules apply when values are assigned in SQL statements to database columns or to host variables.

4.5.1 String assignments

If a string value assigned to a fixed-length or variable-length character column is longer than the defined length of the column (except for trailing spaces), the assignment will fail and an error is returned.

If a string value assigned to a fixed-length character column is shorter than the defined length of the column, the content of the column is padded to the right with blanks after the assignment.

If a string value assigned to a variable-length character column is shorter than the defined maximum length of the column, no blank padding occurs.

Character (both fixed length and variable length) column values assigned to fixed-length host variables in SQL statements are padded with blanks to the right if necessary. Column values assigned to host variables are truncated if they are longer than the declared length of the variable, and a warning is issued.

The following table summarizes the rules for character string assignment:

Assignment	Source too long	Source too short
To column	Error if non-blank character would be truncated.	Pad right with blanks for fixed-length columns. No blank padding for variable length columns.
To variable	Truncate and warn.	Pad right with blanks.

4.5.2 Numerical assignments

Numbers assigned to columns or host variables assume the data type of the item to which they are assigned, regardless of the data type of the source.

Integral parts of numbers are never truncated. Fractional parts of decimal and floating point numbers may be truncated if required. No precision is lost when converting integer values to decimal, but this may happen when converting integer values to floating point.

When decimal or floating point values are converted to integers, the fractional part of the number is truncated (not rounded). Note that the range of numbers represented by integers is smaller than the range represented by floating point numbers. Assignment of a floating point number to an integer produces an overflow error if the source number is too large.

In assigning decimal values to decimal targets, the length of the integer part of the source (i.e. the difference between the precision and scale) may not exceed the precision of the target. The necessary number of leading zeros is appended or eliminated, and trailing zeros are added to or digits truncated from the fractional part as required.

Note: Truncation effects can be avoided by explicitly using the ROUND function (see Section 5.5.25).

In converting decimal values to floating point, the mantissa of the target is treated as a decimal number with the same precision as the source (for example, 1234.56 becomes 1.23456E3).

In converting floating point values to decimal, digits are truncated from the fractional part of the result as required by the scale of the target. An overflow error occurs if the precision of the target cannot accommodate the integral part of the result.

The following table illustrates the main features of numerical assignments. (Leading zeros are shown where appropriate to indicate the maximum number of digits available. Leading zeros in numerical data are not normally displayed on output.):

	Target:				
Source value:	INTEGER(9)	INTEGER(2)	DECIMAL(9,2)	FLOAT(9)	FLOAT(2)
INTEGER(6) 987654	000987654	Overflow	0987654.00	9.87654000E5	9.8E5
DECIMAL(6,3) 987.654	000000987	Overflow	0000987.65	9.87654000E2	9.8E2
FLOAT(6) 9.87654E5	000987654	Overflow	0987654.00	9.87654000E5	9.8E5
FLOAT(6) 9.87654E49	Overflow	Overflow	Overflow	9.87654000E49	9.8E49
FLOAT(6) 9.87654E-49	000000000	00	0000000.00	9.87654000E-49	9.8E-49

4.5.3 Datetime and interval assignments

The following compatibility rules apply when assigning DATETIME values to one another:

- If the value to be assigned is a DATE, the target must also be a DATE.
- If the value to be assigned is a TIME, the target must also be a TIME.
- If the value to be assigned is a TIMESTAMP, the target must also be a TIMESTAMP.
- The CAST function can be used in order to cross-assign.

The following compatibility rules apply when assigning INTERVAL values to one another:

- When assigning a non-null value to an INTERVAL column, the leading precision of the target must be sufficient to represent the value.
- All YEAR-MONTH INTERVAL values are compatible with one another.
- All DAY-TIME INTERVAL values are compatible with one another.

4.5.4 Standard compliance

This section summarizes standard compliance concerning assignments.

Standard	Compliance	Comments
X/Open-95 SQL92	YES	Fully compliant.

4.6 Comparisons

Values to be compared must be of compatible data type. If values with incompatible data types are compared, an error occurs.

4.6.1 Character string comparisons

Both fixed-length and variable-length character strings are compared character by character from left to right. If the strings are of different length, the shorter string is conceptually padded to the right with blanks before the comparison is made (i.e. character differences take precedence over length differences).

This means, for example, that the variable-length column with the value 'HOTEL ' (one trailing blank) is equal to the variable-length column with the value 'HOTEL ' (two trailing blanks).

It is the collating sequence that determines whether one character is less than or greater than another.

The collating sequence for characters is an extended ASCII character set as defined by ISO 8859-1, LATIN1 (see Appendix B).

4.6.2 Numerical comparisons

Numerical values are always compared according to their algebraic values.

Integer values compared with decimal or floating point values are treated as decimal or floating respectively. When decimal values are compared with decimal, the lower precision value is conceptually padded with leading and trailing zeros as necessary. Decimal values compared with floating point values are treated as floating. Thus all the following comparisons evaluate to "true":

```
   1 = 1.0
   2 < 2.3E0
35.3 = 035.300
35.3 > 3.5E1
```

4.6.3 Datetime and interval comparisons

Two DATETIME values may be compared if they are assignment-compatible (as defined in Section 4.5.3).

DATETIME comparisons are performed in accordance with chronological ordering.

When two TIME or two TIMESTAMP values are compared, the seconds precision of the value with the lowest seconds precision is extended by adding trailing zeros.

Two INTERVAL values may be compared if they are assignment-compatible (as defined in Section 4.5.3).

INTERVAL comparisons are performed in accordance with their sign and magnitude.

It is not possible to compare YEAR-MONTH intervals with DAY-TIME intervals.

Comparable INTERVAL types with different interval precisions are conceptually converted to the same interval precision, prior to any comparison, by adding fields as required.

4.6.4 NULL comparisons

All comparisons involving NULL on either side of the comparison operator evaluate to "unknown". NULL is never equal to, greater than or less than anything else. SQL provides a special NULL predicate to test for the presence or absence of NULL in a column (see Section 5.9.6).

Considerable care is required in writing search conditions involving columns which may contain NULL. It is often very easy to overlook the effect of NULL comparisons, with the result that rows which should be included in the result table are omitted or vice versa. See Section 4.3 of the Mimer SQL User's Manual for further discussion of this point.

4.6.5 Truth tables

The following truth tables summarize the outcome of conditional expressions where comparisons are negated by NOT or joined by AND or OR. A question mark represents the truth value “unknown”.

NOT	
T	F
F	T
?	?

AND	T	F	?
T	T	F	?
F	F	F	F
?	?	F	?

OR	T	F	?
T	T	T	T
F	T	F	?
?	T	?	?

4.6.6 Standard compliance

This section summarizes standard compliance concerning comparisons.

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	YES	Fully compliant.

4.7 Result data types

This section describes the syntax rules for, and the resulting data types of, UNION or UNION ALL operations specified in a *query-expression* (see SELECT) and CASE expressions (see Section 5.6).

- The data type of all specified expressions must be comparable.
- If any of the specified expressions is a variable-length character string, then the data type of the result will be variable-length character with maximum length equal to the maximum length of the largest of the specified expressions.
- If all specified expressions are fixed-length character strings, then the data type of the result will be a fixed-length character string with a length equal to the maximum length of the largest of the specified fixed-length character string values.
- If any of the specified expressions is variable-length binary, then the data type of the result will be variable-length binary with maximum length equal to the maximum length of the largest of the specified expressions.
- If all specified expressions are fixed-length binary, then the data type of the result will be fixed-length binary with the same length.
- If all specified expressions are exact numeric, then the data type of the result will be exact numeric with precision and scale equal to the maximum precision and scale of the specified expressions.

- If any of the specified expressions is approximate numeric, then the data type of the result will be approximate numeric with precision equal to the maximum precision of the specified expressions.
- If two numeric data types are specified, the precision and scale of the result is determined by the rules in the table below and which are described in the points that follow:

	FLOAT(p")	INTEGER(p")	DECIMAL(p",s")
FLOAT(p')	FLOAT(p)$^{1)}$	FLOAT(p)$^{1)}$	FLOAT(p)$^{1)}$
INTEGER(p')	FLOAT(p)$^{1)}$	INTEGER(p)$^{1)}$	DECIMAL(p,s)$^{2)}$
DECIMAL(p',s')	FLOAT(p)$^{1)}$	DECIMAL(p,s)$^{2)}$	DECIMAL(p,s)$^{2)}$

1) p = max(p',p")

2) p = min(45, max(p'-s',p"-s")+max(s',s"))
   s = max(s',s")

- If either of the specified expressions is floating point, the result is floating point. The precision of the result is the highest operand precision.

  Thus:

  FLOAT(4) UNION FLOAT(2) $\longrightarrow$ FLOAT(4)

  FLOAT(4) UNION FLOAT(20) $\longrightarrow$ FLOAT(20).

- If both the specified expressions are integer, the result is integer. The precision of the result is the highest operand precision.

  Thus:

  INTEGER(4) UNION INTEGER(2) $\longrightarrow$ INTEGER(4)

  INTEGER(4) UNION INTEGER(20) $\longrightarrow$ INTEGER(20).

- If both the specified expressions are decimal, or one is decimal and the other is integer, the result is decimal. For expressions mixing decimal and integer operands, INTEGER(p) is treated as DECIMAL(p,0).

  The number of positions to the left of the decimal point (i.e. the difference between precision and scale) in the result is the greatest number of positions in either operand. The scale of the result is the greatest scale of the operands. The precision may not exceed 45.

  Thus:

  INTEGER(3) UNION DECIMAL(6,4) $\longrightarrow$ DECIMAL(6,4)

  INTEGER(9) UNION DECIMAL(6,4) $\longrightarrow$ DECIMAL(9,4)

  DECIMAL(9,2) UNION DECIMAL(6,4) $\longrightarrow$ DECIMAL(9,4).

- For INTERVAL operands (see Section 4.3.3), the interval precision of the result is the combined interval precision of the two operands, the scale (seconds precision) is the greatest of the two operands and the leading precision of the result is the greatest of the two operands, expressed in terms of the most significant field of the result.

  DAY TO HOUR UNION MINUTE TO SECOND
  $\rightarrow$ DAY TO SECOND

  HOUR TO SECOND(2,2) UNION MINUTE TO SECOND(1,6)
  $\rightarrow$ HOUR TO SECOND(2,6)

  DAY(2) TO HOUR UNION HOUR(6) TO MINUTE
  $\rightarrow$ DAY(5) TO MINUTE.

Standard	Compliance	Comments
X/Open-95 SQL92	YES	Fully compliant.

5 SQL LANGUAGE ELEMENTS

This chapter describes syntactical elements of the SQL language which occur in SQL statements. The elements described here are generally represented in the statement descriptions in Chapter 6 by single parameter words. Users not accustomed to writing SQL statements will need to refer to the syntax descriptions in this chapter in order to construct many of the full SQL statements.

The language elements dealt with here are summarized in the table below:

Language Element	Summary	Section
Operators	Arithmetical, string operators, comparison, relational operators and logical operators.	5.1
Value specifications	Specifying fixed values in expressions.	5.2
Default values	Specifying default values in CREATE statements.	5.3
Set functions	AVG, COUNT, MAX, MIN, SUM.	5.4
Scalar functions	Summary descriptions for these appear in Section 5.5.1.	5.5
CASE expression	Expression by which a conditional value can be specified.	5.6
CAST specification	Data type conversion specification.	5.7
Expressions	General value specifications.	5.8
Predicates	Conditional statements (basic, quantified, BETWEEN, IN, LIKE, NULL, EXISTS, OVERLAPS predicates).	5.9
Search conditions	Composite predicates defining row subsets from tables.	5.10
Select specification	Language elements defining row/column subsets from tables.	5.11
Joined tables	Inner and outer join specifications.	5.12
Select statements	Combinations of one or more subselects.	5.13
Escape clause	Escape clause for vendor-specific SQL.	5.14

5.1 Operators

5.1.1 Arithmetical and string operators

Arithmetical and string operators are used in forming expressions (see Section 5.8).

The operators are:

unary arithmetical	+	leaves operand unchanged
	-	changes sign of operand
binary arithmetical	+	addition
	-	subtraction
	*	multiplication
	/	division
string	||	concatenation

5.1.2 Comparison and relational operators

Comparison operators are used to compare operands in basic and quantified predicates. Relational operators are used to compare operands in all other predicates (see Section 5.9). Both comparison and relational operators perform essentially similar functions. The comparison operators are however common to most programming languages, while the relational operators are more or less specific to SQL.

Comparison operators	=	equal to
	<>	not equal to
	<	less than
	<=	less than or equal to
	>	greater than
	>=	greater than or equal to

Quantifiers	**ALL**
	SOME
	ANY
Relational operators	**BETWEEN**
	EXISTS
	IN
	IS
	LIKE
	OVERLAPS

5.1.3 Logical operators

Logical operators	**AND**
	OR
	NOT

The operators AND and OR are used to combine several predicates to form search conditions (see Section 5.10).

The operator NOT may be used to reverse the truth value of a predicate in forming search conditions. This operator is also available in predicate constructions to reverse the function of a relational operator (see Section 5.10).

5.1.4 Standard compliance

This section summarizes standard compliance concerning operators.

Standard	Compliance	Comments
X/Open-95 SQL92	YES	Fully compliant.

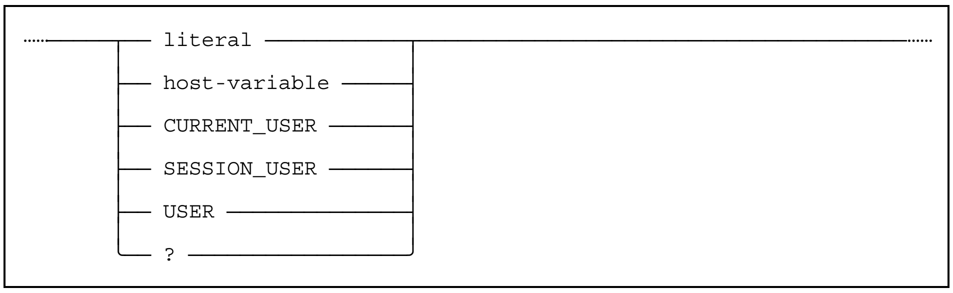
5.2 Value specifications

5.2.1 Description

Value specifications are values which are fixed within the context of one SQL statement (as distinct from values derived from column contents, which may change as different rows or sets of rows are addressed). The value specifications which may be used in expressions are:

- **literals** (see Section 4.4)
- **host variables** (see Section 3.1 in the *Mimer SQL Programmer's Manual*)
- the keyword **CURRENT_USER**, **SESSION_USER** or **USER** representing the name of the current ident (a varying character string with maximum length 128 bytes)
- the parameter marker character (“**?**”), used in dynamic SQL (see Section 7.1 in the *Mimer SQL Programmer's Manual*).

In the syntax diagrams, the term *value-specification* may be replaced by the following construction:



5.2.2 Standard compliance

This section summarizes standard compliance concerning value expressions.

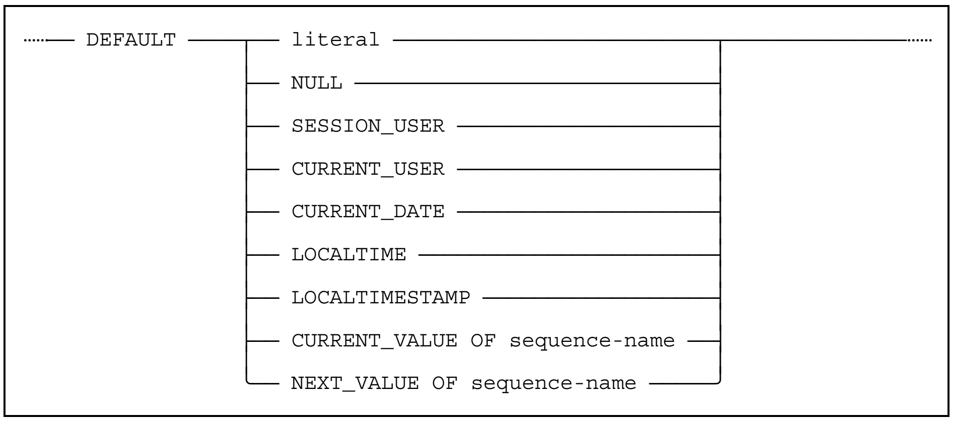
Standard	Compliance	Comments
X/Open-95 SQL92	YES	Fully compliant.

5.3 Default values

5.3.1 Description

There are various places in the Mimer SQL syntax where a default value can be specified. The value resulting from a default value specification must always be assignment-compatible with the data type of the context to which it will be applied.

In the syntax diagrams, the term *default-value* may be replaced by the following construction:



For more information about what can be specified for *literal*, see Section 4.4.

5.3.2 Standard compliance

This section summarizes standard compliance concerning the specification of default values.

Standard	Compliance	Comments
X/Open-95 SQL92	EXTENDED	The support for the use of CURRENT_VALUE and NEXT_VALUE is a Mimer SQL extension.

5.4 Set functions

5.4.1 Description

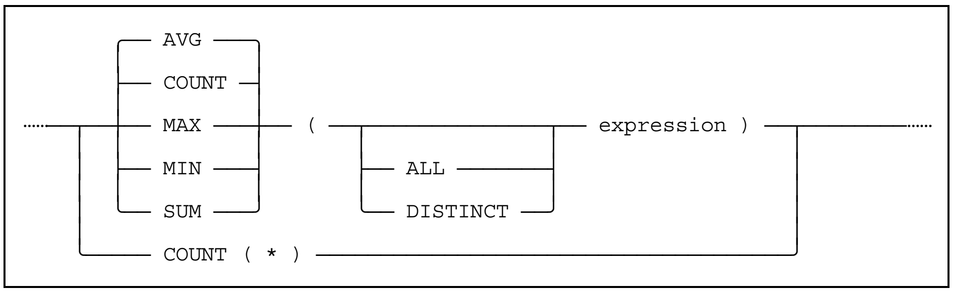
Set functions are pre-defined functions used in select specifications. They operate on the set of values in one column of the result of the SELECT statement, or on the subset in a group if the statement includes a GROUP BY clause. The result of a set function is a single value for each operand set.

The available set functions are:

Set function	Result
AVG	Average of the values in the set.
COUNT	Number of values in the set.
MAX	Largest value in the set.
MIN	Smallest value in the set.
SUM	Sum of the values in the set.

The functions SUM and AVG can only be applied to numerical values.

The general syntax for the *set-function* is:



The operational mode of a set function is determined by the use of the keywords ALL and DISTINCT.

When ALL is specified or no keyword is used:

- Any duplicate values in the operand set are retained.

When DISTINCT is specified:

- Redundant duplicate values are eliminated from the operand set before the function is applied.
- The result of the set function must not be combined with other terms using binary arithmetic operators.
- For the set functions MAX and MIN, the keyword DISTINCT will be ignored if it is used.

For all set functions except COUNT(*), any NULL values in the operand set are eliminated before the set function is applied, regardless of whether DISTINCT is specified or not.

The special form COUNT (*) returns the number of rows in the result table, including any NULL values. The keywords ALL and DISTINCT may not be used with this form of COUNT.

If the operand set is empty, the COUNT function returns the value zero. All other functions return NULL for an empty operand set.

The COUNT function returns an integer with precision 10. The MAX and MIN functions return a value with the same type and precision as the operand. The precision of the result returned by SUM and AVG is considered below.

5.4.2 Restrictions

Column references in the argument of a set function may not address view columns which are themselves derived from set functions.

The argument of a set function must contain at least one column reference and cannot contain any set function references. If the column is an outer reference, then the expression should not include any operators.

If a set function contains a column that is an outer reference, then the set function must be contained in a subselect of a HAVING clause.

5.4.3 Results of set functions

When the argument of a set function is a numerical value, the precision and scale of the set function result is evaluated in accordance with the rules given below. If the argument is an expression, the expression is first evaluated as described in Section 5.8 before the set function is applied.

Evaluation of set functions

	FLOAT(p')	INTEGER(p')	DECIMAL(p', s')
SUM	FLOAT(p)[1)]	INTEGER(p)[3)]	DECIMAL(p, s)[4)]
AVG	FLOAT(p)[1)]	INTEGER(p)[2)]	DECIMAL(p, s)[5)]
MAX, MIN	FLOAT(p)[2)]	INTEGER(p)[2)]	DECIMAL(p, s)[6)]
COUNT	INTEGER(10)	INTEGER(10)	INTEGER(10)

1) p = max(15,p')
2) p = p'
3) p = min(45, 10+p')
4) p = min(45, 10+p')
   s = s'
5) p = min(45, 10+p')
   s = p-(p'-s')
6) p = p'
   s = s'

Some examples of SUM and AVG follow:

SUM(INTEGER(3))	gives	INTEGER(13)
SUM(INTEGER(12))	gives	INTEGER(22)
SUM(DECIMAL(38,10))	gives	DECIMAL(45,10)
SUM(DECIMAL(4,2))	gives	DECIMAL(14,2)
AVG(INTEGER(3))	gives	INTEGER(3)
AVG(INTEGER(12))	gives	INTEGER(12)
AVG(DECIMAL(38,10))	gives	DECIMAL(45,17)
AVG(DECIMAL(4,2))	gives	DECIMAL(14,12)

Note: Often, the average of a series of integers is required as a decimal rather than an integer. This may be achieved by casting the value to a decimal using the CAST function. For example, if the values in the integer column COL are 1, 3 and 6, then AVG(COL) returns 3 but AVG(CAST(COL as decimal(5,4))) returns 3.3333.

5.4.4 Standard compliance

This section summarizes standard compliance concerning set functions.

Standard	Compliance	Comments
X/Open-95 SQL92	YES	Fully compliant.

5.5 Scalar functions

5.5.1 Description

The available scalar functions are:

Scalar numeric functions	Result
ABS	Returns the absolute value of the given numeric expression.
ASCII_CODE	Returns the ASCII code value of the leftmost character in the given string expression, as an integer.
BIT_LENGTH	Returns the number of bits in a string
CHAR_LENGTH (or CHARACTER_LENGTH)	Returns the length of a string.
CURRENT_VALUE	Returns the current value of a sequence.
DAYOFWEEK	Returns the day of the week for the given date expression, expressed as an integer value in the range 1-7, where 1 represents Monday.
DAYOFYEAR	Returns the day of the year for the given date expression, expressed as an integer in the range 1-366.
EXTRACT	Extracts a single field from a DATETIME or INTERVAL value.
IRAND	Returns a random integer number
MOD	Returns the remainder (modulus) of a specified integer expression divided by a second specified integer expression.
NEXT_VALUE	Returns the next value in the series of values defined by a sequence, provided that last value in that series has not already been reached.
OCTET_LENGTH	Returns the octet (byte) length of a string. For single-octet character sets this is the same as CHARACTER_LENGTH.
POSITION	Returns the starting position of the first occurrence of a specified string expression in a given character string, starting from the left of the character string.
ROUND	Returns the given numeric expression rounded to the number of places to the right of the decimal point specified by a given integer expression. If the integer expression is negative, the numeric expression is rounded to a number of places to the **left** of the decimal point specified by the absolute value of the integer expression.
SIGN	Returns an indicator of the sign of the given numeric expression. If the numeric expression is less than zero, -1 is returned. If the numeric expression is equal to zero, 0 is returned. If the numeric expression is greater than zero, 1 is returned.

TRUNCATE	Returns the given numeric expression truncated to a number of places to the right of the decimal point specified by a given integer expression. If the integer expression is negative, the numeric expression is truncated to a number of places to the **left** of the decimal point specified by the absolute value of the integer expression.
WEEK	Returns the week of the year for the given date expression, expressed as an integer value in the range 1-53.

Scalar string functions	**Result**
ASCII_CHAR	Returns the character that has the given ASCII code value. The given ASCII code value should be in the range 0-255.
CURRENT_PROGRAM	Returns the name of a entered program.
LOWER	Converts all uppercase letters in a character string to lowercase.
PASTE	Returns a character string where a specified number of characters, beginning at a given position, have been deleted from a character string and replaced with a given string expression.
REPEAT	Returns a character string composed of a specified string expression repeated a given number of times.
REPLACE	Replaces all occurrences of a given string expression with another string expression in a character string.
SOUNDEX	Returns a character string value containing six digits that represent an encoding of the sound of the given string expression.
SUBSTRING	Extracts a substring from a given string, according to specified start position and length of the substring.
TAIL	Returns the specified number of rightmost characters in a given character string.
TRIM	Removes leading and/or trailing instances of a specified character from a string.
UPPER	Converts all lowercase letters in a character string to uppercase.

Scalar interval functions	**Result**
ABS	Returns the absolute value of the given interval expression.

Datetime "pseudo literals"	**Result**
CURRENT_DATE	Returns a DATE value denoting the current date (i.e. "today").
LOCALTIME	Returns a TIME value denoting the current time (i.e. "now").
LOCALTIMESTAMP	Returns a TIMESTAMP denoting the current date and time.

String "pseudo literals"	Result
CURRENT_USER	Returns the name of the currently connected user or OS_USER ident or the program ident that is currently entered. When used in a routine or trigger, the returns the name of the creator of the schema to which the routine or trigger belongs.
SESSION_USER	Returns the name of the currently connected ident.
USER	Returns the same value as CURRENT_USER (it is recommended that CURRENT_USER be used in preference).

5.5.2 The ABS function

Syntax for the ABS function:

```
·····—— ABS ( value ) ——————————————————————————————————————————·····
```

value is a numeric or an interval value expression.

Rules:

- The function returns the absolute value of *value*.
- If the value of *value* is NULL, then the result of the function is NULL.

5.5.3 The ASCII_CHAR function

Syntax for the ASCII_CHAR function:

```
·····—— ASCII_CHAR ( code ) ————————————————————————————————————·····
```

code is a numeric expression representing an ASCII value.

Rules:

- If the value of *code* is between 0 and 255, the function returns a single character value, i.e. CHAR(1).
- If the value of *code* is **not** between 0 and 255, the function returns NULL.
- If the value of *code* is NULL, then the result of the function is NULL.

5.5.4 The ASCII_CODE function

Syntax for the ASCII_CODE function:

```
·····—— ASCII_CODE ( source-string ) ———————————————————————————·····
```

source-string is a character or binary string expression.

Rules:

- A single INTEGER value is returned, representing an ASCII code.

- If the *source-string* contains more than one character, the ASCII code of the leftmost octet is returned.
- If the length of *source-string* is zero, then the result of the function is NULL.
- If the value of *source-string* is NULL, then the result of the function is NULL.

5.5.5 The BIT_LENGTH function

Syntax for the BIT_LENGTH function:

```
·····—— BIT_LENGTH ( source-string ) ————————————————————————————————·····
```

source-string is a character or binary string expression.

Rules:

- BIT_LENGTH returns an INTEGER value.
- If the data type of *source-string* is variable-length character or variable-length binary, then the result of BIT_LENGTH is the same as the actual length of *source-string* multiplied with 8. (The number of bits in an octet)
- If the data type of *source-string* is fixed-length character or fixed-length binary, then the result of BIT_LENGTH is the same as the fixed-length of *source-string* multiplied by 8.
- If the value of *source-string* is NULL, then the result of the function is NULL.

5.5.6 The CHAR_LENGTH function

Syntax for the CHAR_LENGTH (or CHARACTER_LENGTH) function:

```
·····┬— CHAR_LENGTH ————————┬—— ( source-string ) ——————————————————·····
     └— CHARACTER_LENGTH ———┘
```

source-string is a character or binary string expression.

Rules:

- CHAR_LENGTH returns an INTEGER value.
- If the data type of *source-string* is variable-length character or variable-length binary, then the result of CHAR_LENGTH is the same as the actual length of *source-string*.
- If the data type of *source-string* is fixed-length character or fixed-length binary, then the result of CHAR_LENGTH is the same as the fixed-length of *source-string*.
- If the value of *source-string* is NULL, then the result of the function is NULL.

5.5.7 The CURRENT_DATE function

Syntax for the CURRENT_DATE function:

```
·····──── CURRENT_DATE ──────────────────────────────────────────────────·····
```

Rules:

- The result is the current date (i.e. “today”) as a DATE value.
- In any given SQL statement, all references to CURRENT_DATE are effectively evaluated simultaneously from a single reading of the server clock. Thus the conditional expression CURRENT_DATE = CURRENT_DATE is guaranteed to always evaluate to *true*.
- In any given SQL statement, the value of CURRENT_DATE will always be equal to the DATE portion of LOCALTIMESTAMP.

5.5.8 The CURRENT_PROGRAM function

Syntax for the CURRENT_PROGRAM function:

```
·····──── CURRENT_PROGRAM ( ) ───────────────────────────────────────────·····
```

Rules:

- The function returns the value of the most recently entered program as character varying value with an maximum length of 128.
- If no prgram has been entered the result of the function is NULL

5.5.9 The CURRENT_USER function

Syntax for the CURRENT_USER function:

```
·····──── CURRENT_USER ──────────────────────────────────────────────────·····
```

Rules:

- When used in a routine or trigger, the result is the name of the creator of the schema to which the routine or trigger belongs, otherwise the value is the name of the connected ident or the program that was entered

5.5.10 The CURRENT_VALUE function

Syntax for the CURRENT_VALUE function:

```
──── CURRENT_VALUE OF sequence_name ────────────────────────────────
```

Rules:

- The result is the current value of the sequence specified in *sequence_name*. This is the value that was returned when the NEXT_VALUE function was used for this sequence in this session.
- This function cannot be used until the initial value has been established for the sequence by using NEXT_VALUE (i.e. using it immediately after the sequence has been created will raise an error).
- The function can be used where a value-expression would normally be used. It can also be used after the DEFAULT clause in the CREATE DOMAIN, CREATE TABLE and ALTER TABLE statements.
- USAGE privilege must be held on the sequence in order to use it.

5.5.11 The DAYOFWEEK function

Syntax for the DAYOFWEEK function:

```
──── DAYOFWEEK ( date-or-timestamp ) ───────────────────────────────
```

date-or-timestamp is a date or timestamp value expression.

Rules:

- The result is an integer value, 1 through 7, where 1 = Monday, 2 = Tuesday and so on.
- If the value of *date-or-timestamp* is NULL, then the result of the function is NULL.

5.5.12 The DAYOFYEAR function

Syntax for the DAYOFYEAR function:

```
──── DAYOFYEAR ( date-or-timestamp ) ───────────────────────────────
```

date-or-timestamp is a date or timestamp value expression.

Rules:

- The result is an integer value, 1 through 366, where 1 = January 1st.
- The value for a day after February 28th depends on whether the year is a leap year or not.
- If the value of *date-or-timestamp* is NULL, then the result of the function is NULL.

5.5.13 The EXTRACT function

Syntax for the EXTRACT function:

```
·····—— EXTRACT ( field-name FROM value ) ——————————————————————————·····
```

Rules:

- *field-name* is one of: YEAR, MONTH, DAY, HOUR, MINUTE or SECOND.
- *value* must be of type DATETIME or INTERVAL and it must contain the field specified by *field-name*, otherwise an error is raised.
- The data type of the result is exact numeric with a precision equal to the leading precision of *value* and a scale of 0. The exception is when *field-name* is SECOND, in which case the precision is equal to the sum of the leading precision and the seconds precision of *value*, with a scale equal to the seconds precision.
- When *value* is a negative INTERVAL, the result is a negative value. In all other cases the result is a positive value.
- If the value of *value* is NULL, then the result of the function is NULL.

5.5.14 The IRAND function

Syntax for the IRAND function:

```
·····—— IRAND ( ——————————————————— ) ——————————————————————————·····
                   └——— seed ———┘
```

seed is an integer value expression

Rules:

- The result is a random integer value.
- If a seed is given, this value is used to calculate the random value. If no seed is given, the value is calculated from the previous value.It is thus possible to generate the same random sequence by using the same seed.

5.5.15 The LOCALTIME function

Syntax for the LOCALTIME function:

```
·····—— LOCALTIME ——————————————————————————————————————————————·····
                     └—— ( seconds-precision ) ——┘
```

seconds-precision is an unsigned integer value denoting the seconds precision for the returned TIME value.

Rules:

- The result is the current time (i.e. “now”) as a TIME value.
- The value of *seconds-precision* must be between 0 and 9.

- If *seconds-precision* is not specified, the default value of 0 is assumed.
- In any given SQL statement, all references to LOCALTIME are effectively evaluated simultaneously from a single reading of the server clock. Thus the conditional expression LOCALTIME = LOCALTIME is guaranteed to always evaluate to *true*.
- In any given SQL statement, the value of LOCALTIME will always be equal to the TIME portion of LOCALTIMESTAMP.

5.5.16 The LOCALTIMESTAMP function

Syntax for the LOCALTIMESTAMP function:

```
·····──── LOCALTIMESTAMP ─────┬──────────────────────────┬──────────────────────·····
                              └── ( seconds-precision ) ─┘
```

seconds-precision is an unsigned integer value denoting the seconds precision for the returned TIME value.

Rules:

- The result is the current date and time as a TIMESTAMP value.
- The value of *seconds-precision* must be between 0 and 9.
- If *seconds-precision* is not specified, the default value of 6 is assumed.
- In any given SQL statement, all references to LOCALTIMESTAMP are effectively evaluated simultaneously from a single reading of the server clock. Thus the conditional expression LOCALTIMESTAMP = LOCALTIMESTAMP is guaranteed to always evaluate to *true*.
- In any given SQL statement, the value of LOCALTIMESTAMP will always be equal to the combined value of CURRENT_DATE and LOCALTIME.

5.5.17 The LOWER function

Syntax for the LOWER function:

```
·····──── LOWER ( source-string ) ──────────────────────────────────────────·····
```

source-string is a character string expression.

Rules:

- The data type of the result is the same as the data type of *source-string*.
- *source-string* is assumed to be in ISO 8859-1 format. All conversions to lowercase defined in the ISO 8859-1 standard are handled by LOWER, including national characters.
- If the value of *source-string* is NULL, then the result of the function is NULL.

5.5.18 The MOD function

Syntax for the MOD function:

```
·····——— MOD ( integer-expression-1 , integer-expression-2 ) ————————————······
```

integer-expression-1 and *integer-expression-2* are integer value expressions.

Rules:

- The result is the remainder of *integer-expression-1* divided by *integer-expression-2*.
- If the value of *integer-expression-2* is zero, a divide-by-zero error will be raised.
- The sign of the result is the same as the sign of *integer-expression-1*.
- If the value of either operand is NULL, then the result of the function is NULL.

5.5.19 The NEXT_VALUE function

Syntax for the NEXT_VALUE function:

```
·····——— NEXT_VALUE OF sequence_name ————————————————————————————————————······
```

Rules:

- The result will be the next value in the series of the values defined by the sequence specified in *sequence_name* (this value will then become the current value for the sequence).
- If the sequence is unique and the current value of the sequence specified in *sequence_name* is already equal to the last value in the series of the values defined by it an error will be raised and the current value of the sequence will remain unchanged.
- If the sequence is non-unique, the function will always succeed. If the current value of the sequence specified in *sequence_name* is equal to the last value in the series of values generated by the sequence, the initial value of the sequence will be returned.
- The function can be used where a value-expression would normally be used. It can also be used after the DEFAULT clause in the CREATE DOMAIN, CREATE TABLE and ALTER TABLE statements.
- This function is used to establish the initial value of the sequence after it has been created using the CREATE SEQUENCE statement.
- USAGE privilege must be held on the sequence in order to use it.

5.5.20 The OCTET_LENGTH function

Syntax for the OCTET_LENGTH function:

```
·····—— OCTET_LENGTH ( source-string ) ——————————————————————————————————·····
```

source-string is a character or binary string expression.

Rules:

- OCTET_LENGTH returns an INTEGER value.
- If the data type of *source-string* is variable-length character or variable length binary, then the result of OCTET_LENGTH is the same as the actual length of *source-string* in octets.
- If the data type of *source-string* is fixed-length character or fixed-length binary, then the result of OCTET_LENGTH is the same as the fixed-length of *source-string*.
- If the value of *source-string* is NULL, then the result of the function is NULL.

5.5.21 The PASTE function

Syntax for the PASTE function:

```
·····—— PASTE ( string-1 , start , length , string-2 ) ————————————————·····
```

string-1 and *string-2* are character or binary string expressions. *string-1* and *string-2* must be of the same type, i.e. either both character or both binary.

start and *length* are integer value expressions.

Rules:

- The *length* characters in *string-1*, starting from position *start* are deleted from *string-1*. Then *string-2* is inserted into *string-1*, at the “point of deletion”. The resulting character or binary string is returned.
- If the value of *length* is positive, the *length* characters **to the right** of *start* are deleted. If the value of *length* is negative, the *length* characters **to the left** of *start* are deleted. The “point of deletion” is where the cursor would be if you had just used a text editor to select the characters, as described, and performed an edit-cut operation.
- A value for *start* of less than 1 (zero or negative) specifies a position to the **left** of the beginning of *string-1*. It is possible that the specified “deletion” may not actually affect any of the characters of *string-1*, in which case the paste operation produces the effect of a prepend.
- If the value of any operand is NULL, then the result of the function is NULL.

5.5.22 The POSITION function

Syntax for the POSITION function:

```
·····—— POSITION ( sub-string IN source-string ) ——————————————————————·····
```

sub-string and *source-string* are character or binary string expressions. *sub-string* and *source-string* must be of the same type, i.e. either both character or both binary.

Rules:

- The position of the first occurrence of *sub-string* in *source-string* is returned, starting from position 1 in *source-string* (the leftmost position).
- If *sub-string* does not occur in *source-string*, the functions returns zero.
- If the length of *source-string* is zero, the function returns zero.
- If the length of *sub-string* is zero, the function returns 1.
- If the value of either operand is NULL, then the result of the function is NULL.

5.5.23 The REPEAT function

Syntax for the REPEAT function:

```
·····—— REPEAT ( sub-string , repeat-count ) ——————————————————————————·····
```

sub-string is a character or binary string expression.
repeat-count is an integer expression.

Rules:

- The result is a character or binary string consisting of *sub-string* repeated *repeat-count* times.
- If the value of *repeat-count* is zero, then the result of the function is a character or binary string of length zero.
- If the value of *repeat-count* is less than zero, then the result of the function is NULL.
- If the value of either operand is NULL, then the result of the function is NULL.

5.5.24 The REPLACE function

Syntax for the REPLACE function:

```
·····—— REPLACE ( source-string , string-1 , string-2 ) ———————————————·····
```

source-string, string-1 and *string-2* are character or binary string expressions. *source-string, string-1* and *string-2* must be of equal type, i.e. either all are character or all are binary.

Rules:

- All occurrences of *string-1* found in *source-string* are replaced with *string-2*, the resulting character or binary string is returned.
- If the value of any of the operands is NULL, then the result of the function is NULL.

5.5.25 The ROUND function

Syntax for the ROUND function:

```
·····——— ROUND ( numeric-value , integer-value ) ————————————————————————·····
```

numeric-value is an integer or a float value expression.

integer-value is an integer value expression.

Rules:

- If *integer-value* is positive, the value describes the number of digits permitted in *numeric-value*, after rounding, to the **right** of the decimal point, if it is negative it describes the number of digits allowed to the **left** of the decimal point.
- The value returned depends on the data type of *numeric-value.*
- If the value of either operand is NULL, then the result of the function is NULL.

5.5.26 The SESSION_USER function

Syntax for the SESSION_USER function:

```
·····——— SESSION_USER ———————————————————————————————————————————————————·····
```

Rules:

- The result is the name of the current ident (i.e. the ident who established the current database connection).

5.5.27 The SIGN function

Syntax for the SIGN function:

```
·····——— SIGN ( numeric-value ) —————————————————————————————————————————·····
```

numeric-value is an integer or a float value expression.

Rules:

- The function returns an indicator of the sign of *numeric-value*. If *numeric-value* is less than zero, -1 is returned. If *numeric-value* equals zero, 0 is returned. If *numeric-value* is greater than zero, 1 is returned.
- If the value of *numeric-value* is NULL, then the result of the function is NULL.

5.5.28 The SOUNDEX function

Syntax for the SOUNDEX function:

```
·····─── SOUNDEX ( source-string ) ──────────────────────────────────·····
```

source-string is a character string expression.

Rules:

- The function returns a character string value containing six digits that represent an encoding of the sound of *source-string*.
- If *source-string* contains two or more words, they are effectively concatenated into a single word by ignoring the separating space characters.
- If the SOUNDEX values for two strings compare to be equal then they sound the same.

5.5.29 The SUBSTRING function

Syntax for the SUBSTRING function:

```
·····─── SUBSTRING ( source-string FROM start-position ─────────────·····

·····──┬─────────────────────┬─ ) ───────────────────────────────────·····
       └─ FOR string-length ─┘
```

source-string is a character or binary string expression.
start-position and *string-length* are integer value expressions.

Rules:

- SUBSTRING returns a character or binary string containing *string-length* characters of *source-string*, starting at the character specified by *start-position*, and in the same sequence as they appear in *source-string*. If any of these positions are before the start or after the end of *source-string*, then no character is returned for that position. If all positions are outside the source string, an empty string is returned.
- The first character in *source-string* has position 1.
- If the data type of *source-string* is variable-length character, then the result of the SUBSTRING function is a variable-length character with maximum string length equal to the maximum length of *source-string*.
- If the data type of *source-string* is fixed-length character, then the result of the SUBSTRING function is a variable-length character with maximum string length equal to the fixed length of *source-string*.
- If the data type of *source-string* is variable-length binary, then the result of the SUBSTRING function is a variable-length binary with maximum string length equal to the maximum length of *source-string*.
- If the data type of *source-string* is fixed-length binary, then the result of the SUBSTRING function is a variable-length binary with maximum string length equal to the fixed length of *source-string*.

- If *string-length* is negative, or if *start-position* is greater than the number of characters in *source-string,* the function fails and an error is returned.
- If *string-length* is omitted then it is assumed to be:
  `CHAR_LENGTH(`*`source-string`*`) + 1 -`*`start-position`*`.`
  i.e. the remainder of *source-string*, starting at *start-position*, is returned.
- If the value of any operand is NULL, then the result of the function is NULL.

5.5.30 The TAIL function

Syntax for the TAIL function:

```
·····─── TAIL ( source-string , count ) ──────────────────────────────·····
```

source-string is a character or binary string expression.
count is an integer value expression.

Rules:

- The rightmost *count* characters of *source-string* are returned.
- If *count* is zero, an empty string is returned.
- If *count* is less than zero, then the result of the function is NULL.
- If the value of either operand is NULL, then the result of the function is NULL.

5.5.31 The TRIM function

Syntax for the TRIM function:

```
·····─── TRIM ( ─┬──────────────────────────────────────────┬─────────·····
                 ├──────────────── trim-character ─┬─ FROM ─┘
                 ├─ LEADING ──┬───────────────────┤
                 ├─ TRAILING ─┤  └─ trim-character ─┘
                 └─ BOTH ─────┘

·····─── source-string ) ─────────────────────────────────────────────·····
```

trim-character is a character or binary string expression of length 1.
source-string is a character or binary string expression. *source-string* and *trim-character* must be of equal type, i.e. either must both be character or both binary.

Rules:
(LEADING, TRAILING or BOTH is referred to as the *trim-specification* below)

- If the data type of *source-string* is variable-length character, then the result of the TRIM function is a variable-length character with maximum string length equal to the maximum length of *source-string*.

- If the data type of *source-string* is fixed-length character, then the result of the TRIM function is a variable-length character with maximum string length equal to the length of *source-string*.
- If the data type of *source-string* is variable-length binary, then the result of the TRIM function is a variable-length binary with maximum string length equal to the maximum length of *source-string*.
- If the data type of *source-string* is fixed-length binary, then the result of the TRIM function is a variable-length binary with maximum string length equal to the length of *source-string*.
- If *trim-specification* is not specified, BOTH is implicit.
- If *trim-character* is not specified, ' ' (space) is implicit.
- If the length of *trim-character* is not 1, an error is returned.
- If the value of either operand is NULL, then the result of the function is NULL.

5.5.32 The TRUNCATE function

Syntax for the TRUNCATE function:

```
·····—— TRUNCATE ( numeric-value , integer-value ) ————————————————————————······
```

numeric-value is an integer or a float value expression.

integer-value is an integer value expression.

Rules:

- If *integer-value* is positive, the value describes the number of digits permitted in *numeric-value*, after truncation, to the **right** of the decimal point, if it is negative it describes the number of digits allowed to the **left** of the decimal point.
- The value returned depends on the data type of *numeric-value.*
- If the value of either operand is NULL, then the result of the function is NULL.

5.5.33 The UPPER function

Syntax for the UPPER function:

```
·····—— UPPER ( source-string ) ————————————————————————————————————————————······
```

source-string is a character string expression.

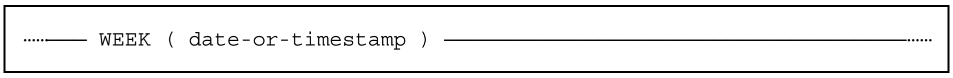
Rules:

- The data type of the result is the same as the data type of *source-string*.
- *source-string* is assumed to be in ISO 8859-1 format. All conversions to uppercase defined in the ISO 8859-1 standard is handled by UPPER, including national characters.

- If the value of *source-string* is NULL, then the result of the function is NULL.

5.5.34 The WEEK function

Syntax for the WEEK function:



date-or-timestamp is a date or timestamp value expression.

Rules:

- The result is an integer value, 1 through 53, representing the week number in the year, calculated in accordance with the ISO 8601 standard.
- If the value of *date-or-timestamp* is NULL, then the result of the function is NULL.

5.5.35 Standard compliance

This section summarizes standard compliance concerning the scalar functions.

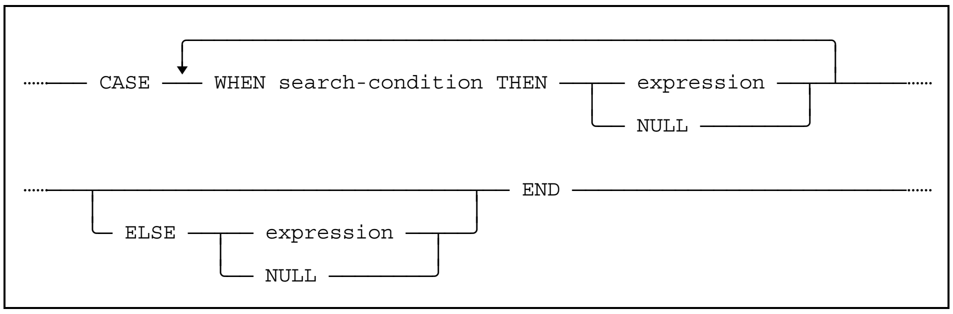
Standard	**Compliance**	**Comments**
X/Open-95 SQL92	EXTENDED	Support for the ABS, ASCII_CHAR, ASCII_CODE, CURRENT_VALUE, DAYOFWEEK, DAYOFYEAR, IRAND, MOD, NEXT_VALUE, PASTE, REPEAT, REPLACE, ROUND, SIGN, SOUNDEX, TAIL, TRUNCATE and WEEK functions is a Mimer SQL extension.

5.6 CASE expression

5.6.1 Description

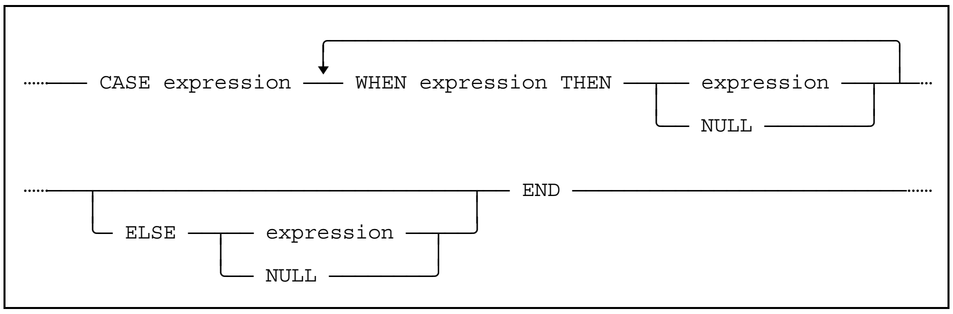
With a case expression it is possible to specify a conditional value. Depending on the result of one or more conditional expressions the case expression can return different values.

A CASE expression can be in one of the following two forms:



Example:

```
CASE WHEN col1 < 10  THEN 1
     WHEN col1 >= 10 THEN 2
     ELSE 3
     END
```



Example:

```
CASE col1 WHEN  0  THEN NULL
          WHEN -1  THEN -999
          ELSE col1
          END
```

Rules:

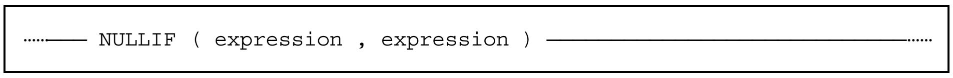
- If one or more *search-conditions* are true, then the result of the CASE expression is the result of the first (leftmost) WHEN clause which has a *search-condition* that is true.
- If none of the *search-conditions* are true, then the result of the CASE expression is the result of the explicit or implicit ELSE clause.
- If no ELSE clause is specified then ELSE NULL is implicit.
- At least one result in a CASE expression must express a value different from NULL.

See Section 4.7 for a description of how the data type of the result of the CASE expression is determined.

5.6.2 Short forms for CASE

There are two short forms for special CASE expressions.

The first short form for CASE is NULLIF:



where

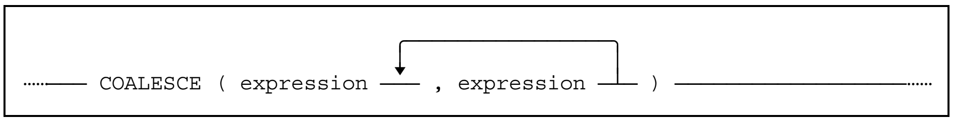
```
NULLIF(x1, x2)
```

is equivalent to

```
CASE WHEN x1=x2 THEN NULL ELSE x1 END
```

i.e. if both operands are equal, the NULLIF expression has the value NULL, otherwise it has the value of the first operand.

The second short form for CASE is COALESCE:



where

```
COALESCE(x1,x2)
```

is equivalent to

```
CASE WHEN x1 IS NOT NULL THEN x1 ELSE x2 END
```

and

```
COALESCE(x1,x2,...,xn)
```

is equivalent to

```
CASE WHEN x1 IS NOT NULL THEN x1
ELSE COALESCE(x2,...,xn) END
```

i.e. the COALESCE expression returns the value of the first non-NULL operand, found by working from left to right, or NULL if all the operands equal NULL.

5.6.3 Standard compliance

This section summarizes standard compliance concerning the CASE expression.

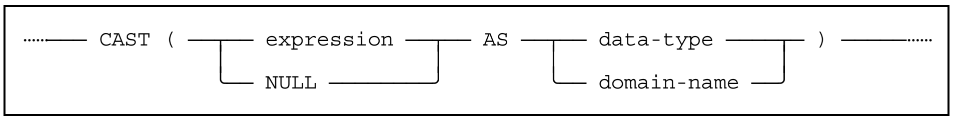
Standard	**Compliance**	**Comments**
X/Open-95 **SQL92**	YES	Fully compliant.

5.7 CAST specification

5.7.1 Description

With the CAST specification it is possible to specify a data type conversion. CAST converts the value of an expression to a specified data type.

Syntax for CAST:



Example:

```
SELECT CAST(floatcol AS DECIMAL(15,3)),
       CAST(charcol AS VARCHAR(10)),
       CAST(intcol AS CHAR(15)),
       CAST(decimcol AS FLOAT(10)) FROM types_tab;
```

Rules:

- *data-type* can be any SQL data type supported by Mimer SQL.
- *domain-name* must be a user-defined domain. The use of domains is not permitted in routines or triggers.
- When converting a numeric or character value to fixed-length character, the value of the source expression is padded with trailing spaces, if the length of the converted value is shorter than the length of the target data type.
- When converting a numeric or character value to variable-length character, no trailing spaces are padded.
- A character value can be converted to a character value of another character type and/or another length if the value to convert is not longer than the length of the target (for CHARACTER) or the maximum length of the target (for VARCHAR).
- A character value can be converted to a fixed-length binary value of equal length.
- A character value can be converted to a variable-length binary value of another length if the value to convert is not longer than the length of the target (for CHARACTER) or the maximum length of the target (for VARCHAR).
- Character values can be converted to a numeric data type if the character string consists of a valid literal representation of the target data type.
- Character values can be converted to a DATETIME or INTERVAL data type provided *expression* conforms to the natural limits placed on date/time values by the Gregorian calendar.
- When a DATE is converted to a TIMESTAMP, the HOUR, MINUTE and SECOND fields of the target are set to zero. The other fields are set to the corresponding values in the source expression.
- When a TIME is converted to a TIMESTAMP, the respective values for the YEAR, MONTH and DAY fields of the target are obtained by evaluating CURRENT_DATE. The other fields are set to the corresponding values in the source expression.
- When a TIMESTAMP is converted to a DATE or TIME, the fields of the target are set to the corresponding values in the source expression. Any values in the source expression for which there are no corresponding fields in the target are ignored.
- When converting from a single field INTERVAL to an exact numeric value, it must be possible to represent the INTERVAL value as an exact numeric value without the loss of leading significant digits.
- When converting from an exact numeric value to a single field INTERVAL, it must be possible to represent the exact numeric as an INTERVAL value without the loss of leading significant digits.
- If CAST is applied on NULL, or if *expression* results in NULL, then CAST returns NULL.

5.7.2 Standard compliance

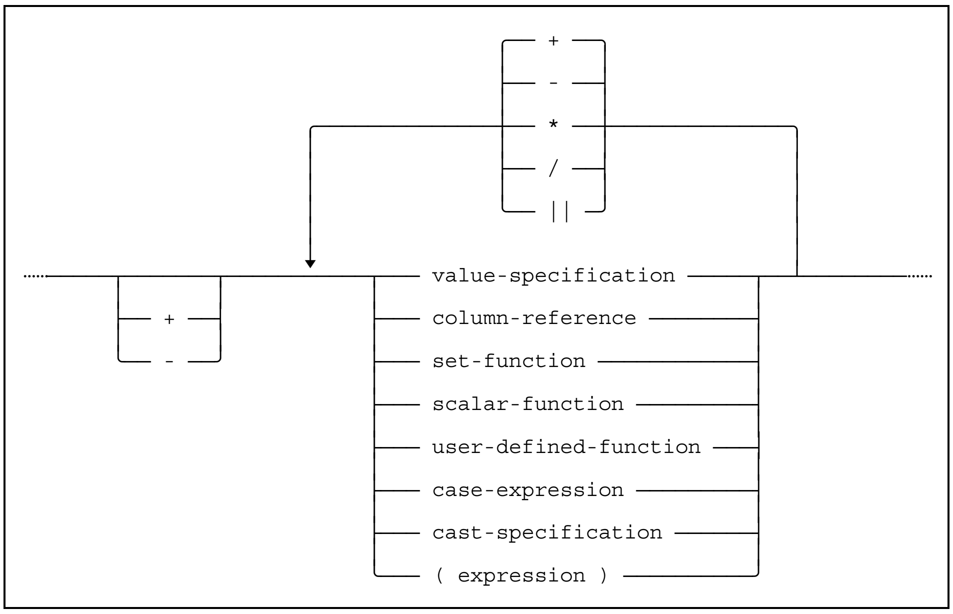
This section summarizes standard compliance concerning the CAST specification.

Standard	Compliance	Comments
X/Open-95 SQL92	YES	Fully compliant.

5.8 Expressions

Expressions are used in a variety of contexts within SQL statements, particularly in search condition predicates and the SET clause in UPDATE statements respectively. An expression always evaluates to a single value.

The syntax of an expression is as follows:



<u>Note:</u> A *user-defined-function* is created by using the CREATE FUNCTION statement.

5.8.1 Unary operators

The prefix operator **+** (unary plus) does not change its operand.

The prefix operator **-** (unary minus) reverses the sign of its operand.

5.8.2 Binary operators

The binary operators specify addition (+), subtraction (-), multiplication (*) and division (/) for numerical operands, and concatenation (||) for string operands.

The operand of a binary operator may not be a set function that includes the keyword DISTINCT.

5.8.3 Operands

When a column name is used as an operand, it represents the single value contained in the column for the row currently addressed when the expression is evaluated. The column name may be qualified by the name of the table or view (see Section 4.2).

5.8.4 Evaluating arithmetical expressions

Expressions within parentheses are evaluated first. When the order of evaluation is not specified by parentheses, the customary arithmetical rules apply, i.e.

- multiplication and division are performed before addition and subtraction
- operators with the same precedence are applied from left to right.

If any operand in an expression is NULL, the whole expression evaluates to NULL. No other expressions evaluate to NULL. Division by zero results in a run-time error.

Arithmetical expressions with mixed numerical and character data are illegal. Note however that where host variables are used in expressions, type conversion may result in apparently incompatible data types being accepted (see Section 4.3).

The type and precision of the result of an arithmetical expression is determined in accordance with the rules described below. If there are more than two operands in an expression, the type and precision of the result is derived in accordance with the sequence in which the component binary operations are performed.

Formally, the arithmetical rules are summarized as follows:

	FLOAT(p")	INTEGER(p")	DECIMAL(p", s")
FLOAT(p')	FLOAT(p)1)	FLOAT(p)1)	FLOAT(p)1)
INTEGER(p')	FLOAT(p)1)	INTEGER(p)2)	DECIMAL(p, s)3)
DECIMAL(p', s')	FLOAT(p)1)	DECIMAL(p, s)3)	DECIMAL(p, s)3)

1)	p = max(15, p', p")	
2)	operator +, -	p = min(45, max(p', p")+1)
	operator *	p = min(45, p'+p")
	operator /	p = p'
3)	operator +, -	p = min(45, max(p'-s', p"-s")+max(s', s")+1)
		s = max(s', s")
	operator *	p = min(45, p'+p")
		s = min(45, s'+s")

operator /	p = min(45, max(15, p'+p"))
	s = p-(p'-s')-s"

In descriptive terms, the rules are as follows:

- If any of the operands is floating point, the result is floating point.

  For all arithmetic expressions, the precision of the result is the highest operand precision. However, the precision is never less than 15. Thus

FLOAT(4) + FLOAT(6)	gives	FLOAT(15)
FLOAT(20) - FLOAT(32)	gives	FLOAT(32)
FLOAT(4) * FLOAT(4)	gives	FLOAT(15)
FLOAT(4) / FLOAT(20)	gives	FLOAT(20)

- If all the operands are integer, the result is integer.

  For addition and subtraction, the precision of the result is the precision of the highest operand plus 1. However, the precision may not exceed 45.

  For multiplication, the precision of the result is the sum of the precisions of the operands. However, the precision may not exceed 45.

  For division, the precision of the result is the precision of the dividend. Thus

INTEGER(3) + INTEGER(5)	gives	INTEGER(6)
INTEGER(20) - INTEGER(30)	gives	INTEGER(31)
INTEGER(5) * INTEGER(18)	gives	INTEGER(23)
INTEGER(4) / INTEGER(6)	gives	INTEGER(4)

- If all the operands are decimal, or decimal and integer operands are mixed, the result is decimal. For expressions mixing decimal and integer operands, INTEGER(p) is treated as DECIMAL(p,0).

  For addition and subtraction, the number of positions to the left of the decimal point (i.e. the difference between precision and scale) in the result is the greatest number of positions in any operand plus 1. The scale of the result is the greatest scale of any of the operands. The precision may not exceed 45. Thus

INTEGER(3) + DECIMAL(6,3)	gives	DECIMAL(7,3)
DECIMAL(4,2) - DECIMAL(8,5)	gives	DECIMAL(9,5)

  For multiplication, the precision of the result is the sum of the precisions of the operands. The scale of the result is the sum of the scales of the operands. Neither the precision nor the scale may exceed 45. If the value of the result does not fit into the precision and scale, overflow occurs. Thus

INTEGER(3) * DECIMAL(6,3)	gives	DECIMAL(9,3)
DECIMAL(4,2) * DECIMAL(8,5)	gives	DECIMAL(12,7)
DECIMAL(12,7) * DECIMAL(10,2)	gives	DECIMAL(22,9)
DECIMAL(25,0) * DECIMAL(25,25)	gives	DECIMAL(45,25)

For division, the precision of the result is the sum of the precisions of the operands. The precision is however never less than 15 and may not exceed 45. The scale of the result is calculated as the precision of the result, less the number of positions to the left of the decimal point in the dividend, less the scale of the divisor. An error occurs if this calculation gives a negative value for the scale. Thus

INTEGER(3) / DECIMAL(6,3)	gives	DECIMAL(15,9)
DECIMAL(4,2) / DECIMAL(8,5)	gives	DECIMAL(15,8)
DECIMAL(12,7) / DECIMAL(10,2)	gives	DECIMAL(22,15)
DECIMAL(25,0) / DECIMAL(25,25)	gives	DECIMAL(45,0)
DECIMAL(20,0) / DECIMAL(20,20)	gives	DECIMAL(40,0)
DECIMAL(45,0) / DECIMAL(45,45)	gives	error (scale = -45)

5.8.5 Evaluating string expressions

The result of a string concatenation expression is a string containing the first operand string directly followed by the second. If string literals or fixed-length host variables are concatenated, any trailing blanks in the operands are retained. If a fixed-length character column value is directly concatenated with another string, any trailing blanks in the column value up to the defined fixed length of the column are retained. If a variable-length character column value is directly concatenated with another string, any trailing blanks in the column value up to the actual length of the column value are retained.

If a fixed-length character value and a variable-length character value are concatenated, the result will be a variable-length character value.

If either of the operands in a concatenation expression is NULL, the result of the expression is NULL.

5.8.6 Standard compliance

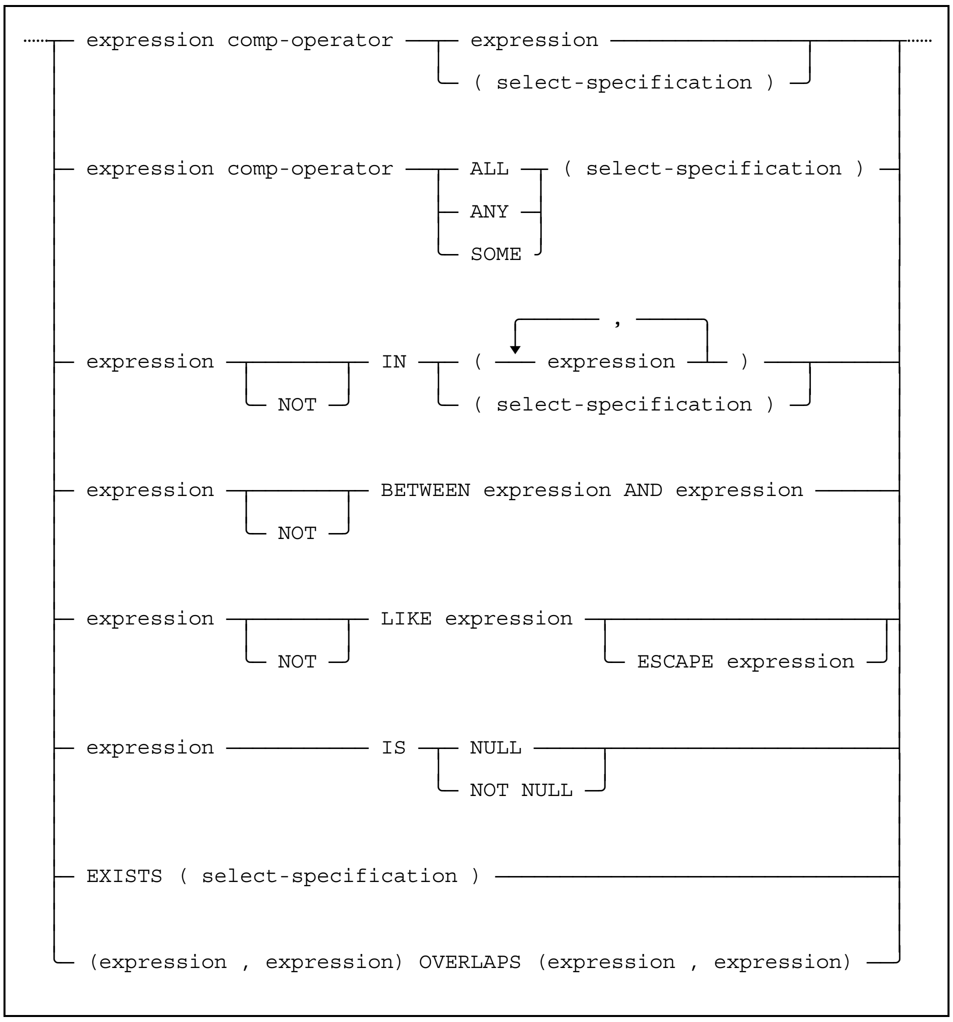
This section summarizes standard compliance concerning the expressions.

Standard	Compliance	Comments
X/Open-95 SQL92	YES	Fully compliant.

5.9 Predicates

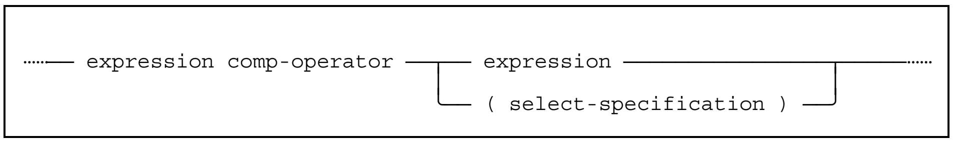
A predicate is a single conditional expression which evaluates to either true, false or unknown. Predicates are used in constructing search conditions (see Section 5.10).

The general predicate syntax is given below. Each individual predicate construction is explained in more detail in the following sections.



5.9.1 The basic predicate

A basic predicate compares a value with **one and only one** other value, and has the syntax:



The comparison operators (*comp-operator*) are described in Section 5.1.2.

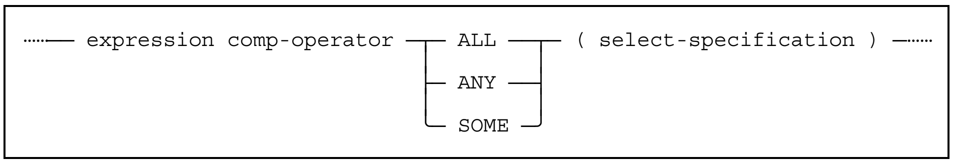
The expressions on either side of the comparison operator must have compatible data types (see Section 4.6).

Within the context of a basic predicate, a select-specification must result in either an empty set or a single value.

The result of the predicate is unknown if either of the expressions used evaluates to NULL, or if the select-specification used results in an empty set.

5.9.2 The quantified predicate

A quantified predicate compares an expression with a **set of values** addressed by a subselect (as opposed to a basic predicate which compares two single-valued expressions). The form of the quantified expression is:



The comparison operators (*comp-operator*) are described in Section 5.1.2.

Within the context of a quantified predicate, a select-specification must result in either an empty set or a set of single values.

ALL predicate

The result is true if the select-specification results in an empty set or if the comparison is true for every value returned by the select-specification.

The result is false if the comparison is false for at least one value returned by the select-specification.

The result is unknown if any of the values returned by the select-specification is NULL and no value is false.

ANY or SOME predicates

The keywords ANY and SOME are equivalent.

The result is true if the comparison is true for at least one value returned by the select-specification.

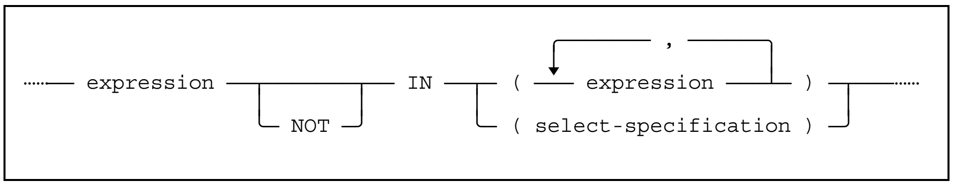
The result is false if the select-specification results in an empty set or if the comparison is false for every value returned by the select-specification.

The result is unknown if any of the values returned by the select-specification is NULL and no value is true.

Quantified predicates may always be replaced by alternative formulations using EXISTS, which can often clarify the meaning of the predicates.

5.9.3 The IN predicate

An IN predicate tests whether a value is contained in a set of discrete values and has the form:



If the set of values on the right hand side of the comparison is given as an explicit list, an IN predicate may always be expressed in terms of a series of basic predicates linked by one of the logical operators AND or OR:

IN predicate	Equivalent basic predicates
x IN (a,b,c)	x = a OR x = b OR x = c
x NOT IN (a,b,c)	x <> a AND x <> b AND x <> c

If the set of values is given as a select-specification, an IN predicate is equivalent to a quantified predicate:

IN predicate	Equivalent quantified predicate
x IN (subselect)	x = ANY (subselect)
x NOT IN (subselect)	x <> ALL (subselect)

The result of the IN predicate is unknown if the equivalent predicates give an unknown result.

5.9.4 The BETWEEN predicate

A BETWEEN predicate tests whether or not a value is within a range of values (including the given limits). It has the form:

```
·····─── expression ─────┬─────────┬─── BETWEEN expression AND expression ───────·····
                         └── NOT ──┘
```

The BETWEEN predicate can always be expressed in terms of two basic predicates. Thus

BETWEEN predicate	Equivalent basic predicates
x BETWEEN a AND b	x >= a AND x <= b
x NOT BETWEEN a AND b	x < a OR x > b

All expressions in the predicate must have compatible data types.

The result of the predicate is unknown if the equivalent basic predicates give an unknown result.

5.9.5 The LIKE predicate

A LIKE predicate compares the value in a string expression with a character string pattern which may contain wildcard characters (meta-characters). It has the form:

```
·····─── string-value ─────┬─────────┬─── LIKE character-pattern ──────────────·····
                           └── NOT ──┘

·····──────────┬──────────────────────────────┬────────────────────────────·····
               └── ESCAPE character-value ────┘
```

The *string-value* on the left hand side of the LIKE operator must be a string expression.

The *character-pattern* on the right hand side of the LIKE operator is a string expression that can be specified as a string literal or by using a host variable. The following meta-characters (wildcards) may be used in the *character-pattern*:

_ stands for any single character

% stands for any sequence of zero or more characters.

Note: Wildcard characters are only used as such in LIKE predicates. In any other context, the characters _ and % have their exact values.

The optional escape character is used to allow matching of the special characters _ and **%.** When the escape character prefixes _ and **%,** they are interpreted without any special meaning. The *character-value* must be a string expression of length 1. To search for the escape character itself it must appear twice in immediate succession.

An escape character used in a pattern string may only be followed by another escape character or one of the wildcard characters, unless it is itself escaped (i.e. preceded by an escape character).

Examples:

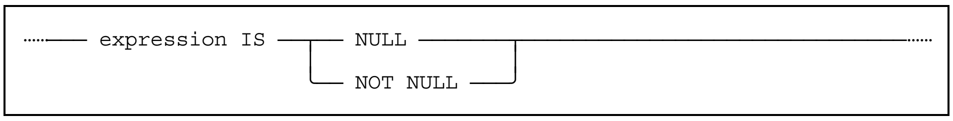
LIKE predicate	Matches
LIKE '%A%'	any string containing an uppercase A
LIKE '%A\%\\' ESCAPE '\'	any string ending with A%\
LIKE '_ABC'	any 4-character string ending in ABC

A LIKE predicate where the pattern string does not contain any wildcard characters is essentially equivalent to a basic predicate using the "=" operator. The comparison strings in the LIKE predicate are **not** conceptually padded with blanks, in contrast to the basic comparison Thus

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

5.9.6 The NULL predicate

The NULL predicate is used to test if the specified expression is the NULL value, and has the form:



If the predicate specifies *expression IS NULL*, then the result is true if any operand in the expression is NULL. The result is false if no operand in the expression is NULL.

The result of the NULL predicate is never unknown.

The use of composite expressions in NULL predicates provides a shorthand for testing whether any of the operands is NULL. Thus the predicate “A+B IS NULL” is an alternative to “A IS NULL OR B IS NULL”, provided that A+B does not result in overflow.

Note: The actual arithmetical operator(s) used in numerical expressions in NULL predicates is irrelevant since all arithmetical operations involving a NULL value evaluate to the NULL value.

The NULL predicate is the only way to test for the presence of the NULL value in a column, since all other predicates where at least one of the operands is NULL evaluate to unknown.

5.9.7 The EXISTS predicate

The EXISTS predicate tests whether the set of values addressed by a select-specification is empty or not, and has the form:

```
·····──── EXISTS ( select-specification ) ─────────────────────────────────·····
```

The result of the predicate is true if the select-specification does not result in an empty set. Otherwise the result of the predicate is false. A set containing only NULL values is not empty. The result is never unknown.

The EXISTS predicate is the only predicate which does not compare a value with one or more other values. The columns selected in the select-specification of an EXISTS predicate are irrelevant. Most commonly, the SELECT * shorthand is used.

The EXISTS predicate may be negated in the construction of search conditions. Observe however that NOT EXISTS predicates must be handled with care, particularly if empty result sets arise in the selection condition. Consider the four following examples, and note particularly that the last example is true if all guests have undefined names:

```
EXISTS (SELECT * FROM BOOK_GUEST
        WHERE GUEST = 'DATE')
```

demands that at least one guest is called DATE

```
NOT EXISTS (SELECT * FROM BOOK_GUEST
            WHERE GUEST = 'DATE')
```

demands that no guest may be called DATE

```
EXISTS (SELECT * FROM BOOK_GUEST
        WHERE NOT GUEST = 'DATE')
```

demands that at least one guest is not called DATE

```
NOT EXISTS (SELECT * FROM BOOK_GUEST
            WHERE NOT GUEST = 'DATE')
```

demands that no guest may not be called DATE, i.e. every guest must be called DATE (or be NULL).

5.9.8 The OVERLAPS predicate

The OVERLAPS predicate tests whether two “events” cover a common point in time or not, and has the form:

```
·····— ( expression , expression ) OVERLAPS ( expression , expression ) —···
```

Each of the two “events” specified on either side of the OVERLAPS keyword is a period of time between two specified points on the timeline. The two points can be specified as a pair of datetime values or as one datetime value and an INTERVAL offset.

Each “event” is defined by a two expressions constituting a row value expression having **two** columns.

The **first** column in each row value expression must be a DATE, TIME or TIMESTAMP and the value in the first column of the first “event” must be comparable (see Section 4.6.3) to the value in the first column of the second “event”.

The **second** column in each row value expression may be either a DATE, TIME or TIMESTAMP that is comparable with the value in the first column or an INTERVAL with a precision that allows it to be added to the value in the first column.

The value in the first column of each row value expression defines one of the points on the timeline for the event.

If the value in the second column of the row value expression is a datetime, it defines the other point on the timeline for the event.

If the value in the second column of the row value expression is an INTERVAL, the other point on the timeline for the event is defined by adding the values in the two column of the row value to expression together.

The NULL value is assumed to be a point that is infinitely late in time.

Either of the two points may be the earlier point in time.

If the value in the **first** column of the row value expression is the NULL value, then this is assumed to be the later point in time.

5.9.9 Standard compliance

This section summarizes standard compliance concerning predicates.

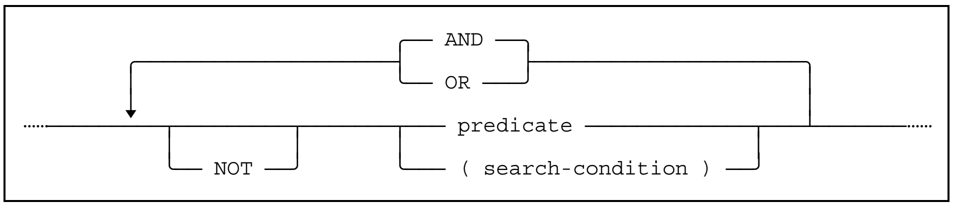
Standard	Compliance	Comments
X/Open-95 SQL92	EXTENDED	Support for an IN predicate with only one element is a Mimer SQL extension.

5.10 Search conditions

5.10.1 Description

Search conditions are used in WHERE and HAVING clauses to qualify the selection of rows and groups respectively; in CHECK clauses to define sets of acceptable values; in CASE expressions to conditionally return different values; in CASE and IF statements to control conditional execution in a routine or trigger; in WHILE and REPEAT statements to control conditional iteration in a routine or trigger; and in the WHEN clause of a trigger to control conditional execution of the trigger action.

A search condition is built from one or more predicates linked by the logical operators AND and OR and qualified if desired by the operator NOT. The general syntax of a search condition is:



Search conditions enclosed in parentheses may be used as part of more complex search condition constructions. A search condition is evaluated as follows:

- Conditions in parentheses are evaluated first.
- Within the same level of parentheses, NOT is applied before AND, AND is applied before OR.
- Operators at the same precedence level are applied in an order determined by internal optimization routines.

The result of a search condition is evaluated by combining the results of the component predicates. Each predicate evaluates to true, false or unknown (truth tables are shown in Section 4.6.5).

WHERE and HAVING clauses select the set of values for which the search condition evaluates to true. CHECK clauses define the set of values for which the search condition does not evaluate to false, i.e. is either true or unknown.

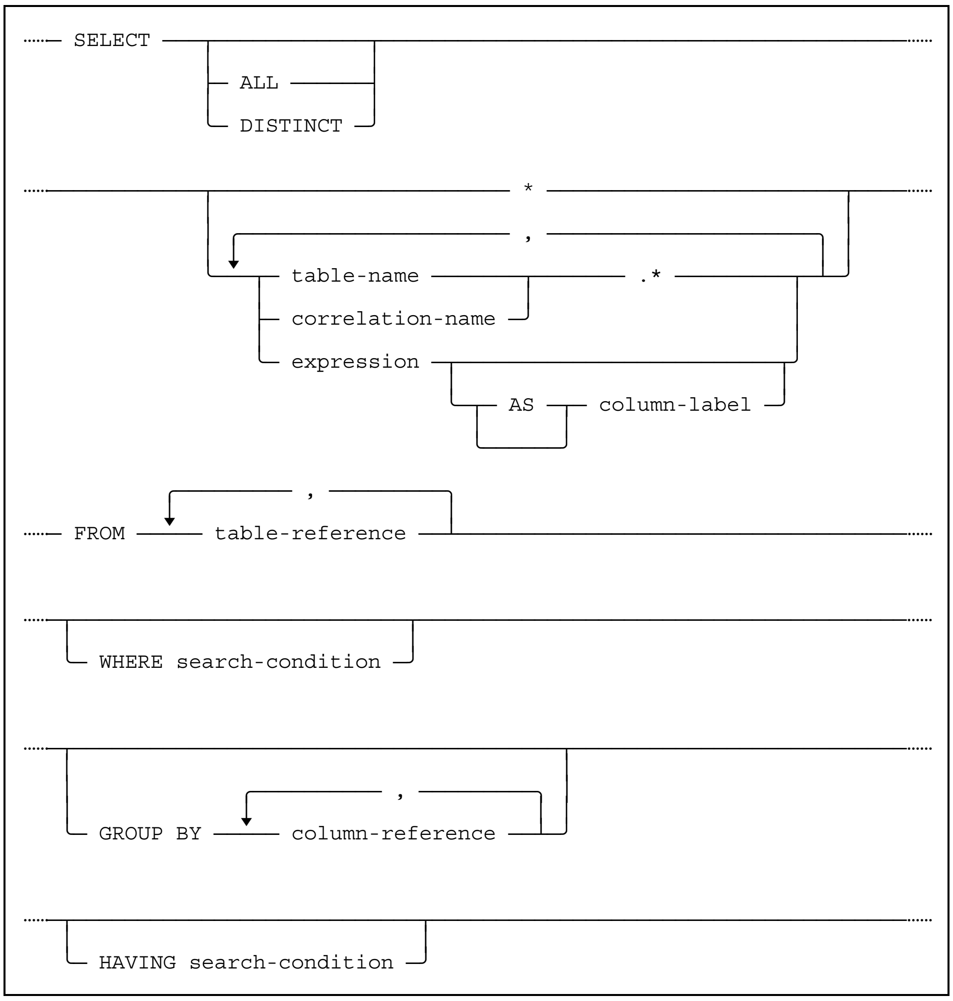
5.10.2 Standard compliance

This section summarizes standard compliance concerning search conditions.

Standard	Compliance	Comments
X/Open-95 SQL92	YES	Fully compliant.

5.11 Select-specification

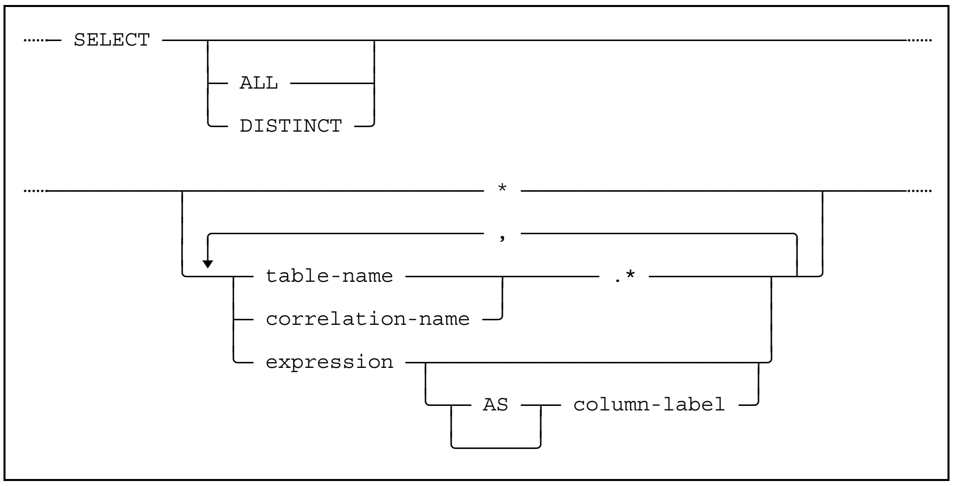
A select specification defines a set of data (rows and columns) extracted from one or more tables or views. The syntax is:



Each clause in the select specification construction is described in detail in the following sections.

5.11.1 The SELECT clause

The SELECT clause defines which values are to be selected. Values are specified by column references or expressions; where columns are addressed, the value selected is the content of the column.



SELECT *

This form of the SELECT clause specifies all columns in the Cartesian product of the tables specified in the FROM clause. The single asterisk may not be combined with any other value specification.

Example:

```
SELECT * FROM HOTEL ...
```

Note: Use of SELECT * is discouraged in embedded SQL programs (except in EXISTS predicates) since the asterisk is expanded to a column list when the statement is compiled, and any subsequent alterations in the table or view definitions may cause the program to function incorrectly.

SELECT table.*

If a named table or view (*table-reference* or *correlation-name*) is followed by an asterisk in the SELECT clause, all columns are selected from that table or view. This formulation may be used in a list of select specifications.

Example:

```
SELECT HOTEL.*, ROOMS.ROOMNO
FROM   HOTEL,ROOMS
WHERE  HOTEL.HOTELCODE = ROOMS.HOTELCODE
```

If a *correlation-name* is used, it must be defined in the associated FROM clause (see Section 5.11.2).

Note: Use of SELECT table.* is discouraged in embedded SQL programs (except in EXISTS predicates) since the asterisk is expanded to a column list when the statement is compiled, and any subsequent alterations in the table or view definitions may cause the program to function incorrectly.

SELECT expression

Values to be selected may be specified as expressions (using column references, set functions and literals, see Section 5.8). Column names used in expressions must refer to columns in the tables addressed in the FROM clause. A column name must be qualified if more than one column in the set of table references addressed in the FROM clause has the same name.

Example:

```
SELECT 'Room type ',ROOMTYPE,' costs ',
       PRICE * :EXCHANGE_RATE, 'in dollars'
FROM   ROOM_PRICES
```

SELECT ... AS column-label

A column-label may be added after each separate expression in the SELECT clause. Column-label is an SQL identifier which becomes the name of the column in the result set. If no name is given the original column name is used, unless the new column was created by an expression, in which case the new column has no name. e.g. “SELECT COLUMN_NAME” would result in column called COLUMN_NAME in the result set, but “SELECT COLUMN_NAME + 1” would result in a column in the result set with no name.

Examples:

```
SELECT PRICE AS INTERNAL, PRICE*RATE AS DOLLAR
FROM   ROOM_PRICES, EXCHANGE_RATE
WHERE  CURRENCY = 'USD'

SELECT PRICE AS INTERNAL
FROM   ROOM_PRICES
WHERE  PRICE > 300
ORDER  BY INTERNAL
```

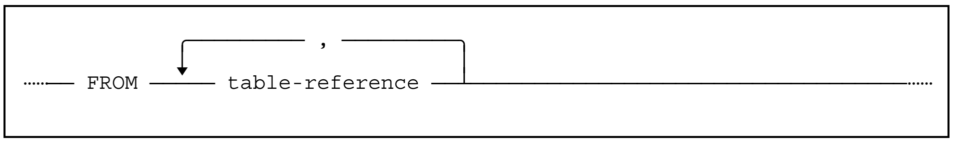
Note: In the second example, the PRICE column is renamed INTERNAL so the ORDER BY clause must use the new name. However, column-label’s cannot be used in a WHERE clause, which is why PRICE is used.

The keywords ALL and DISTINCT

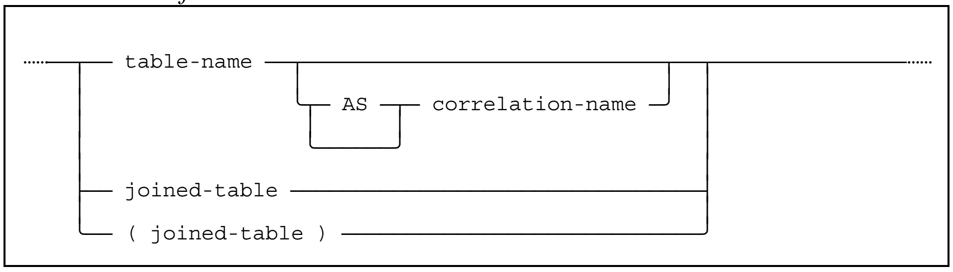
If ALL is specified or if no keyword is given, duplicate rows are not eliminated from the result of the select-specification. If DISTINCT is specified, duplicate rows are eliminated.

5.11.2 The FROM clause and table-reference

The FROM clause defines an intermediate result set for the select-specification, and may define correlation names for the table references used in the result set.



where *table-reference* is:



General syntax

All source tables and views referenced in the SELECT clause and at the top level in the WHERE clause (but not in any subselect used in the WHERE clause) must be named in the FROM clause.

Intermediate result sets

If a single table or view is named in the FROM clause, the intermediate result set is identical to the table or view.

If the FROM clause names more than one table or view, the intermediate result set may be regarded as the complete Cartesian product of the named tables or views.

Note: The intermediate result set is a conceptual entity, introduced to aid in understanding of the selection process. The complete result set does not have any direct physical existence, so that the machine resources available do not need to correspond to the (sometimes very large) Cartesian product tables implied by multiple table references in a FROM clause.

Correlation names

Correlation names introduced in the FROM clause redefine the form of the table name which may be used to qualify column names (see Section 4.2.3). Correlation names may be used for two purposes:

- to shorten table names, which saves typing and makes statements easier to follow and less error-prone. For example,

```
SELECT   G.GUESTNO, SUM(AMOUNT)
FROM     BOOKADM.BILL AS B, BOOKADM.BOOK_GUEST AS G
WHERE    B.GUESTNO = G.GUESTNO
GROUP BY G.GUESTNO
```

- to relate a table to a logical copy of itself, as in the following example which selects all unique pairs of hotels located in the same city:

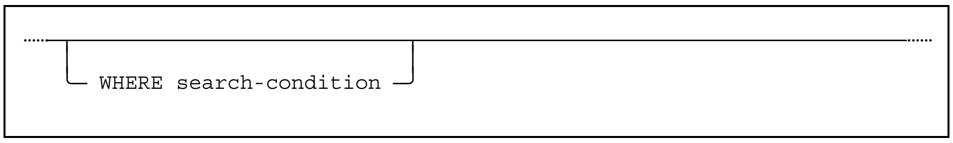
```
SELECT HOTEL.NAME, HOTELCOPY.NAME
FROM   HOTEL, HOTEL AS HOTELCOPY
WHERE  HOTEL.CITY = HOTELCOPY.CITY
AND    HOTEL.NAME > HOTELCOPY.NAME
```

A table or view name is exposed in the FROM clause if it does not have a correlation name. The same table or view name cannot be exposed more than once in the same FROM clause.

The same correlation name may not be introduced more than once in the same FROM clause, and it cannot be the same as an exposed table or view name.

5.11.3 The WHERE clause

The WHERE clause selects a subset of the rows in the intermediate result set on the basis of values in the columns. If no WHERE clause is specified, all rows of the intermediate result set are selected.

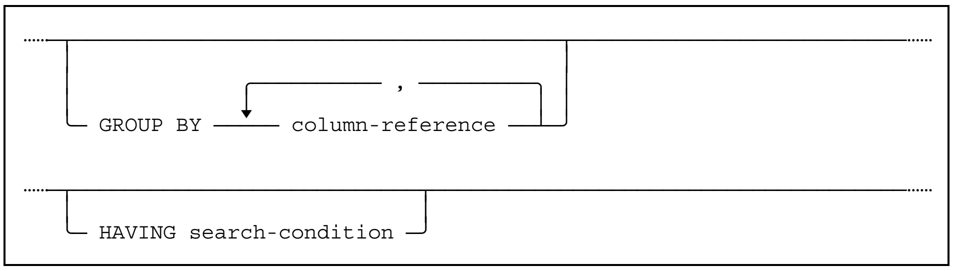


All column references in the *search-condition* must uniquely identify a column in the intermediate result set defined by the FROM clause or be an outer reference. Column references must be qualified if more than one column in the intermediate result set has the same name, or if the column is an outer reference.

5.11.4 The GROUP BY and HAVING clauses

The GROUP BY clause determines grouping of the result table for the application of set functions specified in the SELECT clause. The HAVING clause restricts selection of groups in the same way that a WHERE clause restricts selection of rows.

The GROUP BY and HAVING clauses have the syntax:



If a GROUP BY clause is specified, each column reference in the SELECT list must either identify a grouping column or be the argument of a set function. The rows of the intermediate result set are (conceptually) arranged in groups, where all values in the grouping column(s) are identical within each group. Each group is reduced to a single row in the final result of the select-specification.

The following example lists the number of rooms for each room type in each hotel:

```
SELECT   H.NAME, R.ROOMTYPE, COUNT(*)
FROM     HOTEL H, ROOMS R
WHERE    H.HOTELCODE = R.HOTELCODE
GROUP BY H.NAME, R.ROOMTYPE
```

If a GROUP BY clause is **not** specified, the SELECT list must **either** be a list that does not include any set functions **or** a list of set functions and optional literal expressions.

The search condition in the HAVING clause defines restrictions on the values in the elements of the SELECT list. Column references in the search condition of the having clause must identify a grouping column, or be used in set functions, or be outer references.

The following example is similar to the example above, but room types with less than 10 occurrences are not listed.

```
SELECT   H.NAME, R.ROOMTYPE, COUNT(*)
FROM     HOTEL H, ROOMS R
WHERE    H.HOTELCODE = R.HOTELCODE
GROUP BY H.NAME, R.ROOMTYPE
HAVING   COUNT(*) >= 10;
```

Most commonly, HAVING is used together with GROUP BY, in which case the search conditions relate either values in grouping columns or results of set functions to expressions. If the HAVING clause is used without a GROUP BY clause, all rows in the result table are treated as a single group. In this case, the HAVING clause must refer to a set function (since there are no grouping columns).

The GROUP BY or HAVING clause may not be specified in a select-specification whose FROM clause names a view that uses GROUP BY or HAVING, or in a select-specification used in a basic predicate.

5.11.5 Standard compliance

This section summarizes standard compliance concerning select-specifications.

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	YES	Fully compliant.

5.12 Joined Tables

The JOIN syntax provides methods of combining information in tables. INNER joins, and LEFT and RIGHT OUTER joins are supported. A join may be performed as a NATURAL JOIN or where the conditions for the join are more explicitly specified.

The different ways in which the various join options can be combined makes the overall join syntax quite convoluted, so to simplify the explanation, each variant will be described on its own with an accompanying syntax diagram.

In order to understand the join syntax generally, it is important to appreciate the difference between an INNER and an OUTER join. It is also important to be aware of the three fundamental join variants (NATURAL, JOIN USING and JOIN ON) which are **mutually exclusive** and apply equally to INNER and OUTER joins.

5.12.1 The INNER JOIN

An **INNER JOIN** produces a result table containing composite rows created by combining rows from two tables where some pre-defined, or explicitly specified, join condition evaluates to **true**.

Rows that do not satisfy the join condition will **not** appear in the result table of an inner join.

The inner join is the default join type, therefore the keyword INNER is optional and may be omitted.

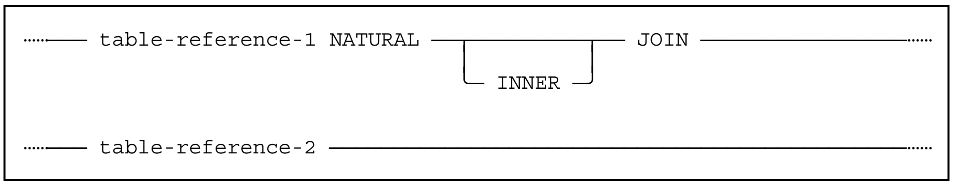
5.12.1.1 NATURAL JOIN

The result table of a NATURAL JOIN contains one row for each case where **all** the **common columns** in the two tables contain values that are **equal**.

Common columns are those which have the **same name** in each table. The common columns must have a data type that allows values in the respective tables to be compared.

A row in the result table contains the combined set of columns from each table, except that the common columns appear only once. The common columns appear first (at the left of the table) followed by the remaining columns from table-1, followed by those from table-2.

The syntax for a NATURAL JOIN is:



If there are **no rows** where all the common columns have equal values, the result table is an empty table (i.e. it has a set of columns as just described, but the number of rows is zero).

Example: two tables contain different sets of information on people, and each table has a FIRST_NAME column and a SURNAME column to identify the person to whom the information applies. When both the FIRST_NAME column and the SURNAME column contain the same values in a row in each table, it means those rows are related. A NATURAL JOIN between these two tables would produce a result table with a single composite row for each person, containing all the information held in both tables, with the SURNAME and FIRST_NAME columns appearing once in the rows of the result.

It is actually possible to perform a NATURAL JOIN between two tables which have **no common columns** at all. In this case the result table is the Cartesian product (sometimes called the CROSS JOIN) of the two tables.

5.12.1.2 JOIN USING

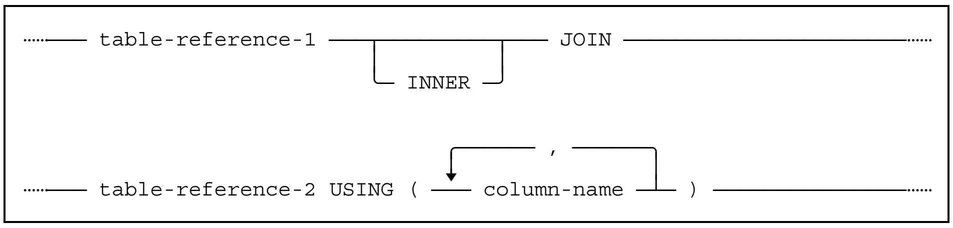
JOIN USING allows a **list** of column names to be specified. This kind of join is conceptually the same as a NATURAL JOIN except that the join is based on the **specified columns** rather than on all the common columns.

Specifying the columns explicitly instead of using the entire set of common columns is useful in situations where some of the common columns may not contain identical values even though the respective rows are related (e.g. as in a REMARKS column).

The columns specified in the list **must be common** to both tables, they must be specified in an unqualified manner and must have a data type that allows the values in the respective tables to be compared.

A row in the result table contains the combined set of columns from each table, except that the common columns appear only once. The columns specified in the list of column names appear first (at the left of the table) followed by the remaining columns from table-1, followed by those from table-2.

The syntax for JOIN USING is:



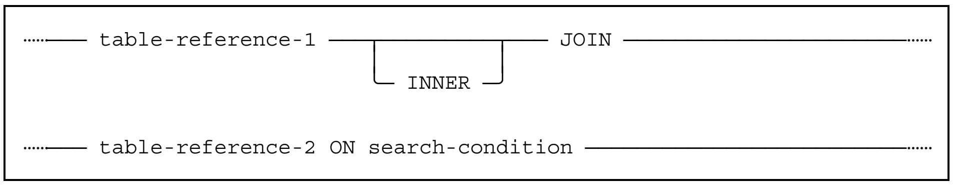
5.12.1.3 JOIN ON

JOIN ON allows a **join condition** to be specified. The result table of this kind of join is produced by applying the specified join condition to the Cartesian product of the two tables. The result table will contain only those rows for which the join condition evaluates to **true**.

The join condition cannot reference common columns unless they are qualified (e.g. by table name).

A row in the result table contains the combined set of columns from each table. The columns from table-1 appear first followed by those from table-2. Common columns will therefore appear twice.

The syntax for JOIN ON is:



5.12.2 The OUTER JOIN

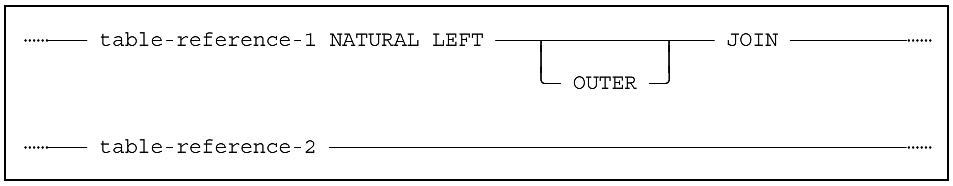
A table resulting from an inner join, as just described, will only contain those rows that satisfy the applicable join condition. This means that a row in either table which does not match a row in the other table will be excluded from the result.

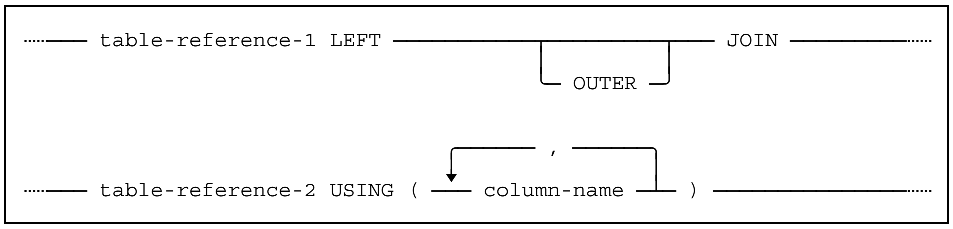
In an **OUTER JOIN**, however, a row that does not match a row in the other table is also **included** in the result table. Such a row appears once in the result and the columns that would normally contain information from the other table will contain the NULL value.

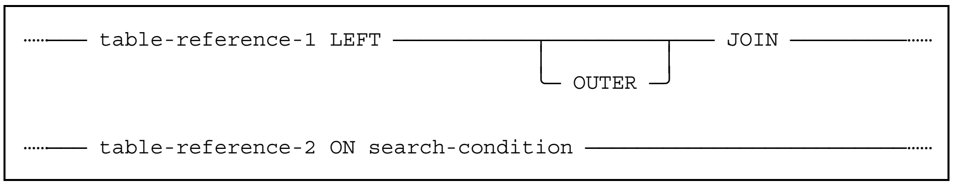
5.12.2.1 LEFT OUTER JOIN

The inner join variants (NATURAL JOIN, JOIN USING and JOIN ON) can be applied as a LEFT OUTER JOIN. The LEFT OUTER JOIN **includes** the rows from table-reference-1 (the table on the **left** of the JOIN) which do **not** satisfy the join condition.

The syntax for the variants of the LEFT OUTER JOIN is as follows:



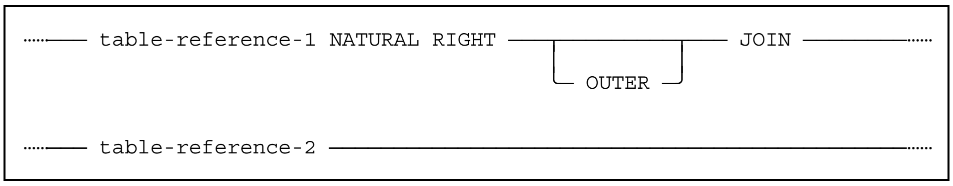


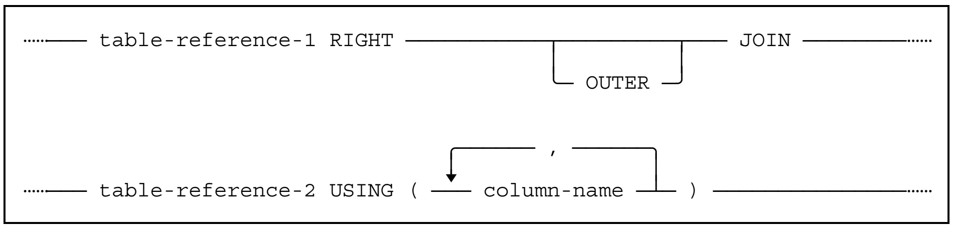


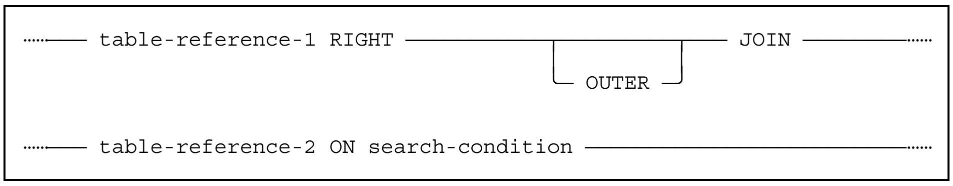
5.12.2.2 RIGHT OUTER JOIN

The inner join variants (NATURAL JOIN, JOIN USING and JOIN ON) can be applied as a RIGHT OUTER JOIN. The RIGHT OUTER JOIN **includes** the rows from table-reference-2 (the table on the **right** of the JOIN) which do **not** satisfy the join condition.

The syntax for the variants of the RIGHT OUTER JOIN is as follows:







5.12.2.3 FULL OUTER JOIN

A full outer join can be achieved by performing a UNION between a LEFT OUTER JOIN and RIGHT OUTER JOIN.

For example, a full outer join of the SUPPLIERS table with the PARTS table on the CITY column is specified as follows:

```
SUPPLIERS LEFT OUTER JOIN PARTS USING (CITY)
UNION
SUPPLIERS RIGHT OUTER JOIN PARTS USING (CITY)
```

5.12.3 Standard compliance

This section summarizes standard compliance concerning JOIN.

Standard	**Compliance**	**Comments**
X/Open-95 **SQL92**	YES	Fully compliant.

5.13 SELECT statements

Simple SELECT statements are built from a select-specification optionally followed by either an ORDER BY or a FOR UPDATE OF clause. More complex statements can combine two or more select-specifications with the UNION operator. Select statements are used in embedded SQL (including Procedural usage contexts) to define cursors and as the input to dynamic PREPARE statements. The embedded SELECT statement is syntactically equivalent to the interactive data retrieval SELECT statement. In embedded contexts however, the statement cannot be used to retrieve data directly but must be implemented through a cursor.

The full syntax of the SELECT statement is given in Chapter 6.

5.13.1 Updatable result sets

A result set is only updatable if **all** of the following conditions are true (otherwise the result set is read-only):

- the keyword DISTINCT is not specified
- all the result columns are specified as column-names and no column-name appears more than once
- the FROM clause specifies exactly one table reference and that table reference refers either to a base table or an updatable view
- the result set is not the product of an explicit INNER or OUTER JOIN
- the GROUP BY clause is not included
- the HAVING clause is not included
- the keyword UNION is not included
- the ORDER BY clause is not included
- it is not the result of a call to a result set procedure.

A cursor which addresses a read-only result table may **not** be used for DELETE CURRENT or UPDATE CURRENT statements.

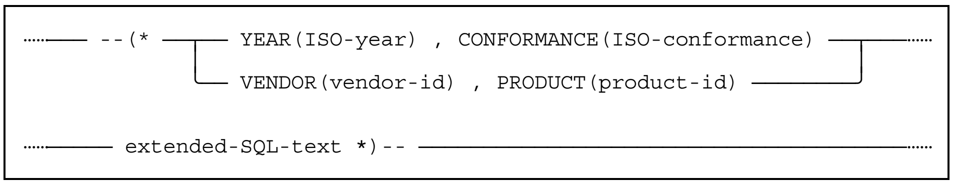
5.14 Vendor-specific SQL

5.14.1 Description

An escape clause is a syntactic mechanism for using vendor-specific SQL extensions in a standardized SQL application. Using escape clauses, an application can request a vendor-specific function in a way that does not keep it from compiling or executing in an environment that does not support the function. However, if the application depends on the vendor-specific SQL functions it will be restricted in its portability, since a standard-compliant SQL implementation need not provide the vendor-specific SQL extensions that are used.

Escape clauses are allowed in SQL statements submitted using dynamic SQL (see Chapter 7 of the Mimer SQL Programmer's Manual), and in interactive SQL.

Syntax for the escape clause:


--(*
YEAR(ISO-year) , CONFORMANCE(ISO-conformance)
VENDOR(vendor-id) , PRODUCT(product-id)
extended-SQL-text *)--

Example:

```
SELECT name, salary
FROM   employee
WHERE  company = 'Important Stuff Organization'
--(* VENDOR(MIMER), PRODUCT(MIMER)
AND    TAIL(manager,4) <> 'Boss'
*)--
AND    salary > 20000;
```

Note: Mimer SQL strictly follows the defined standards and has only a few vendor-specific extensions so vendor-specific statements in Mimer SQL are usually quite simple.

The escape clause allows coherent interpretation of different SQL dialects. The *extended-SQL-text* contains all, or part of, a valid SQL statement in the SQL dialect that the escape clause specifies. The text must not contain an SQL-prefix (EXEC SQL), an SQL-terminator (“;”), a dynamic parameter or a host variable.

There are two forms for the escape clause:

ISO-based escape clauses

This form of the escape clause introduces SQL syntax that is supported by a specified dialect of the International Standard, but is not supported by X/Open-92. Through this escape clause, the database can reject the clause if it does not support the specified SQL dialect. *ISO-year* specifies the publication year of the standard, e.g. “1989” for the SQL89 standard. *ISO-conformance* specifies the conformance level: CONFORMANCE(1) for Entry level conformance, CONFORMANCE(2) for Intermediate level conformance, or CONFORMANCE(3) for Full level conformance.

Vendor-based escape clauses

This form of the escape clauses introduces vendor-specific SQL syntax. The *vendor-id* for Mimer SQL is “MIMER”, and the *product-id* is “MIMER”.

An SQL statement that contains an escape clause is processed in the following way:

1. If Mimer SQL supports the specified SQL dialect, then the entire escape clause is replaced with the text provided in *extended-SQL-text* of the escape clause.
2. If Mimer SQL does not support this SQL dialect, then it conceptually deletes the entire escape clause.
3. The edited SQL statement is processed in the usual way.

5.14.2 Standard compliance

This section summarizes standard compliance concerning vendor-specific SQL.

Standard	Compliance	Comments
SQL92	EXTENDED	Support for the escape clause for vendor-specific SQL is a Mimer SQL extension.
X/Open-95	YES	Fully compliant.

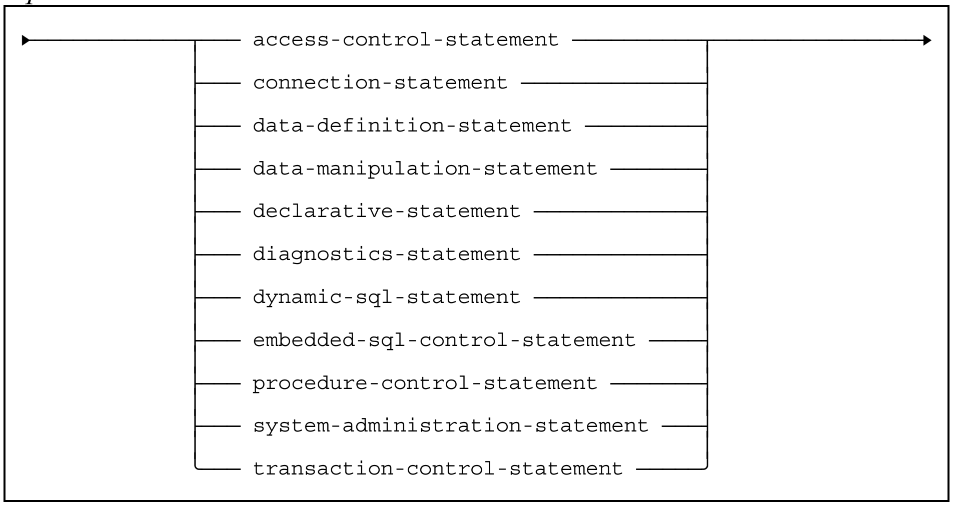
6 SQL STATEMENT DESCRIPTIONS

Statements in Mimer SQL may be divided into functional groups as shown below in the definition of *sql-statement*.

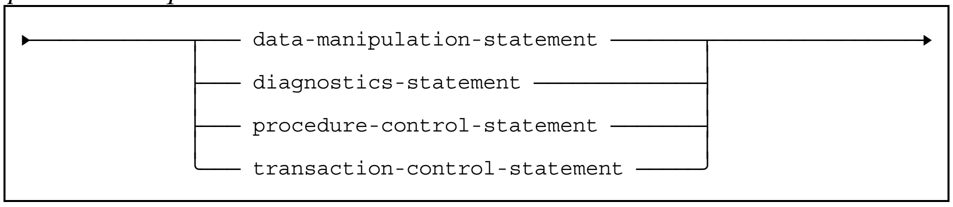
A special group, described by the definition of *procedural-sql-statement*, is those statements which may be used within a function, procedure or trigger (see Chapter 8 of the *Mimer SQL Programmer's Manual* for a detailed discussion of the use of SQL in a Procedural usage context).

The exact usage mode allowed for each statement is described with the syntactic definition for it appearing in this chapter (see Section 6.1 for a description of the different usage modes).

sql-statement


access-control-statement
connection-statement
data-definition-statement
data-manipulation-statement
declarative-statement
diagnostics-statement
dynamic-sql-statement
embedded-sql-control-statement
procedure-control-statement
system-administration-statement
transaction-control-statement

procedural-sql-statement

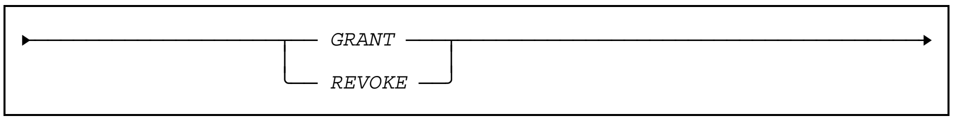

data-manipulation-statement
diagnostics-statement
procedure-control-statement
transaction-control-statement

The definitions of the statement groups mentioned above follow.

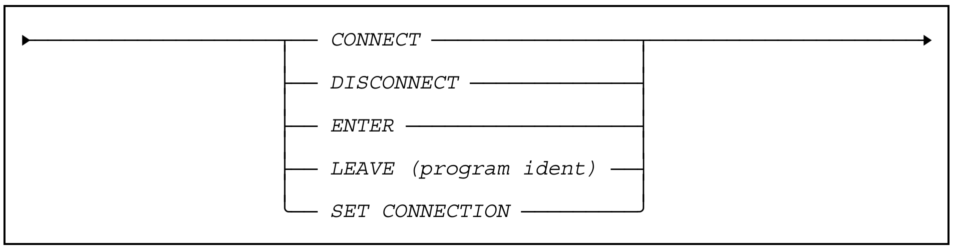
The statement group definitions that follow are shown as syntax diagrams because this is the most convenient presentational form.

The text shown in these definitions does not necessarily represent partial or complete syntax but instead presents the **titles** for the actual statement syntax definitions which are alphabetically arranged in the remainder of this chapter.

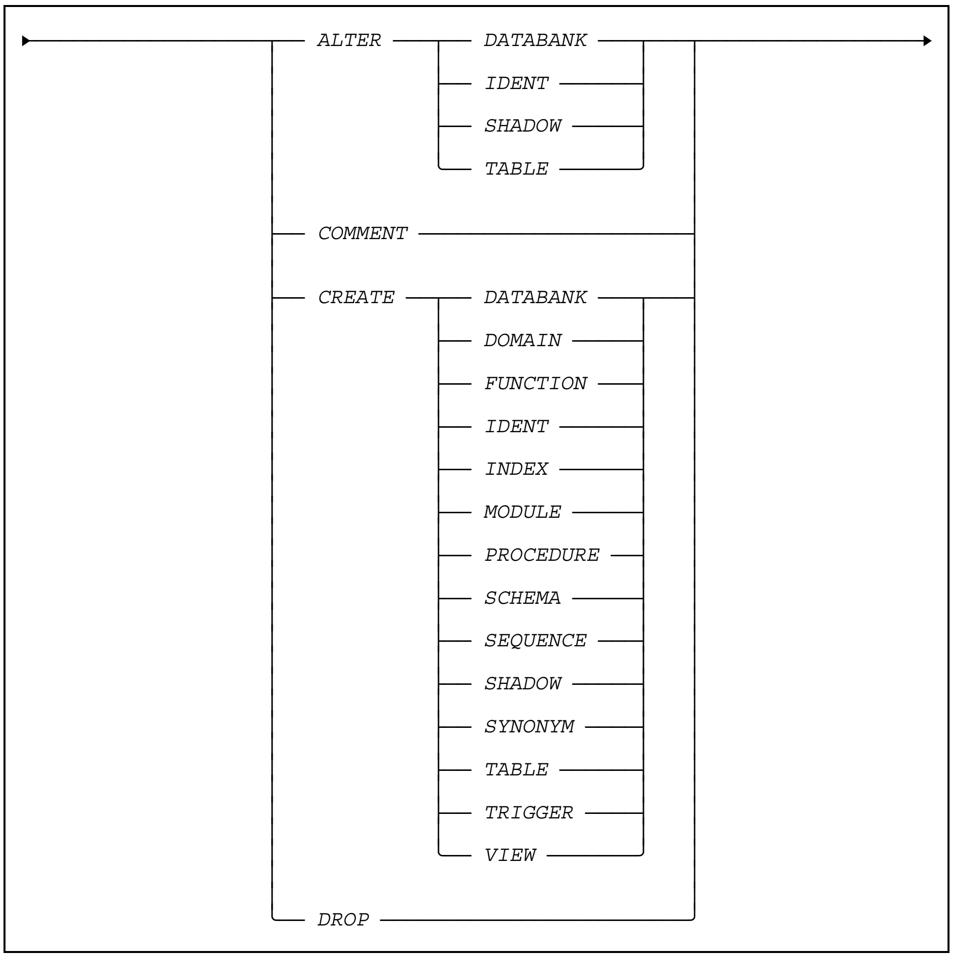
access-control-statement



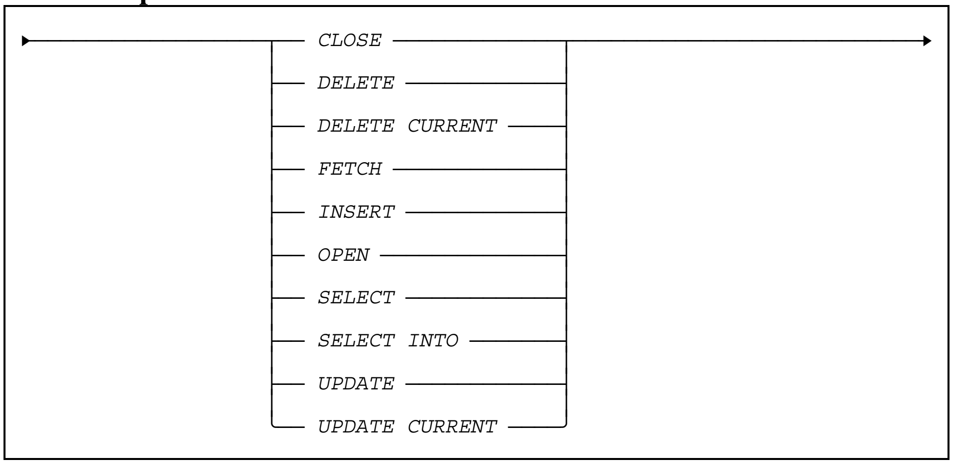
connection-statement



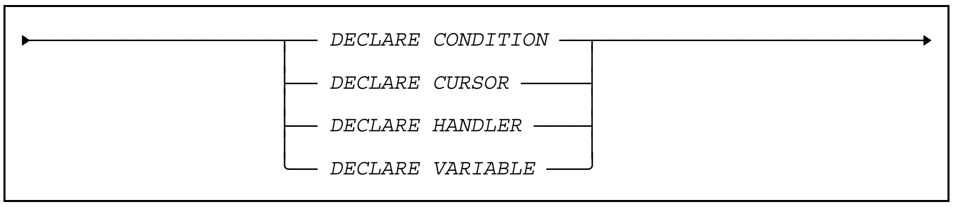
data-definition-statement



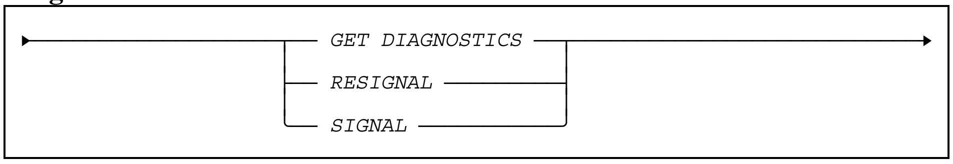
data-manipulation-statement



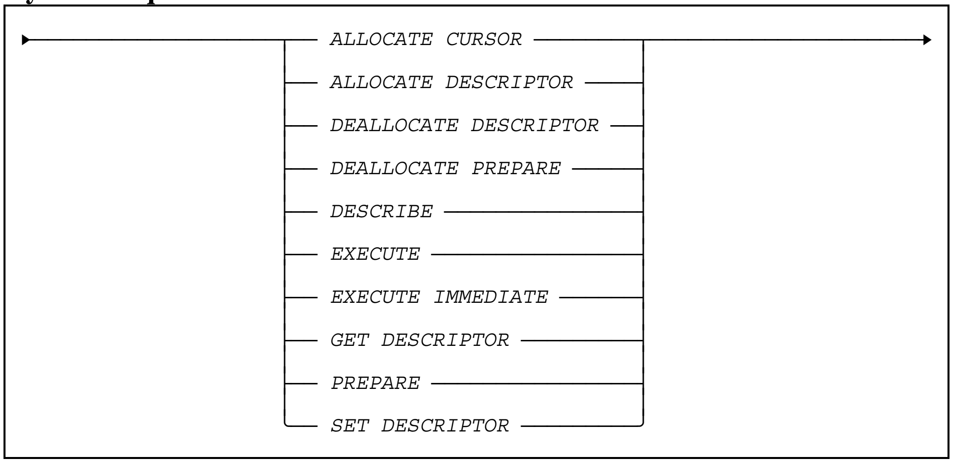
declarative-statement



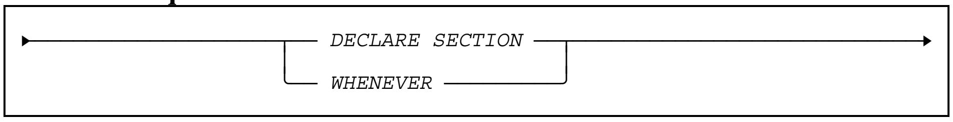
diagnostics-statement



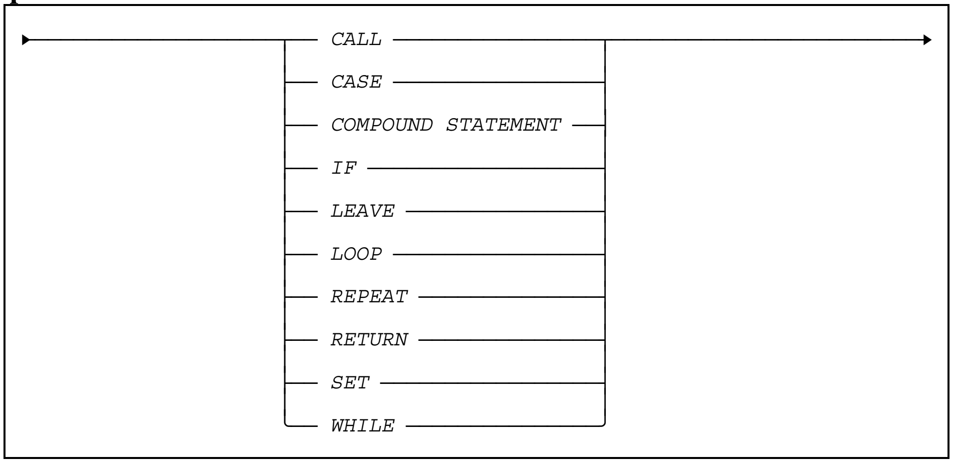
dynamic-sql-statement



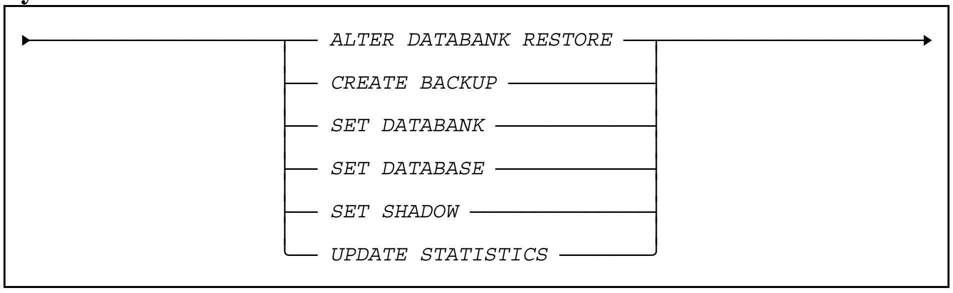
embedded-sql-control-statement



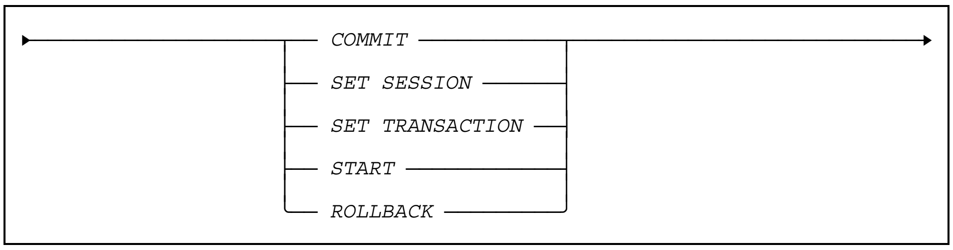
procedure-control-statement



system-administration-statement



transaction-control-statement



The rest of this chapter contains a detailed syntactic and functional description of each of the statements listed above, in alphabetical order.

Refer to Chapter 5 for a formal description of the language elements used in the syntax descriptions.

6.1 Usage modes

The following usage modes apply to the statements described in this chapter:

Embedded	The statement may be embedded in an application program.
Interactive	The statement is valid for use in interactive SQL.
ODBC	The statement is valid for use via the Microsoft Open Database Connectivity (ODBC) interface.
Procedural	The statement may be used within a function, procedure or trigger.

ALLOCATE CURSOR

Allocates an extended cursor name.

```
►── ALLOCATE extended-cursor-name ──┬──────────────┬──────────────······
                                    ├─ NO SCROLL ──┤
                                    ├─ SCROLL ─────┤
                                    └─ REOPENABLE ─┘

·····── CURSOR FOR extended-statement-name ──────────────────────────►
```

Usage

Embedded.

Description

The value of the *extended-cursor-name* is associated with the prepared statement specified by the *extended-statement-name*. Extended cursors and statements differ from "normal" cursors and statements in that they are identified by a host variable or a literal, instead of by an identifier. The host variable must be declared in the DECLARE SECTION of the compilation unit as a character string variable.

The association between the cursor and the statement is preserved until the prepared statement is destroyed (see DEALLOCATE PREPARE statement), at which time the cursor is also destroyed.

A cursor allocated as REOPENABLE may be opened several times in succession and previous cursor states are saved on a stack (see OPEN). Saved cursor states are restored when the current state is closed (see CLOSE).

A cursor allocated as SCROLL will be a scrollable cursor. For a scrollable cursor, records can be fetched using an orientation specification. See the description of FETCH later in this chapter for a description of how the orientation is specified.

Restrictions

None.

Notes

The extended statement must identify a statement previously prepared in the scope of the *extended-statement-name*. That prepared statement must be a query expression.

There must be no other extended cursor with the same name allocated in the same compilation unit.

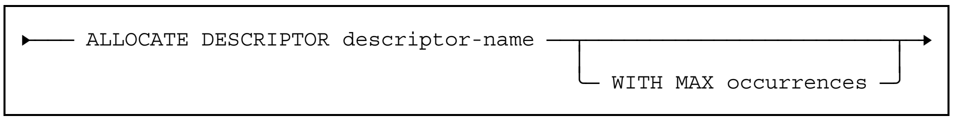
A reopenable cursor can be used to solve the “Parts explosion” problem. Refer to Section 5.3.1 of the Mimer SQL Programmer’s Manual for a description of this.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95	MIMER EXTENSION	Support for the ALLOCATE CURSOR statement is a Mimer SQL extension.
SQL92	EXTENDED	Support for REOPENABLE is a Mimer SQL extension.

ALLOCATE DESCRIPTOR

Allocates an SQL descriptor area.



Usage

Embedded.

Description

An SQL descriptor area is allocated. The SQL descriptor area is used to provide information about variables used for input and output between the application and the database. The *descriptor-name* is identified by a host variable or a literal.

The allocated SQL descriptor area will have as many item descriptor areas as specified by the *WITH MAX occurrences* clause. If *WITH MAX occurrences* is omitted, 100 item descriptor areas are allocated.

The SQL descriptor area has the following structure:

COUNT
item descriptor area 1
item descriptor area 2
...
item descriptor area n

The COUNT field specifies how many item descriptor areas contain data.

See GET DESCRIPTOR for a description of the descriptor fields.

Restrictions

None.

Notes

The maximum length of the descriptor name is 128 characters.

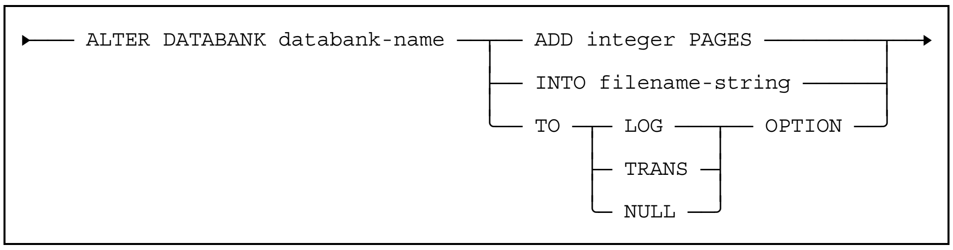
The scope of a descriptor name is limited to a single compilation unit and there cannot be more than one descriptor with the same name in a single compilation unit.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	YES	Fully compliant.

ALTER DATABANK

Alters the file location, transaction control option or size of a databank.



Usage

Embedded/Interactive/ODBC.

Description

If the ADD ... PAGES clause is specified, the databank file is extended by the number of Mimer SQL pages given by the *integer* parameter.

If the INTO clause is specified, the databank location stored in the data dictionary is changed to the location given in the *filename-string* parameter. The file specified by *filename-string* must exist when the ALTER DATABANK statement is executed.

The new file must be identifiable as a copy of the databank created for the current Mimer SQL database. The first page of the databank file is read to verify that the data in the databank can be accessed and that the file was closed correctly the last time it was used.

If the file is flagged internally as not being closed correctly, a full DBC check is effectively done on it (see Chapter 6 of the Mimer SQL System Management Handbook for details on the DBC functionality).

The ALTER DATABANK statement will fail if the new file does not verify correctly against the checks performed.

If the timestamp information in the databank file indicates that additional information must be restored to it to bring it up to date, an information message is written to the database server log file (this message will be returned to the user if the database is being accessed in single user mode).

This situation will **not** cause the ALTER DATABANK statement to fail, but any attempt to subsequently access the databank will raise an error indicating that additional information must be restored to the databank. Once the additional information has been restored, the databank can be used normally.

If the databank is OFFLINE, however, the new file will be accepted by the ALTER DATABANK statement **without** any verification. In this case the file is validated when the databank is next set ONLINE and the SET DATABANK statement will fail if the file does not verify correctly against the checks performed.

If the TO ... OPTION clause is specified, the transaction control option of the databank is changed. The possible 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. Set operations (DELETE, UPDATE and INSERT on several rows) which are interrupted will not be rolled back. All secondary indexes contained in the databank are flagged as “not consistent” (a secondary index that is flagged as “not consistent” will not offer optimal performance when used in a query).

Restrictions

Only the creator of the databank may alter it.

The databank option may only be set to TRANS or LOG for a databank that is shadowed or contains a table defined with foreign or unique keys, a table referenced in a foreign key context or a table on which a UNIQUE index or trigger has been created.

If the databank contains tables whose primary key column(s) are to be updated at some time in the future, the databank option must be set to TRANS or LOG.

The ADD ... PAGES clause may not be used if the databank is OFFLINE.

Notes

If the extension of the databank exceeds the available disk space, the databank is extended as much as possible.

If a databank is full, it will be extended automatically on operating systems supporting dynamic file extension (provided that there is free space on the disk). However, such incremental extensions may lead to the disk becoming fragmented, so the use of explicit ALTER DATABANK ... ADD can help avoid disk fragmentation.

Changing the location of a databank with the ALTER DATABANK ... INTO statement only changes the file location stored in the data dictionary, it does not move any physical files in the host operating system. You must first copy or move the databank file to its new location using operating system commands and then use the ALTER DATABANK statement to correct the location stored in the data dictionary.

The value of *filename-string* must always be enclosed in string delimiters. The maximum length of the filename string is 256 characters.

Refer to Section 3.1.2.1 for details concerning the specification of pathname components in *filename-string*.

When the databank option is altered to NULL, all secondary indexes contained in the databank will be flagged as “not consistent”.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 **SQL92**	MIMER EXTENSION	The ALTER DATABANK statement is a Mimer SQL extension.

ALTER DATABANK RESTORE

Restores a databank from a backup of LOGDB or from the information currently in LOGDB.

```
►── ALTER DATABANK databank-name RESTORE USING ─┬─ LOG ─────────────┬─►
                                                └─ filename-string ─┘
```

Usage

Embedded/Interactive/ODBC.

Description

This form of ALTER DATABANK is used to recover a databank in the event of it being damaged or destroyed. It is possible to use this SQL statement to restore the databank from information contained in the log records for the databank held in the current LOGDB or from a LOGDB backup (refer to Chapter 5 of the Mimer SQL System Management Handbook for details on Backup and Restore).

The recovery operation must start from a usable backup copy of the databank file, which has been created using the host file system backup or from a backup taken using CREATE BACKUP.

Once the restored databank file is in place, ALTER DATABANK is used to bring the databank up to date by applying any updates made to it since that copy of the databank file was taken. The updates may have been recorded in one or more backups of LOGDB and the latest updates will be contained in the log records in the LOGDB system databank.

The syntax options are:

filename-string	When a file name is specified, it names a backup LOGDB file. The updates contained in the file will be applied to the databank. Note that the timestamp information contained in both the databank file and the backup file must match, otherwise a backup sequence error is returned.
LOG	When this option is used, the updates for the databank recorded in the log records currently in LOGDB will be applied to the databank. Note that the timestamp information contained in both the databank file and the LOGDB records must match, otherwise a backup sequence error is returned.

Restrictions

Only the creator of the databank or an ident with BACKUP privilege (e.g. SYSADM), may use the ALTER DATABANK RESTORE statement to restore it.

If the databank does not have LOG option, there will be no operations recorded in LOGDB.

Notes

It is possible to restore a databank that has been set offline.

The recovery operations performed using ALTER DATABANK RESTORE can only be applied to a copy of a databank file which has been placed in the original file location used by the databank.

If the copy of the databank file must be restored to a new location for some reason (e.g. a disk has been lost), then ALTER DATABANK is first used to change the databank file location.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 **SQL92**	MIMER EXTENSION	The ALTER DATABANK RESTORE statement is a Mimer SQL extension.

ALTER IDENT

Alters the password for an existing user, OS_USER or program ident.

```
►── ALTER IDENT ident-name USING password-string ──────────────────►
```

Usage

Embedded/Interactive/ODBC.

Description

The password of the ident given by *ident-name* is changed to the string given in *password-string*.

Restrictions

Group idents do not have passwords, therefore the ALTER IDENT statement cannot be used on a group ident.

The password can only be changed by the ident or by the creator of the ident.

Notes

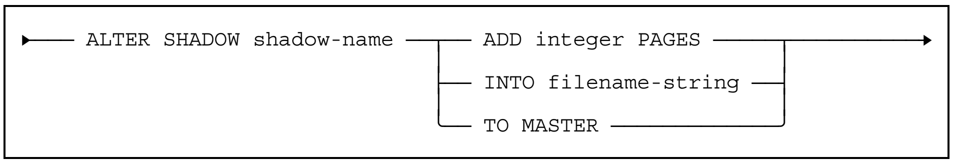
The password must be at least 1 and at most 18 characters long and may contain any characters except the space character. The case of alphabetical characters is significant. The password string must be enclosed in string delimiters, which are not included as part of the password.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	MIMER EXTENSION	The ALTER IDENT statement is a Mimer SQL extension.

ALTER SHADOW

Alters the file location or the size of a shadow, or switches a databank shadow to be the master databank.



Usage

Embedded/Interactive/ODBC.

Description

Alters an existing databank shadow (see the Mimer SQL System Management Handbook for details on databank shadowing).

If the ADD ... PAGES clause is specified, the shadow file is extended by the number of Mimer SQL pages given by the *integer* parameter.

If the INTO clause is specified, the shadow file location stored in the data dictionary is changed to the location specified in the *filename-string* parameter. The file specified by *filename-string* must exist when the ALTER SHADOW statement is executed.

The new file must be identifiable as a copy of the databank shadow created for the current Mimer SQL database. The first page of the databank file is read to verify that it was closed correctly the last time it was used and that the internal timestamp information is consistent with the current contents of LOGDB.

If the file is flagged internally as not being closed correctly, a full DBC check is effectively done on it (see Chapter 6 of the Mimer SQL System Management Handbook for details on the DBC functionality).

The ALTER SHADOW statement will fail if the new file does not verify correctly against the checks performed.

If the shadow is OFFLINE, however, the new file will be accepted by the ALTER SHADOW statement **without** any verification. In this case the file is validated when the shadow is next set ONLINE and the SET SHADOW statement will fail if the file does not verify correctly against the checks performed.

If the TO MASTER clause is specified, the data dictionary is changed so that the file location stored for the shadow file is set for the databank file and vice versa, i.e. the shadow becomes the master and the master becomes a shadow. If the shadow is OFFLINE when the TO MASTER clause is specified, it is automatically set ONLINE before the data dictionary is updated.

Restrictions

ALTER SHADOW is only for use with the optional Mimer SQL Shadowing module.

Only an ident with SHADOW privilege may use the ALTER SHADOW statement.

ALTER SHADOW may not be used if the **master** databank is OFFLINE.

The ADD ... PAGES clause may not be used if the shadow is OFFLINE.

The databank for which the shadow exists cannot be used by any other user while the shadow is being altered.

Shadows for the system databanks SYSDB, TRANSDB, and LOGDB cannot be altered with the ALTER SHADOW statement. These shadows must be altered by the Mimer SQL Utility program.

Notes

If the extension of the shadow exceeds the available disk space, the shadow is expanded as much as possible.

If a shadow is full, it will be extended automatically in systems supporting dynamic file extension (provided that there is space on the disk). However, such incremental extensions may lead to the file becoming fragmented and use of explicit ALTER SHADOW ... ADD is generally recommended (used at the same time as ALTER DATABANK ... ADD is used to extend the master databank).

Changing the location of a shadow with the ALTER SHADOW ... INTO statement only changes the location as stored in the data dictionary, it does not move any physical files in the host operating system. You must first copy or move the shadow file to its new location using operating system commands and then use the ALTER SHADOW statement to correct the location stored in the data dictionary.

The value of *filename-string* must always be enclosed in string delimiters. The maximum length of the filename string is 256 characters.

Refer to Section 3.1.2.1 for details concerning the specification of pathname components in *filename-string*.

The TO MASTER option is used when the original databank file has been lost or is inaccessible for any reason. Since this option swaps the information about the shadow and the master databank stored in the data dictionary, the command may be followed by a DROP SHADOW command to dispose of the original databank.

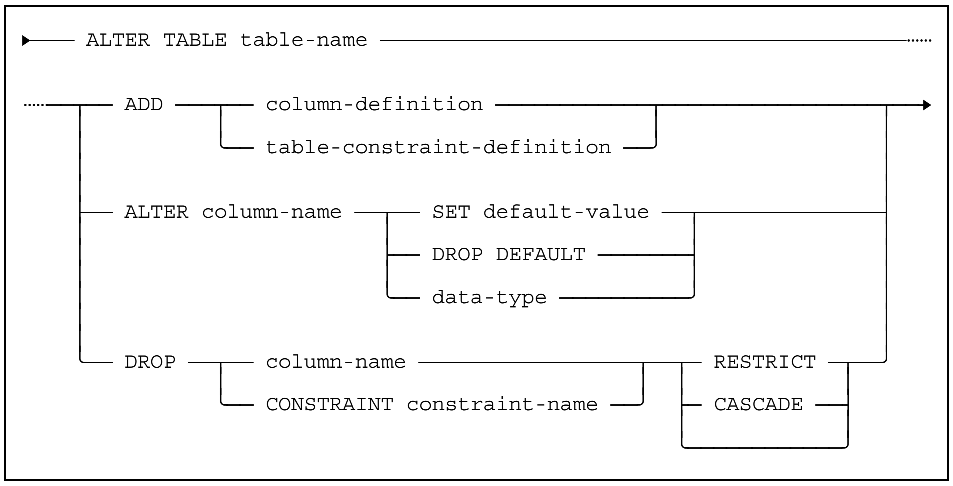
Note that the ALTER SHADOW ... TO MASTER and DROP SHADOW commands used together are equivalent to transforming a shadow into a master using the Shadowing utilities in the UTIL program (see the Mimer SQL System Management Handbook).

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 **SQL92**	MIMER EXTENSION	The ALTER SHADOW statement is a Mimer SQL extension.

ALTER TABLE

Alters a table definition by: adding a column or table constraint; dropping a table constraint or a column; changing the data-type or the default value for a column.



Usage

Embedded/Interactive/ODBC.

Description

When a column is **added**, the existing table is extended with the addition of a new column, which is placed at the end of the table definition.

For each existing row in the table, the column will be assigned the default value (which will be the column default value if one is defined, the domain default if the column belongs to a domain or otherwise the NULL value).

Note: If the *column-definition* of the column being added includes the NOT NULL column constraint, then the column must either have a non-NULL default value defined or belong to a domain with a non-NULL default value. Otherwise an attempt would be made to insert the NULL value into a column which cannot accept it.

When a column is **altered**, it is possible to change the data-type of the data in it and to set or drop the column default value.

If a new data-type is set for the column, it must be assignment-compatible with the values that already exist in the column.

Note: If the column is part of a foreign key in this table or referenced by a foreign key of another table, the data-type cannot be changed.

If a column default value is set for the column, it must be assignment-compatible with the values that already exist in the column.

When the column default value is dropped, the column takes its default value from the domain to which the column belongs (if it uses a domain), otherwise the column default becomes the NULL value.

When a column is **dropped**, it is removed from the table. The keywords CASCADE and RESTRICT specify the action to be taken if other objects (such as views, table constraints, indexes, routines and triggers) exist which reference the column being dropped.

If CASCADE is specified, referencing objects will be dropped as well. If RESTRICT is specified, an error will be raised if referencing objects exist and neither the column nor the referencing objects will be dropped. If neither keyword is specified, RESTRICT behavior is the default.

It is possible to **add** a new **table constraint** to the table, which is specified in the same way it would be when a new table is created. If the table constraint is explicitly named, it cannot have the same name as a constraint that already exists on the table (see CREATE TABLE for details of table constraints).

If the existing data in a table violates the table constraint being added, the ALTER TABLE statement will fail and the new constraint will not be added to the table.

It is also possible to **drop** an existing **table constraint** in order to remove the constraint from the table. The keywords CASCADE and RESTRICT specify the action to be taken in the case of a referential constraint being dropped.

If CASCADE is specified when a referential constraint is dropped, any other referential constraints which are referencing the unique key being dropped will also be dropped.

If RESTRICT is specified an error will be raised, and nothing will be dropped, if there are other referential constraints referencing the one to be dropped. If neither keyword is specified, RESTRICT behavior is the default.

Language elements

column-definition	see CREATE TABLE.
table-constraint-definition	see CREATE TABLE.
default-value	see Section 5.3.

Restrictions

A table can only be altered by the creator of the schema to which the table belongs.

A column cannot be dropped if it is the **only** column in a table (i.e. a drop column operation may not result in a table with no columns).

The ident performing an ALTER TABLE operation must have USAGE privilege on any domain or sequence involved, EXECUTE privilege on any function involved and REFERENCES privilege on all columns specified in *references* of a referential constraint.

Notes

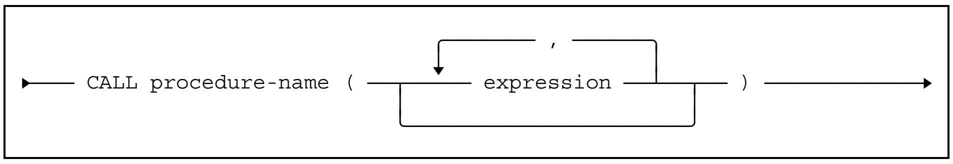
See Appendix C for information on the maximum length of a row in a table.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	EXTENDED	Support for the ALTER option which allows the data type to be changed is a Mimer SQL extension.

CALL

Calls a procedure.



Usage

Embedded/Interactive/ODBC/Procedural.

Description

The CALL statement is used to invoke a procedure. The values specified for *expression* must correspond to the parameters defined for the procedure.

The nature of each *expression* depends on the parameter it applies to. For parameters with mode OUT or INOUT, *expression* must be a target-variable (see Section 4.2.6). For parameters with mode IN, *expression* may be a value-expression.

The value of *expression* must be assignment-compatible with the data type of the parameter to which it is applied (see Section 4.5).

Restrictions

In embedded SQL the CALL statement is **not** used to invoke result set procedures - see DECLARE CURSOR for information about calling result set procedures.

In interactive SQL the CALL statement is used to invoke all types of procedures.

Recursion is permitted, an error will be raised if the internal recursion limit is exceeded.

In a procedural usage context, the called procedure must have an *access-clause* which is lower or equal to that of the calling procedure (- see CREATE PROCEDURE for details about procedure access clause values).

Notes

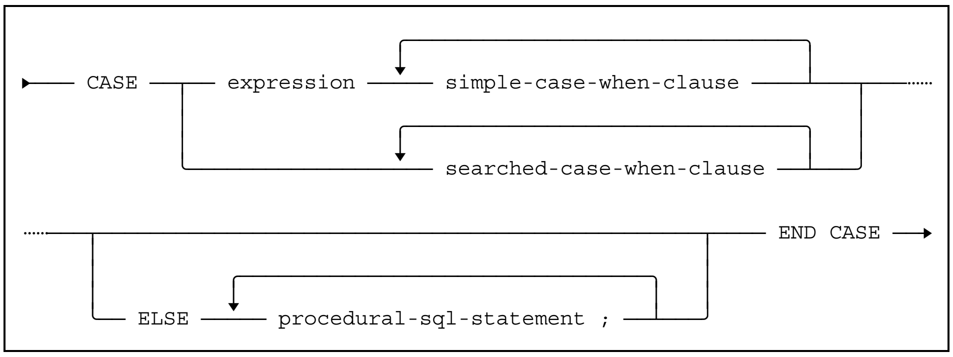
The CALL statement is **not** used to invoke a function. A function is invoked by specifying its name and parameter list in a context where a value expression would normally be used.

Standard compliance

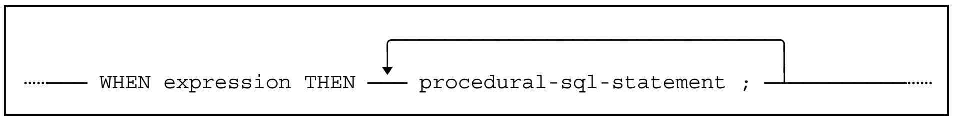
Standard	**Compliance**	**Comments**
X/Open-95 SQL92	MIMER EXTENSION	Support for the CALL statement is a Mimer SQL extension.
SQL/PSM	EXTENDED	Support for procedures that return a result-set (i.e. Result Set Procedures) is a Mimer SQL extension.

CASE

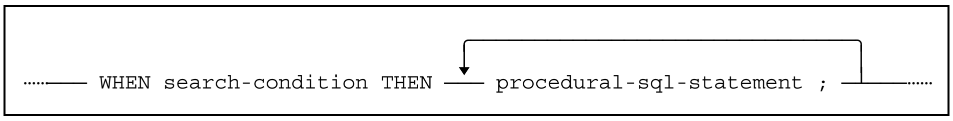
Allows sequences of SQL statements to be selected for execution based on search or comparison criteria.



where *simple-case-when-clause* is



where *searched-case-when-clause* is



Usage

Procedural.

(For information on the *case-expression*, which provides a mechanism for conditionally selecting values, see Section 5.6).

Description

The CASE statement provides a mechanism for conditional execution of SQL statements. It exists in two forms, the “simple” case and the “searched” case.

The **simple case** involves an equality comparison between one expression and a number of alternative expressions, each following a WHEN clause.

The **searched case** involves the evaluation for truth of a number of alternative search conditions, each following a WHEN clause.

In each form of the CASE it is the **first** WHEN clause to evaluate to true, working from the top down, that determines which sequence of SQL statements will be executed.

There may be one or more SQL statements following the THEN clause for each WHEN.

If none of the WHEN clauses evaluates to true, the SQL statements following the ELSE clause are executed. If none of the WHEN clauses evaluates to true and there is **no ELSE clause**, an exception condition is raised to indicate that a case was not found.

Providing an ELSE clause supporting an empty compound statement will avoid an exception condition being raised, in cases where no “else” action is required, when none of the WHEN alternatives evaluates to true.

Restrictions

None.

Notes

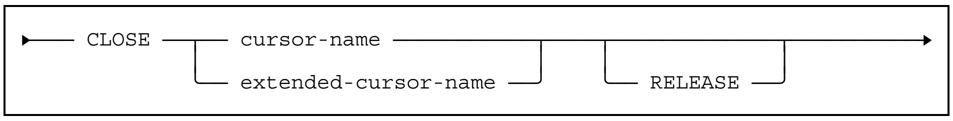
Flow of control leaves the CASE statement as soon as the SQL statements following the selected THEN, or the ELSE, have been executed (i.e. there is no “fall through” as is found in a case statement in, for example, the C programming language).

Standard compliance

Standard	**Compliance**	**Comments**
SQL/PSM	YES	Fully compliant.

CLOSE

Closes a cursor.



Usage

Embedded/Procedural.

Description

The current state of the named cursor is closed.

If any states of the cursor have been saved on a stack by successive OPEN statements (see OPEN), the most recently saved cursor state is restored. Information about whether there are cursor states remaining on the stack is returned as diagnostic information. Otherwise the cursor is closed (deactivated), and may not be used until it has been re-opened with a new OPEN statement.

If the optional keyword RELEASE is used, all resources allocated to the cursor including any stacked references are destroyed. The cursor must be re-prepared (in dynamic SQL) and reopened before it can be used again.

See ALLOCATE CURSOR for a description of extended cursors.

Restrictions

In a **Procedural** usage context, a cursor cannot be specified by extended-cursor-name.

Notes

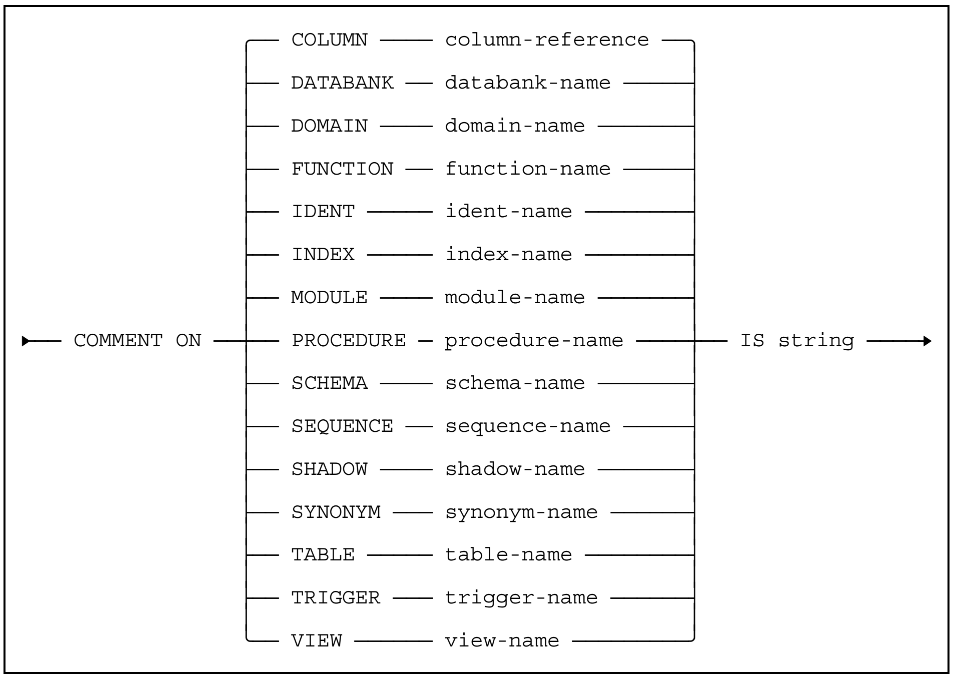
For the statement to be valid, the cursor in question must be open.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	MIMER EXTENSION	Support for the RELEASE keyword is a Mimer SQL extension.

COMMENT

Inserts or replaces a comment on a database object.



Usage

Embedded/Interactive/ODBC.

Description

The comment string for the specified object is stored in the data dictionary. Any previously defined comment for the object is replaced by the new.

Restrictions

A comment can only be stored on a **system** database object by the creator of the object.

A comment can only be stored on a **private** database object by the creator of the schema to which the object belongs.

See Section 3.1.1 for a description of system and private database objects.

Only users with SHADOW privilege may store a comment on a shadow.

Notes

A comment string may have a maximum length of 254 characters and must be enclosed in string delimiters.

Comments may not be altered or dropped directly. However, since the COMMENT statement replaces any existing comment with the new text, a comment may be altered simply by issuing a new COMMENT statement. A comment may be effectively dropped by issuing a COMMENT statement with an empty string.

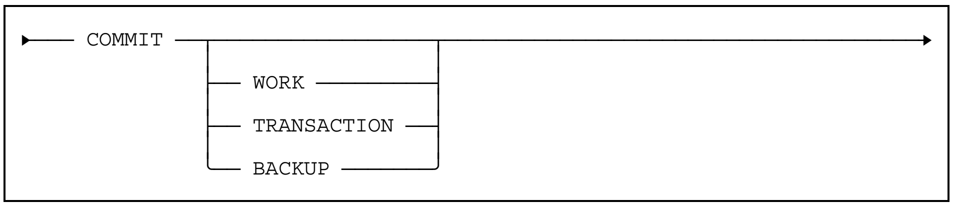
When a comment is written for a column, the column name must be qualified by a table-name or a view-name in the form *table-name.column-name* or *view-name.column-name*.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	MIMER EXTENSION	Support for the COMMENT ON statement is a Mimer SQL extension.

COMMIT

Commits the current transaction.



Usage

Embedded/Interactive/Procedural.

Description

The current transaction is terminated. Database alterations requested in the transaction build-up are executed against the database, provided that no transaction conflict is detected.

If a transaction conflict is detected, no changes are made in the database, and an error code is set to indicate a transaction conflict. The intention list established during the transaction build-up is dropped.

All cursors opened by the current ident are closed.

If there is no currently active transaction, any cursors opened by the current ident are closed, but the COMMIT statement is otherwise ignored. No error code is returned in this case.

Committing a BACKUP transaction performs online backup for all databanks for which a CREATE BACKUP command has been performed since START BACKUP. Please note that this command may be lengthy if backups for large databank files are made.

Restrictions

The COMMIT statement cannot be used in a **result set procedure** because this would close the cursor which is calling it.

The COMMIT statement cannot be used within an **atomic** compound SQL statement (see COMPOUND STATEMENT).

The COMMIT BACKUP statement can only be used when a corresponding START BACKUP command has been given. The statement is not supported in procedural mode.

Notes

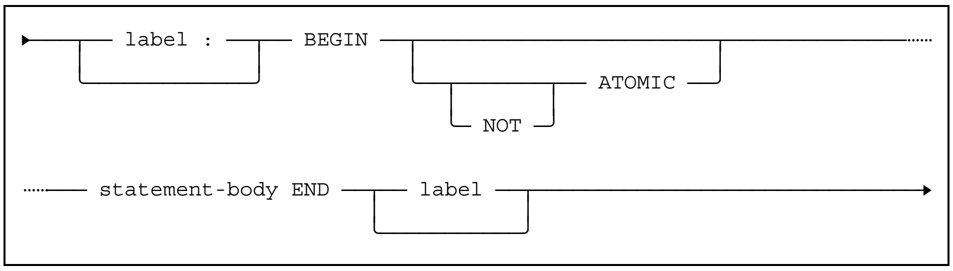
See Chapter 6 of the Mimer SQL Programmer’s Manual for a detailed discussion of transaction control.

Standard compliance

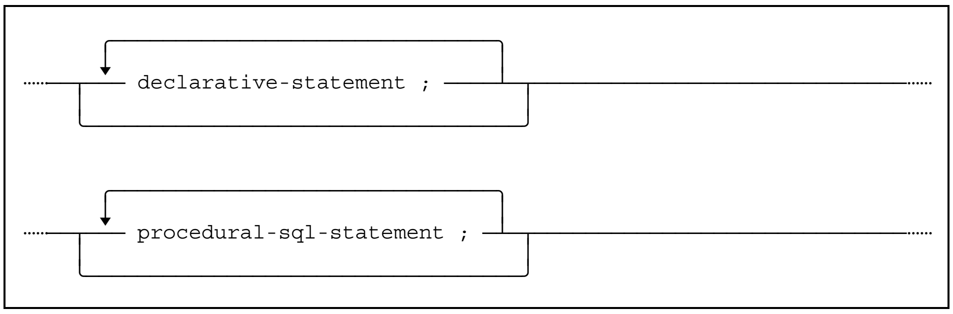
Standard	**Compliance**	**Comments**
X/Open-95 SQL92	EXTENDED	Support for the BACKUP and TRANSACTION keywords is a Mimer SQL extension.

COMPOUND STATEMENT

The compound SQL statement.



where *statement-body* is



Usage

Procedural.

Description

The compound statement is used in a routine or trigger to create an environment within which variables, cursors, exception condition names and exception handlers can be declared. A number of *procedural-sql-statement*'s can also be specified.

The *procedural-sql-statement*'s in a compound statement are executed in sequence whenever the compound statement is executed.

The compound statement may be used wherever a single *procedural-sql-statement* is permitted. Thus, it provides a mechanism for executing a sequence of statements in places where the syntax rules permit only a single statement to be specified.

Compound statements can be nested and the optional *label* value can be used to qualify the names of objects declared within the compound statement.

The *label* value can also be used in conjunction with the LEAVE statement to control the execution flow by exiting from the compound statement.

The compound statement can be defined as **atomic** by specifying ATOMIC next to the BEGIN keyword.

When a compound statement is defined as atomic, an “atomic execution context” becomes active while it, or any sub-query contained in a statement within it, is executing.

While an atomic execution context is active:

- It is not possible to explicitly terminate a transaction.
- If an SQL statement fails to execute successfully, RESIGNAL is effectively executed and any changes already successfully made can be potentially committed, depending on the error handling in effect.

Restrictions

If ATOMIC is specified, the ROLLBACK and COMMIT statements must **not** be used in the compound statement.

A compound statement which contains a declaration of an UNDO exception handler **must** be ATOMIC.

Notes

A compound statement without an ATOMIC or NOT ATOMIC specification is assumed to be **NOT ATOMIC**.

The value of *label* must be the same at both ends of the compound statement.

If *label* is specified at the **end** of the compound statement it **must** also be specified at the beginning.

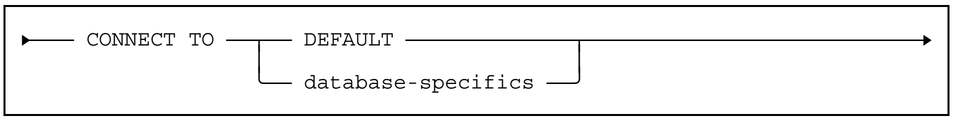
If the LEAVE statement is to be used to exit the compound statement, the label at the **beginning** must be specified.

Standard compliance

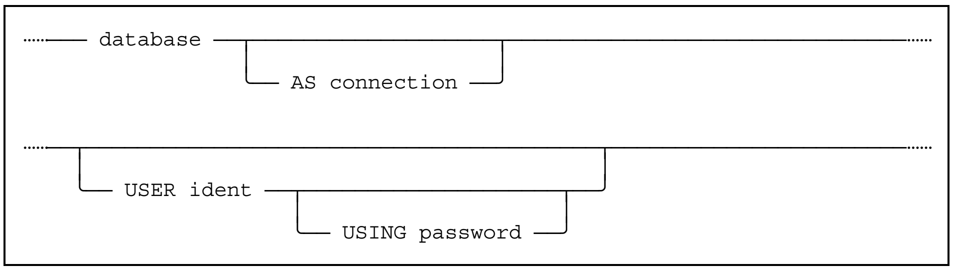
Standard	**Compliance**	**Comments**
SQL/PSM	YES	Fully compliant.

CONNECT

Connects a user ident to a database.



where *database-specifics* is



Usage

Embedded/Interactive.

Description

The ident is logged into the specified database. The database may exist on the local machine (a **local database**) or on another machine in a network configuration (a **remote database**).

The *database*, *connection*, *ident* and the *password* can be supplied either using a host variable or as a literal value.

If an empty string is specified for *database*, a connection is established to the DEFAULT database (see Section 3.7.2 of the Mimer SQL System Management Handbook for details on how the DEFAULT database is defined in the Mimer SQL system).

If *ident* is not specified (or *ident* is specified as blank), the name of the current operating system user is assumed. In that case, if an OS_USER ident with that name exists in the selected database, a connect will be established without any password verification.

Note: It is only possible to establish a connection to a **remote database** without specifying *ident* (or specifying a blank *ident*) if both the node on which the database resides and the node from which the connection is attempted are running the Windows operating system, and the NamedPipes protocol is used for the network communication. It is not possible for the database server node to determine the name of the operating system user currently using the remote machine in other network configurations.

When connected, the ident is able to access the database and becomes the current ident (i.e. the name the returned by SESSION_USER).

If *connection* is specified, the name must be a valid identifier or an empty string.

Note: Connection names must be unique. If an empty string is specified, or if no connection name is given, the value of *database* will be used as the connection name.

Ident, *database* and *connection* are **not** case-sensitive in the CONNECT statement.

Password **is** case-sensitive in the CONNECT statement.

Restrictions

Only idents of type USER and OS_USER can connect to a database using the CONNECT statement.

Notes

If it is desired that a CONNECT TO DEFAULT be effectively performed, but with the possibility of specifying one or more of *connection*, *ident* or *password*, then specify *database-specifics* but supply an empty string for *database*.

The maximum length of *database*, *ident* and *connection* is 128 characters.

The maximum length of *password* is 18 characters.

If an SQL statement is executed in an application without first executing a CONNECT statement, an implicit CONNECT TO DEFAULT is performed (this requires that an OS_USER ident exist in the default database with the same name as the operating system user and that the default database either be a local database or a remote database residing on a node which allows the name of the current operating system user to be determined - see the related note in the ***Description*** section for details).

Such an implicit default connection will only be established if the CONNECT statement has not been previously executed in the application. This means that if an **explicit** connection has been previously established and then disconnected, any subsequent attempt to execute an SQL statement without a current connection will result in either a “Connection does not exist” error or a transaction rollback depending on the context of the SQL statement.

If only the **implicit default** connection has been previously established and then disconnected, any subsequent attempt to execute an SQL statement without a current connection will result in that connection being re-established.

Observe that it is possible for the implicit default connection to exist but not be currently active (this will be the case if a connection is has been subsequently established and then disconnected).

It is recommended that Mimer SQL applications always establish **explicit** connections and reliance on the implicit default connection is discouraged.

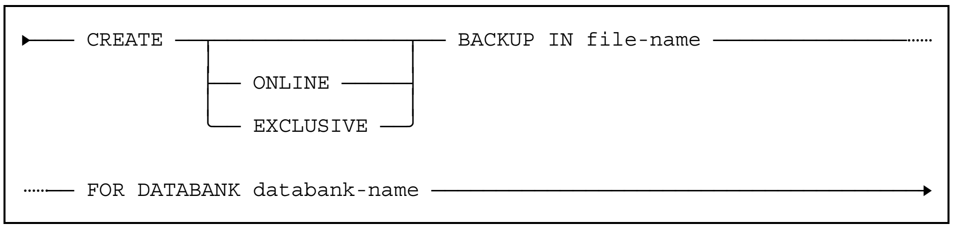
Earlier versions of Mimer SQL used a different syntax for the CONNECT statement (see Appendix D). This syntax is still supported for backward compatibility, but its use is not recommended in new applications.

Standard compliance

Standard	Compliance	Comments
X/Open-95 SQL92	EXTENDED	Support for the USING *password* clause is a Mimer SQL extension.

CREATE BACKUP

Takes a backup copy of a databank file.



Usage

Embedded/Interactive/ODBC.

Description

This SQL statement is used to take a backup of a databank.

A **backup** is a copy of the current databank file and may be used as the basis for a databank recovery operation (see ALTER DATABANK RESTORE).

The backup will be recorded in a file on disk, the name of the file is specified in the CREATE BACKUP statement.

In order to preserve the consistency of the backup between related databanks, a backup of each of the databanks must be taken at exactly the same point in time, from the point of view of transactions updating the databanks. This is done by starting a transaction for the online backup operations using the START BACKUP statement, then executing a CREATE BACKUP statement for each databank to be backed up. Finally conclude the transaction by executing the COMMIT BACKUP statement or ROLLBACK BACKUP statement.

It is recommended that **all** databanks (including System databanks) in a database are backed up together in this way.

The CREATE BACKUP command creates the backup file. The actual copying of data from the databank to the backup file is not done until a COMMIT BACKUP is executed.

When the keyword EXCLUSIVE is used, the backup of the databank will be taken without allowing any concurrent operations. Otherwise, the backup will be taken online, i.e. other operations can be executed concurrently.

When a backup of LOGDB is taken, changes made on all databanks are copied to the backup. I.e. this corresponds to taking an incremental backup of all databanks. The entire log is dropped when the backup transaction is committed.

When LOGDB is not included in the backup, only the information that applies to the backed up databanks is dropped from the database log. Note that, in this case, it will not be possible to restore the databanks from a previous backup, as the log records are not saved. Therefore, it is highly recommended to always include LOGDB whenever any databank is backed up.

Restrictions

CREATE BACKUP requires that the current ident be the creator of the databank or have BACKUP privilege.

The CREATE BACKUP statement cannot be executed unless a transaction, that was started by executing a START BACKUP statement, is currently active.

A backup requires read access to all tables in the databank. It is therefore not possible to take a backup when commands, such as ALTER TABLE and CREATE INDEX, are executing. When a backup has been initiated, commands that require exclusive access will get an error indicating the table is in use by another user.

Notes

The value of *filename-string* must always be enclosed in string delimiters. The maximum length of *filename-string* is 256 characters. Refer to Section 3.1.2.1 for details concerning specification of the pathname components in *filename-string*.

The CREATE BACKUP command can be used with all databanks in a database including SYSDB, TRANSDB, LOGDB, and SQLDB.

The databank option will affect the backup copy:

LOG	A consistent backup is made of the databank. Transaction logging is used and it will be possible to redo operations made after the backup.
TRANS	A consistent backup is made of the databank. But as transaction logging is not used, it will not be possible to redo operations made after the backup. I.e. if a disk is corrupted, it is only possible to revert to the state of the lastest backup.
NULL	An online backup of the databank will give a backup which is not completly consistent as the system uses the transaction system to make backups. For a completly consistent backup to be made, the keyword EXCLUSIVE must be used in the CREATE BACKUP command.

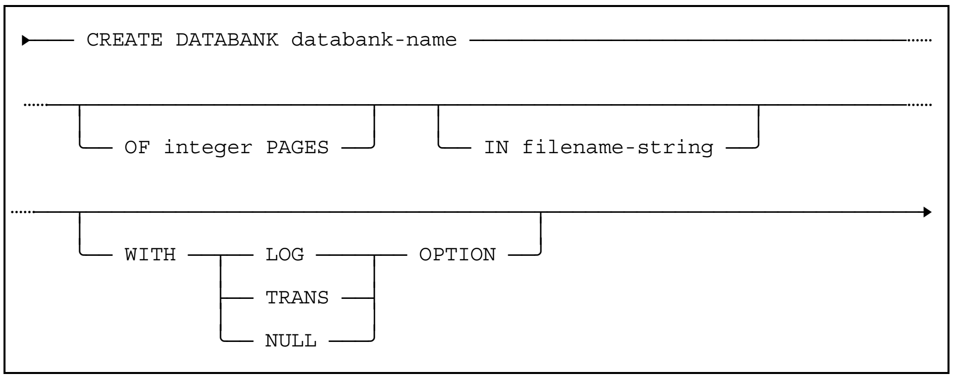
The removal of records from the database log to maintain consistency with the backups is handled automatically by these statements, i.e. no additional commands are needed.

Standard compliance

Standard	Compliance	Comments
X/Open-95 SQL92	MIMER EXTENSION	The CREATE BACKUP statement is a Mimer SQL extension.

CREATE DATABANK

Creates a new databank.



Usage

Embedded/Interactive/ODBC.

Description

A new databank is created, i.e. a physical file is created in the host file system and formatted for use as a Mimer SQL databank. The initial file size is the number of Mimer SQL pages (2K in size) specified by the *integer* parameter. A value of 1000 Mimer SQL pages is assumed if an initial file size is not specified.

The *filename-string* specifies the name of the new databank file in the host file system and this is stored in the data dictionary as the location of the databank file. If a file name is not specified, it will be the same as *databank-name* (and the databank will be created in the database home directory).

The databank is created with the transaction and logging options as specified:

LOG	All operations on the databank are performed under transaction control. All transactions are logged, i.e. it will be possible to restore the databank from a backup.
TRANS	All operations on the databank are performed under transaction control. No transactions are logged. (This databank option is assumed if one is not explicitly specified.)
NULL	All operations on the databank are performed without transaction control (even if they are requested within a transaction) and they are not logged. Set operations (DELETE, UPDATE and INSERT on several rows) which are interrupted, will not be rolled back. All secondary indexes created in the databank are flagged as “not consistent” (a secondary index that is flagged as “not consistent” will not offer optimal performance when used in a query - see UPDATE STATISTICS for information on how to ensure that secondary indexes are consistent).

Restrictions

CREATE DATABANK requires that the current ident has DATABANK privilege.

The databank name must not be the same as that of an existing databank or shadow.

The databank must be created with either the TRANS or LOG option if any of the following are true:

- the databank is to be shadowed
- the databank will be used to store tables defined with foreign or unique keys
- the databank will be used to store tables that are referenced in a foreign key context
- the databank will be used to store tables holding UNIQUE indexes
- the databank will be used to store tables on which triggers have been created
- the databank contains tables that will accept updates in their primary key column(s)

Notes

The creator of the databank is granted TABLE privilege on the new databank, with the WITH GRANT OPTION.

The value of *filename-string* must always be enclosed in string delimiters. The maximum length of the filename string is 256 characters.

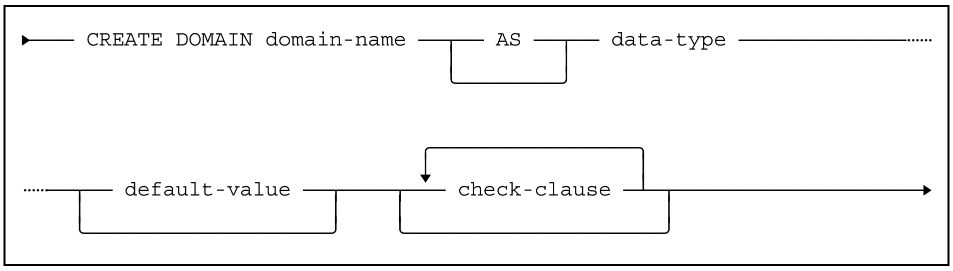
Refer to Section 3.1.2.1 for details concerning the specification of pathname components in *filename-string*.

Standard compliance

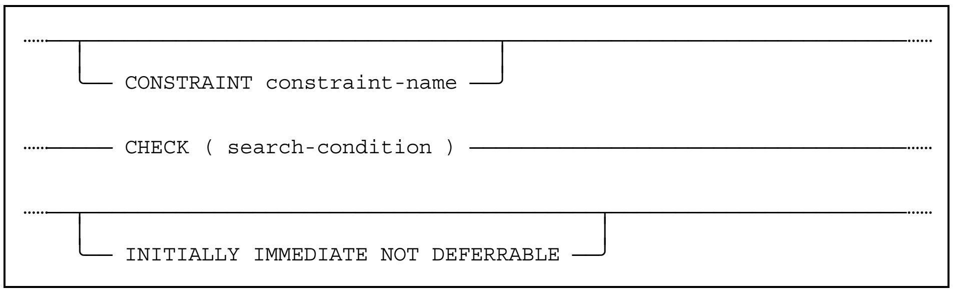
Standard	**Compliance**	**Comments**
X/Open-95 SQL92	MIMER EXTENSION	The CREATE DATABANK statement is a Mimer SQL extension.

CREATE DOMAIN

Creates a domain.



where *check-clause* is:


CONSTRAINT constraint-name
CHECK (search-condition)
INITIALLY IMMEDIATE NOT DEFERRABLE

Usage

Embedded/Interactive/ODBC.

Description

A domain is created with the properties specified in the statement. Domains may be used instead of explicit data type specifications to define column formats in the CREATE and ALTER TABLE statements.

If *domain-name* is specified in its unqualified form, the domain will be created in the schema which has the same name as the current ident.

If *domain-name* is specified in its fully qualified form (i.e. *schema-name.domain-name*) the domain will be created in the named schema (in this case, the current ident must be the creator of the specified schema).

Refer to Section 4.3 for a description of how the various data types are specified for the domain.

If *default-value* is specified, this value will be assigned to a column defined using the domain whenever a new table row is created or an existing table row is updated without an explicit value being specified for that column.

The CHECK clause
Specification of a CHECK clause means that only values for which the search condition does **not** evaluate to **false** may be assigned to a column defined using the domain.

The search condition (see Section 5.10) in the CHECK clause may only reference the domain (by using the keyword VALUE), literals, user-defined function invocations or the keyword NULL. The CHECK clause must **not** contain any non-deterministic expressions, e.g. CURRENT_DATE.

Example

```
CHECK (VALUE IN (-1,0,3)
 OR VALUE BETWEEN 5 AND 9)
```

References to columns, subselects, set functions or host variables are not allowed.

Specifying INITIALLY IMMEDIATE NOT DEFERRABLE explicitly states that the check constraint will be, by default, verified at the time the relevant data manipulation operation is performed rather than when the transaction is committed and that the verification may never be explicitly deferred until the time the transaction is committed. This is also the default behavior. (This is to allow for future extensions to the Mimer SQL syntax.)

Language elements

default-value	see Section 5.3.

Restrictions

An ident must have USAGE privilege on the domain in order to use it.

Notes

The domain name may not be the same as the name of any other domain belonging to the same schema.

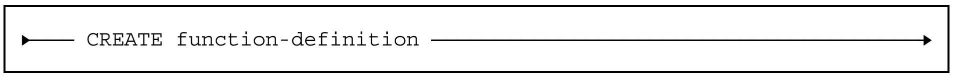
The CREATE DOMAIN statement does **not** verify that any specified default value conforms to the restrictions of any specified CHECK clause. It is, therefore, possible to create a domain definition where attempts to store the default value in a column defined using the domain will fail.

Standard compliance

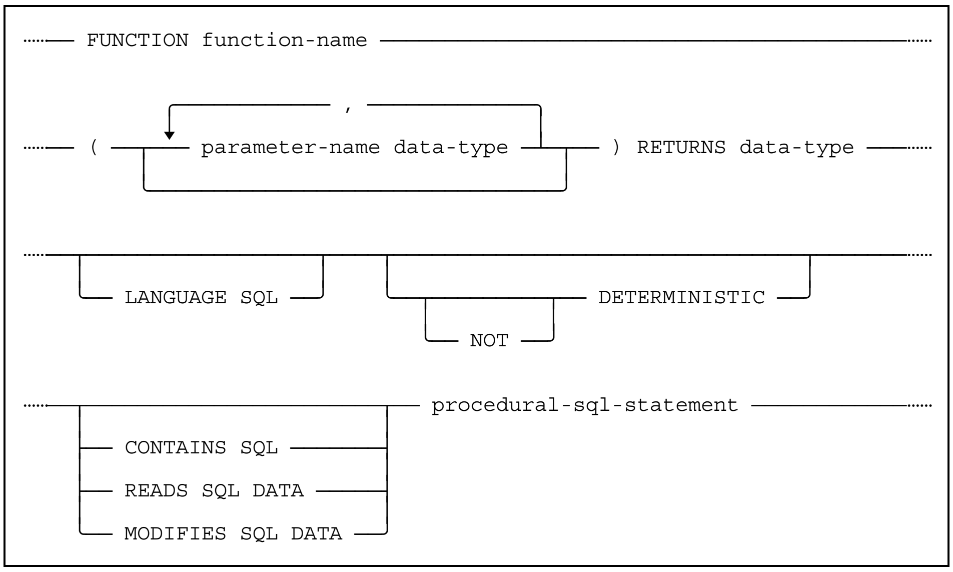
Standard	**Compliance**	**Comments**
X/Open-95	MIMER EXTENSION	The CREATE DOMAIN statement is a Mimer SQL extension.
SQL92	YES	Fully compliant.

CREATE FUNCTION

Creates a new stored user-defined function.



where *function-definition* is



Usage

Embedded/Interactive/ODBC.

Description

The *function-name* should follow the normal rules for naming database objects (see Section 4.2.2).

If *function-name* is specified in its unqualified form, the function will be created in the schema which has the same name as the current ident.

If *function-name* is specified in its fully qualified form (i.e. *schema-name.function-name*) the function will be created in the named schema (in this case, the current ident must be the creator of the specified schema).

The fully qualified function name must be used by all idents except the ident that has the same name as the schema to which the function belongs.

The *parameter-name* should follow the normal rules for naming SQL identifiers (see Section 4.2).

The permitted values for *data-type* are described in Section 4.3.

If neither DETERMINISTIC nor NOT DETERMINISTIC is specified, then NOT DETERMINISTIC is implicit.

If DETERMINISTIC is specified, then the function is guaranteed to produce the same result every time it is invoked with the same set of input values and repeated invocations of it can, therefore, be optimized.

The following access options may be specified:

CONTAINS SQL	The function may **not** contain any data-manipulation-statements. All other procedural-sql-statements are permitted. The function may only invoke CONTAINS SQL functions and procedures. This option effectively prevents a routine from performing read or write operations on data in the database.
READS SQL DATA	All procedural-sql-statements are permitted except those performing updates (i.e. DELETE, INSERT and UPDATE). The function may only invoke CONTAINS SQL or READ SQL DATA functions and procedures. This option effectively prevents a routine from performing write operations on data in the database.
MODIFIES SQL DATA	All procedural-sql-statements are permitted and any function or procedure may be invoked from this type of function. This option allows a routine to performread and write operations on data in the database.

If neither CONTAINS SQL, READS SQL DATA nor MODIFIES SQL DATA is specified, then CONTAINS SQL is implicit.

Restrictions

A function created this way cannot be added to a module.

Two functions with the same name cannot belong to the same schema.

It is not possible to create a synonym for a function name.

A parameter name must be unique within the function.

The parameter mode cannot be specified for a function parameter (as it is for a procedure parameter).

The ROW data type cannot be specified in *data-type*.

If DETERMINISTIC is specified, the *procedural-sql-statement* of the function may not contain, or be, a reference to: SESSION_USER, CURRENT_PROGRAM, CURRENT_DATE, LOCALTIME or LOCALTIMESTAMP and the function may not invoke functions or procedures that are not deterministic.

If an invoked function attempts to execute a COMMIT or ROLLBACK statement in a context where this is not permitted, (i.e. after being invoked from within a result set procedure, from within an atomic compound statement or from a data manipulation statement in one of these contexts) an exception will be raised.

An ident must have EXECUTE privilege on the function in order to invoke it.

Notes

A function is invoked by specifying its name and parameter list where a value-expression would be used.

All function parameters have the default mode (which is **IN**). See CREATE PROCEDURE for details on the parameter modes.

A *parameter-name* can be the same as the name of the function, but this is not recommended.

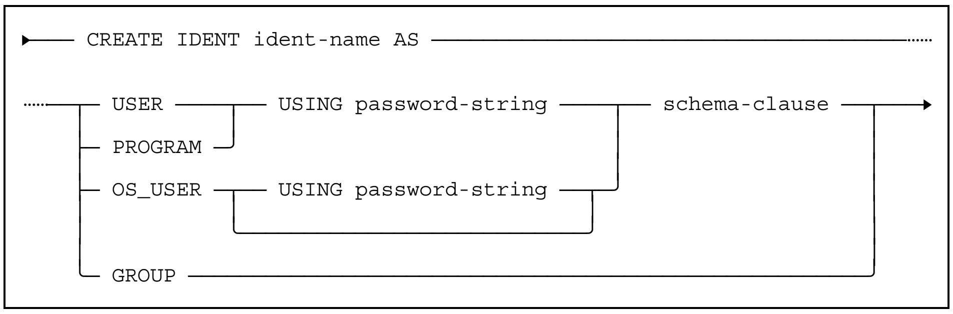
Refer to Section 7.6 of the Mimer SQL User's Manual for details on using the CREATE FUNCTION statement in BSQL, where the "@" delimiter is required.

Standard compliance

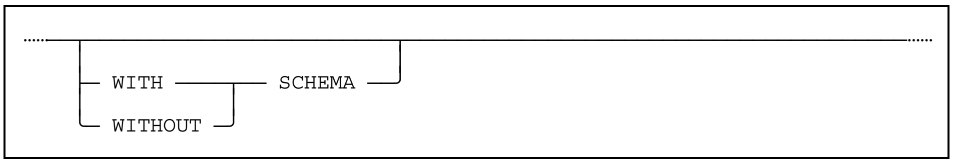
Standard	**Compliance**	**Comments**
SQL/PSM	YES	Fully compliant.

CREATE IDENT

Creates a group, OS_USER, program or user ident.



where *schema-clause* is



Usage

Embedded/Interactive/ODBC.

Description

A new ident is created. If the ident is a USER, OS_USER or PROGRAM ident, a **schema** with the same name as the ident can also be created. A schema is created by default and when WITH SCHEMA is explicitly specified.

If the ident is a USER or PROGRAM ident, a password **must** be specified.

If the ident is an OS_USER, a password can be optionally specified.

USER idents are authorized to access a Mimer SQL database by using the CONNECT statement. In interactive contexts, e.g. when BSQL is started, a USER ident is used to log in.

OS_USER idents are a special type of USER ident which can be used to connect or log in a more automatic way. Once the connection has been established, an OS_USER ident will access the database as a USER ident.

If the CONNECT statement is used without specifying an ident name (or if <return> is pressed at the username prompt when logging into BSQL), the connect attempt uses the name of the operating system user id. In this case, the connection process will automatically attempt to use an OS_USER ident with that name. If an OS_USER ident exists in the database with that name, a connection is established without any password verification.

The same is true if the ident name specified in the CONNECT statement (or at the username prompt of BSQL) is the same as the name of the current operating system user id and an OS_USER ident exists in the database with that name.

If an OS_USER ident is created with a password, it can be used as if it were a USER ident in situations where the operating system user id does not match the OS_USER ident name.

PROGRAM idents cannot be used to connect to a database. After a connection has been established (by using a USER or OS_USER ident), the ENTER statement can used to make a PROGRAM ident the current ident. The access rights to the database defined for the PROGRAM ident will thus come into effect.

The ident executing the ENTER statement must have EXECUTE privilege on the PROGRAM ident (the ENTER statement can be executed by a PROGRAM ident).

The ident that executed the ENTER statement will become the current ident again after the LEAVE statement has been executed.

GROUP idents cannot be used to connect to a database. They are used to implement collective authorization of access rights to the database. Other idents become members of a GROUP ident when MEMBER privilege on the GROUP ident is granted to them.

While an ident is a member of a GROUP ident, that ident is effectively granted the privileges held by the GROUP ident.

For a more detailed description of idents, see Chapter 4 of the Mimer SQL Programmer's Manual.

Restrictions

CREATE IDENT requires that the current ident have IDENT privilege.

The ident must not have the same name as an ident that already exists in the database.

Notes

All letters in OS_USER names are treated as uppercase in Mimer SQL, regardless of operating system conventions.

The creator of a GROUP ident is automatically granted MEMBER privilege on it, with the WITH GRANT OPTION.

The creator of a PROGRAM ident is automatically granted EXECUTE privilege on it, with the WITH GRANT OPTION.

Ident passwords must be at least 1 and at most 18 characters long and may contain any characters except space. The case of alphabetic characters is **significant**. The password string must be enclosed in string delimiters, which are not stored as part of the password.

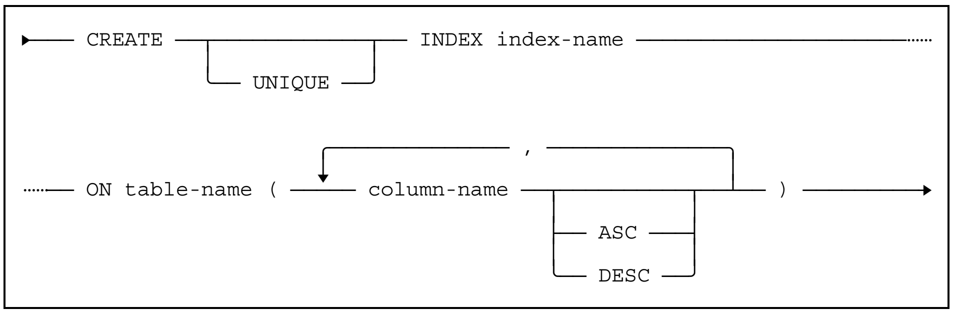
An ident who is authorized to created new idents (by having IDENT privilege) can also create new schemas.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	MIMER EXTENSION	The CREATE IDENT statement is a Mimer SQL extension.

CREATE INDEX

Creates a secondary index on one or more columns of a table.



Usage

Embedded/Interactive/ODBC.

Description

A secondary index is created on the column(s) in the table as specified. The index is stored in the data dictionary under the given name. The secondary index is used internally by the optimizer to improve the efficiency of a search.

If *index-name* is specified in its unqualified form, the index will be created in the schema which has the same name as the current ident.

If *index-name* is specified in its fully qualified form (i.e. *schema-name.index-name*) the index will be created in the named schema (in this case, the current ident must be the creator of the specified schema).

If UNIQUE is specified each index value (i.e. the value of all index columns together) is only allowed once. In this context two null values are considered equal. It is recommended that UNIQUE is used in the create table statement rather than in the create index statement.

ASC and DESC indicate the sort order of the column within the index. If neither is specified, then ASC is implicit. This makes an index appropriate for queries with a matching ORDER BY specification.

Restrictions

Two indexes with the same name cannot belong to the same schema.

An index must belong to the same schema as the table on which it is created.

Secondary indexes may only be created on base tables, not on views.

UNIQUE indexes may only be created on tables in databanks defined with the LOG or TRANS transaction option.

Notes

Each column name must identify an existing column of the table. The same column may not be identified more than once.

Mimer SQL can make use of an index in both the forward and backward direction. It is therefore immaterial whether ASC or DESC is specified if all the index columns have the same sorting direction.

Secondary indexes are automatically maintained and are invisible to the user. The index is used automatically when it provides better efficiency.

Any column may be specified as a secondary index.

Note: Table columns that are in the primary key, a unique key or used in a foreign key reference are automatically indexed (in the order in which they are defined in the key). Therefore, explicitly creating an index on these columns will not improve performance.

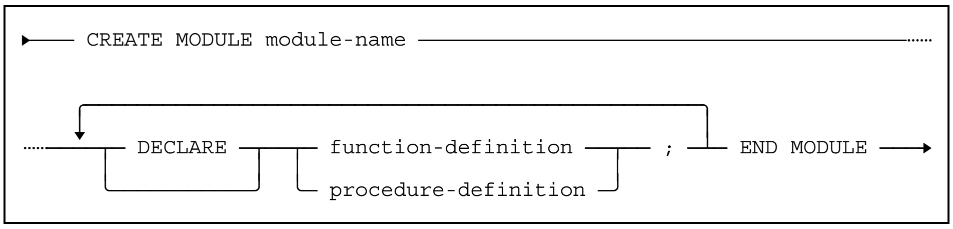
Example: Consider a table with columns A, B and C of which A and B form the primary key, in that order. An index is automatically created for the column combination A,B. Therefore, there is no advantage in explicitly creating an index on column A or on the column combination A,B. Secondary indexes may, however, be advantageous on column B alone or on combinations such as B,A or A,C.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95	YES	Fully compliant.
SQL92	MIMER EXTENSION	The CREATE INDEX statement is a Mimer SQL extension.

CREATE MODULE

Creates a new module.



Usage

Embedded/Interactive/ODBC.

Description

If *module-name* is specified in its unqualified form, the module will be created in the schema which has the same name as the current ident.

If *module-name* is specified in its fully qualified form (i.e. *schema-name.module-name*) the module will be created in the named schema (in this case, the current ident must be the creator of the specified schema).

A module is simply a convenient enclosure for the collection of one or more routines that are declared as belonging to the module when it is created.

Language elements

function-definition	- see CREATE FUNCTION.
procedure-definition	- see CREATE PROCEDURE.

Restrictions

Two modules with the same name cannot belong to the same schema.

All the functions and procedures declared as belonging to the module **must** be created in the same schema as the module.

Two functions with the same name cannot belong to the same schema.

Two procedures with the same name cannot belong to the same schema.

It is not possible to create a **synonym** for a module name.

Notes

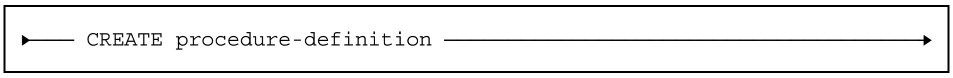
The names of the functions and procedures declared as belonging to the module are qualified by using the name of schema to which they belong and **not** the name of the module.

Standard compliance

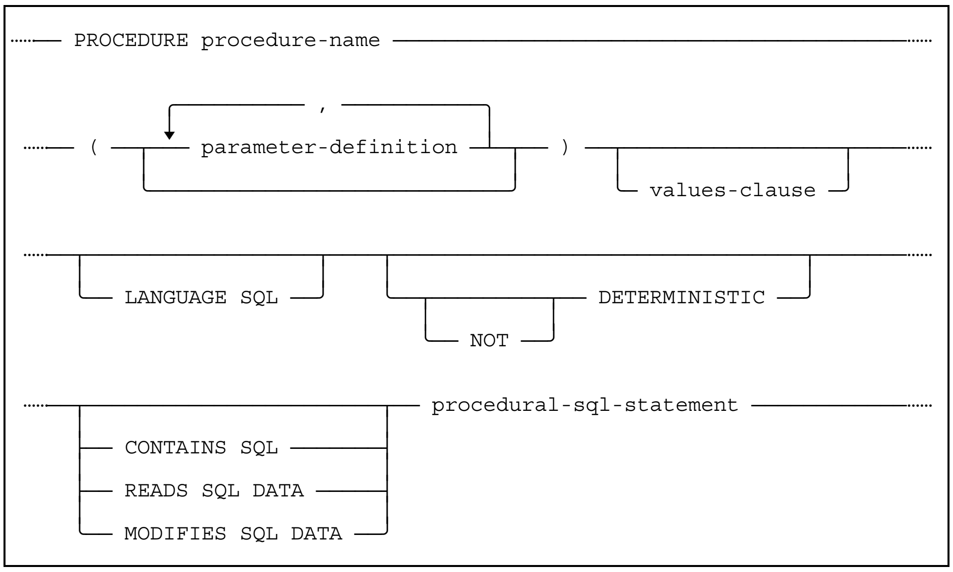
Standard	**Compliance**	**Comments**
SQL/PSM	YES	Fully compliant.

CREATE PROCEDURE

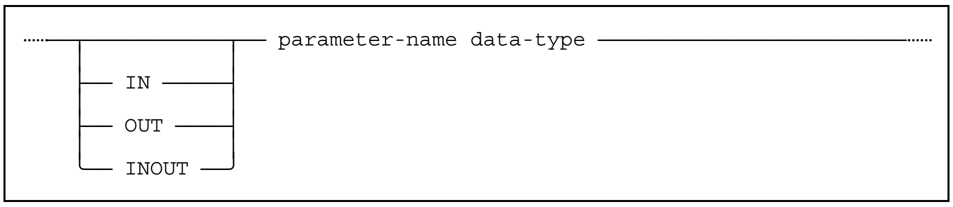
Creates a new stored procedure.



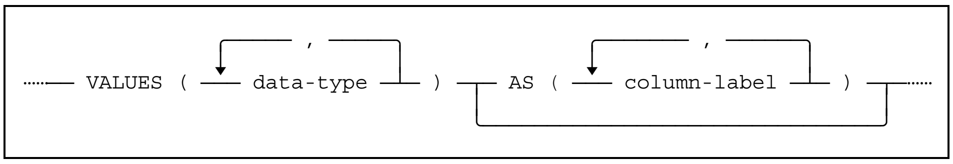
where *procedure-definition* is



and *parameter-definition* is



and *values-clause* is



Usage

Embedded/Interactive/ODBC.

Description

The *procedure-name* should follow the normal rules for naming database objects (see Section 4.2.2).

If *procedure-name* is specified in its unqualified form, the procedure will be created in the schema which has the same name as the current ident.

If *procedure-name* is specified in its fully qualified form (i.e. *schema-name.procedure-name*) the procedure will be created in the named schema (in this case, the current ident must be the creator of the specified schema).

The fully qualified procedure name must be used by all idents except the ident that has the same name as the schema to which the procedure belongs.

The *parameter-name* in the *parameter-definition* should follow the normal rules for naming SQL identifiers (see Section 4.2).

The following mode values may be specified in a *parameter-definition*:

IN	the parameter is effectively "read-only", i.e. it cannot be used as the target in an assignment, fetch or select into statement in the procedure
OUT	the parameter is effectively "write-only", i.e. it can only be used as the target for an assignment and cannot be used in a value expression in the procedure. This type of parameter must be a variable in the procedure CALL statement
INOUT	the parameter can be used both as an IN and OUT parameter, this type of parameter must be a variable in the procedure CALL statement.

The permitted values for *data-type*, specified in *parameter-definition*, are described in Section 4.3.

If a *values-clause* is specified, the procedure is created as a **result set procedure**. A result set procedure is a special type of procedure which returns a result-set and is called by being specified in a cursor declaration (see DECLARE CURSOR) rather than by using the CALL statement.

If neither DETERMINISTIC nor NOT DETERMINISTIC is specified, then NOT DETERMINISTIC is implicit.

If DETERMINISTIC is specified, then the procedure is guaranteed to produce the same result every time it is invoked with the same set of input values and repeated invocations of it can, therefore, be optimized.

The following access options may be specified:

CONTAINS SQL	The procedure may **not** contain any data-manipulation-statements. All other procedural-sql-statements are permitted. The procedure may only invoke CONTAINS SQL functions and procedures. This option effectively prevents a routine from performing read or write operations on data in the database.
READS SQL DATA	All procedural-sql-statements are permitted except those performing updates (i.e DELETE, INSERT and UPDATE). The procedure may only invoke CONTAINS SQL or READ SQL DATA functions and procedures. This option effectively prevents a routine from performing write operations on data in the database.
MODIFIES SQL DATA	All procedural-sql-statements are permitted and any function or procedure may be invoked from this type of procedure. This option allows a routine to perform read and write operations on data in the database.

If neither CONTAINS SQL, READS SQL DATA nor MODIFIES SQL DATA is specified, then CONTAINS SQL is implicit.

Restrictions

A procedure created this way cannot be added to a module.

Two procedures with the same name cannot belong to the same schema.

It is not possible to create a synonym for a procedure name.

A parameter name must be unique within the procedure.

The ROW data type cannot be specified in *parameter-definition* or in a *values-clause*.

If the procedure contains a COMMIT or ROLLBACK statement, it must **not** be invoked from within a result set procedure.

A result set procedure may only have parameters with mode IN.

A result set procedure must **not** execute a COMMIT or ROLLBACK statement because this will interfere with the cursor used when the result set procedure is called.

A result set procedure must **not** invoke a function that executes a COMMIT or ROLLBACK statement.

If DETERMINISTIC is specified, the *procedural-sql-statement* of the procedure may not contain, or be, a reference to: SESSION_USER, CURRENT_DATE, CURRENT_PROGRAM, LOCALTIME or LOCALTIMESTAMP.

The option MODIFIES SQL DATA cannot be used for a result set procedure.

An ident must have EXECUTE privilege on the procedure in order to invoke it.

Notes

If neither IN, OUT nor INOUT is specified in a *parameter-definition* then IN is implicit.

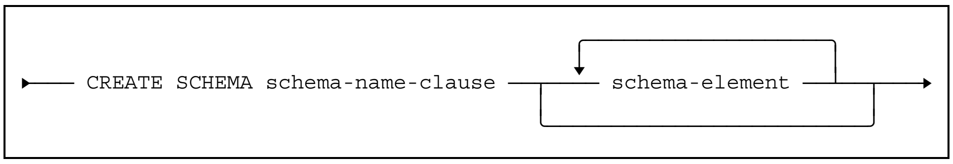
Refer to Section 7.6 of the Mimer SQL User's Manual for details on using the CREATE PROCEDURE statement in BSQL where the "@" delimiter is required.

Standard compliance

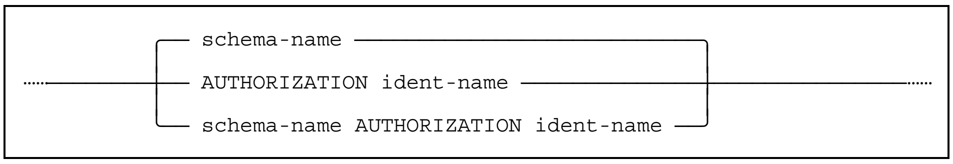
Standard	**Compliance**	**Comments**
SQL/PSM	EXTENDED	Support for the VALUES clause, i.e. the concept of a procedure returning a result-set, is a Mimer SQL extension.

CREATE SCHEMA

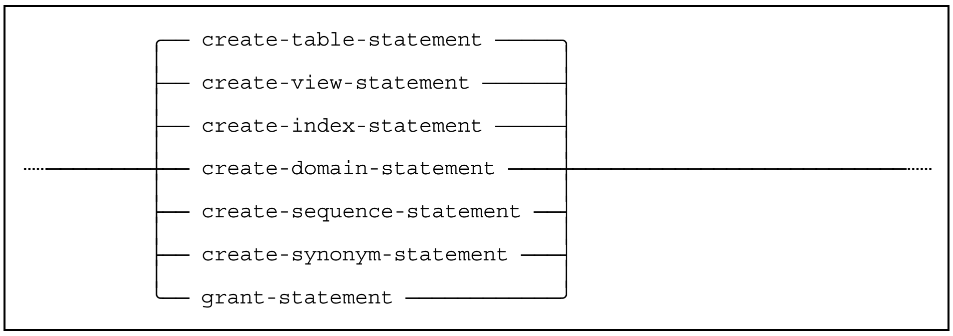
Creates a new schema.



where *schema-name-clause* is



where *schema-element* is



Usage

Embedded/Interactive/ODBC.

Description

A new schema is created with the name specified in *schema-name-clause*. If *schema-name* is specified, the schema is created with that name, otherwise the name of the schema will be the same as *ident-name*.

If *ident-name* is specified, the schema and all the other objects created by the CREATE SCHEMA statement are created with the named ident as the effective current ident.

A *schema-element* is a CREATE or GRANT statement that is specified using the normal syntax for such a statement and which is executed by the CREATE SCHEMA statement in the normal way.

Language elements

create-table-statement	- see CREATE TABLE.
create-view-statement	- see CREATE VIEW.
create-index-statement	- see CREATE INDEX.
create-domain-statement	- see CREATE DOMAIN.
create-sequence-statement	- see CREATE SEQUENCE.
create-synonym-statement	- see CREATE SYNONYM.
grant-statement	- see GRANT ACCESS PRIVILEGE or GRANT OBJECT PRIVILEGE.

Restrictions

The schema name must not be the same as that of a schema which already exists in the database.

CREATE SCHEMA requires that the current ident has SCHEMA or IDENT privilege.

The value for *ident-name* is currently restricted to be the name of the current ident.

A *grant-statement* is restricted to GRANT ACCESS PRIVILEGE, GRANT USAGE ON DOMAIN or GRANT USAGE ON SEQUENCE (i.e. it is only possible to grant privileges on **private** database objects).

If a *schema-element* contains a CREATE statement where the name of the object to be created is specified in its fully qualified form (i.e. *schema_name.object_name*), the *schema-name* component **must** be the same as the name of the schema being created by the CREATE SCHEMA statement.

Notes

If a *schema-element* contains a CREATE statement where the name of the object to be created is specified in an unqualified form, it will be created in the schema created by the CREATE SCHEMA statement (and not the schema with the same name as the current ident as is usual for CREATE statements).

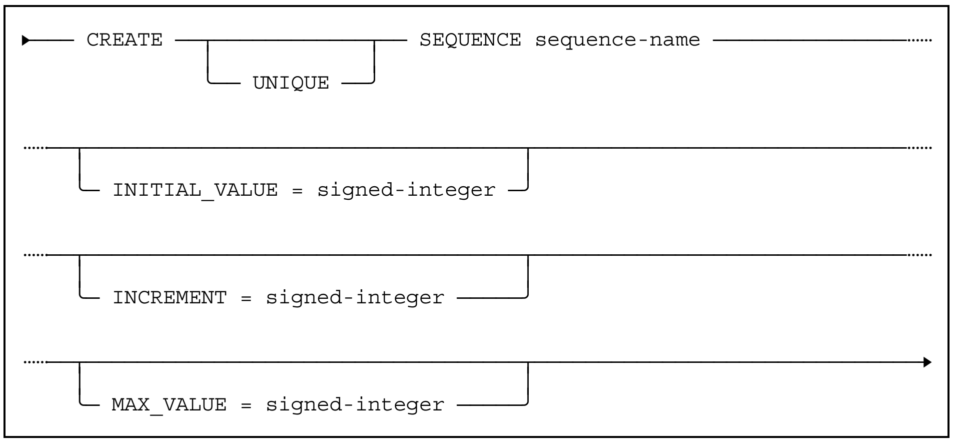
It is possible for one *schema-element* to reference the objects created by other *schema-element*'s regardless of the order of creation of the objects. All object references are verified at the **conclusion** of the CREATE SCHEMA statement when all the *schema-element*'s have been executed and all objects have been created.

Standard compliance

Standard	**Compliance**	**Comments**
SQL92	EXTENDED	Support for *create-index-statement*, *create-sequence-statement* and *create-synonym-statement* in *schema-element* is a Mimer SQL extension.

CREATE SEQUENCE

Creates a new sequence.


CREATE
UNIQUE
SEQUENCE sequence-name
INITIAL_VALUE = signed-integer
INCREMENT = signed-integer
MAX_VALUE = signed-integer

Usage

Embedded/Interactive/ODBC.

Description

A new sequence is created.

The *sequence-name* should follow the normal rules for naming database objects (see Section 4.2.2).

If *sequence-name* is specified in its unqualified form, the sequence will be created in the schema which has the same name as the current ident.

If *sequence-name* is specified in its fully qualified form (i.e. *schema-name.sequence-name*) the sequence will be created in the named schema (in this case, the current ident must be the creator of the specified schema).

The construct "CURRENT_VALUE OF *sequence-name*" will return the current value of the sequence (see Section 5.5.10).

The construct "NEXT_VALUE OF *sequence-name*" will return the next value for the sequence and this will be established as the current value of the sequence (see Section 5.5.19).

A sequence can be defined as unique or non-unique (the default is **non-unique**).

A sequence generates a series of values by starting at the initial value and proceeding in increment steps. When the addition of an increment would produce a value that is greater than the specified maximum, the sequence cycles back into the beginning of the value series by the appropriate number of steps to generate the next value.

A sequence can make more than one pass between the initial value and the maximum value to generate its series of values, depending on the initial value, increment step and the maximum value.

A non-unique sequence will continue to cycle indefinitely and thus may repeatedly generate its series of values.

A unique sequence will only go through its value-generating cycle once and is guaranteed never to return the same value twice.

Examples:
A **non-unique** sequence with initial value 1, increment 3 and maximum 10 will generate the following (repeating) series of values: 1, 4, 7, 10, 3, 6, 9, 2, 5, 8,   1, 4, 7, 10, 3, 6, 9, 2, 5, 8,   1, 4, 7, 10, 3, 6… .

A **unique** sequence with initial value 1, increment 3 and maximum 10 will generate the following series of values: 1, 4, 7, 10, 3, 6, 9, 2, 5, 8.

Note: It is possible that not every value in the series of values defined by the sequence will be generated. If a database server crash etc. occurs during the life of a sequence is possible that some of the values in the series might be skipped.

Restrictions

Two sequences with the same name cannot belong to the same schema.

An ident must have USAGE privilege on the sequence in order to use it.

Notes

If an initial value is not specified, a value of 1 is assumed.

If an increment is not specified, a value of 1 is assumed.

If a maximum value is not specified, a value equal to the largest INTEGER (with no precision specified) value possible in Mimer SQL is assumed (i.e. 2,147,483,647).

The sequence is created with an **undefined** current value initially. When “NEXT_VALUE OF *sequence-name*” is used for the first time after the sequence is created, the initial value for the sequence is returned and established as the current value of the sequence.

If “CURRENT_VALUE OF *sequence-name*” is used when the current value of the sequence is undefined, an error will be raised.

The “CURRENT_VALUE OF *sequence-name*” and “NEXT_VALUE OF *sequence-name*” constructs may be used in contexts where a value-expression would normally be used. They may also be used in the DEFAULT clause of the ALTER TABLE, CREATE DOMAIN and CREATE TABLE statements.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	MIMER EXTENSION	The CREATE SEQUENCE statement is a Mimer SQL extension.

CREATE SHADOW

Creates a new shadow for a databank.

```
►─── CREATE SHADOW shadow-name ─────────────────────────────────·····

·····─── FOR databank-name IN filename-string ──────────────────────►
```

Usage

Embedded/Interactive/ODBC.

Description

A new shadow for a databank is created, i.e. a new physical file is created in the host file system and the contents of an existing Mimer SQL databank is copied to the file (see the Mimer SQL System Management Handbook for more details). The *filename-string* specifies the name of the new file in the host system and is stored in the data dictionary as the physical location of the shadow.

Restrictions

CREATE SHADOW is only for use with the optional Mimer SQL Shadowing module.

The ident executing the CREATE SHADOW statement must hold SHADOW privilege.

The CREATE SHADOW statement cannot be used if the databank to be shadowed is OFFLINE.

The databank to be shadowed (specified by *databank-name*) cannot be used by any other user while the shadow is being created.

A databank must have either the TRANS or LOG option if it is to be shadowed, since the shadowing facility requires transaction handling.

Notes

The *shadow-name* may not be the same as that of any existing databank or shadow.

The value of *filename-string* must always be enclosed in string delimiters. The maximum length of the filename string is 256 characters.

Refer to Section 3.1.2.1 for details concerning the specification of pathname components in *filename-string*.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 **SQL92**	MIMER EXTENSION	The CREATE SHADOW statement is a Mimer SQL extension. Note: Databank shadowing does **not** affect application portability in any way because its runtime operation is completely transparent.

CREATE SYNONYM

Creates an alternative name for a table, view or another synonym.

```
►── CREATE SYNONYM synonym-name FOR object-name ──────────────►
```

Usage

Embedded/Interactive/ODBC.

Description

A synonym is created for the table, view or synonym specified in *object-name*.

If *synonym-name* is specified in its unqualified form, the synonym will be created in the schema which has the same name as the current ident.

If *synonym-name* is specified in its fully qualified form (i.e. *schema-name.synonym-name*) the synonym will be created in the named schema (in this case, the current ident must be the creator of the specified schema).

Restrictions

A synonym may only be created if the creator has some access privilege on the object specified in *object-name*.

A synonym can only be created for an existing table, view or synonym.

The synonym name may not be the same as the name of any other table, view or synonym already belonging to the schema in which the synonym is created.

Notes

The synonym is stored in the data dictionary and it may be used to refer to the associated table or view wherever *object-name* would normally be used in the syntax specifications for SQL.

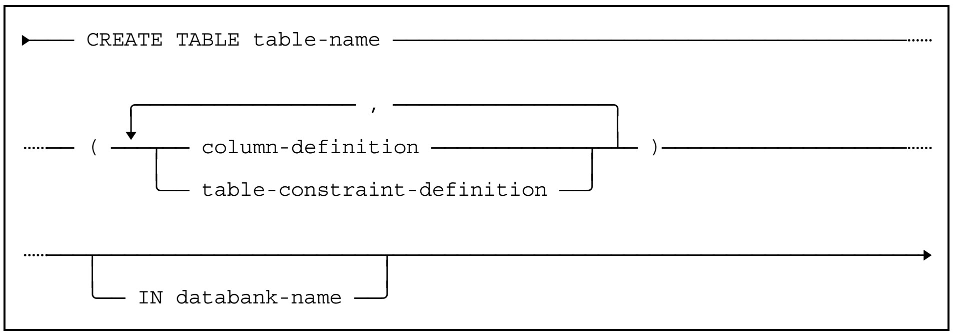
Note: Synonyms are **not** the same as correlation names. The latter are defined in the FROM clause of select-specifications (see Section 5.11) and apply only within the context of the statement where they are defined.

Standard compliance

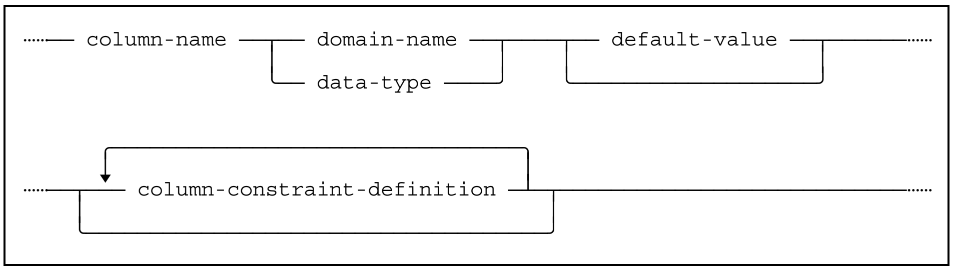
Standard	**Compliance**	**Comments**
X/Open-95 **SQL92**	MIMER EXTENSION	The CREATE SYNONYM statement is a Mimer SQL extension. Note**:** Equivalent standard-compliant functionality can be achieved by creating a VIEW statement with no WHERE clause and granting the same access rights on the view as apply on the table.

CREATE TABLE

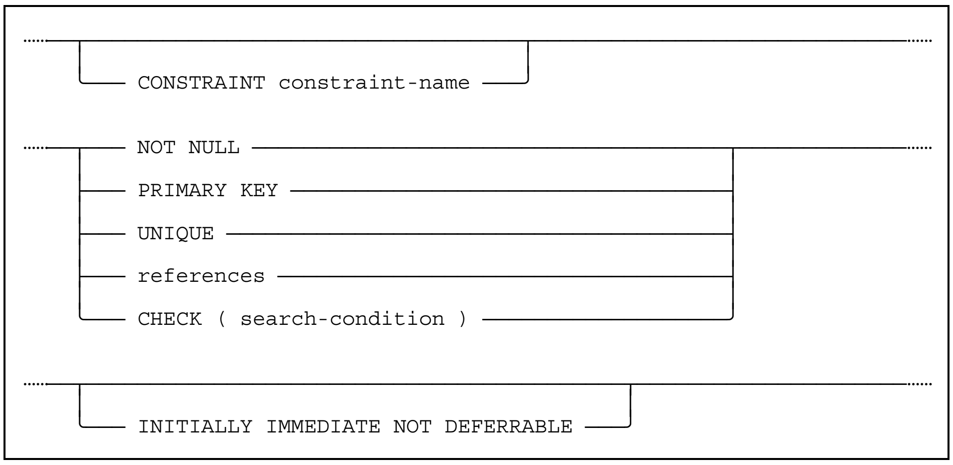
Creates a new table.



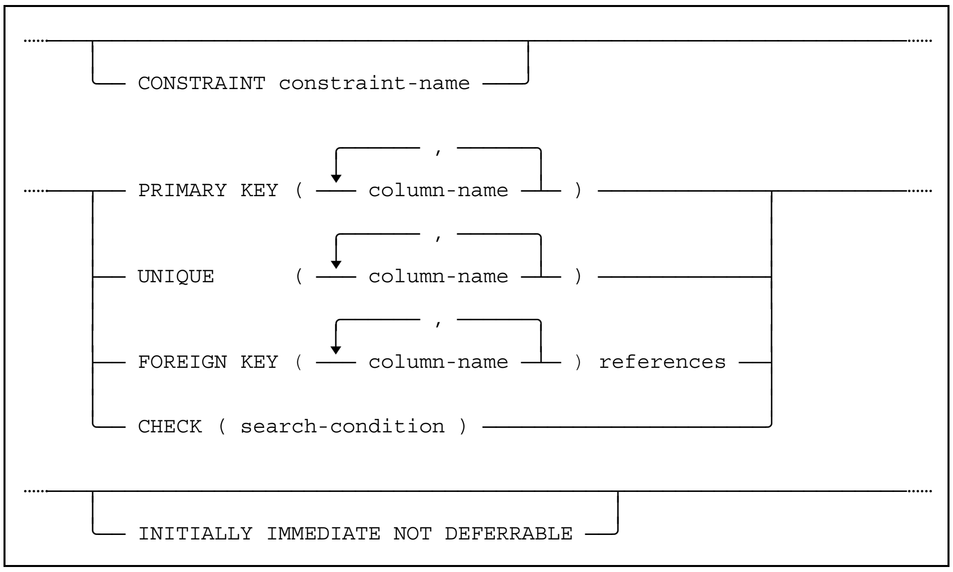
where *column-definition* is



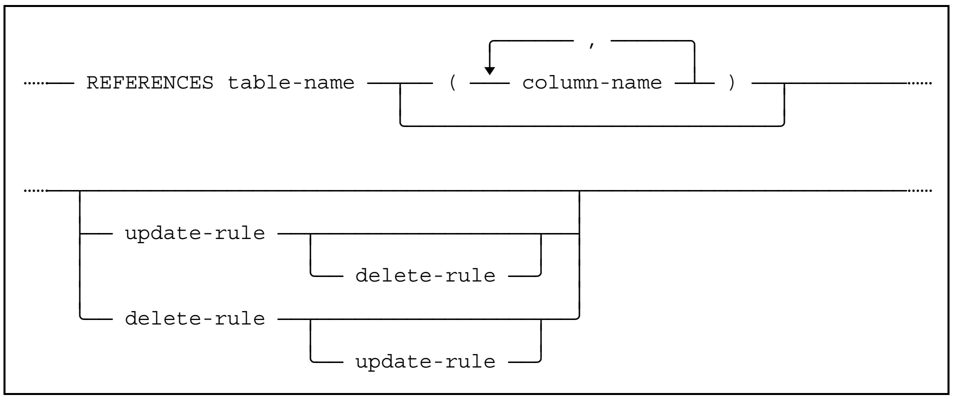
and *column-constraint-definition* is


CONSTRAINT constraint-name
NOT NULL
PRIMARY KEY
UNIQUE
references
CHECK (search-condition)
INITIALLY IMMEDIATE NOT DEFERRABLE

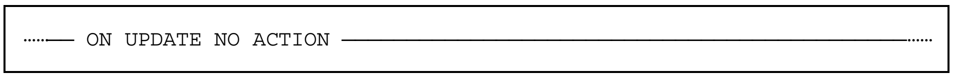
and *table-constraint-definition* is


CONSTRAINT constraint-name
PRIMARY KEY (column-name)
,
UNIQUE (column-name)
,
FOREIGN KEY (column-name) references
,
CHECK (search-condition)
INITIALLY IMMEDIATE NOT DEFERRABLE

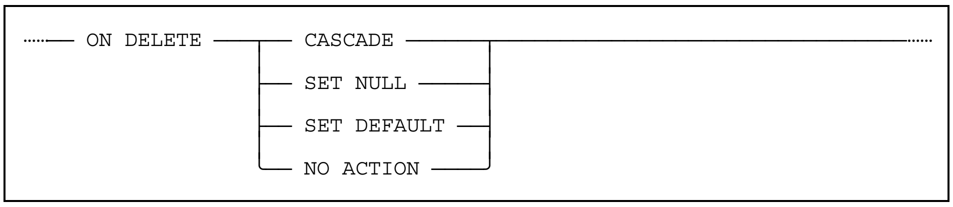
and *references* is



and *update-rule* is



and *delete-rule* is



Usage

Embedded/Interactive/ODBC.

Description

A new table is created as specified.

If *table-name* is specified in its unqualified form, the table will be created in the schema which has the same name as the current ident.

If *table-name* is specified in its fully qualified form (i.e. *schema-name.table-name*) the table will be created in the named schema (in this case, the current ident must be the creator of the specified schema).

The table definition includes a list of *column-definition*'s and *table-constraint-definition*'s.

The table must be created in a databank on which the current ident has TABLE privilege. If "IN *databank-name*" is not specified, the system will choose a databank on which the user has TABLE privilege. If more than one such databank exists, databanks created by the current ident are chosen in preference to others and the databank with the most secure transaction option is chosen (i.e. a databank with LOG option would be chosen in preference to one with TRANS option and one with TRANS option in preference to one with NULL option).

The new table is empty until data is inserted.

Column definitions

The columns will appear in the table in the order specified. Each column name must be unique within the table. Column formats may be specified either by explicit data type (see Section 4.3) or by specifying the name of a domain to which the column will belong. In the latter case, all the properties of the domain apply to the column.

A default value can be defined for the column by specifying *default-value* in *column-definition* or by having the column belong to a domain for which a default value is defined. A default value specified in *default-value* will take precedence over a domain default value and the data type of the value specified in *default-value* must conform to the data type of the column.

The default value will be assigned to a column whenever an INSERT is performed with no explicit value supplied. If the defined default value does not conform to other constraints, e.g. a CHECK constraint, then an INSERT **must** supply a value.

Table constraints

One or more constraints may be defined on the table, either by specifying a *column-constraint-definition* in a *column-definition* or by the specifying a *table-constraint-definition* in the table element list.

All table constraints may be named by specifying a *constraint-name* in the *column-constraint-definition* or *table-constraint-definition*. If a constraint is defined without specifying an explicit name, an automatically generated name will be assigned to it.

Note: Automatically generated constraint names start with "SQL_", so it is recommended that this initial character sequence be avoided when explicitly specifying a constraint name.

Constraint names are shown in the appropriate INFORMATION_SCHEMA views (see Chapter 7).

The constraint name is used to identify a constraint when it is dropped using the ALTER TABLE statement.

NOT NULL constraints
If this constraint is specified in a *column-constraint-definition* in the *column-definition* for a column, the column will not accept an attempt to insert the NULL value. (This constraint can also be effectively defined by specifying by a CHECK constraint for the table, see below, however this is not recommended because the column would not then be flagged as not accepting NULL values in any DESCRIBE functionality.)

PRIMARY KEY constraint
One PRIMARY KEY can be defined for the table, composed of one or more of the table columns. The same column must not occur more than once in the primary key. A column that is a part of the primary key will implicitly be constrained as NOT NULL, regardless of any NOT NULL constraints explicitly defined on the table. The NULL value cannot, therefore, occur in a primary key column.

The purpose of a primary key is to define a key value that uniquely identifies each table row, therefore the primary key value for each row in the table **must** be unique.

The primary key value for a table row is the combined value of the column(s) making up the primary key. The column(s) of the primary key (and their order in the key) can be defined using the PRIMARY KEY clause in a *table-constraint-definition*.

If the primary key for the table is to be composed of only a **single column**, then it can be defined by specifying PRIMARY KEY in a *column-constraint-definition* in the *column-definition* for that column.

UNIQUE constraints
One or more UNIQUE constraints can be defined on the table. A UNIQUE constraint defines a **unique key** for the table. A unique key is composed of one or more table columns, just like the primary key. A column must not occur more than once in the same unique key.

A unique key defines a key value that uniquely identifies each row in the table, therefore a table **cannot** contain two rows which have the same value for a unique key unless all columns are null.

A unique key must not be composed of the same set of column(s) (occurring in the same order) as either the primary key or an existing unique key defined for the table.

A unique key value for a table row is the combined value of the column(s) making up the unique key. The column(s) of the unique key (and their order in the key) can be defined using the UNIQUE clause in a *table-constraint-definition*.

If a unique key is to be composed of only a single column, then it can be defined by specifying UNIQUE in a *column-constraint-definition* in the *column-definition* for that column.

Note*:* Multiple occurrences of the NULL-value do **not** violate a UNIQUE constraint.

REFERENTIAL constraints

A referential constraint defines a **foreign key** relationship between the table being created (the “referencing table”) and another table in the database (the “referenced table”).

A foreign key relationship exists between a key (the “foreign key”) in the referencing table and the **primary key** or one of the **unique keys** of the referenced table.

The foreign key in the referencing table is defined by using the FOREIGN KEY clause in *table-constraint-definition* and is composed of one or more columns of the referencing table. The same referencing table column **cannot** occur more than once in the foreign key.

The corresponding key in the referenced table is specified by using the REFERENCES clause in *references*. If a list of column names is not specified after the name of the referenced table, then the **primary key** of the referenced table is assumed.

More than one foreign key can be defined for a table and the same table column can occur in more than one of the foreign keys.

The name of the referenced table must be specified in its fully qualified form if the name of the schema to which it belongs is not the same as the current ident.

The *i*-th column in the referencing table foreign key corresponds to the *i*-th column in the specified key of the referenced table and both keys must be composed of the same number of columns.

The data type and data length of each column in the referencing table foreign key must be **identical** to the data type and data length of the corresponding column in the specified key of the referenced table.

The effect of a referential constraint is to constrain table data in a way that only allows a row in the referencing table which has a foreign key value that matches the specified key value of a row in the referenced table.

One or more of the columns in a foreign key may permit the NULL value (this will be the case if there is no NOT NULL constraint or equivalent CHECK constraint in effect for the column).

A referencing table row which has a foreign key value with the NULL value in at least one of the columns will **always** fulfil the referential constraint and therefore be acceptable as a row in the referencing table.

If **all** of the columns in a foreign key are constrained not to accept the NULL value, then the only rows that will be accepted in the referencing table are those with a foreign key value that already exists in the corresponding key of the referenced table.

Rules can be defined in *references* that specify an action to be performed on the affected row(s) of the referencing table when a DELETE or UPDATE operation in the referenced table causes a referential constraint to be violated (because rows would consequently exist in the referencing table those foreign key value did not match the corresponding key value of a row in the referenced table).

It is possible to explicitly specify ON UPDATE NO ACTION in *update-rule* (this will also be assumed by default if no *update-rule* is specified). This is to allow for future extensions to the Mimer SQL syntax.

The following actions can be specified in *delete-rule*:

CASCADE	this will cause the affected rows of the referencing table to be deleted.
SET NULL	the relevant foreign key columns of the affected rows in the referencing table will be set to the NULL value.
SET DEFAULT	the relevant foreign key columns of the affected rows in the referencing table will be set to the value that would be applied to that column if an INSERT operation is performed without specifying an explicit value (i.e. column default, domain default or the NULL value).
NO ACTION	an error will be raised because the referential constraint has been violated.

If a *delete-rule* is not specified, then the action NO ACTION is assumed.

A referential constraint can be defined by specifying a FOREIGN KEY clause in *table-constraint-definition*.

If a referencing table foreign key is to be composed of only a single column, then the referential constraint can be defined by specifying *references* in a *column-constraint-definition* in the *column-definition* for that column.

CHECK constraints

One or more check constraints can be defined on the table, which will determine whether the changes resulting from an INSERT or UPDATE operation will be accepted or rejected.

The *search-condition* which defines the check constraint must **not** contain a select-specification, an invocation of a set function, a reference to a host variable or a non-deterministic expression.

If the *search-condition* of a check constraint specified in a *table-constraint-definition* contains column references, they **must** be columns in the table being created.

If the *search-condition* of a check constraint specified in a *column-constraint-definition* in a *column-definition* contains a column reference, it **must** be the column identified by *column-name* of the *column-definition*.

The *search-condition* of a check constraint defined on the table will be evaluated whenever a new row is inserted into the table and whenever an existing row is updated.

The values for any column reference(s) contained in the *search-condition* will be taken from the row being inserted or updated.

The data change operation will only be accepted if the search condition does **not** evaluate to false.

Language elements

default-value	see Section 5.3.
search-condition	see Section 5.10.

Restrictions

CREATE TABLE requires TABLE privilege on the databank in which the table is to be created.

The table name must not be the same as the name of any other table, view or synonym belonging to the same schema.

If a domain name is specified for *column-definition*, USAGE privilege must be held on the domain.

Each *table-constraint-definition* can only be specified once in the CREATE TABLE statement.

If a UNIQUE constraint is defined on the table, it must be stored in a databank with the TRANS or LOG option.

If a REFERENTIAL constraint is defined, both the referencing table and the referenced table must be stored in a databank with the TRANS or LOG option.

A schema cannot contain two constraints with the same name.

The creator of the table must hold REFERENCES privilege on all the columns specified in *references*.

The name of a **view** cannot be specified for *table-name* in *references*.

Notes

The creator of the table is granted all access privileges to the table WITH GRANT OPTION.

In a REFERENTIAL constraint, the referenced table can be the same as referencing table. In this situation, the table data is constrained in a way that only allows the foreign key columns to contain key values that are already present in the referenced (primary or unique) key.

The action specified in the *delete-rule* of a REFERENTIAL constraint does not apply to columns in the referencing table that are already marked for deletion.

If a name is not specified for a table or column constraint, a system generated name is applied to it. System generated names will begin with “SQL_” so it is recommended that this starting character sequence be avoided for explicitly specified constraint names.

The constraint characteristics INITIALLY IMMEDIATE NOT DEFERRABLE can be explicitly specified to ensure that constraints will be, by default, verified at the time the relevant data manipulation operation is performed rather than when the transaction is committed and that the verification may never be explicitly deferred until the time the transaction is committed. This is also the default behavior. (This is to allow for future extensions to the Mimer SQL syntax.)

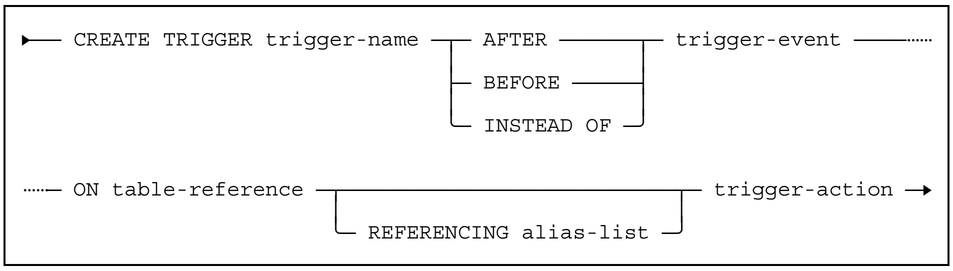
The primary key and the unique keys for a table are not dissimilar in their function and they constrain data in the same way apart from the fact that primary key columns are always defined as not null, however a unique key should **not** be used **instead of** a primary key. One reason for this is that the primary key is handled more efficiently than the unique keys, so there is a performance advantage. See “Relational Databases - Selected Writings” by C. J. Date for a discussion of primary and unique keys.

Standard compliance

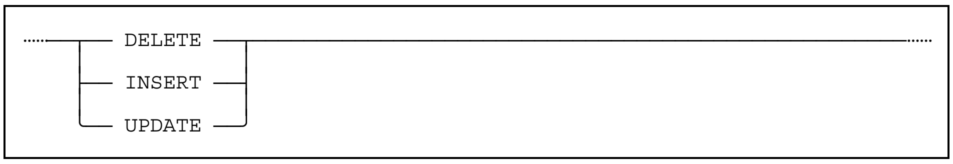
Standard	**Compliance**	**Comments**
X/Open-95 **SQL92**	EXTENDED	Support for the IN *databank-name* clause is a Mimer SQL extension. (Note: Support for domains is included in SQL92 Intermediate level).

CREATE TRIGGER

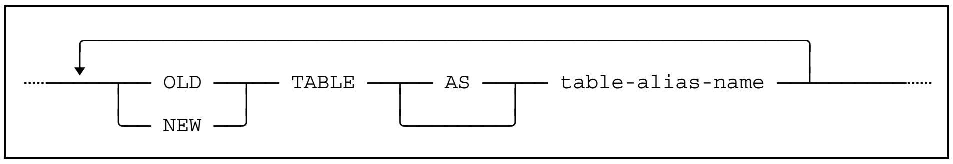
Creates a trigger which is invoked by data changes in a named table or view.



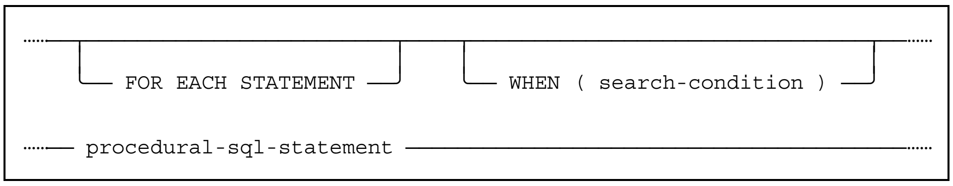
where *trigger-event* is



and *alias-list* is



and *trigger-action* is



Usage

Embedded/Interactive/ODBC.

Description

A trigger is created on a table or view (table reference).

For a complete description of triggers see Chapter 9 of the Mimer SQL Programmer's Manual.

The *trigger-name* should follow the normal rules for naming database objects (see Section 4.2.2).

If *trigger-name* is specified in its unqualified form, the trigger will be created in the schema which has the same name as the current ident.

If *trigger-name* is specified in its fully qualified form (i.e. *schema-name.trigger-name*) the trigger will be created in the named schema (in this case, the current ident must be the creator of the specified schema).

The *trigger-action* will be executed when the data manipulation operation specified by *trigger-event* occurs on *table-reference* and any *search-condition* specified in the WHEN clause of the *trigger-action* evaluates to true.

If the trigger time is AFTER, the trigger will be invoked after the data manipulation operation specified by *trigger-event* has been performed on *table-reference*.

If the trigger time is BEFORE, the trigger will be invoked before the data manipulation operation specified by *trigger-event* has been performed on *table-reference*.

If the trigger time is INSTEAD OF, the trigger will be executed in place of the data manipulation operation specified by *trigger-event*, i.e. the data manipulation statement invoking the trigger will **not** actually change any data in *table-reference*. In this case the only data changes possible are those performed by the *trigger-action*.

In the *alias-list*, OLD TABLE and NEW TABLE are used to create two special temporary tables showing those rows of *table-reference* **selected by** the data manipulation statement invoking the trigger.

The temporary table created by using OLD TABLE shows the affected rows as they were before the data manipulation statement was executed.

The temporary table created by using NEW TABLE shows the affected rows in a state consistent with the data manipulation statement **having been executed**. In the case of an AFTER trigger, the same rows can be found in *table-reference* but for an INSTEAD OF trigger, the matching rows of *table-reference* will actually appear as they do in the OLD TABLE alias because the data manipulation statement is not actually executed.

Restrictions

The trigger and *table-reference* **must** belong to the same schema.

Two triggers with the same name cannot belong to the same schema.

If the trigger time is AFTER or BEFORE, then *table-reference* must be the name of a base table which is located in a databank with the TRANS or LOG option.

If the trigger time is INSTEAD OF, then *table-reference* must be the name of a view.

OLD TABLE and NEW TABLE may each be specified only once in the *alias-list* and the same *table-alias-name* must not appear twice in the same list.

OLD TABLE may not be specified if the *trigger-event* is INSERT.

NEW TABLE may not be specified if the *trigger-event* is DELETE.

If the trigger time is BEFORE, the REFERENCING keyword and *alias-list* must not be specified.

If the *procedural-sql-statement* of the *trigger-action* is a COMPOUND STATEMENT, it **must** be ATOMIC.

The creator of the trigger must hold the appropriate access rights, **with grant option**, for all operations performed in the trigger action.

The *trigger-action* must not contain a COMMIT or ROLLBACK statement.

If the trigger time is BEFORE, the following restrictions apply to the *trigger-action*:

- the *trigger-action* must not contain any SQL statement that performs an update (i.e. DELETE, INSERT and UPDATE statements are **not** permitted)
- a routine whose access clause is MODIFIES SQL DATA must **not** be invoked from within the *trigger-action*.

Notes

The *trigger-action* is always executed in the transaction started for the data manipulation operation which caused the trigger to be invoked. Thus, if the data manipulation operation is subject to a rollback, all operations performed in the *trigger-action* will also be undone and an unhandled error occurring in the *trigger-action* will be treated like an error in the triggering data manipulation statement.

During the execution of the *trigger-action*, the effect of changes made in the transaction are visible.

The scope of the *trigger-action* is the optional WHEN clause and the *procedural-sql-statement*.

The tables specified by using OLD TABLE and NEW TABLE in the *alias-list* are temporary and are local to scope of the *trigger-action*. It is not possible to perform any data change operations on either table and the data contained in each will not otherwise change during the time it exists.

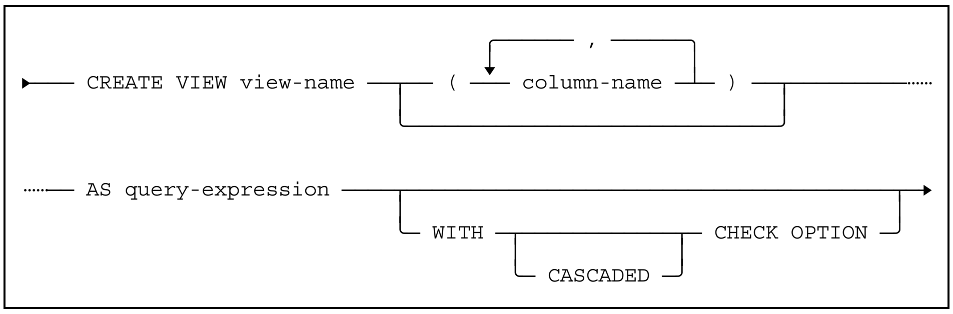
Data manipulation operations performed in the *trigger-action* may cause the trigger to be invoked recursively. Trigger execution in a recursive situation will proceed normally in every respect.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	MIMER_EXTENSION	Support for triggers is a Mimer SQL extension.
SQL99	EXTENDED	Support for INSTEAD OF triggers is a Mimer SQL extension.

CREATE VIEW

Creates a view on one or more tables or views.



Usage

Embedded/Interactive/ODBC.

Description

If *view-name* is specified in its unqualified form, the view will be created in the schema which has the same name as the current ident.

If *view-name* is specified in its fully qualified form (i.e. *schema-name.view-name*) the view will be created in the named schema (in this case, the current ident must be the creator of the specified schema).

A view is created in accordance with the specification in the *query-expression*.

If a list of column names is given in parentheses before the *query-expression*, the columns in the view are named in accordance with this list. There must be the same number of names in the column list as there are columns addressed by the *query-expression*. The names must be unique within the list.

If the column name list is omitted, the columns in the view will be given the same names as they have in the source table(s) or view(s) addressed in the select-specification. The column names in the source must all be unique in the view being created. If this is not the case, an explicit column name list must be given. An explicit column name list must also be given if columns in the view are defined as expressions.

Specification of WITH CHECK OPTION indicates that any data inserted into the view by INSERT or UPDATE statements will be checked for conformity with the definition of the view. Attempts to insert data which do not conform to the view definition will be rejected.

The optional keyword CASCADED can be explicitly specified in the WITH CHECK OPTION clause to ensure that any data inserted into a view which is **based on** this view will be also be checked for conformity with the definition of **this** view.

Thus, if an INSERT or UPDATE in a view based on this one results in an attempt to insert data into this view which does conform to the view definition, the data change operation will be rejected.

If CASCADED is not specified, it is assumed by default (use of the keyword CASCADED is now permitted to allow for future extensions to the Mimer SQL syntax).

Language elements

query-expression	see SELECT.

Restrictions

CREATE VIEW requires SELECT access to the tables or views from which the view is created.

Notes

The view name may not be the same as the name of any other table, view or synonym belonging to the same schema.

The creator of the view is always granted SELECT access to the view. If the view is updatable (see below), any access the creator may hold on the underlying table or view at the time the new view is created is also granted on the new view. Access to the view is granted WITH GRANT OPTION only if the corresponding access to all underlying table(s) or view(s) is held WITH GRANT OPTION.

SELECT and UPDATE statements can only be performed on data accessible from the view. Insertion of a new row assigns the default value or NULL value to columns in the base table excluded from the view, in accordance with the definition of the columns. Deletion of a row from a view removes the entire row from the underlying base table, including columns invisible from the view.

The select-specification defining the view may not contain references to host variables.

The WITH CHECK OPTION clause is illegal if the view is not updatable. A result set is only updatable if all of the following conditions are true:

- the keyword DISTINCT is not specified
- the select list consists only of column specifications, and no column specification appears more than once
- the FROM clause specifies exactly one table reference and that table reference refers either to a base table or an updatable view
- a GROUP BY clause is not included
- a HAVING clause is not included.

Note: A view will **always** be updatable if an INSTEAD OF trigger exists on the view, regardless of the conditions previously mentioned. If there is an instead of trigger any possible with check option for the view is ignored. If **all** the INSTEAD OF triggers on the view are dropped, the view will revert to not updatable if one or more of those conditions are not true.

If an updatable view is based on other views, insert and update operations are checked against all view definitions for which WITH CHECK OPTION is specified. Thus if view-2 is defined with check option on view-1, which in turn is defined with check option on a base table, no changes may be made in the base table through either view-1 or view-2 which violate the definition of view-1.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	YES	Fully compliant.

DEALLOCATE DESCRIPTOR

Deallocates an SQL descriptor area.

```
►── DEALLOCATE DESCRIPTOR descriptor-name ──────────────────────►
```

Usage

Embedded.

Description

This statement deallocates an SQL descriptor area that was previously allocated with the specified *descriptor-name*.

Restrictions

None.

Notes

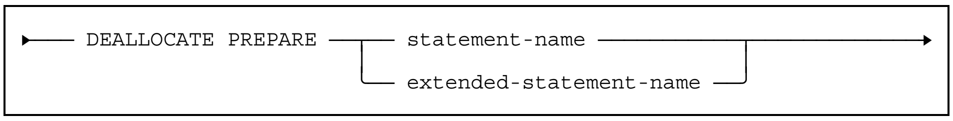
An SQL descriptor area with the specified name must have been previously allocated.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	YES	Fully compliant.

DEALLOCATE PREPARE

Deallocates a prepared SQL statement.



Usage

Embedded.

Description

The prepared statement associated with the statement name is destroyed. Any cursor allocated with an ALLOCATE CURSOR statement that is associated with the prepared statement is also destroyed.

See ALLOCATE CURSOR for a description of extended statements.

Restrictions

None.

Notes

The statement name must identify a statement prepared in the same compilation unit.

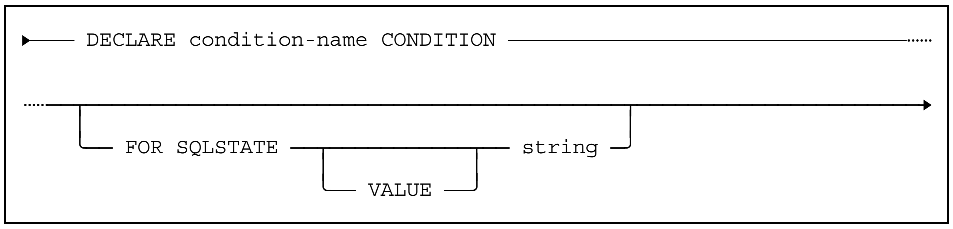
The statement must not identify an existing prepared statement that is associated with an open cursor.

Standard compliance

Standard	Compliance	Comments
X/Open-95 SQL92	YES	Fully compliant.

DECLARE CONDITION

Declares a condition name for an exception condition value.



Usage

Procedural.

Description

An exception condition may be specified by using an SQLSTATE value. It is also possible, within a compound SQL statement, to declare a condition name to represent an exception condition.

The condition name may be declared to represent an SQLSTATE value or it may exist on its own.

If an exception is raised (by executing SIGNAL or RESIGNAL) using a condition name that is not associated with an SQLSTATE value, the SQLSTATE value 45000 (“RETURNED SQLSTATE”) will be retrieved from the diagnostics area by the GET DIAGNOSTICS statement when the RETURNED_SQLSTATE option is used.

Condition names allow representations of exception conditions to be declared that are more meaningful and readable than an SQLSTATE value.

Restrictions

A condition name which represents an SQLSTATE value may only be declared to represent a **specific** SQLSTATE value, i.e. it is **not** possible to declare a condition name to represent an exception class group (for a description of exception class groups see DECLARE HANDLER).

The scope of a condition name covers all the *procedural-sql-statements* in the compound statement declaring it, including any other compound statements nested within it.

The general naming rules for a condition name are the same as those for other database objects.

If a condition name is declared to represent a particular SQLSTATE value, another condition name **cannot** exist for that same SQLSTATE value which has **exactly the same** scope.

A condition name **cannot** be declared for an SQLSTATE value with class “successful completion”, this covers all SQLSTATE values starting with “00”.

Notes

The SQLSTATE value string is five characters long and contains only alphanumeric characters.

In Mimer SQL any SQLSTATE value that falls outside the range of standard SQLSTATE values is treated as an implementation-defined value.

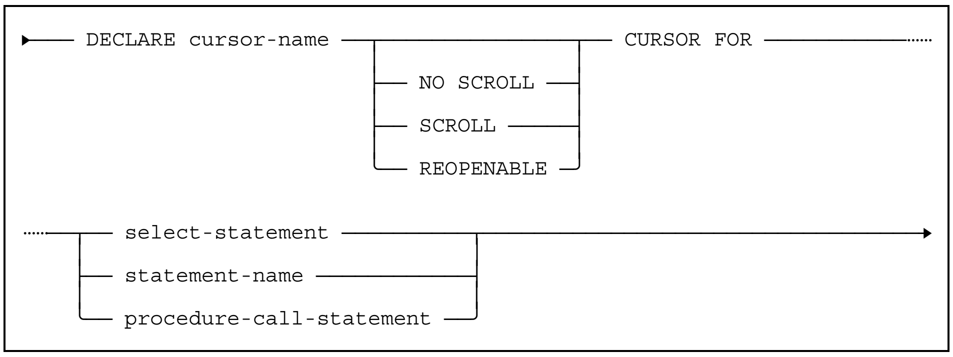
Standard SQLSTATE values begin with the characters “**A-I**”, “**S**”, “**0-4**” and “**7**”, while implementation-defined SQLSTATE values begin with the characters “**J-R**”, “**T-Z**”, “**5-6**” and “**8-9**”.

Standard compliance

Standard	**Compliance**	**Comments**
SQL/PSM	YES	Fully compliant.

DECLARE CURSOR

Declares a cursor definition.



Usage

Embedded/Procedural.

Description

A cursor is declared in accordance with the *select-statement* or the result set procedure call specified in *procedure-call-statement*.

The *select-statement* may be specified explicitly in ordinary embedded SQL applications or by the name of a prepared select-statement, identified by *statement-name*, in dynamic SQL statements (see Chapter 7 of the Mimer SQL Programmer's Manual).

The cursor is identified by *cursor-name*, and may be used in FETCH, DELETE CURRENT and UPDATE CURRENT statements. The cursor must be activated with an OPEN statement before it can be used.

A cursor declared as REOPENABLE may be opened several times in succession, and previous cursor states are saved on a stack (see OPEN). Saved cursor states are restored when the current state is closed (see CLOSE).

A cursor declared as SCROLL will be a scrollable cursor. For a scrollable cursor, records can be fetched using an orientation specification. See the description of FETCH later in this chapter for a description of how the orientation can be specified.

A cursor will be non-scrollable if NO SCROLL is explicitly specified. The cursor will be non-scrollable and not reopenable by default.

Language elements

select statement	see Section 5.13.
procedure-call-statement	see CALL.

Restrictions

If a *procedure-call-statement* is specified, it **must** specify a result set procedure.

The following restrictions apply to **Procedural** usage:

- The cursor cannot be declared as REOPENABLE
- The select-statement cannot be in the form of a prepared dynamic SQL statement, i.e. specifying s*tatement-name* is not allowed
- If the cursor declaration contains a select statement, the *access-clause* for the procedure must be READS SQL DATA or MODIFIES SQL DATA (- see CREATE PROCEDURE).

Notes

The DECLARE CURSOR statement is declarative, not executable. In an **Embedded** usage context, access rights for the current ident are checked when the cursor is opened, not when it is declared.

In a **Procedural** usage context, access rights for the current ident are checked when the cursor is declared, i.e. when the procedure containing the declaration is created.

The value of *cursor-name* may not be the same as the name of any other cursor declared within the same compound statement (**Procedural** usage) or in the same compilation unit (**Embedded** usage).

The *select-statement* is evaluated when the cursor is opened, not when it is declared. This applies both to *select-statement*'s identified by statement name, and to host variable references used anywhere in the select statement.

The execution of the result set procedure specified in a CALL-statement is controlled by the opening of the cursor and subsequent fetches - see Section 8.6 of the Mimer SQL Programmer's Manual.

REOPENABLE cannot be used if evaluation of *select-statement* uses a work table, or if the cursor declaration occurs within a procedure.

If the declared cursor is a dynamic cursor, the DECLARE statement must be placed before the PREPARE statement.

A reopenable cursor can be used to solve the “Parts explosion” problem. Refer to Section 5.3.1 of the Mimer SQL Programmer's Manual for a description of this.

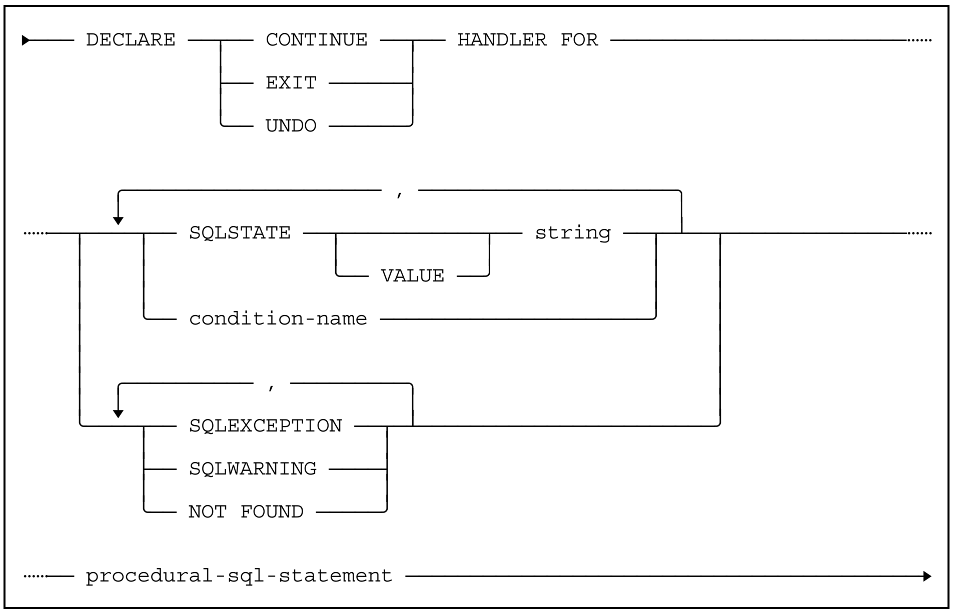
Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	EXTENDED	Support for the keyword REOPENABLE is a Mimer SQL extension.

Note: See also standard compliance for SELECT.

DECLARE HANDLER

Declares an exception handler.



Usage

Procedural.

Description

An exception handler may be declared to respond to a specific exception condition or one or more of the exception class groups SQLEXCEPTION, SQLWARNING or NOT FOUND. An exception handler that responds to one or more exception class groups is called a **general exception handler**.

A specific exception condition may be specified by using an SQLSTATE value or a condition name (see DECLARE CONDITION). An exception handler that responds to one or more specific exception conditions is called a **specific exception handler**.

The keywords CONTINUE, EXIT and UNDO affect the flow of control behavior subsequent to the execution of the exception handler.

If CONTINUE is specified, the flow of control continues by executing the SQL statement immediately following the statement that raised the error, after the handler has executed.

If EXIT is specified, the flow of control exits the compound statement within which the exception handler is declared after the handler has executed.

If UNDO is specified, all the changes made by the SQL statements in the ATOMIC compound statement, within which the handler is declared, (or by any SQL statements triggered by those changes) are cancelled. Then the handler is executed and the flow of control exits the compound statement.

Restrictions

An UNDO exception handler can only be declared within an ATOMIC compound statement.

An exception handler must be **either** a general or a specific exception handler, it cannot respond to **both** an exception class group and a specific exception condition.

The same exception condition must **not** be specified more than once, whether by SQLSTATE value or by condition name, in an exception handler declaration.

Within a given scope, only **one** specific exception handler may be declared for a particular exception condition.

If *string* is specified it must have length five and contain only alphanumeric characters.

Notes

Within a given compound statement, if both a general and a specific exception handler have been declared to respond to a given exception condition, the **specific** exception handler will handle the exception.

If no exception handlers have been declared to respond to an exception condition in the compound statement within which the error was raised, the exception condition is propagated out to the enclosing compound statement or to the calling environment.

The above does not apply to exception conditions with the **NOT FOUND** and **SQLWARNING** class, these are not propagated by the exception handling mechanism in absence of an exception handler defined to handle them.

SQLWARNING covers SQLSTATE values beginning with “01”.

NOT FOUND covers SQLSTATE values beginning with “02”.

SQLEXCEPTION covers all other SQLSTATE values (including implementation-defined values), except those beginning with “00”.

Standard compliance

Standard	**Compliance**	**Comments**
SQL/PSM	YES	Fully compliant.

DECLARE SECTION

Identifies declaration of host variables used in SQL statements.

```
►─── BEGIN DECLARE SECTION ──────────────────────────────────────────►

     host-variable-declarations

►─── END   DECLARE SECTION ──────────────────────────────────────────►
```

Usage

Embedded.

Description

Host variables declared between the BEGIN DECLARE SECTION and END DECLARE SECTION statements are declared for both the host language compiler and the SQL compiler. Host variables used in SQL statements must be declared in an SQL DECLARE SECTION.

Restrictions

None.

Notes

Host variables declared in the SQL DECLARE SECTION must be elementary data items or pointers to charater strings. They may not be declared as part of a record or structure.

The rules for placing the DECLARE SECTION in the host program code and for declaring variables within the SECTION follow the host language syntax. Language-specific issues are considered in Appendix A of the Mimer SQL Programmer's Manual.

SQL statements may not be placed within the SQL DECLARE SECTION.

Note: BEGIN DECLARE SECTION and END DECLARE SECTION are two separate SQL statements, **each** of which must be preceded by EXEC SQL and followed by the (language-specific) statement terminator.

The following example in C shows the correct usage of both statements:

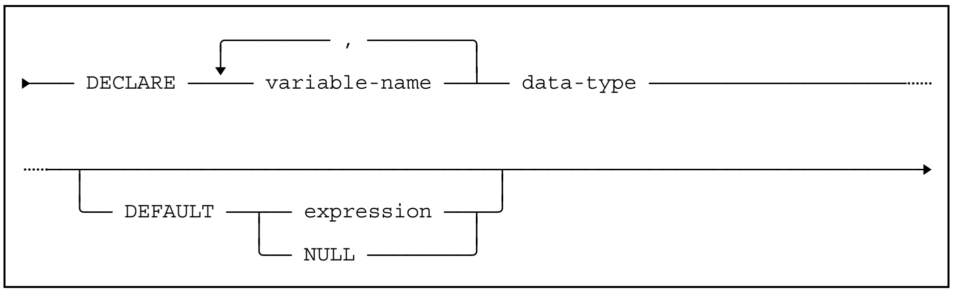
```
exec sql BEGIN DECLARE SECTION;
char usernm[129];
char passwd[19];
exec sql END DECLARE SECTION;
```

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	YES	Fully compliant.

DECLARE VARIABLE

Declares a variable.



Usage

Procedural.

Description

The value for *data-type* can be any data type supported by Mimer SQL (see Section 4.3).

The value for *data-type* may also be a ROW type definition. See Section 8.2.7 of the Mimer SQL Programmer's Manual for details of the ROW data type.

More than one variable of the same type can be declared in a single declaration.

The optional DEFAULT clause may be used to specify an initial value for the variable(s). A value of NULL is permitted as the value for the DEFAULT clause.

If a ROW data type definition has been specified for *data-type*, a row value expression can be specified for *expression* in the DEFAULT clause. See Section 8.2.8 of the Mimer SQL Programmer's Manual for a description of a row value expression.

If the DEFAULT clause is not specified, the variable(s) will be set to NULL initially. In the case of a variable declared with the ROW data type, **each field** in the variable is set to NULL initially if a DEFAULT clause is not specified.

Restrictions

A domain **cannot** be used to specify the data type in a variable declaration.

The name of a variable cannot be the same as any of the routine parameter names.

A function with MODIFIES SQL DATA specified for its access clause cannot be used as *expression* in the DEFAULT clause.

Notes

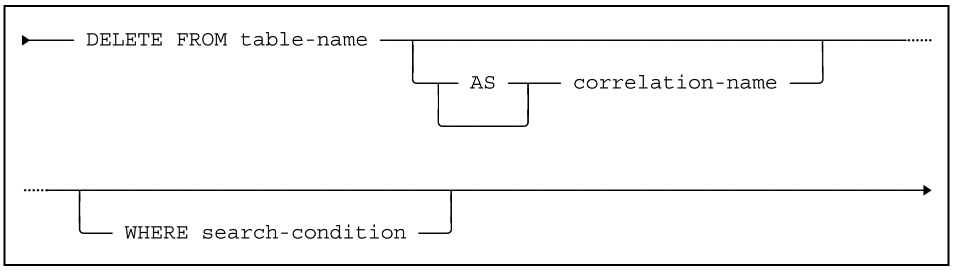
It is possible to declare a variable with the same name as that of a column in a table. In such a situation, an unqualified name will always resolve to the table column name and not the variable, so it is recommended that a suitable naming convention be adhered to that distinguishes between the two.

Standard compliance

Standard	**Compliance**	**Comments**
SQL/PSM	EXTENDED	Support for the specification of an expression rather than a literal value in the DEFAULT clause is a Mimer SQL extension.

DELETE

Deletes a set of rows from a table or view.



Usage

Embedded/Interactive/ODBC/Procedural.

Description

All rows in the set defined by the WHERE clause are deleted from the base table or view identified by *table-name*. If no WHERE clause is specified, all rows are deleted.

If *table-name* identifies a view rather than a base table, entire rows, including columns invisible from the view, are deleted from the base table on which the view is defined. For a delete to be performed on a view, the view must be updatable (see CREATE VIEW).

A NOT FOUND condition code is returned if no row is deleted (see Appendix E).

Language elements

search-condition	see Section 5.10.

Restrictions

DELETE privilege must be held on the table or view identified by *table-name*.

In a **procedural** usage context, the DELETE statement is only permitted if the procedure *access-clause* is MODIFIES SQL DATA (see CREATE PROCEDURE).

Notes

If a *correlation-name* is specified after the table name in the DELETE FROM clause, the correlation name **must** be used to refer to the table in the WHERE clause of the DELETE statement.

If the table name addressed by the DELETE statement is subject to any referential constraints, the delete operation must not create a situation where these constraints are violated. The effect of the delete operation on any referential constraints depends on the *delete-rule* in effect for each constraint (see CREATE TABLE for a description of the *delete-rule* options).

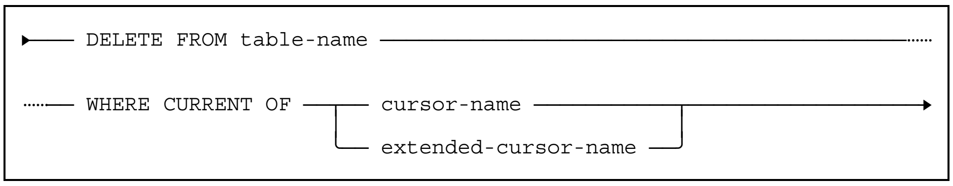
A DELETE statement is executed as a single statement. If an error occurs at any point during the execution, none of the rows in the set defined by the WHERE clause will be deleted (however, if the table is stored in a databank with the NULL option it is possible that some rows will be deleted).

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 **SQL92**	EXTENDED	Support for the AS *correlation-name* clause is a Mimer SQL extension.

DELETE CURRENT

Deletes the current row indicated by a cursor.



Usage

Embedded/ODBC/Procedural.

Description

The row currently indicated by the cursor is deleted from the base table implied by the cursor definition.

The row indicated by the cursor is that retrieved by the most recent FETCH statement (see FETCH).

After the deletion, the cursor is no longer positioned on a row in the set defined by the cursor declaration. A FETCH statement is required to position the cursor on the next row in the set before another DELETE CURRENT or an UPDATE CURRENT statement can be executed.

See ALLOCATE CURSOR for a description of extended cursors.

If an *extended-cursor-name* is specified in a DELETE CURRENT statement, it specifies the name of a host variable whose value is the name of the cursor.

Restrictions

DELETE access to the table or view identified by *table-name* is required when the cursor used for the DELETE CURRENT statement is opened. If DELETE access is not held, the cursor may be opened but DELETE CURRENT statements will fail.

The DELETE CURRENT statement may not be used for read-only cursors.

A cursor cannot be identified by specifying *extended-cursor-name* in a **procedural** usage context.

In a **procedural** usage context, the DELETE CURRENT statement is only permitted if the procedure *access-clause* is MODIFIES SQL DATA (see CREATE PROCEDURE).

Notes

The table name given in the DELETE CURRENT statement must be the same as that specified in the FROM clause of the cursor declaration. If a synonym is used in one of the statements, the same synonym must also be used in the other.

If the table name addressed by the DELETE CURRENT statement is subject to any referential constraints, the delete operation must not create a situation where these constraints are violated. The effect of the delete operation on any referential constraints depends on the *delete-rule* in effect for each constraint (see CREATE TABLE for a description of the *delete-rule* options).

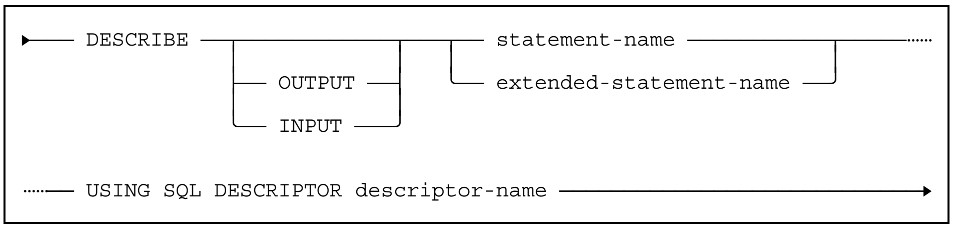
If an error occurs during execution of a DELETE CURRENT statement, the row is not deleted.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	YES	Fully compliant.

DESCRIBE

Describes the SELECT list or dynamic input variable specifications in a prepared statement into a descriptor area.



Usage

Embedded.

Description

Information concerning the prepared statement identified by *statement-name* is placed in the named SQL descriptor area.

The two forms “DESCRIBE statement...” and “DESCRIBE OUTPUT statement...” are equivalent, and describe the items in the SELECT list of the statement into the SQL descriptor area.

The form “DESCRIBE INPUT statement...” describes the input variables of the statement into the SQL descriptor area.

If descriptions of both the SELECT list and the input variables are required for a prepared statement, the statement must be described separately with each form of DESCRIBE.

The *descriptor-name* is identified by a host variable or a literal.

See ALLOCATE DESCRIPTOR for a description of the SQL descriptor area.

See ALLOCATE CURSOR for a description of extended statements.

For a description of how to use the SQL descriptor area in dynamic application programs, see Chapter 7 of the Mimer SQL Programmer’s Manual.

Restrictions

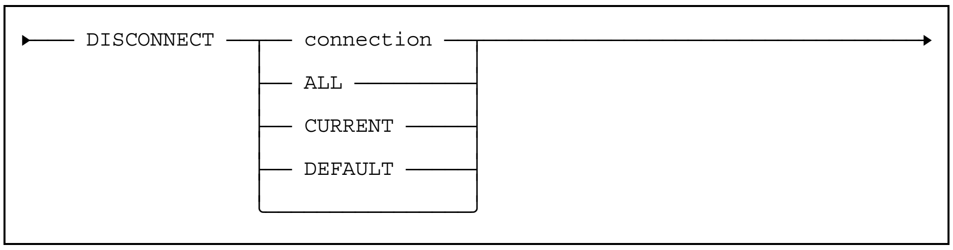
The DESCRIBE statement is only applicable to dynamically prepared SQL statements.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	YES	Fully compliant.

DISCONNECT

Disconnects a user from the specified connection.



Usage

Embedded/Interactive.

Description

The specified connection is disconnected. Any current transaction is rolled back, cursors are closed, and compiled statements are destroyed. No further access to the database using that connection is possible.

If the specified connection is not the current connection the application still has access to the current connection. Otherwise, to continue with another connection the application must either perform a SET CONNECTION or a CONNECT statement.

The *connection* is **not** case-sensitive and may be specified as a literal value or by using a host variable.

If no disconnect option is specified, CURRENT is assumed by default.

Restrictions

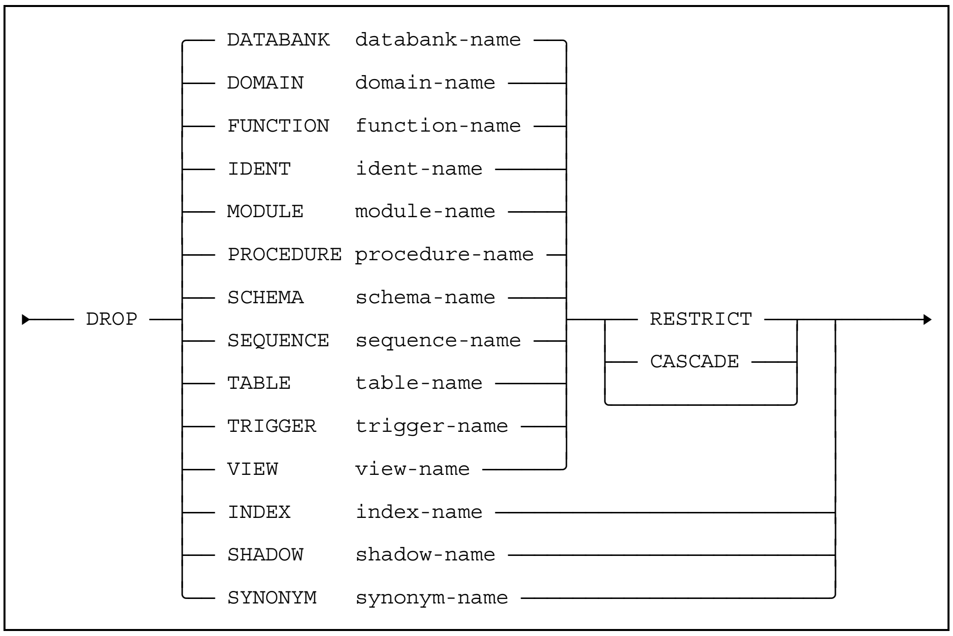
None.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 **SQL92**	YES	Specifying one of: *connection-name*, ALL, CURRENT or DEFAULT after the keyword DISCONNECT is optional in Mimer SQL, not mandatory. Using the DISCONNECT statement on a connection that has an active transaction is not forbidden in Mimer SQL (the transaction is rolled back).

DROP

Drops an object from the database.



Usage

Embedded/Interactive/ODBC.

Description

The named object is deleted from the database. The object name is free to be reused for other objects.

The CASCADE and RESTRICT keyword specifies the action to be taken if other objects exist that are dependent on the object being dropped. If CASCADE is specified, such objects will be dropped as well. If RESTRICT is specified, an error is returned if other objects are affected, and no objects are dropped.

If neither RESTRICT nor CASCADE is specified, then RESTRICT is implicit.

Restrictions

A private database object can only be dropped by the creator of the schema to which it belongs, unless it is implicitly dropped because of cascade effects when another object is dropped, see Notes below.

A system database object can only be dropped by its creator, unless it is implicitly dropped because of cascade effects when another object is dropped, see Notes below.

You must have exclusive use of a table to drop the table or an index on the table, and of a databank to drop the databank.

DROP SHADOW is only for use with the optional Mimer SQL Shadowing module and requires SHADOW privilege.

The databank for which the shadow exists cannot be used by any other user while the shadow is being dropped.

Notes

Databank
When a databank is dropped, all tables in the databank are deleted.All shadows defined on the databank are also dropped. An attempt is made to delete the physical file in which the databank is stored. If the file deletion is unsuccessful for any reason (e.g. the disk is not mounted), the databank is dropped from the database but the file remains.

If the databank is OFFLINE, no attempt is made to delete the physical databank file or any shadow file(s).

Domain
When a domain is dropped, existing columns defined using the domain retain all the properties of the domain. No new columns may however use the domain. All routines, triggers or views whose definitions contain a CAST involving the domain will be dropped.

Function
When a function is dropped with the CASCADE option in effect, all constraints, functions, procedures, triggers or views invoking it will be dropped. Dropping any object referenced from the SQL statements in the body of a function will drop the **function** when the CASCADE option is in effect.

Ident
When an ident is dropped, all objects owned by the ident are dropped, and all privileges granted by the ident are revoked. (Remember that revocation of privileges, in particular, may have recursive effects on other objects.)

Module
When a module is dropped, all the routines belonging to the module are also dropped.

Procedure
When a procedure is dropped with the CASCADE option in effect, all other routines or triggers calling it will be dropped. Dropping any object referenced from the SQL statements in the body of a procedure will drop the **procedure** when the CASCADE option is in effect.

Schema
When a schema is dropped and CASCADE is in effect, all the objects belonging to the schema are also dropped. If RESTRICT is in effect, the schema will be dropped only if it is empty.

Sequence
When a sequence is dropped and CASCADE is in effect, all the objects (i.e. domains, functions, procedures, table columns, triggers and views) referencing the sequence are also dropped.

Shadow
DROP SHADOW deletes the named shadow from the data dictionary.

An attempt is made to delete the physical shadow file in the same way as for dropping a databank. If the shadow or the master databank is OFFLINE however, no attempt is made to delete the physical shadow file.

Synonym
There are no cascade effects when a synonym is dropped because it is resolved to the associated table or view when an SQL statement containing the synonym is executed. Thus, it is a table or view reference that is actually stored in the database, not the synonym reference. Once dropped, of course, the synonym can no longer be used in new SQL statements.

Table
When a table is dropped, all views based on that table and all triggers created on it are also dropped.

When a table referenced from within a routine or trigger is dropped with the CASCADE option in effect, the routine or trigger will also be dropped (see also the notes above for *Function, Module, Procedure* and *Trigger* for full cascade implications).

If a table used as a REFERENCES table in a FOREIGN KEY clause is dropped, the referential integrity constraint is lost from the table with the foreign key clause.

All cursors defined for a table must be closed before the table can be dropped.

Trigger
If a trigger has been created on a **non-updatable** view, the creator of the trigger implicitly gets the appropriate privilege for the trigger event on that view, with WITH GRANT OPTION.

The creator of the trigger may then have granted the privilege to other idents or may have used the privilege to perform updates on the view in one or more routines subsequently created.

If the trigger is then dropped, with the CASCADE option in effect, any routines using the privilege to update the view will be dropped and the privilege will be revoked from any idents to whom the trigger creator granted it.

View
When a view is dropped, all other views based on that view and all triggers created on it are also dropped.

When a view referenced from within a routine or trigger is dropped with the CASCADE option in effect, the routine or trigger will also be dropped (see also the notes above for *Function, Procedure* and *Module* for full cascade implications).

Comments
Comments may not be dropped from the data dictionary, but they may be replaced by blank comments (see COMMENT).

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 **SQL92**	EXTENDED	Support for dropping DATABANK's, IDENT's, SEQUENCE's, SHADOW's, SYNONYM's and TRIGGER's is a Mimer SQL extension. Specification of CASCADE or RESTRICT is optional in Mimer SQL, not mandatory.

ENTER

Connects a program ident to the system.

```
►── ENTER ident USING password ──────────────────────────────►
```

Usage

Embedded/Interactive/ODBC.

Description

The specified ident becomes the current ident, provided that the password is correct.

The *ident* and the *password* can be specified either as a host variable or as a literal.

A program ident is set up to have certain privileges and access to the database which apply after the ENTER statement has been used and replace those that apply to the ident executing the ENTER statement.

Restrictions

The ENTER statement may only be issued for idents of type PROGRAM. ENTER may not be used to log on to the system.

To perform an ENTER operation for a PROGRAM ident, the current ident must have EXECUTE privilege on that ident.

The ENTER statement may not be issued inside a transaction.

Notes

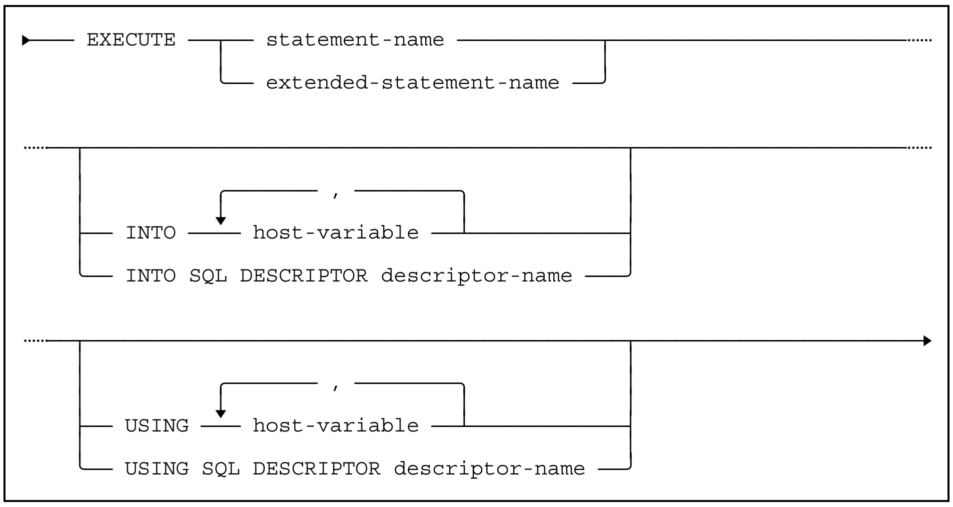
When the ENTER operation is successfully performed, the privileges granted to the new current ident apply and those granted to the previous current ident are suspended until the PROGRAM ident is left, see LEAVE (program ident).

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	MIMER EXTENSION	Support for the ENTER statement is a Mimer SQL extension.

EXECUTE

Associates parameter values with a previously prepared SQL statement and executes the statement.



Usage

Embedded.

Description

The prepared statement identified by the statement name is executed.

If the source form of the prepared statement contains parameter markers, the EXECUTE statement must be used with the USING clause and possibly also the INTO clause to correctly associate the statement parameters with an appropriate number of host variables or allocated SQL descriptor areas.

The DESCRIBE INPUT statement can be used to determine the PARAMETER_MODE of each of the parameters in the prepared statement.

If parameters are present with parameter mode IN or INOUT, the USING clause must be used to specify an SQL descriptor area or a list of host variables for these parameters. There must be one variable in the host variables list or referenced in the SQL descriptor area for each IN or INOUT parameter in the prepared statement. The variables are associated with the input parameter markers in a left to right manner as they appear in the prepared statement.

If parameters are present with parameter mode INOUT or OUT, the INTO clause must be used to specify an SQL descriptor area or a list of host variables for these parameters. There must be one variable in the host variables list or referenced in the SQL descriptor area for each INOUT or OUT parameter in the prepared statement. The variables are associated with the output parameter markers in a left to right manner as they appear in the prepared statement.

The data types of the variables must be compatible with their usage in the source statement.

The *descriptor-name* is identified by a host variable or a literal.

See ALLOCATE CURSOR for a description of extended statements.

For a fuller discussion of the use of EXECUTE statements in dynamic application programs, see Chapter 7 of the Mimer SQL Programmer's Manual.

Restrictions

The EXECUTE statement may only be used for dynamically prepared statements. Dynamically prepared SELECT statements may ***not*** be executed; these should be performed using a sequence of OPEN, FETCH and CLOSE statements.

Notes

When parameters with parameter mode INOUT are present, both a USING clause and an INTO clause is required. Each INOUT parameter must be associated with a variable in both the USING clause and the INTO clause.

The Mimer SQL Programmer's Manual contains information about the format of the host variable.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 **SQL92**	YES	Fully compliant.

EXECUTE IMMEDIATE

Prepares and executes a dynamically submitted SQL statement.

```
►── EXECUTE IMMEDIATE host-variable ──────────────────────────►
```

Usage

Embedded.

Description

The SQL source statement contained in the host variable is prepared and executed.

For a fuller discussion of the use of EXECUTE statements in dynamic application programs, see Chapter 7 of the Mimer SQL Programmer's Manual.

Restrictions

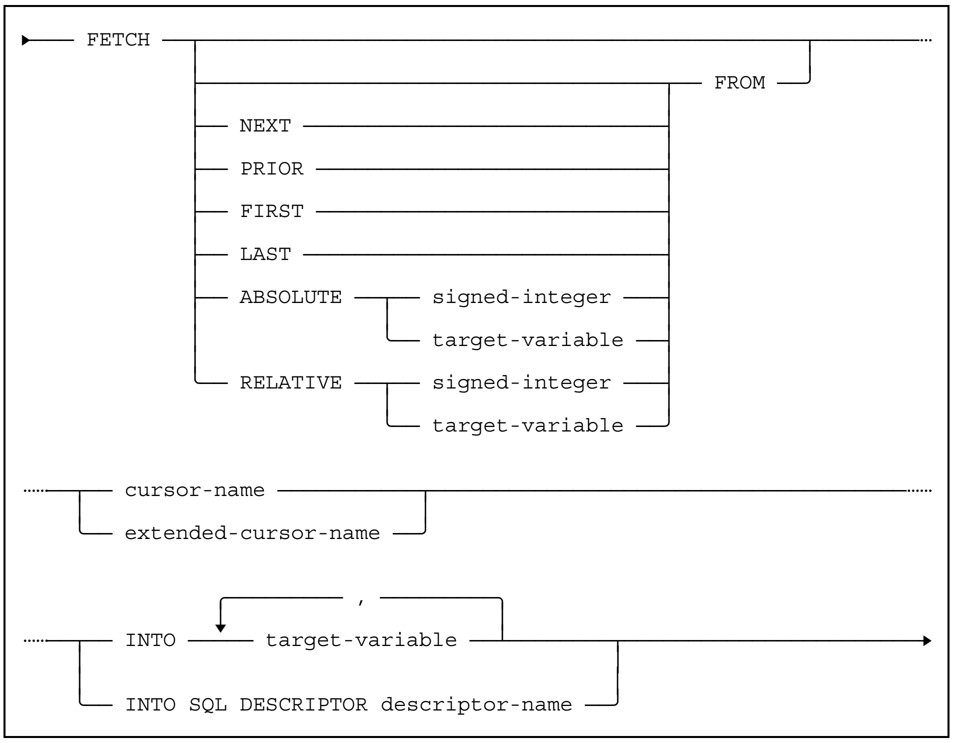
The EXECUTE IMMEDIATE statement may only be used for dynamically submitted statements with no parameter markers. Dynamically submitted SELECT statements may not be executed with EXECUTE IMMEDIATE, these should be performed using a sequence of OPEN, FETCH and CLOSE statements.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 **SQL92**	YES	Fully compliant.

FETCH

Positions a cursor on a specified row in the set addressed by the cursor, and assigns values from the row to host variables.



Usage

Embedded/Procedural.

Description

The named cursor is positioned on the specified row in the result set defined by the cursor declaration. This row becomes the current row for the cursor.

There are two imaginary row positions for a cursor, “one row prior to FIRST” and “one row after LAST”. The cursor will be positioned on these “rows” if any of the orientation specifications cause the cursor position to move before the FIRST row or after the LAST row of the result-set respectively. Once the cursor position reaches one of the imaginary rows it will not advance any further in that direction.

The *descriptor-name* is identified by a host variable or a literal.

See ALLOCATE CURSOR for a description of extended cursors.

When using a scrollable cursor, the position of the cursor can be specified with an orientation specification. The orientation can be specified in one of the following ways:

NEXT	Position the cursor on the row next to the current row
PRIOR	Position the cursor on the row prior to the current row
FIRST	Position the cursor on the first row in the result set
LAST	Position the cursor on the last row in the result set
ABSOLUTE	Position the cursor on the row with a specified absolute row number in the result set. Row zero does not exist and will return NOT FOUND. Negative numbers count from the **end** of the result set (i.e. -1 = LAST)
RELATIVE	Position the cursor on the row specified with a row number relative to the current row in the result set. Zero is the current row, positive numbers count toward the end of the result set and negative numbers toward the beginning.

Values from the current row are assigned to target variables as listed in the INTO clause or specified in the named SQL descriptor area. The form FETCH ... USING SQL DESCRIPTOR is used when an appropriate SQL descriptor area has been established. See Section 7.3 of the Mimer SQL Programmer's Manual for a discussion of the use of SQL descriptor areas.

The columns retrieved from the database are defined by the SELECT clause in the cursor declaration. The value from the first column in the SELECT clause is assigned to the first variable, that from the second column to the second variable, and so on. The data type of each variable must be assignment-compatible with the value in the corresponding column.

The number of columns in a row must be the same as the number of variables specified in the INTO clause.

If there is no next row in the set of rows, the cursor is placed "after the last row", no new values are assigned to the variables and a NOT FOUND condition code is returned (see Appendix E).

Language elements

target-variable	see Section 4.2.6.

Restrictions

SELECT access to the table or view addressed by the table reference is checked when the cursor used for the FETCH statement is opened. Access to the base table is not required for a FETCH operation on a view.

The cursor cannot be identified by specifying *extended-cursor-name* in a **Procedural** usage context. The INTO SQL DESCRIPTOR clause cannot be used in a **Procedural** usage context.

Notes

If the cursor is not declared as scrollable, the FETCH operation always positions the cursor at the next row in the result set. For such a cursor, orientation specification is not allowed (except for NEXT).

If the orientation specification is omitted for a scrollable cursor, NEXT is implicit.

If the cursor that is used by the FETCH statement is not declared with an “ORDER BY” clause, the sort order for the result set is undefined, even if the cursor is defined as scrollable. This means that the sort order may change if new indexes are created, if indexes are dropped, if new statistics are gathered, or if a new version of the SQL optimizer is installed. To assure a correct sort order, “ORDER BY” must be used.

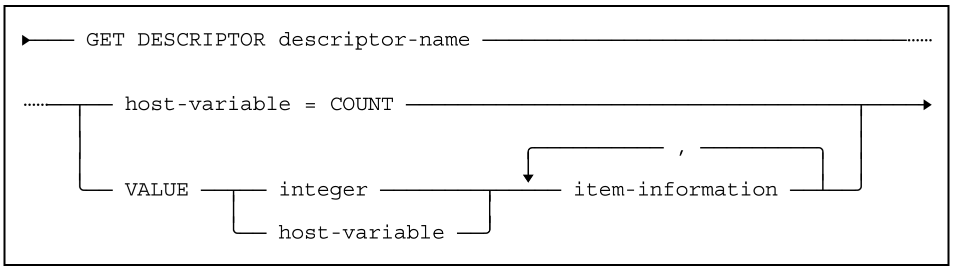
If a data conversion error occurs in assigning a value to a variable, an error code is returned and the execution of the FETCH statement is terminated. All variables successfully assigned before the error occurred retain their assigned values.

Standard compliance

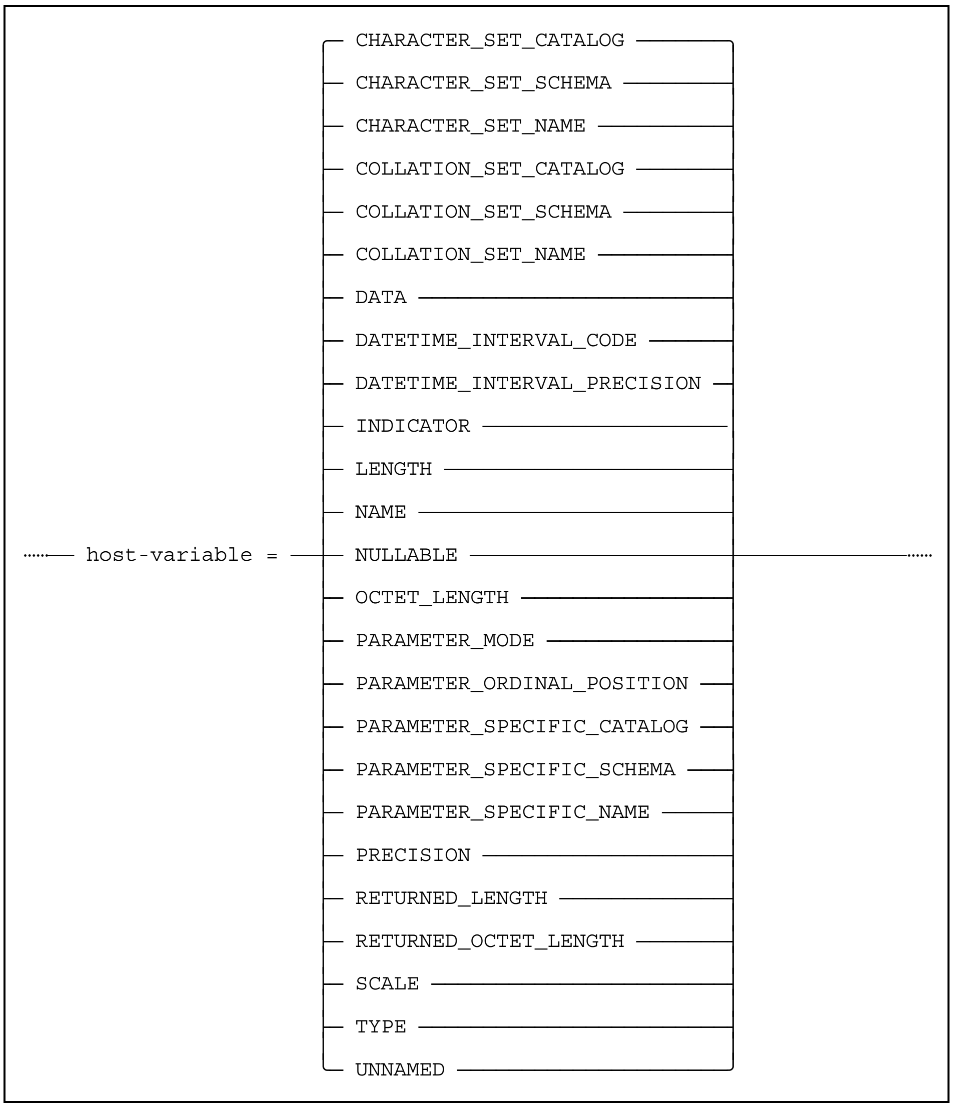
Standard	**Compliance**	**Comments**
X/Open-95	EXTENDED	Support for scrollable cursors is a Mimer SQL extension.
SQL92	YES	Fully compliant.

GET DESCRIPTOR

Gets values from an SQL descriptor area.



where *item-information* is:



Usage

Embedded.

Description

Values are retrieved from the specified SQL descriptor area. The GET DESCRIPTOR statement can be used in two forms. To determine the number of active item descriptor areas for the specified SQL descriptor, the form "...host-variable = COUNT" is used. The VALUE form is used to retrieve SQL descriptor field values from the item descriptor area specified by item-number.

The *descriptor-name* is identified by a host variable or a literal.

An item descriptor area contains the following fields:

CHARACTER_SET_CATALOG
: catalog name for the character set which the described item is using

CHARACTER_SET_SCHEMA
: A character string containing the schema name for the character set which the described item is using

CHARACTER_SET_NAME
: A character string containing the name of the character set which the described item is using

COLLATION_CATALOG
: A character string containing the catalog name for the collation which the described item is using

COLLATION_SCHEMA
: A character string containing the schema name for the collation which the described item is using

COLLATION_NAME
: A character string containing the name of the collation which the described item is using

DATA
: If the INDICATOR field does not indicate a NULL value, this field contains an input (OPEN or EXECUTE) or output (FETCH or EXECUTE with an INTO clause) value with the data type specified by the TYPE field, and with the attributes specified by the applicable fields in the item descriptor area.

DATETIME_INTERVAL_CODE
: If the TYPE field contains 9 or 10 (i.e. for DATETIME and INTERVAL data types), this column will contain an exact numeric value with scale 0 which specifies the DATETIME or INTERVAL subtype. See below for descriptions of the codes that apply to these two data types.

DATETIME_INTERVAL_PRECISION
: An exact numeric value with scale 0, which specifies the leading field precision for the INTERVAL data type (i.e. when the TYPE value is 10).

INDICATOR
An exact numeric value with scale 0, used as a NULL indicator for input (OPEN or EXECUTE) or output (FETCH or EXECUTE with an INTO clause) values. INDICATOR=-1 indicates a NULL value, and INDICATOR=0 indicates a non-NULL value. If INDICATOR is > 0 after a FETCH. operation or an EXECUTE operation with INTO clause, it indicates that a truncation occurred and the value of INDICATOR is the required length.

LENGTH
An exact numeric value with scale 0, containing the string length of a character or binary string data type. Terminating null bytes are excluded.

NAME
A character string containing the column name, returned by DESCRIBE OUTPUT. After DESCRIBE INPUT this field contains a question mark (“?”).

NULLABLE
An exact numeric value with scale 0, indicating whether a resulting column can contain NULL or not. NULLABLE=1 indicates that NULL is allowed. NULLABLE=0 indicates that NULL is not allowed.

OCTET_LENGTH
An exact numeric value with scale 0, containing the number of octets of a character or binary string data type. Terminating null bytes are excluded.

PARAMETER_MODE
An exact numeric value with scale 0, which specifies the MODE of a routine parameter. See below for a description of the code values for this field.

PARAMETER_ORDINAL_POSITION
An exact numeric value with scale 0, indicating the ordinal position of the parameter in the parameter list of the routine definition.

PARAMETER_SPECIFIC_CATALOG
A character string representing the catalog name for the invoked procedure, if the prepared statement is a call statement

PARAMETER_SPECIFIC_SCHEMA
A character string representing the schema name for the invoked procedure, if the prepared statement is a call statement

PARAMETER_SPECIFIC_NAME
A character string representing the name of the invoked procedure, if the prepared statement is a call statement

PRECISION

An exact numeric value with scale 0, specifying the precision for a numeric data type value. For the data types INTERVAL DAY TO SECOND, INTERVAL HOUR TO SECOND, INTERVAL MINUTE TO SECOND, INTERVAL SECOND, TIME and TIMESTAMP the value in this field describes the precision of the fractional SECONDS component.

RETURNED_LENGTH

An exact numeric value with scale 0, set by FETCH or EXECUTE with an INTO clause, which returns the actual length of a VARCHAR or VARBINARY output value.

RETURNED_OCTET_LENGTH

An exact numeric value with scale 0, set by FETCH or EXECUTE with an INTO clause, which returns the actual number of octets of a VARCHAR or VARBINARY output value

SCALE

An exact numeric value with scale 0, specifying the scale for a numeric data type value.

TYPE

An exact numeric value with scale 0, containing a coded representation of the data type. See below for a description of the codes.

UNNAMED

An exact numeric value with scale 0, indicating whether NAME contains an actual column name or a situation dependent generated name (for example if the value is a result of an aggregate function). UNNAMED=0 indicates that NAME contains an actual column name. UNNAMED=1 means that NAME does not contain an actual column name.

The TYPE field in the item descriptor area can contain one of the following values:

Code	**SQL data type**
1	CHARACTER
2	NUMERIC
3	DECIMAL
4	INTEGER
5	SMALLINT
6	FLOAT
7	REAL
8	DOUBLE PRECISION
9	DATETIME
10	INTERVAL
12	CHARACTER VARYING
-2	BINARY**
-3	BINARY VARYING
-11	INTEGER(p)*
-13	CHARACTER VARYING

* INTEGER(p) is a Mimer SQL specific data type used for integer data with a specified precision.

** Null padding is applied to the fixed size BINARY data type.

For DATETIME data types, the DATETIME_INTERVAL_CODE field in the item descriptor area can contain one of the following values:

Code	**DATETIME subtype**
1	DATE
2	TIME
3	TIMESTAMP

For INTERVAL data types, the DATETIME_INTERVAL_CODE field in the item descriptor area can contain one of the following values:

Code	INTERVAL subtype
1	YEAR
2	MONTH
3	DAY
4	HOUR
5	MINUTE
6	SECOND
7	YEAR TO MONTH
8	DAY TO HOUR
9	DAY TO MINUTE
10	DAY TO SECOND
11	HOUR TO MINUTE
12	HOUR TO SECOND
13	MINUTE TO SECOND

For routine parameters, the PARAMETER_MODE field in the item descriptor area can contain one of the following values:

Code	PARAMETER_MODE
1	PARAMETER_MODE_IN
2	PARAMETER_MODE_INOUT
4	PARAMETER_MODE_OUT

Restrictions

None.

Notes

The value of the DATA field is undefined if the INDICATOR field indicates the NULL value.

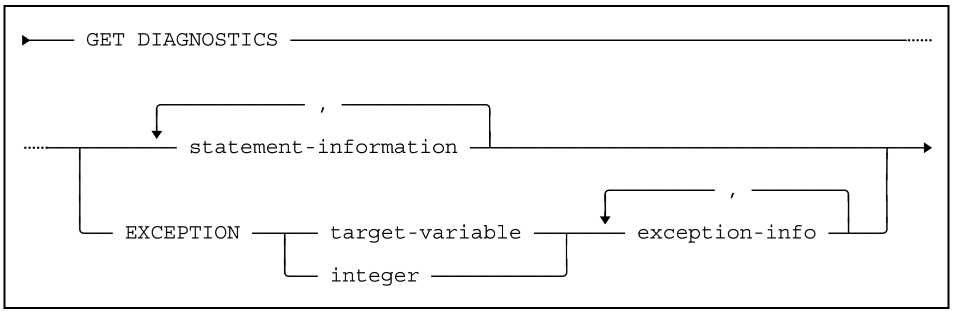
The data type of the host variables must be compatible with the data type of the associated field name.

Standard compliance

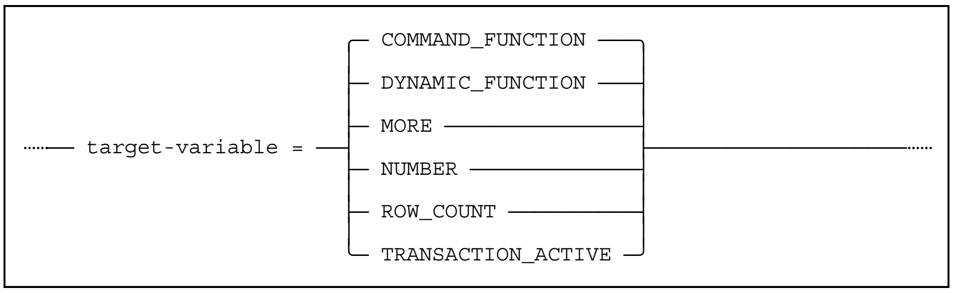
Standard	Compliance	Comments
X/Open-95 **SQL92**	YES	Fully compliant.

GET DIAGNOSTICS

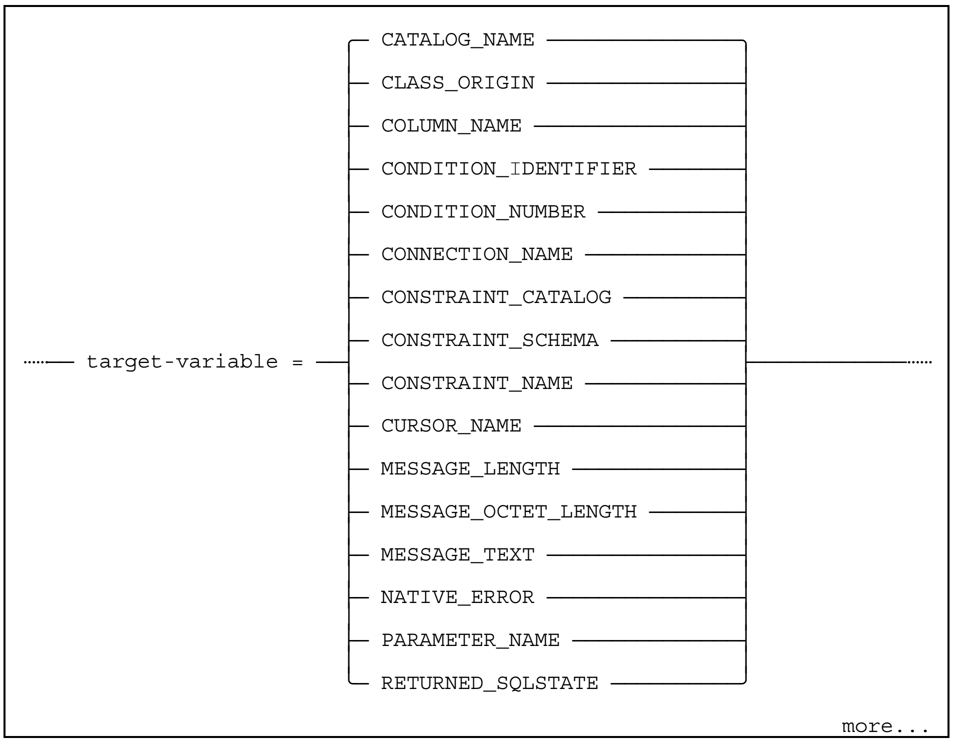
Gets statement or exception information from the diagnostics area.



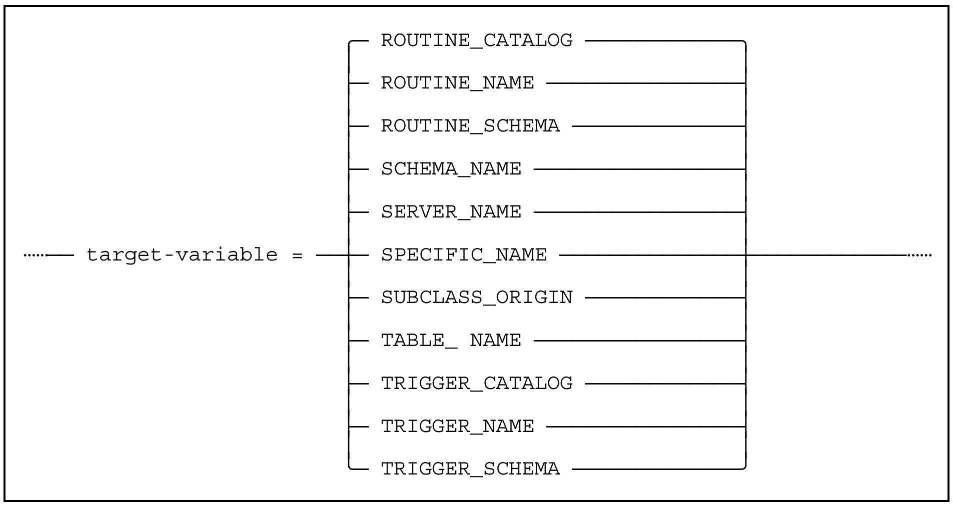
where *statement-information* is:



and *exception-info* is:



exception-info continued:



Usage

Embedded/Procedural.

Description

Selected status information from the diagnostics area is retrieved. The diagnostics area holds information about the most recently executed SQL statement. There is only one diagnostics area for each application, independent of the number of connections that the application holds. Observe that the GET DIAGNOSTICS statement itself does not change the diagnostics area, although it does set SQLSTATE.

The GET DIAGNOSTICS statement can be in two forms: the first form retrieves statement information about the most recent SQL statement executed. The second form of GET DIAGNOSTICS is the “EXCEPTION” form, which retrieves exception information for the most recently executed SQL statement. The ordinal number of the exception to be returned is specified immediately following the keyword EXCEPTION.

The information items for *statement-information* are described in the following table:

Information item	**Data type**	**Description**
COMMAND_FUNCTION	VARCHAR(128)	A string identifying the preceding embedded SQL statement executed.
DYNAMIC_FUNCTION	VARCHAR(128)	A string identifying the preceding prepared SQL statement executed.
MORE	CHAR(1)	Indicates if there are any exceptions for which no exception information has been stored. "N" if all detected exceptions are stored in the diagnostics area, otherwise "Y".
NUMBER	INTEGER	The number of exception messages stored for the most recently executed SQL statement.
ROW_COUNT	INTEGER	The number of rows inserted, updated or deleted if the last statement was INSERT, searched UPDATE or searched DELETE.
TRANSACTION_ACTIVE	INTEGER	Indicates if a transaction is active or not. 0 = transaction not active, 1 = transaction is active.

The information items for *exception-info* are described in the following table:

Information item	**Data type**	**Description**
CATALOG_NAME	VARCHAR(128)	The catalog name of the schema containing the table on which the violated constraint is defined, always an empty string ("").
CLASS_ORIGIN	VARCHAR(128)	The defining source of the two first characters (the class portion) of the SQLSTATE value.
COLUMN_NAME	VARCHAR(128)	The name of the table column on which the violated constraint is defined. If the constraint involves more than one column or the data change operation causing the exception is not in the table on which the constraint is defined, this will be an empty string ("").

CONDITION_IDENTIFIER	VARCHAR(128)	The value specified for *condition-name* in the DECLARE CONDITION statement declaring the exception as a named condition. This will be the empty string (“”) if the exception has not been declared as a named condition.
CONDITION_NUMBER	INTEGER	The ordinal number of the exception on the diagnostics exception stack.
CONNECTION_NAME	VARCHAR(128)	The connection name specified in a CONNECT, DISCONNECT or SET CONNECTION statement. The name of the current connection for all other statements.
CONSTRAINT_CATALOG	VARCHAR(128)	The catalog name of the schema containing the violated constraint, always an empty string (“”).
CONSTRAINT_SCHEMA	VARCHAR(128)	The name of the schema containing the violated constraint.
CONSTRAINT_NAME	VARCHAR(128)	The name of the violated constraint.
CURSOR_NAME	VARCHAR(128)	The name of the cursor which is in an invalid state, when the exception 24000 - “Invalid Cursor State” is raised.
MESSAGE_LENGTH	INTEGER	The length of the message text for the specified exception.
MESSAGE_OCTET_ LENGTH	INTEGER	Currently the same as MESSAGE_LENGTH.
MESSAGE_TEXT	VARCHAR(254)	The descriptive message text for the specified exception.
NATIVE_ERROR	INTEGER	The internal Mimer SQL return code relating to the exception (listed in Appendix B of the Mimer SQL Programmer’s Manual).
PARAMETER_NAME	VARCHAR(128)	The name of the routine parameter causing the exception.
RETURNED_SQLSTATE	CHAR(5)	Value of SQLSTATE for the specified exception.
ROUTINE_CATALOG	VARCHAR(128)	The catalog name of the schema containing the function or procedure in which the exception was raised, always an empty string (“”).

ROUTINE_SCHEMA	VARCHAR(128)	The name of the schema containing the function or procedure in which the exception was raised.
ROUTINE_NAME	VARCHAR(128)	The name of the function or procedure in which the exception was raised.
SCHEMA_NAME	VARCHAR(128)	The name of the schema containing the table on which the violated constraint is defined. If the data change operation causing the exception is not in the table on which the constraint is defined, this will be an empty string (“”).
SERVER_NAME	VARCHAR(128)	The database name specified in a CONNECT, DISCONNECT or SET CONNECTION statement. The current database name for all other statements.
SPECIFIC_NAME	VARCHAR(128)	Currently the same as ROUTINE_NAME.
SUBCLASS_ORIGIN	VARCHAR(128)	The defining source of the three last characters (the subclass portion) of the SQLSTATE value.
TABLE_NAME	VARCHAR(128)	The name of the table on which the violated constraint is defined. If the data change operation causing the exception is not in the table on which the constraint is defined, this will be an empty string (“”).
TRIGGER_CATALOG	VARCHAR(128)	The catalog name of the schema containing the table supporting the trigger in which the exception was raised, always an empty string (“”).
TRIGGER_SCHEMA	VARCHAR(128)	The name of the schema containing the table supporting the trigger in which the exception was raised.
TRIGGER_NAME	VARCHAR(128)	The name of the trigger in which the exception was raised.

The COMMAND_FUNCTION and DYNAMIC_FUNCTION information items can contain any of the following values:

ALLOCATE CURSOR
ALLOCATE DESCRIPTOR
ALTER DATABANK
ALTER DATABANK RESTORE
ALTER IDENT
ALTER SHADOW
ALTER TABLE
ASSIGNMENT
CALL
CLOSE CURSOR
COMMENT
COMMIT WORK
CONNECT
CREATE BACKUP
CREATE DATABANK
CREATE DOMAIN
CREATE FUNCTION
CREATE IDENT
CREATE INDEX
CREATE MODULE
CREATE PROCEDURE
CREATE SCHEMA
CREATE SEQUENCE
CREATE SHADOW
CREATE SYNONYM
CREATE TABLE
CREATE TRIGGER
CREATE VIEW
DEALLOCATE DESCRIPTOR
DEALLOCATE PREPARE
DELETE CURSOR
DELETE WHERE
DESCRIBE
DISCONNECT
DROP DATABANK
DROP DOMAIN
DROP FUNCTION
DROP IDENT
DROP INDEX
DROP MODULE
DROP PROCEDURE
DROP SCHEMA
DROP SEQUENCE
DROP SHADOW
DROP SYNONYM
DROP TABLE
DROP TRIGGER
DROP VIEW
DYNAMIC CLOSE
DYNAMIC DELETE CURSOR
DYNAMIC FETCH
DYNAMIC OPEN
DYNAMIC UPDATE CURSOR
ENTER
EXECUTE
EXECUTE IMMEDIATE
FETCH
GET DESCRIPTOR
GET DIAGNOSTICS
GRANT
GRANT OBJECT PRIVILEGE
GRANT SYSTEM PRIVILEGE
INSERT
LEAVE
LEAVE RETAIN
OPEN
PREPARE
REVOKE
REVOKE OBJECT PRIVILEGE
REVOKE SYSTEM PRIVILEGE
ROLLBACK WORK
SELECT
SET CONNECTION
SET DATABANK
SET DATABASE
SET DESCRIPTOR
SET SESSION DIAGNOSTIC SIZE
SET SESSION ISOLATION LEVEL
SET SESSION READ ONLY
SET SESSION READ WRITE
SET SESSION START EXPLICIT
SET SESSION START IMPLICIT
SET SHADOW
SET TRANSACTION DIAGNOSTIC SIZE
SET TRANSACTION ISOLATION LEVEL
SET TRANSACTION READ ONLY
SET TRANSACTION READ WRITE
SET TRANSACTION START EXPLICIT
SET TRANSACTION START IMPLICIT
START TRANSACTION
UPDATE CURSOR
UPDATE WHERE
UPDATE STATISTICS

Language elements

target-variable	see Section 4.2.6.

Restrictions

None.

Notes

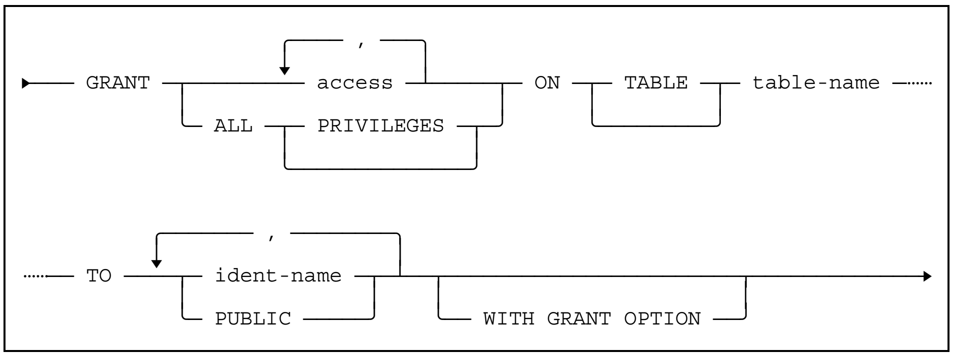
The exception requested by the GET DIAGNOSTICS EXCEPTION form must be one of the exceptions that exist in the diagnostics area, i.e. the exception number must be in the range from 1 up to the value of NUMBER.

Standard compliance

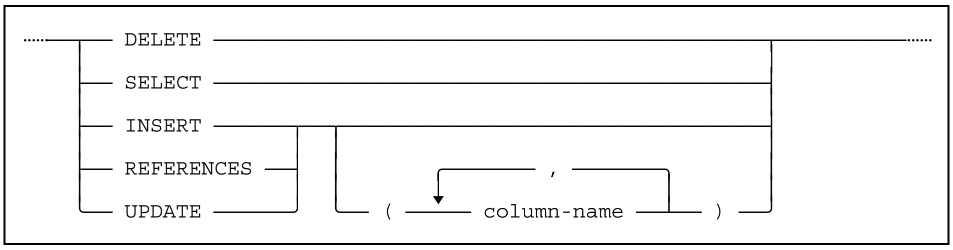
Standard	**Compliance**	**Comments**
X/Open-95 SQL92	YES	Fully compliant.

GRANT ACCESS PRIVILEGE

Grants one or more access privileges on a table or view.



where *access* is



Usage

Embedded/Interactive/ODBC.

Description

The specified access privileges are granted to the ident(s) listed. When WITH GRANT OPTION is specified, the ident may in turn grant the specified access privileges to another ident. If the privileges are granted to a GROUP ident, all members of the group receive the privileges.

The access privileges are as follows:

DELETE	allows the ident to delete rows from the table or view identified by *table-name*. If a selective delete is specified, involving a WHERE clause, appropriate privileges must also to held to permit execution of the search-condition, otherwise the delete operation will fail.
INSERT	allows the ident to insert new rows into the table or view identified by *table-name*. If a column name list is supplied, the access is restricted to the specified columns, otherwise the access applies to the entire table (including any new columns subsequently added).

REFERENCES	allows the ident to use the table identified by *table-name* in the FOREIGN KEY clause of a CREATE TABLE statement. REFERENCES access may only be granted on a base table, not on a view. If a column name list is supplied, the access is restricted to the specified columns, otherwise the access applies to the entire table (including any new columns subsequently added).
SELECT	allows the ident to select rows from the table or view identified by *table-name*.
UPDATE	allows the ident to update the table or view identified by *table-name*. If a column name list is supplied, the access is restricted to the specified columns, otherwise the access applies to the entire table (including any new columns subsequently added). If a selective update is specified, involving a WHERE clause, appropriate privileges must also to held to permit execution of the search-condition, otherwise the update operation will fail.

Access privileges may be granted in any combination. Specification of the keyword ALL (followed by the optional keyword PRIVILEGES) instead of an explicit list of access privileges results in all applicable privileges being granted to the specified ident(s) (i.e. all privileges which the grantor is authorized to grant).

Restrictions

The grantor must have grant option on the access privilege.

Notes

If the grantor loses WITH GRANT OPTION, any access privileges he has granted using it are automatically revoked.

An ident may not grant access privileges to himself.

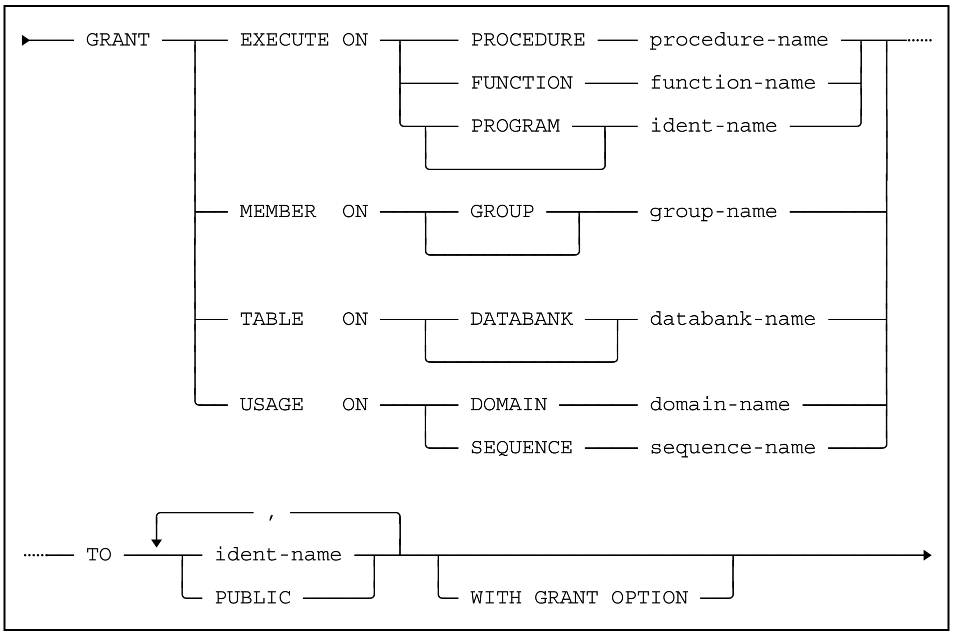
An ident may gain WITH GRANT OPTION because of the effect of a "cascading grant" in the following situation:
If an ident creates an object that references at least one object on which the ident holds an access privilege **without** WITH GRANT OPTION, the ident will not hold WITH GRANT OPTION on the created object.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	YES	Use of the keyword PRIVILEGES after the keyword ALL is optional in Mimer SQL, not mandatory.

GRANT OBJECT PRIVILEGE

Grants object privileges to one or more idents.



Usage

Embedded/Interactive/ODBC.

Description

The specified object privilege is granted to the ident(s) listed. When WITH GRANT OPTION is specified, the ident may in turn grant the specified privilege to another ident. If the privilege is granted to a GROUP ident, all members of the group receive the specified privilege.

The object privileges are as follows:

EXECUTE	allows the ident to **invoke** the specified function or procedure, or to **enter** the specified program ident.
MEMBER	specifies that the ident is a member of the stated **group**. All privileges granted to the group are held by all members of the group.
TABLE	allows the ident to **create** new tables in the specified databank.
USAGE	allows the ident to specify the named **domain** where a data type would normally be specified, in contexts where use of domains is allowed or to use the value of a **sequence**, in contexts where this is allowed.

Restrictions

The grantor must have grant option on the privilege.

Notes

If the grantor loses his grant option, any privileges he has granted using the option are automatically revoked.

An ident may not grant privileges to himself.

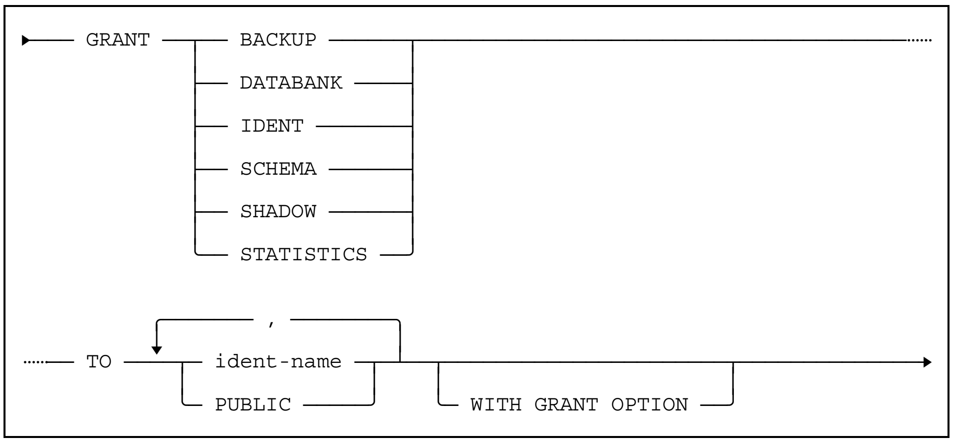
An ident may gain WITH GRANT OPTION because of the effect of a “cascading grant” in the following situation:
If an ident creates an object that references at least one other object on which the ident holds an object privilege **without** WITH GRANT OPTION, the ident will not hold WITH GRANT OPTION on the created object.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	EXTENDED	Support for the keywords EXECUTE, MEMBER and TABLE is a Mimer SQL extension. Support for GRANT USAGE ON SEQUENCE is a Mimer SQL extension.
SQL/PSM	EXTENDED	Support for EXECUTE ON PROGRAM and the privileges MEMBER and TABLE is a Mimer SQL extension.

GRANT SYSTEM PRIVILEGE

Grants system privileges to one or more idents.



Usage

Embedded/Interactive/ODBC.

Description

The specified system privilege is granted to the ident(s) listed. When WITH GRANT OPTION is specified, the ident may in turn grant the specified privilege to another ident. If the privilege is granted to a GROUP ident, all members of the group receive the specified privilege.

The system privileges are as follows:

BACKUP	allows the ident to perform databank backup and restore operations.
DATABANK	allows the ident to create new databanks. The databank file is created by the Mimer SQL system. The privilege authorizes the ident to create files, using the file access used by the database server process, in the operating system.
IDENT	allows the ident to create new Mimer SQL idents and schemas.
SCHEMA	allows the ident to create new schemas.
SHADOW	allows the ident to create and perform operations on databank shadows.
STATISTICS	allows the ident to execute the UPDATE STATISTICS statement.

Restrictions

The grantor must have grant option on the privilege.

Notes

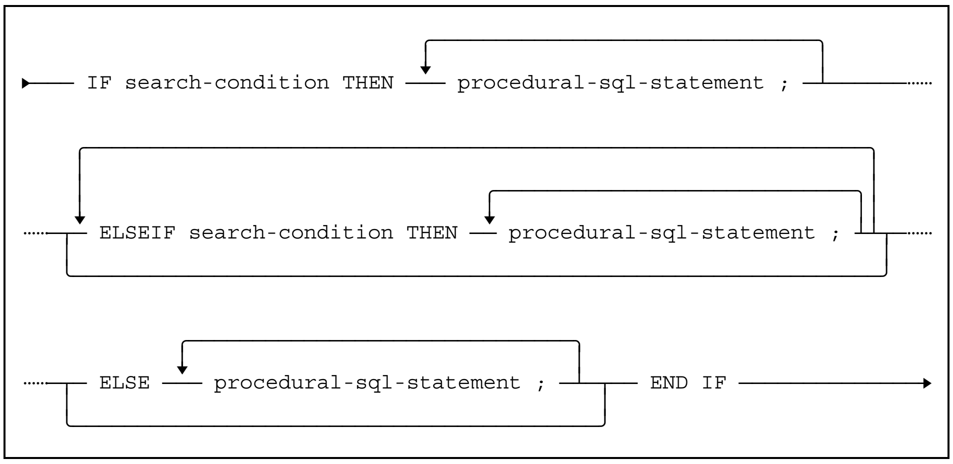
An ident may not grant privileges to himself.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	MIMER EXTENSION	Support for system privileges is a Mimer SQL extension.

IF

Provides conditional execution based on the truth value of a conditional expression.


IF search-condition THEN
procedural-sql-statement ;
ELSEIF search-condition THEN
procedural-sql-statement ;
ELSE
procedural-sql-statement ;
END IF

Usage

Procedural.

Description

The IF-statement allows a sequence of procedural-sql-statements to be conditionally executed depending on the truth value a *search-condition*.

All of the predicates supported by Mimer SQL may be used in the *search-condition* (see Section 5.9).

The *search-condition* is always evaluated directly in the context of the IF-statement, its value **cannot** be pre-assigned to a data item which is used in its place.

In a basic IF-statement, the sequence of procedural-sql-statements in the THEN clause will be executed if *search-condition* evaluates to **true**, otherwise the sequence of procedural-sql-statements in the ELSE clause will be executed.

One or more IF-statements can be nested by using the ELSEIF clause in place of an ELSE clause containing another IF-statement.

Language elements

search-condition	see Section 5.10.

Restrictions

None.

Notes

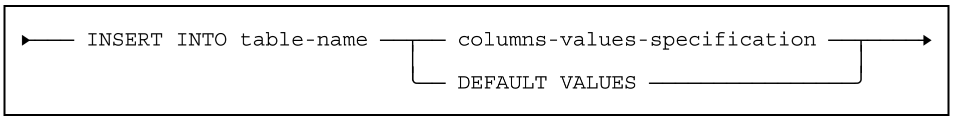
If the *search-condition* equals NULL or directly includes the NULL value, it evaluates to unknown and it treated as **false**. If it is required that the conditional expression test for the NULL value, then the correct behavior is achieved by using the IS NULL predicate (see Section 5.9.6).

Standard compliance

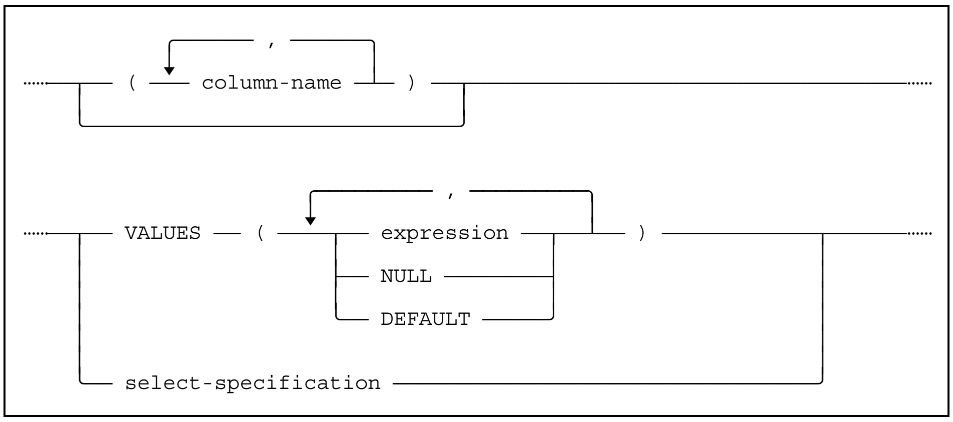
Standard	**Compliance**	**Comments**
SQL/PSM	YES	Fully compliant.

INSERT

Inserts one or more rows into a table or view.



where *columns-values-specification* is:



Usage

Embedded/Interactive/ODBC/Procedural.

Description

One or more new rows are inserted into the table or view specified in *table-name*.

If a list of column names is given in *columns-values-specification*, only the specified columns are assigned values in accordance with the INSERT statement. The columns not listed are assigned their default value or the NULL value in accordance with the column definition (see CREATE TABLE). If *table-name* specifies a view, any columns in the base table which are excluded from the view are also assigned their default value or the NULL value in the same way.

If the column name list is omitted, all columns in the table or view are implicitly specified in the order in which they are defined in the table or view. This practice is, however, not recommended when INSERT statements are embedded in application programs, since the semantics of the statement will change if the table or view definition is changed.

Specification of a DEFAULT VALUES clause inserts a single row into the table with the column default value specified for **each** column in the table.

Values are assigned in order from the items in the VALUES clause or the select-specification to the columns that have been explicitly or implicitly specified. The number of values specified must be the same as the number of columns and the data type of each value must be assignment-compatible with the column into which it is to be inserted.

Specification of a VALUES clause inserts a single row into the table or view. The keyword NULL or DEFAULT can be specified in the VALUES clause to insert the NULL value or the column default value, respectively, into the corresponding column.

Specification of a select-specification instead of a VALUES clause inserts the set of rows resulting from the select-specification into the target table or view. If the set of rows resulting from the select-specification is empty, a NOT FOUND condition code is returned (see Appendix E).

Language elements

expression	see Section 5.8.
select-specification	see Section 5.11.

Restrictions

INSERT access is required on the table or view specified in the INTO clause. If a select-specification is specified, SELECT access is required on the table(s) from which the selection is performed.

In a **procedural** usage context, the INSERT statement is only permitted if the procedure *access-clause* is MODIFIES SQL DATA (see CREATE PROCEDURE).

Notes

Expressions used in the VALUES clause cannot refer to column names or set functions.

If the row or rows inserted do not conform to constraints imposed on the table, no rows are inserted. Constraints are as follows:

- Values in the primary key and unique keys of the base table may not be duplicated. This also applies to unique secondary indexes.
- FOREIGN KEY constraints must be observed.
- CHECK constraints in table, column and domain definitions must be observed for insertions.
- For insertion into views defined WITH CHECK OPTION, inserted values must conform to the view definition.

Standard compliance

Standard	Compliance	Comments
X/Open-95 SQL92	YES	Fully compliant.

LEAVE

Leaves a labeled procedural-sql-statement.

```
►── LEAVE label ──────────────────────────────────────────────►
```

Usage

Procedural.

Description

The LEAVE-statement can be used to terminate the execution of a procedural-sql-statement.

The *label* is the beginning label of a procedural-sql-statement within the scope containing the LEAVE-statement.

Restrictions

A procedural-sql-statement must have a beginning label if LEAVE is to be used to terminate its execution.

Notes

If the LEAVE-statement is contained in a compound statement the following actions occur before execution of the compound statement is terminated, after the LEAVE-statement is executed:

- Every open cursor declared in the compound statement is closed.
- All variables, cursors and handlers declared in the compound statement are destroyed.
- All condition names declared in the compound statement cease to be defined.

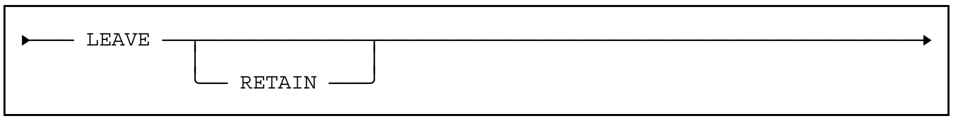
If the LEAVE-statement is executed within a compound statement forming the body of a procedure, execution of the procedure will be terminated.

Standard compliance

Standard	**Compliance**	**Comments**
SQL/PSM	YES	Fully compliant.

LEAVE (program ident)

Leaves a program ident and restores the state of the previous ident.



Usage

Embedded/Interactive/ODBC.

Description

The current program ident is left and the saved environment of the previous ident is restored.

If RETAIN is specified, resources allocated to the ident being left are kept, i.e. cursor declarations. The cursors are however inactivated, and may not be used in any statement until the same ident is re-entered.

Restrictions

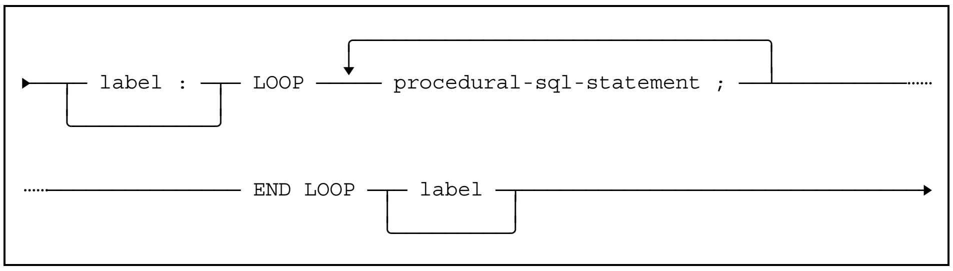
The LEAVE statement may not be issued within a transaction.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	MIMER EXTENSION	Support for PROGRAM idents is a Mimer SQL extension.

LOOP

Allows one or more procedural SQL statements to be iteratively executed.



Usage

Procedural.

Description

The LOOP statement can be used to iteratively execute a sequence of one or more *procedural-sql-statement*'s.

Restrictions

If *label* appears at the beginning **and** at the end of the LOOP statement, the same value must be specified in both places.

Specifying *label* is optional, however, if *label* appears at the end of the LOOP statement, it **must** also appear at the beginning.

A label is required at the beginning if the LEAVE statement is to be used to terminate the LOOP statement.

Notes

The LOOP statement itself does not include a mechanism for terminating the iteration.

The LOOP statement can be terminated by executing the LEAVE statement. The LOOP statement will also terminate if an exception condition is raised, in accordance with the normal exception handling process.

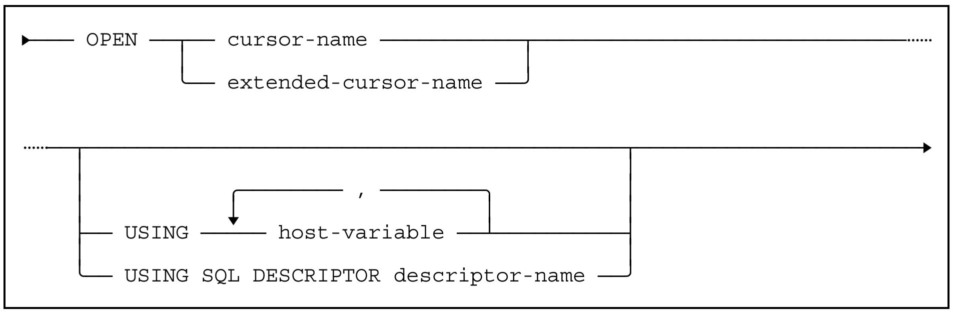
The LOOP statement does **not** establish any form of local scope, as the compound statement does, the *label* is only specified to allow the LEAVE statement to be used to terminate the iteration.

Standard compliance

Standard	**Compliance**	**Comments**
SQL/PSM	YES	Fully compliant.

OPEN

Opens a cursor.



Usage

Embedded/Procedural.

Description

The cursor is opened and references a set of rows in accordance with the definition of the cursor. The result-set is defined when the OPEN statement is executed, any inserts, updates or deletes occurring after the open will not be reflected in the result-set. The cursor is placed “before” the first row in the addressed set.

The *descriptor-name* is identified by a host variable or a literal.

See ALLOCATE CURSOR for a description of extended cursors.

If the cursor is declared for a dynamically prepared SELECT statement, the source form of which contains parameter markers, the OPEN statement must include a USING clause. The first variable in the variable list or referenced in the descriptor area takes the place of the first parameter marker, the second variable takes the place of the second marker, and so on. The number of variables provided in the USING clause must be equal to the number of parameter markers in the source statement, and the data types of the variables must be assignment-compatible with their usage in the source statement. See Chapter 7 of the Mimer SQL Programmer’s Manual for a more detailed description of the use of dynamically prepared statements.

If the cursor is already in the open state and the cursor has been declared as REOPENABLE, the previous state is saved on a cursor stack, and is recalled when the newly opened cursor is closed (see CLOSE). In this context, the “state” of a cursor includes both the set of rows addressed by the cursor and the position of the cursor within the set.

Cursors not declared as REOPENABLE may not be opened more than once in succession, unless it is closed by one of the statements COMMIT, ROLLBACK or CLOSE.

Restrictions

SELECT access is required to the table(s) or view(s) addressed by the cursor.

In a **procedural** usage context, *extended-cursor-name* cannot be used to identify the cursor and neither of the USING options may be used.

Notes

A cursor must be defined with a DECLARE CURSOR statement before it may be opened.

All access rights that the current ident holds to the table(s) or view(s) addressed by the cursor are checked when the cursor is opened. If SELECT access is lacking, the OPEN statement will fail. If UPDATE or DELETE access is lacking, the cursor may be opened but any corresponding UPDATE CURRENT or DELETE CURRENT statements will fail.

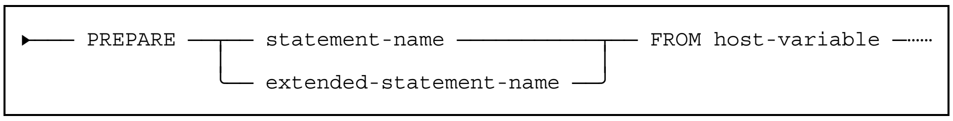
Only cursors declared in dynamically prepared statements may be opened with a USING clause.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	YES	Fully compliant.

PREPARE

Prepares an SQL statement contained in a host string variable for execution.



Usage

Embedded.

Description

The SQL statement contained in the host variable is prepared for execution. This function is equivalent to pre-processing and compiling an embedded SQL statement, but is performed at run-time. The (internal) output form of the statement is identified by the statement-name parameter.

See ALLOCATE CURSOR for a description of extended statements.

The following SQL statements can be used with the PREPARE statement:

Connect statements	ENTER
	LEAVE
Data definition statements	CREATE
	ALTER
	DROP
	COMMENT
Access control statements	GRANT
	REVOKE
Transaction control statements	START
	COMMIT
	ROLLBACK
	SET SESSION
	SET TRANSACTION
Data manipulation statements	INSERT
	DELETE
	DELETE CURRENT
	UPDATE
	UPDATE CURRENT
	SELECT
	SET
	CALL
System administration statements	SET DATABASE
	SET DATABANK
	SET SHADOW

Restrictions

None.

Notes

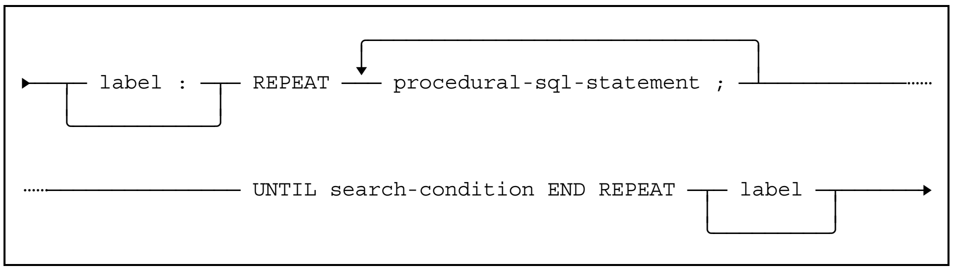
The source form of the SQL statement to be prepared is not preceded by the identifier “EXEC SQL” or terminated by the language-specific delimiter. The source statement may contain question marks as parameter markers to represent variables to be used when the prepared statement is executed. See Chapter 7 of the Mimer SQL Programmer’s Manual for more details.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	YES	Fully compliant.

REPEAT

Allows one or more procedural SQL statements to be iteratively executed.



Usage

Procedural.

Description

The REPEAT-statement can be used to iteratively execute a sequence of one or more *procedural-sql-statements*.

The iteration **continues until** *search-condition* evaluates to **true**.

Restrictions

If *label* appears at the beginning **and** at the end of the REPEAT-statement, the same value must be specified in both places.

Specifying *label* is optional, however, if *label* appears at the end of the REPEAT-statement, it **must** also appear at the beginning.

A label is required at the beginning if the LEAVE-statement is to be used to terminate the REPEAT-statement.

Notes

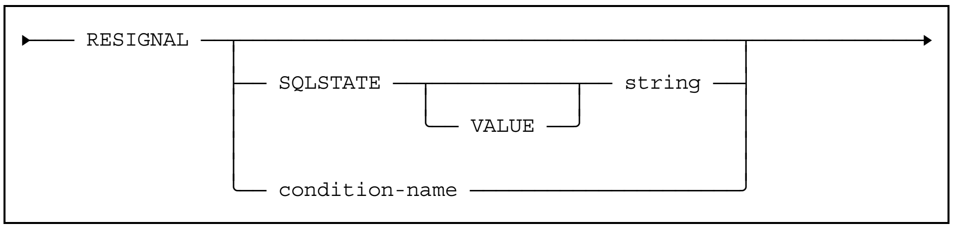
The REPEAT-statement may be terminated by executing the LEAVE-statement using *label*. It will also terminate if an exception condition is raised, in accordance with the normal exception handling process.

Standard compliance

Standard	Compliance	Comments
SQL/PSM	YES	Fully compliant.

RESIGNAL

Raises the current, or the specified, exception condition.



Usage

Procedural.

Description

The RESIGNAL-statement has the effect of raising the **current** exception condition, if specified without an argument, or an alternative exception condition specified by either an SQLSTATE value or a condition name.

Restrictions

The RESIGNAL-statement may only be used **within an exception handler** (see DECLARE HANDLER) to force re-propagation of an exception condition to the scope or calling environment enclosing the scope supporting the exception handler.

Notes

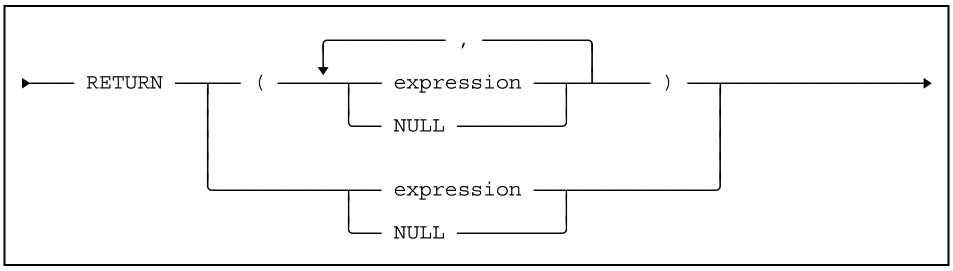
See DECLARE CONDITION for a description of how to declare a condition name.

Standard compliance

Standard	**Compliance**	**Comments**
SQL/PSM	YES	Fully compliant.

RETURN

Returns the specified value(s) from a result set procedure or a function.



Usage

Procedural.

Description

The RETURN statement is used in a function to return the single value of the **function**.

The SQL statements in the body of the function are executed until a RETURN statement is executed. If the end of the function is encountered (because no RETURN statement has been executed) an exception is raised.

The RETURN statement is used in a **result set procedure** to return the value(s) of a row of the result-set to the calling cursor when FETCH is executed for it.

When a FETCH is executed for a cursor calling a result set procedure, the SQL statements in the body of the result set procedure are executed until a RETURN statement is executed, then execution within the result set procedure is suspended until the next FETCH.

Note: An array FETCH will cause more than one RETURN statement to be executed, so there is not necessarily a 1:1 correspondence between the number of FETCH statements executed and the number of RETURN statements executed.

If, following a FETCH, the end of the result set procedure is encountered instead of a RETURN statement, the NOT FOUND exception is raised to indicate the end of the result-set.

Restrictions

If the RETURN statement is used in a procedure, it must be a **result set procedure** (see Section 8.6 of the Mimer SQL Programmer’s Manual).

Notes

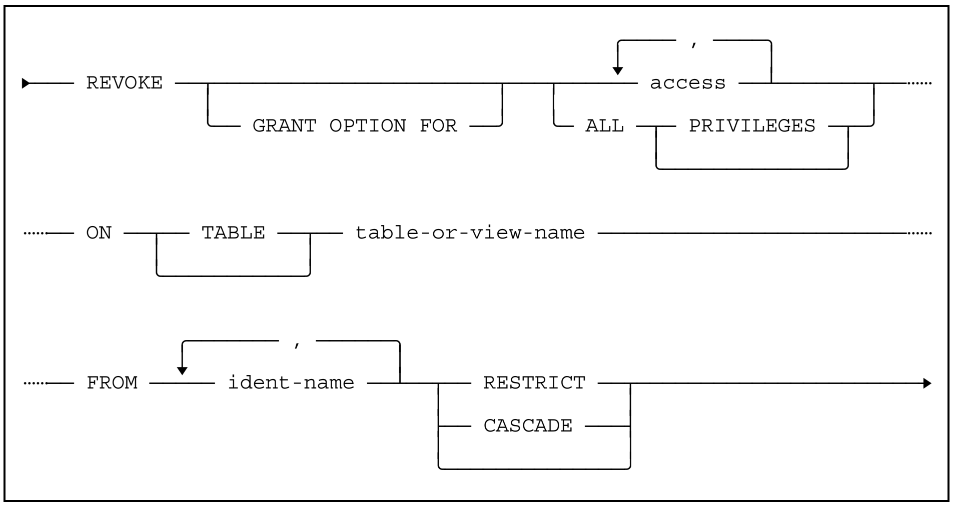
If only **one** value expression is being returned, the parentheses are optional.

Standard compliance

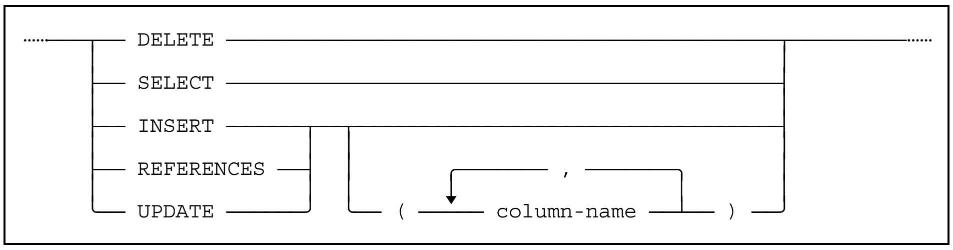
Standard	Compliance	Comments
SQL/PSM	EXTENDED	The use of RETURN to return the row data of a result-set from a result set procedure is a Mimer SQL extension.

REVOKE ACCESS PRIVILEGE

Revokes access privileges on a table or view, from one or more idents.



where *access* is



Usage

Embedded/Interactive/ODBC.

Description

The specified access privileges are revoked from the ident(s) listed. If the privileges are revoked from a GROUP ident, all members of the group lose the privileges.

The access privileges are described under GRANT ACCESS PRIVILEGE.

The access privileges may be revoked in any combination. Specification of the keyword ALL (followed by the optional keyword PRIVILEGES) instead of an explicit list of privileges results in all access privileges on the table or view being revoked from the specified ident(s).

The GRANT OPTION FOR clause specifies that only the WITH GRANT OPTION is to be revoked from the specified instance(s) of the privilege(s).

The keywords CASCADE and RESTRICT specify whether the REVOKE statement will allow the recursive effects that cause views to be dropped or FOREIGN KEY constraints to be removed, as a result of the REVOKE statement. Refer to the ***Notes*** section below for details of the recursive effects. If CASCADE is specified, such recursive effects will be allowed. If RESTRICT is specified, the REVOKE statement will return an error if it would cause such recursive effects and then no access privileges will be revoked.

If neither CASCADE nor RESTRICT is specified, then RESTRICT is implicit.

Restrictions

Privileges may only be revoked explicitly by the grantor.

Notes

If an access privilege has been granted to the same ident more than once (by different grantors), the REVOKE statement will only revoke (or will revoke the WITH GRANT OPTION from) the single instance of the privilege that was granted by the current ident.

The access rights attached to the privilege (or the WITH GRANT OPTION) will only be lost when the **last** instance of the privilege has been revoked.

Revoking access privileges has recursive effects.

When SELECT access on a table or view is revoked, views based on that table or view and created under the authorization of that access, are recursively dropped.

When UPDATE, INSERT, DELETE or REFERENCES access on a table or view is revoked, the same privilege on views based on that table or view and created under the authorization of the access are recursively revoked.

When REFERENCES access on an entire table or on one or more explicitly specified columns of the table is revoked, any FOREIGN KEY constraints in tables created by that ident under the authorization of that privilege are removed.

When INSERT, REFERENCES or UPDATE access is revoked from one or more explicitly specified columns of a table or view, the same privilege on columns of views based on that table or view and created under the authorization of the access are recursively revoked.

Revoking INSERT, REFERENCES or UPDATE access from one or more explicitly specified columns of a table or view will not affect access held on other column(s) of that table or view. If the original access was granted on the entire table or view, the access will stay in effect at the table level and will, therefore, apply to any new columns added to the table.

When the last instance of the required access held by the creator of a routine or trigger on a table is revoked, any routines or triggers created by that ident which contain references to the table will be dropped.

When the last instance of a privilege WITH GRANT OPTION is revoked, all instances of the privilege granted by the ident under that authorization are recursively revoked.

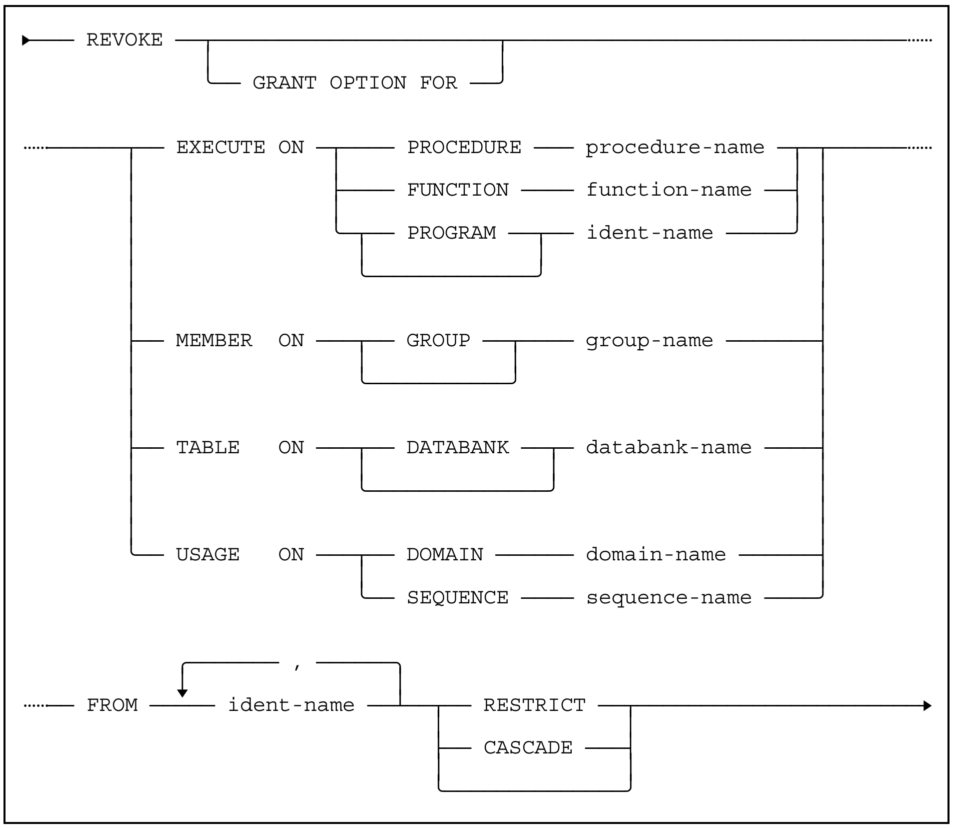
An ident may not revoke access privileges from himself.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	YES	Use of the keyword PRIVILEGES after the keyword ALL is optional in Mimer SQL, not mandatory. Specification of CASCADE or RESTRICT is optional in Mimer SQL, not mandatory.

REVOKE OBJECT PRIVILEGE

Revokes object privileges from one or more idents.



Usage

Embedded/Interactive/ODBC.

Description

The specified object privilege is revoked from the ident(s) listed. If the privilege is revoked from a GROUP ident, all members of the group lose the privilege.

The object privileges are described under GRANT OBJECT PRIVILEGE.

The GRANT OPTION FOR clause specifies that only the WITH GRANT OPTION is to be revoked from the specified instance(s) of the privilege(s).

The keywords CASCADE and RESTRICT specifies whether the REVOKE statement will allow recursive effects that causes views to be dropped or FOREIGN KEY constraints to be removed, because access privileges are revoked as a result of a REVOKE MEMBER statement. See the ***Notes*** section for REVOKE ACCESS PRIVILEGE for a description of when views are dropped and FOREIGN KEY constraints are removed due to recursive effects. If CASCADE is specified, such recursive effects will be allowed. If RESTRICT is specified, the REVOKE statement will return an error if it would cause such recursive effects, and no access privileges will be revoked.

If neither CASCADE nor RESTRICT is specified, then RESTRICT is implicit.

Restrictions

Privileges may only be explicitly revoked by the grantor.

Notes

If an object privilege has been granted to the same ident more than once (by different grantors), the REVOKE statement will only revoke (or will revoke the WITH GRANT OPTION from) the single instance of the privilege that was granted by the current ident.

The object rights attached to the privilege (or the WITH GRANT OPTION) will only be lost when the **last** instance of the privilege has been revoked.

Revoking object privileges has recursive effects. Privileged actions are performed under the authorization of the most recently granted instance of the access.

When the last instance of a privilege WITH GRANT OPTION is revoked, all instances of the privilege granted by the ident under that authorization are recursively revoked.

If MEMBER privilege on a group is revoked from an ident, all privileges granted through the group are revoked from the ident.

An ident may not revoke privileges from himself.

Revoking TABLE privilege does not drop the tables created when the privilege was held.

Revoking USAGE privilege on a domain or sequence preserves the uses of the domain or sequence which were set up when the privilege was held, however, new instances of usage of the domain or sequence are prevented.

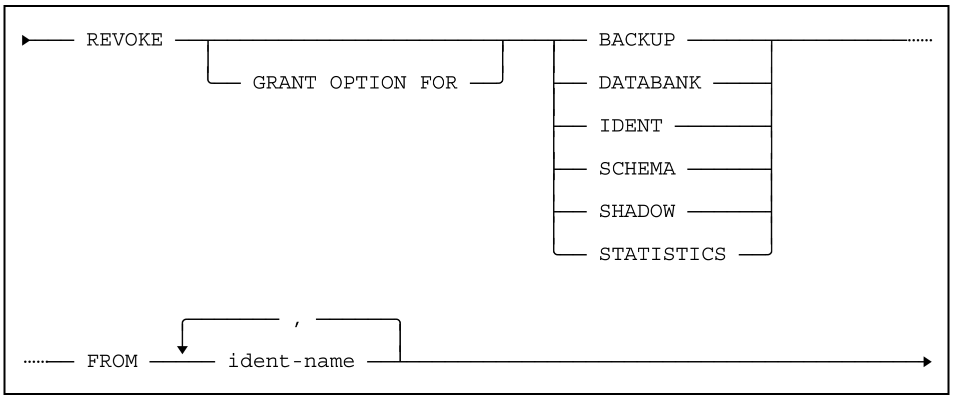
Revoking EXECUTE privilege immediately prevents the ident from invoking the routine or entering the program ident.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 **SQL92**	EXTENDED	Support for EXECUTE privilege on a PROGRAM ident is a Mimer SQL extension. Support for MEMBER privilege on a GROUP ident is a Mimer SQL extension. Support for TABLE privilege on a DATABANK is a Mimer SQL extension. Support for USAGE privilege on a SEQUENCE is a Mimer SQL extension.

REVOKE SYSTEM PRIVILEGE

Revokes system privileges from one or more idents.



Usage

Embedded/Interactive/ODBC.

Description

The specified system privilege is revoked from the ident(s) listed. If the privilege is revoked from a GROUP ident, all members of the group lose the privilege.

The system privileges are described under GRANT SYSTEM PRIVILEGE.

The GRANT OPTION FOR clause specifies that only the WITH GRANT OPTION is to be revoked from the specified instance(s) of the privilege(s).

Restrictions

Privileges can only be revoked explicitly by the grantor.

Notes

If a system privilege has been granted to the same ident more than once (by different grantors), the REVOKE statement will only revoke (or will revoke the WITH GRANT OPTION from) the single instance of the privilege that was granted by the current ident.

The system rights attached to the privilege (or the WITH GRANT OPTION) will only be revoked when the **last** instance of the privilege has been revoked.

Revoking system privileges has recursive effects on instances of the privilege being granted to other idents by virtue of the WITH GRANT option.

When the last instance of a privilege WITH GRANT OPTION is revoked, all instances of the privilege granted by the ident under that authorization are recursively revoked.

Databanks, idents, shadows or schemas created while the privilege was held are **not** dropped when the privilege is revoked.

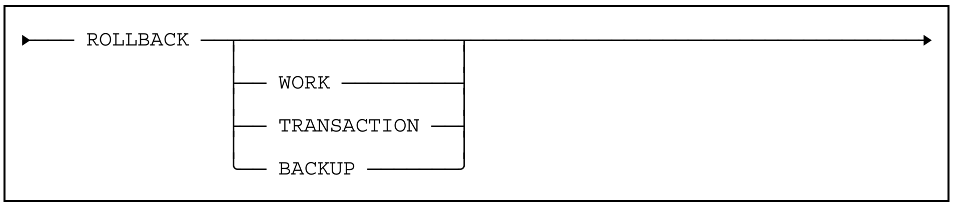
An ident may not revoke privileges from himself.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	MIMER EXTENSION	Support for system privileges is a Mimer SQL extension.

ROLLBACK

Aborts the current transaction.



Usage

Embedded/Interactive/Procedural.

Description

The current transaction is aborted. No database alterations requested in the transaction build-up are executed.

All cursors opened by the current ident are closed.

If there is no currently active transaction, any cursors opened by the current ident are closed, but the ROLLBACK statement is otherwise ignored. No error code is returned in this case.

When a BACKUP transaction is rolled back, all files created with CREATE BACKUP are deleted.

Restrictions

The ROLLBACK statement cannot be used in a **result set procedure**.

The ROLLBACK statement cannot be used within an **atomic** compound SQL statement (see COMPOUND STATEMENT).

The ROLLBACK BACKUP statement must be used to abort a BACKUP transaction.

The ROLLBACK BACKUP statement is not supported in procedural usage contexts.

Notes

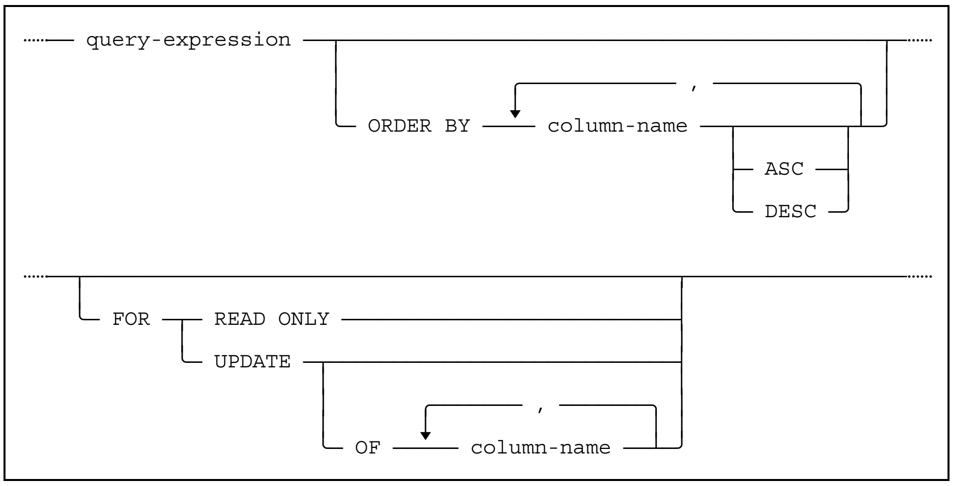
See Chapter 6 of the Mimer SQL Programmer's Manual for a more detailed discussion of transaction handling.

Standard compliance

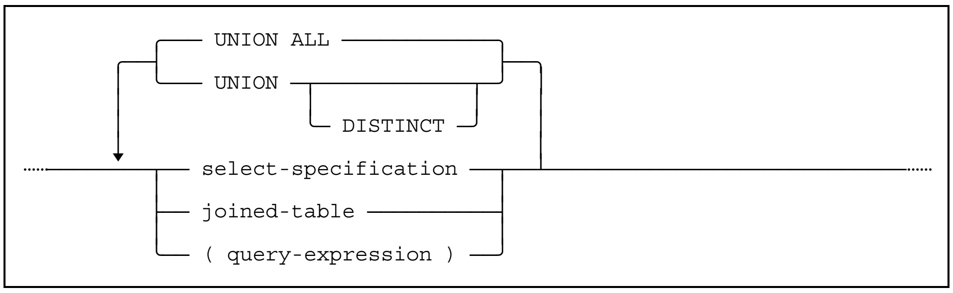
Standard	**Compliance**	**Comments**
X/Open-95 **SQL92**	EXTENDED	Support for the BACKUP and TRANSACTION keywords is a Mimer SQL extension.

SELECT

Retrieves data from the tables in the database.



where *query-expression* is



Usage

Embedded/Interactive/ODBC/Procedural.

In **Embedded** SQL, the SELECT statement may only be used to declare a cursor or as input to a PREPARE statement.

In a **Procedural** usage context, the SELECT statement may only be used to declare a cursor.

In **Interactive** SQL, the SELECT statement is used for interactive data retrieval. See the Mimer SQL User's Manual for more details.

Description

The ORDER BY clause

The final result table may be ordered according to an ORDER BY clause. Columns in the ORDER BY clause may be identified by a column label created with SELECT AS, or by a column name if this is unambiguous, within the final result table. The names in the first select specification are used in UNION constructions.

For each column in the ORDER BY clause, the sort order may be specified as ASC (ascending) - the default, or DESC (descending). If more than one column is specified, the result table is ordered first by values in the first specified column, then by values in the second, and so on.

The collating sequence for sorting purposes follows the ISO 8859-1 standard character set (shown in Appendix B).

The FOR UPDATE OF clause
If the SELECT statement defines a cursor for UPDATE CURRENT statements, a FOR UPDATE OF clause may be optionally used to list the columns to be updated. If the FOR UPDATE OF clause is used, it must include **all** the columns to be updated.

Each column specified in the FOR UPDATE CLAUSE must belong to the table or view named in the FROM clause of the SELECT statement, although the columns in FOR UPDATE OF do not need to be specified in the SELECT clause. No column may be named more than once in the FOR UPDATE OF clause.

Column names in the FOR UPDATE OF clause may not be qualified by the name of the table or view. They are implicitly qualified by the table reference in the FROM clause of the select specification.

FOR UPDATE OF may not be specified if the statement defines a read-only result set (see Section 5.13.1).

The UNION operator
If several SELECT statements are connected by UNION or UNION DISTINCT, the result is derived by first merging all result tables specified by the separate SELECT statements, and then eliminating duplicate rows from the merged set. All columns in the result table are significant for the purpose of eliminating duplicates.

The UNION ALL operator on the other hand retains all duplicates. The operator can be viewed as a way to concatenate several queries.

The rules described below apply to both UNION and UNION ALL.

All separate result tables from SELECT statements connected by UNION must have the same number of columns and the data types of columns to be merged must be compatible.

The columns in the result table are named in accordance with the columns in the first SELECT statement of the UNION construction.

Separate SELECT statements may be enclosed in parentheses if desired. This does not affect the result of a UNION operation.

See Section 4.7 for a description of how the data type of the UNION result is determined.

Restrictions

SELECT access is required on all tables and views specified in a FROM clause.

Notes

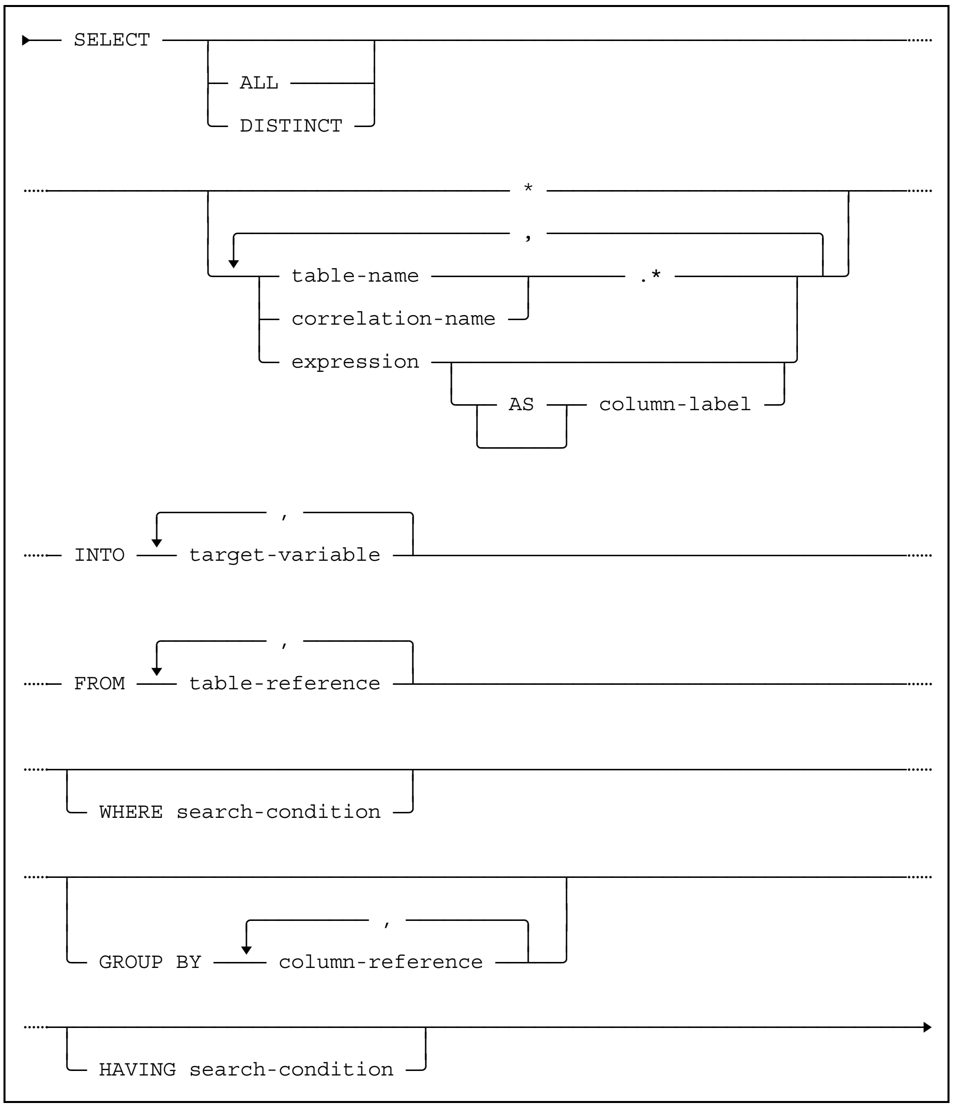
If the SELECT statement is used without the ORDER BY clause, the sort order is undefined. This means that the sort order may change if new indexes are created, indexes are dropped, new statistics are gathered or if a new version of the SQL optimizer is installed.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	YES	Fully compliant.

SELECT INTO

Selects a single-row result table and assigns the values directly to host variables.



Usage

Embedded/Procedural.

Description

Values defined by the SELECT, FROM and WHERE clauses are assigned to target variables as specified in the INTO clause. The value of the first element in the SELECT clause is assigned to the first variable, the value of the second element to the second variable, and so on. The data types of the variables must be assignment-compatible with those of the corresponding values.

The number of elements in the select-list must be the same as the number of elements in the target-variable list.

The result table defined by the SELECT INTO statement may not contain more than one row.

If a table reference or correlation name is used together with an asterisk in the SELECT clause, all columns are selected from the referred table. Columns listed explicitly in the SELECT clause need not be prefixed with the table or view name unless the same column name is used in more than one source table or view.

The whole list of values in the SELECT clause may be replaced by a single asterisk, in which case all columns from the table(s) or view(s) named in the FROM clause are selected, in the order in which they are defined in the source table(s) or view(s).

Note. Use of SELECT * is discouraged in embedded SQL programs (except in EXISTS predicates) since the asterisk is expanded to a column list when the statement is compiled, and any subsequent alterations in the table or view definitions may cause the program to function incorrectly.

When set functions are used in the list of values in the SELECT clause, their evaluation is influenced by the keywords ALL and DISTINCT. If ALL is specified, all rows in the result table are used in calculating the result of the set function. If DISTINCT is specified, duplicate rows are eliminated from the result table before the set function is evaluated. If no keyword is specified, ALL is assumed.

Language elements

expression	see Section 5.8.
search-condition	see Section 5.10.
target-variable	see Section 4.2.6.

Restrictions

SELECT access is required on all tables and views specified in the statement.

In a **procedural** usage context, the SELECT INTO statement is only permitted if the routine *access-clause* is READS SQL DATA or MODIFIES SQL DATA (see CREATE FUNCTION, CREATE PROCEDURE).

Notes

Correlation names used in the SELECT or WHERE clause must be defined in the FROM clause of the same SELECT INTO statement. The same correlation name may not be defined more than once in one FROM clause.

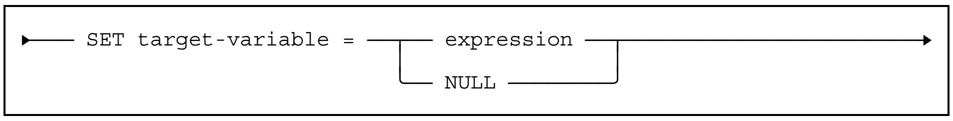
A SELECT INTO statement may include a GROUP BY or HAVING clause. However, care must be exercised to ensure that the HAVING clause selects one and only one group, and that the selected group either contains only one member or is reduced to a single row by a set function.

Standard compliance

Standard	Compliance	Comments
X/Open-95 SQL92	YES	Fully compliant.

SET

Assigns the specified value to a variable or output parameter.



Usage

Procedural/Interactive.

Description

The SET statement directly assigns the specified value-expression to a *target-variable* (see Section 4.2.6). Where the target of the assignment is a routine parameter, it must have mode OUT or INOUT.

Restrictions

The value-expression must be assignment-compatible with the data type of the *target-variable* (see Section 4.5).

Notes

Where the target of the assignment is a declared variable, its name may be qualified with a scope label (see Section 8.2.6 of the Mimer SQL Programmer's Manual).

If the target of the assignment is a variable declared with the ROW data type, a row value expression may be specified for *expression*.

If is possible to assign a value to a field of a variable declared with the ROW data type by using the following syntax to refer to the field: *routine-variable . field-name*.

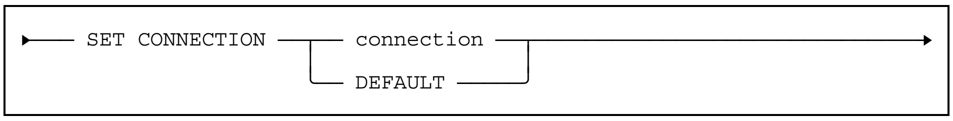
See Section 8.2.7 and Section 8.2.8 of the *Mimer SQL Programmer's Manual* for details of the ROW data type and row value expressions respectively.

Standard compliance

Standard	**Compliance**	**Comments**
SQL/PSM	YES	Fully compliant.

SET CONNECTION

Sets the current connection.



Usage

Embedded/Interactive.

Description

The specified connection becomes current. The connection must specify an existing connection name. If it does not, an error code is returned and the connection status remains unchanged.

The *connection* is case-sensitive.

The *connection* can be specified as a host variable or a literal.

Restrictions

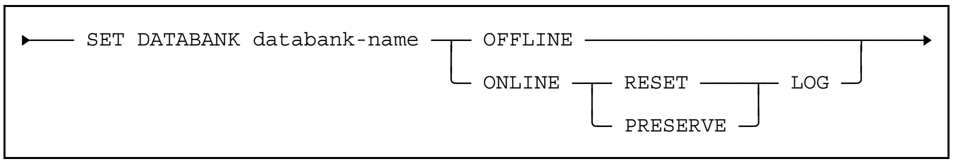
None.

Standard compliance

Standard	Compliance	Comments
X/Open-95 SQL92	YES	Fully compliant.

SET DATABANK

Sets a databank offline or online, with the option of clearing LOGDB records for it.



Usage

Embedded/Interactive/ODBC.

Description

Setting a databank OFFLINE makes it unavailable for all users and closes the databank file. A typical use for this is when taking databank backups using the host file system.

If the databank is being used, an error will be raised and it will not be set offline. When a databank is set offline, all online shadows will be brought up to date.

When a databank is set ONLINE again, you must specify whether to clear the LOGDB records for it (using the RESET LOG option), or whether to preserve these (using the PRESERVE LOG option). The RESET LOG option should be used after a successful backup has been taken.

It is essential to keep LOGDB in a state consistent with the databank backups (see the Mimer SQL System Management Handbook for a discussion of issues relating to Backup & Restore). Clearing records from LOGDB is handled automatically when CREATE BACKUP and CREATE INCREMENTAL BACKUP are used to take databank backups.

Restrictions

The current ident must either be the creator of the databank or have BACKUP privilege.

Notes

While a databank is OFFLINE none of the tables stored in it are accessible and the updating of all its shadows is suspended. It is possible to use ALTER DATABANK and ALTER DATABANK RESTORE to change or recover a databank while it is OFFLINE.

If ALTER DATABANK was used to change the location of the databank file while the databank was OFFLINE, the SET DATABANK statement will verify that the new file contains a valid copy of the databank when the databank is set ONLINE again (it cannot, however, check that the contents of the file is up-to-date).

It is possible to use DROP DATABANK to drop an OFFLINE databank.

While a databank is OFFLINE, it is **not** possible to use ALTER SHADOW on any of its shadows.

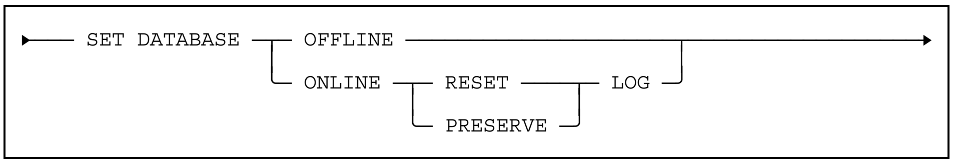
An error will be raised if an attempt is made to set a databank OFFLINE that is already OFFLINE, or ONLINE when it is already ONLINE.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	MIMER EXTENSION	Support for the SET DATABANK statement is Mimer SQL extension.

SET DATABASE

Sets the entire database offline or online, with the option of clearing all records from LOGDB.



Usage

Embedded/Interactive/ODBC.

Description

When the database is set OFFLINE, all **non-system** databanks are closed. During the process of setting the database OFFLINE, all updated databank pages are forced to disk and all databank shadows are brought up to date.

A typical use for this is when backing up all databank files in one go using the host file system utilities.

When the database is set ONLINE again, you must specify whether to clear **all** the LOGDB records (using the RESET LOG option), or whether to preserve these (using the PRESERVE LOG option).

The RESET LOG option should be used only after a complete backup has been taken of **everything** in the database.

Restrictions

The current ident must have BACKUP privilege.

Notes

While the database is OFFLINE no connections to it can be established, the database can only be accessed by a **single** system administrator ident.

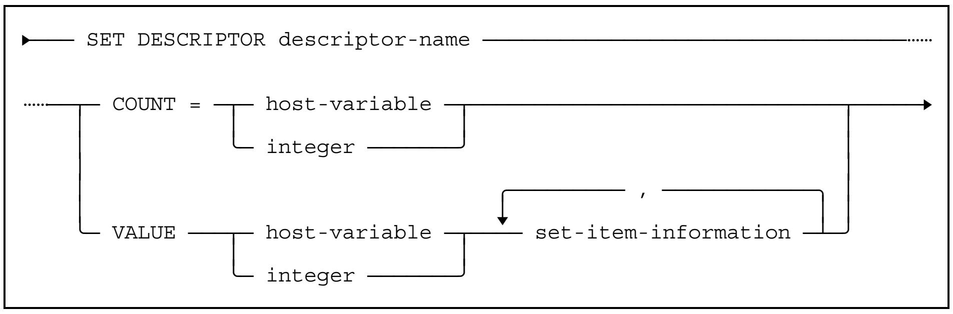
An error will be raised if an attempt is made to set the database OFFLINE when it is already OFFLINE, or ONLINE when it is already ONLINE.

Standard compliance

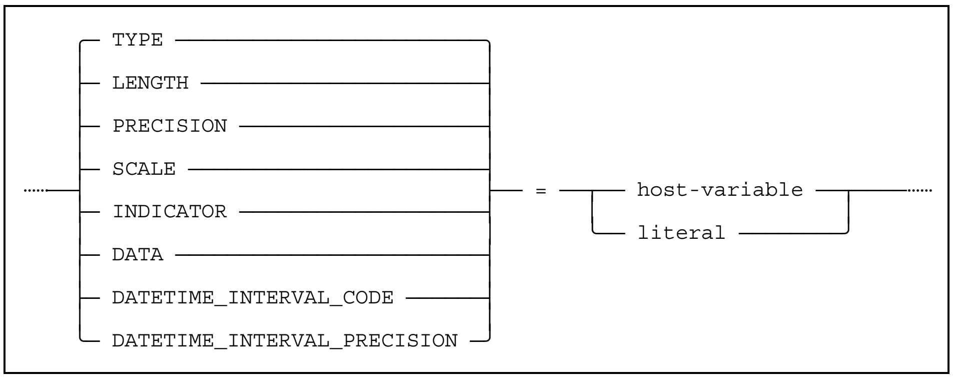
Standard	**Compliance**	**Comments**
X/Open-95 **SQL92**	MIMER EXTENSION	Support for the SET DATABASE statement is Mimer SQL extension.

SET DESCRIPTOR

Set values in an SQL descriptor area.



where *set-item-information* is:



Usage

Embedded.

Description

Fields values are assigned in the specified SQL descriptor area. The SET DESCRIPTOR statement can be used in two forms. The COUNT form sets the number of active item descriptor areas for the specified SQL descriptor. The VALUE form assigns SQL descriptor field values for the item descriptor area specified by *item-number*.

The *descriptor-name* is identified by a host variable or a literal.

See GET DESCRIPTOR for a description of the descriptor fields.

Restrictions

None.

Notes

The data type of the host variables must be compatible with the data type of the associated field name.

If an item descriptor area is specified for any field other than DATA, the DATA field becomes undefined.

When the TYPE field is set, some of the other fields are implicitly set according to the table below:

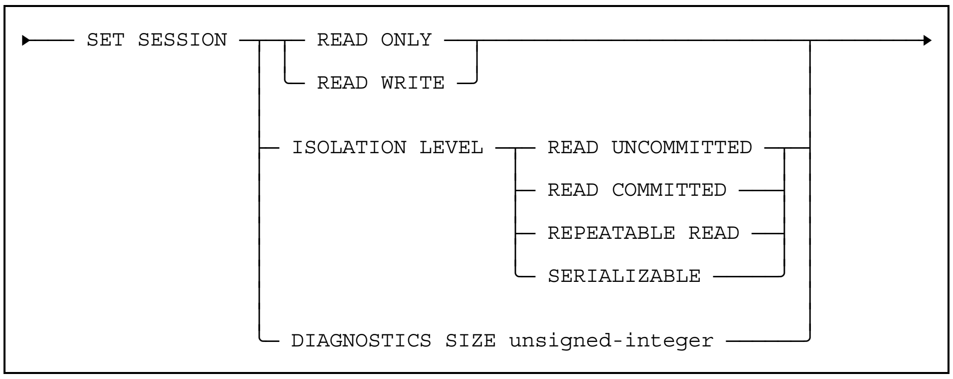
TYPE	**Implicitly set fields**
BINARY	LENGTH set to 1
BINARY VARYING	LENGTH set to 1
CHARACTER	LENGTH set to 1
CHARACTER VARYING	LENGTH set to 1
DATETIME	PRECISION set to 0
DECIMAL, NUMERIC	PRECISION set to 15 SCALE set to 0
FLOAT, DOUBLE PRECISION	PRECISION set to 16
INTEGER	PRECISION set to 10
INTERVAL	DATETIME_INTERVAL_PRECISION set to 2
REAL	PRECISION set to 7
SMALLINT	PRECISION set to 5

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 **SQL92**	YES	Fully compliant.

SET SESSION

Sets the default transaction modes for transactions.



Usage

Embedded/Interactive/Procedural.

Description

SET SESSION READ
The SET SESSION READ option allows the default SET TRANSACTION READ setting to be defined.

The SET TRANSACTION READ statement only affects the **single next** transaction to be started after it has been used.

The default SET TRANSACTION READ setting is normally READ WRITE, however, SET SESSION READ can be used to set whichever default is desired for the current session.

SET SESSION ISOLATION LEVEL
The SET SESSION ISOLATION LEVEL option allows the default SET TRANSACTION ISOLATION LEVEL setting to be defined.

The SET TRANSACTION ISOLATION LEVEL statement only affects the **single next** transaction to be started after it has been used.

The default SET TRANSACTION ISOLATION LEVEL setting is normally REPEATABLE READ, however, SET SESSION ISOLATION LEVEL can be used to set whichever default is desired for the current session.

SET SESSION DIAGNOSTICS SIZE
The SET SESSION DIAGNOSTICS SIZE option allows the default size of the diagnostics area to be defined. The *unsigned-integer* value specifies how many exceptions can be stacked in the diagnostics area, and examined by GET DIAGNOSTICS, in situations where repeated RESIGNAL operations have effectively been performed. The default size is 50.

Restrictions

The SET SESSION statement may not be issued within a transaction.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 **SQL92**	MIMER EXTENSION	The SET SESSION statement is a Mimer SQL extension.

SET SHADOW

Sets a list of databank shadows offline or online, with the option of clearing the LOGDB records for them.

```
►── SET SHADOW shadow-list ─┬─ OFFLINE ──────────────────────────┬──►
                            └─ ONLINE ─┬─ RESET ─────┬─ LOG ─┘
                                       └─ PRESERVE ─┘
```

Usage

Embedded/Interactive/ODBC.

Description

Setting a databank shadow OFFLINE suspends updating of it. A typical use for this is when taking databank backups from shadows using the host file system.

When a databank shadow is set ONLINE again, you must specify whether to clear the applicable LOGDB records (using the RESET LOG option), or whether to preserve these (using the PRESERVE LOG option). The RESET LOG option should be used after a successful backup has been taken.

Clearing records from LOGDB is handled automatically when CREATE BACKUP and CREATE INCREMENTAL BACKUP are used to take databank backups.

Restrictions

The current ident must either be the creator of the databank or have BACKUP privilege in order to use all the SET SHADOW options.

If the current ident holds SHADOW privilege, the shadow can be set offline and online with the PRESERVE LOG option, but the RESET LOG option cannot be used.

SET SHADOW cannot be used if the master databank is OFFLINE.

Notes

While a shadow is OFFLINE, updating of it is suspended. It is possible to use ALTER SHADOW to change the shadow while it is OFFLINE.

If ALTER SHADOW was used to change the location of the shadow file while the shadow was OFFLINE, the SET SHADOW statement will verify that the new file contains a valid copy of the shadow when the shadow is set ONLINE again (it cannot, however, check that the contents of the file is up-to-date).

It is possible to use DROP SHADOW to drop an OFFLINE shadow.

SET SHADOW OFFLINE will succeed with a warning if the shadow exists and is ONLINE, but the file cannot be accessed for some reason.

It is **not** possible to set more than a single shadow of any given databank OFFLINE at a time. If the *shadow-list* contains more than one shadow for a databank, **none** of the shadows for that databank will be set OFFLINE, and an error will be raised.

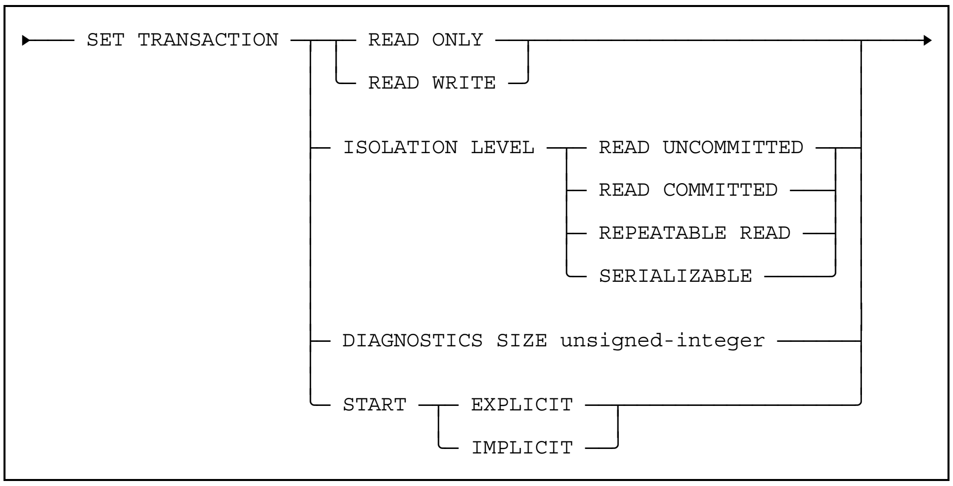
An error will be raised if an attempt is made to set a shadow OFFLINE that is already OFFLINE, or ONLINE when it is already ONLINE.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	MIMER EXTENSION	The SET SHADOW statement is a Mimer SQL extension.

SET TRANSACTION

Sets transaction modes for transactions.



Usage

Embedded/Interactive/Procedural.

Description

SET TRANSACTION READ

The SET TRANSACTION READ setting only affects the **single next** transaction to be started.

The default SET TRANSACTION READ setting (READ WRITE or whatever has been defined to be the default by using SET SESSION) applies unless an alternative is explicitly set before **each** transaction.

The SET TRANSACTION READ ONLY option is provided so that transaction performance can be optimized for those transactions not performing any **updates**.

It is strongly recommended that SET TRANSACTION READ ONLY be used for each transaction that does not require update access to the database and that READ WRITE mode only be used for transactions actually performing updates.

Significant performance gains can be achieved, especially for queries retrieving large numbers of rows, when the SET TRANSACTION READ options are used as recommended.

SET TRANSACTION ISOLATION LEVEL

The SET TRANSACTION ISOLATION LEVEL options are provided to control the degree to which the updates performed by a transaction are affected by the updates performed by concurrent transactions.

The SET TRANSACTION ISOLATION LEVEL setting only affects the **single next** transaction to be started.

The default SET TRANSACTION ISOLATION LEVEL setting (REPEATABLE READ or whatever has been defined to be the default by using SET SESSION) applies unless an alternative is explicitly set before **each** transaction.

All of the isolation levels guarantee that each transaction will be executed completely or not at all and that no updates will be lost.

The execution of concurrent transactions at the most secure isolation level, SERIALIZABLE, guarantees that the execution of the operations of concurrently executing transactions produces the same effect as some serial execution of those same transactions (i.e. an execution where one transaction executes to completion before the next begins).

When the other isolation levels are in effect (READ UNCOMMITTED, READ COMMITTED and REPEATABLE READ), the following effects may occur during the execution of concurrent transactions:

1. "Dirty Read" - this is where uncommitted updates can be read by another transaction. This can lead to a situation, in the event of a rollback occurring in an update transaction after another transaction has performed a read, where data has been read which (because it was never committed) must be considered to have never existed.

2. "Non-repeatable Read" - this is where a transaction reads a row and then another transaction updates or deletes that specific row. A subsequent attempt to re-read the same specific row retrieves modified information or finds that the row no longer exists, thus it can be said that the original read cannot be repeated.

3. "Phantoms" - this is where a transaction reads a set of rows that satisfy some search condition. Another transaction then performs an update which generates one or more new rows that satisfy the search condition. If the original query is repeated (using the same search condition), extra rows appear in the result-set that where previously not found.

The following table summarizes, for each of the four isolation levels, which of the affects described above are guaranteed never to occur, or must be accepted as possible, where there are concurrent transactions:

Isolation Level	**1. Dirty Read**	**2. Non-repeatable Read**	**3. Phantoms**
READ UNCOMMITTED	POSSIBLE	POSSIBLE	POSSIBLE
READ COMMITTED	NEVER OCCURS	POSSIBLE	POSSIBLE
REPEATABLE READ	NEVER OCCURS	NEVER OCCURS	POSSIBLE
SERIALIZABLE	NEVER OCCURS	NEVER OCCURS	NEVER OCCURS

SET TRANSACTION DIAGNOSTICS SIZE
The SET TRANSACTION DIAGNOSTICS SIZE option allows the size of the diagnostics area to be defined. The *unsigned-integer* value specifies how many exceptions can be stacked in the diagnostics area, and examined by GET DIAGNOSTICS, in situations where repeated RESIGNAL operations have effectively been performed.

The SET TRANSACTION DIAGNOSTICS SIZE setting only affects the **single next** transaction to be started.

The default SET TRANSACTION DIAGNOSTICS SIZE setting (50 or whatever has been defined to be the default by using SET SESSION) applies unless an alternative is explicitly set before **each** transaction.

SET TRANSACTION START
Transactions are started either by an explicit START statement or by an implicit transaction start. The procedure that is followed is determined by using the SET TRANSACTION START statement.

When START is set to IMPLICIT, the first operation involving a databank with either the TRANS or LOG option will start a transaction. The transaction must then be terminated explicitly by either COMMIT or ROLLBACK.

The SET TRANSACTION START setting has effect in the current session until SET TRANSACTION START is next used.

The default setting is START IMPLICIT.

Restrictions

The SET TRANSACTION statement may not be issued within a transaction.

Notes

The SET TRANSACTION START statement is generally issued at the beginning of a session, to set the start mode for transactions. Changing the start mode for transactions in the middle of a session is not generally recommended.

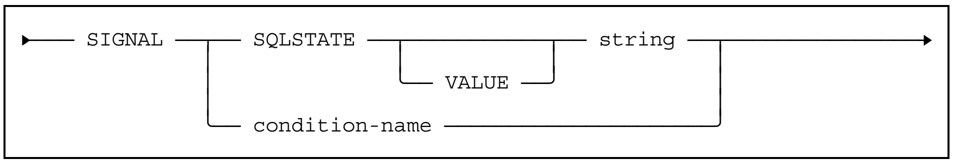
The SET SESSION statement can be used to define the default settings for the SET TRANSACTION READ, SET TRANSACTION ISOLATION LEVEL and SET TRANSACTION DIAGNOSTICS SIZE options.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 **SQL92**	EXTENDED	Support for SET TRANSACTION START is a Mimer SQL extension.

SIGNAL

Raises the specified exception condition.



Usage

Procedural.

Description

The SIGNAL statement has the effect of raising an exception condition specified by an SQLSTATE value or a condition name.

Restrictions

None.

Notes

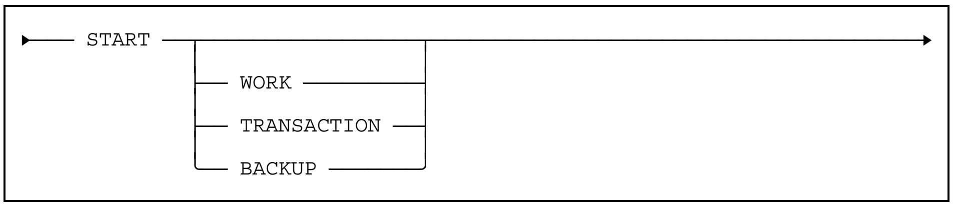
See DECLARE CONDITION for a description of how to declare a condition name.

Standard compliance

Standard	**Compliance**	**Comments**
SQL/PSM	YES	Fully compliant.

START

Starts a transaction build-up.



Usage

Embedded/Interactive/Procedural.

Description

A new transaction is started, regardless of whether the transaction start mode is set to IMPLICIT or EXPLICIT (see the SET TRANSACTION statement).

The START BACKUP command starts a transaction in which the CREATE ONLINE BACKUP statements of an online backup sequence are executed, see the description of CREATE BACKUP (Online) for more information.

Restrictions

The START statement may not be executed from within a transaction.

The START statement may not be used in a **result set procedure**.

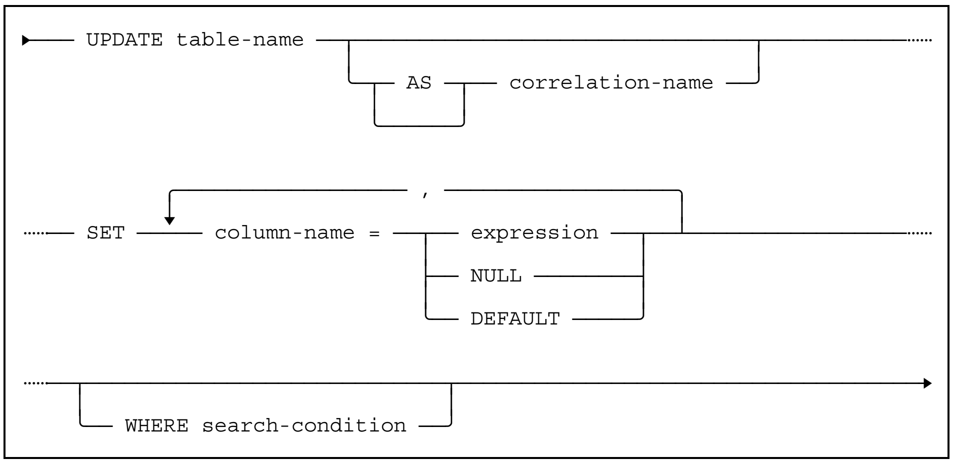
The COMMIT BACKUP command is not supported in procedural mode.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 **SQL92**	MIMER EXTENSION	Support for the START statement is a Mimer SQL extension.

UPDATE

Updates a set of rows in a table or view.



Usage

Embedded/Interactive/Procedural.

Description

The table or view identified by the table name is updated in the rows which satisfy the condition in the WHERE clause by assigning new values to the columns as specified in the SET clause. If no WHERE clause is specified, all rows are updated.

Values to be assigned to columns may be specified either as expressions or by using the keywords NULL or DEFAULT. Expressions must have a data type compatible with the definition of the column to which they are assigned. If column names are used in expressions, they must refer to columns in the table or view addressed in the UPDATE clause. The value specified by a column name in an expression is the value for the column in the row concerned before any update operation is performed.

If no row is updated a NOT FOUND condition code is returned (see Appendix E).

Language elements

expression	see Section 5.8
search-condition	see Section 5.10.

Restrictions

UPDATE access is required on the columns specified in the SET clause.

If the UPDATE statement is used on a primary key column of a table, the table must be stored in a databank with the TRANS or LOG option.

In a **procedural** usage context, the UPDATE statement is only permitted if the procedure *access-clause* is MODIFIES SQL DATA (see CREATE PROCEDURE).

Notes

Column names on the left-hand side of the assignment operator in the SET clause may not be qualified by the table reference.

Columns may not be specified more than once on the left-hand side of the assignment operator in the SET clause in a single UPDATE statement.

Expressions used in the SET clause cannot refer to set functions.

UNIQUE constraints in the table being updated may not be violated (this is evaluated **at the end** when all the modifications involved in the UPDATE statement have been made).

Column names in the search condition of the WHERE clause must identify columns in the table or view to be updated.

If a correlation name is introduced after the table reference in the UPDATE clause, the correlation name must be used to refer to the table in the WHERE clause of the same UPDATE statement.

If the table name specified in the UPDATE statement is subject to any referential constraint, the values in all updated rows must conform to that constraint. If a view defined WITH CHECK OPTION is to be updated, the values assigned to the columns must conform to the view definition.

Read-only views may not be updated - see CREATE VIEW.

An UPDATE statement is executed as a single statement. If an error occurs at any point during the execution, no rows will be updated (however, if the table is stored in a databank with the NULL option it is possible that some rows will be updated).

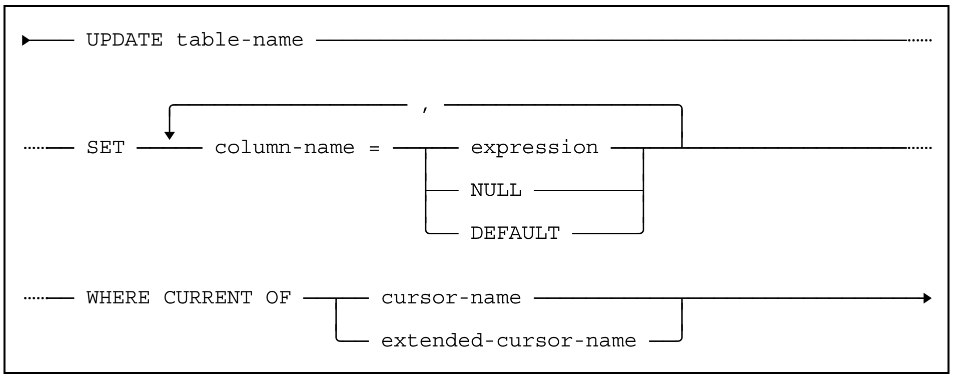
Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 **SQL92**	EXTENDED	Support for the AS *correlation-name* construct is a Mimer SQL extension.

Note: See also standard compliance table for assignment operations, in Section 4.5.4.

UPDATE CURRENT

Updates the current row indicated by a cursor.



Usage

Embedded/ODBC/Procedural.

Description

The current row addressed by the cursor is updated by assigning new values to the columns as specified in the SET clause.

See ALLOCATE CURSOR for a description of extended cursors.

If an extended cursor is used in a UPDATE CURRENT statement, the cursor is represented following these rules:

- If the UPDATE CURRENT statement is executed with static SQL, i.e. using EXEC SQL, the extended cursor is represented by the host variable containing the cursor.
- If the UPDATE CURRENT statement is executed with dynamic SQL, the extended cursor must be represented by the cursor value contained in the host variable.

Values to be assigned to columns may be specified either as expressions or by using the keywords NULL or DEFAULT. Expressions must have a data type compatible with the definition of the column to which they are assigned. If column names are used in expressions, they must refer to columns in the table or view addressed in the UPDATE CURRENT clause. The value specified by a column name in an expression is the value for the column in the row concerned before the update operation is performed.

Language elements

expression	see Section 5.8
search-condition	see Section 5.10.

Restrictions

UPDATE access to the appropriate columns in the table or view identified by the table name is required when the cursor used for the UPDATE CURRENT statement is opened. If UPDATE access is not held, the cursor may be opened but UPDATE CURRENT statements will fail. Direct access to the base table is not required for an update operation on a view.

If the UPDATE CURRENT statement is used on a primary key column of a table, the table must be stored in a databank with the TRANS or LOG option.

In a **procedural** usage context, *extended-cursor-name* cannot be used to identify the cursor.

In a **procedural** usage context, the UPDATE CURRENT statement is only permitted if the procedure *access-clause* is MODIFIES SQL DATA (see CREATE PROCEDURE).

Notes

Column names on the left-hand side of the assignment operator in the SET clause may not be qualified by the table name.

Columns may not be specified more than once on the left-hand side of the assignment operator in the SET clause in a single UPDATE statement.

Expressions used in the SET clause cannot refer to set functions.

UNIQUE constraints in the table being updated may not be violated.

If columns are listed in the FOR UPDATE OF clause of the cursor definition (described under SELECT) they must match.

The table name specified in the UPDATE CURRENT clause must be exactly the same as that in the FROM clause of the SELECT statement used to declare the cursor. If a synonym is used in one of the statements, the same synonym must also be used in the other.

If the table name specified in the UPDATE statement is subject to any referential constraint, the values in the row to be updated must conform to that constraint. If a view defined WITH CHECK OPTION is to be updated, the values assigned to the columns must conform to the view definition.

The UPDATE CURRENT statement may not be used for read-only cursors.

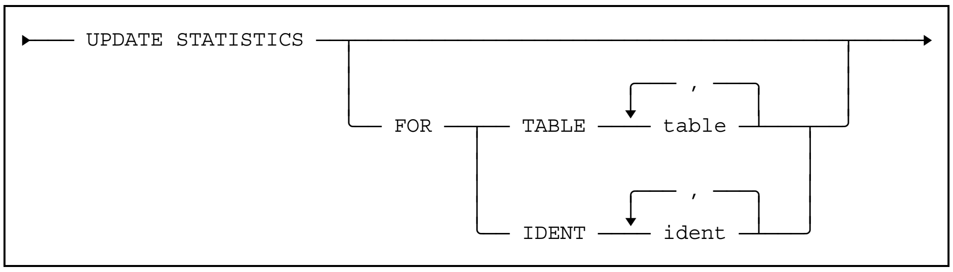
Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 **SQL92**	YES	Fully compliant.

Note: See also standard compliance table for assignment operations, in Section 4.5.4.

UPDATE STATISTICS

Updates the statistics recorded for all tables in the database, a specified list of tables or all the tables belonging to the schemas owned by a specified list of idents.



Usage

Embedded/Interactive/ODBC.

Description

The **default** operation is to update statistics for **all** tables, including data dictionary tables, in the database.

It is possible to update statistics for a specified list of tables by using the FOR TABLE option or for all the tables belonging to the schemas created by a specified list of idents by using the FOR IDENT option.

Update statistics includes an automatic operation which ensures the consistency of secondary indexes (both explicitly created indexes and those created by the system when certain constraints are defined). The operation is transparent to users of the database and is performed on indexes selected by the UPDATE STATISTICS statement that are contained in a databank with the TRANS or LOG option and which are flagged as “not consistent”.

The process of ensuring the consistency of an index, and updating statistics for all tables in the database (the default operation), can be rather time-consuming. Therefore, it is generally recommended that these operations be performed in batch mode and at off-peak times (refer to the Mimer SQL System Management Handbook for more information on Database Statistics).

A secondary index is flagged as “not consistent” if it is contained in a databank with the NULL option or if the databank containing it has been upgraded from Mimer SQL version 7 or 8.1.

The IS_CONSISTENT column in the data dictionary table TABLE_CONSTRAINTS shows which indexes in the database are flagged as “not consistent”.

Restrictions

The current ident must be the creator of all the tables involved or must have STATISTICS privilege.

Notes

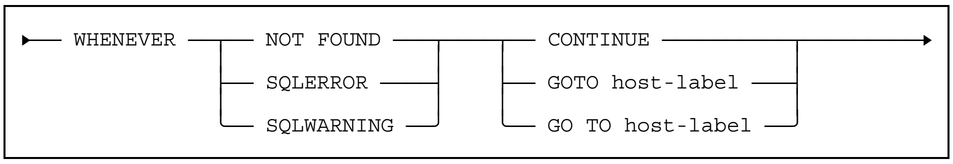
The UPDATE STATISTICS statement can be used concurrently with other SQL statements.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 **SQL92**	MIMER EXTENSION	The UPDATE STATISTICS statement is a Mimer SQL extension.

WHENEVER

Defines action to be taken for errors and exception conditions.



Usage

Embedded.

Description

The action taken in the event of a condition arising during execution of an SQL statement is governed by the most recently issued WHENEVER statement. There are three different types of conditions: NOT FOUND, SQLERROR and SQLWARNING. See Appendix E for a description of the different condition types.

The action taken is as follows:

CONTINUE	Program execution continues at the next sequential statement of the source program.
GOTO	Program execution continues at the source code statement identified by *host-label*, where *host-label* is a program label in a program written according to the host language.

Restrictions

None.

Notes

If a condition in an SQL statement is not covered by an explicit WHENEVER statement issued earlier in the host program code, CONTINUE will be assumed.

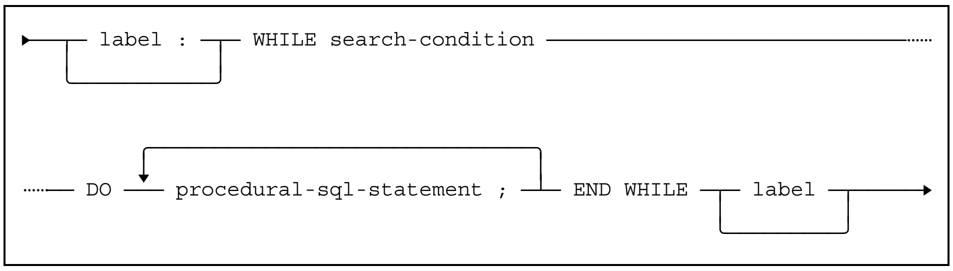
See Chapter 10 of the Mimer SQL Programmer's Manual for a discussion of the use of WHENEVER in control of different conditions.

Standard compliance

Standard	**Compliance**	**Comments**
X/Open-95 SQL92	EXTENDED	Support for the SQLWARNING keyword is a Mimer SQL extension.

WHILE

Allows one or more procedural SQL statements to be iteratively executed.



Usage

Procedural.

Description

The WHILE statement can be used to iteratively execute a sequence of one or more *procedural-sql-statement*'s.

The iteration continues **as long as** *search-condition* evaluates to **true**.

Restrictions

If *label* appears at the beginning **and** at the end of the WHILE-statement, the same value must be specified in both places.

Specifying *label* is optional, however, if *label* appears at the end of the WHILE-statement, it **must** also appear at the beginning.

A label is required at the beginning if the LEAVE-statement is to be used to terminate the WHILE-statement.

Notes

The WHILE-statement may be terminated by executing the LEAVE-statement using *label*. It will also terminate if an exception condition is raised, in accordance with the normal exception handling process.

Standard compliance

Standard	**Compliance**	**Comments**
SQL/PSM	YES	Fully compliant.

7 DATA DICTIONARY VIEWS

This chapter documents the predefined system views on the data dictionary tables. PUBLIC holds SELECT access on these views, so that they may be examined by any user.

INFORMATION_SCHEMA views are used to retrieve information about the objects in the data dictionary.

INFO_SCHEM is an older version of this with shorter names, supported for backward compatibility.

FIPS_DOCUMENTATION views contain information about SQL standards compliance features and Mimer SQL limits.

An INFORMATION_SCHEMA view can be read with the statement (note the qualified form of *view-name*):

```
SELECT column-list
FROM   INFORMATION_SCHEMA.view-name
WHERE  condition
```

Many of the system views include only objects, privileges and so on relevant to the current ident (the description for each view indicates exactly what information is shown in the view).

Note: Some of the system views have columns designed to display information that is not currently supported by Mimer SQL (e.g. catalog names for database objects), in this situation the empty string (“”) will be shown in these columns.

The tables in the data dictionary may be read directly only by the system administrator ident SYSADM in the default installation. The base tables in the data dictionary are documented in the Mimer SQL System Management Handbook. The system administrator may, if desired, grant SELECT access on the dictionary tables to other users.

No user may access the data dictionary views or tables directly for any purpose other than SELECT. All data dictionary maintenance is performed by internal routines and is invisible to the user.

7.1 INFORMATION_SCHEMA

The table below summarizes the system views that are part of the schema INFORMATION_SCHEMA (view names appear in their unqualified form):

View name	Description
ASSERTIONS	Owned assertions.
CHARACTER_SETS	Accessible character sets.
CHECK_CONSTRAINTS	Owned check constraints.
COLLATIONS	Accessible collations.
COLUMN_DOMAIN_USAGE	Columns defined using to owned domains.
COLUMN_PRIVILEGES	Privileges granted on accessible table columns.
COLUMNS	Accessible table columns.
CONSTRAINT_COLUMN_USAGE	Columns referenced by owned referential, unique, check or assertion constraints.
CONSTRAINT_TABLE_USAGE	Tables on which owned referential, unique, check or assertion constraints are defined.
DOMAIN_CONSTRAINTS	Constraints of accessible domains.
DOMAINS	Accessible domains.
EXT_COLUMN_REMARKS	Comments for accessible table columns.
EXT_DATABANKS	Accessible databanks.
EXT_IDENTS	Accessible authorization idents.
EXT_INDEX_COLUMN_USAGE	Accessible table columns on which indexes depend.
EXT_INDEXES	Accessible indexes.
EXT_OBJECT_IDENT_USAGE	Accessible objects created by authorization ident.
EXT_OBJECT_OBJECT_USED	Accessible objects used by other objects
EXT_OBJECT_OBJECT_USING	Accessible objects using other objects
EXT_OBJECT_PRIVILEGES	Object privileges granted to an authorization ident.
EXT_ROUTINE_MODULE_DEFINITION	Source definition for routines defined in modules
EXT_ROUTINE_MODULE_USAGE	Accessible routines in a module
EXT_SEQUENCES	Accessible sequences.
EXT_SHADOW_DATABANK_USAGE	Owned databanks on which shadows depend.
EXT_SHADOWS	Accessible shadows.
EXT_SOURCE_DEFINITION	Text definition for owned objects.
EXT_STATISTICS	Statistics for table
EXT_SYNONYMS	Accessible synonyms.
EXT_SYSTEM_PRIVILEGES	System privileges granted to an authorization ident.
EXT_TABLE_DATABANK_USAGE	Owned databanks on which tables depend.
KEY_COLUMN_USAGE	Table columns constrained as keys by owned constraints.
MODULES	Owned modules.
PARAMETERS	Parameters of accessible routines.

REFERENTIAL_CONSTRAINTS	Owned referential constraints.
ROUTINE_COLUMN_USAGE	Owned table columns on which routines depend.
ROUTINE_PRIVILEGES	Privileges held on accessible routines.
ROUTINE_TABLE_USAGE	Owned tables on which routines depend.
ROUTINES	Accessible routines.
SCHEMATA	Owned schemas.
SQL_LANGUAGES	Conformance levels for supported SQL language options and dialects.
TABLE_CONSTRAINTS	Owned table constraints.
TABLE_PRIVILEGES	Privileges held on accessible tables.
TABLES	Accessible tables.
TRANSLATIONS	Accessible character set translations.
TRIGGERED_UPDATE_COLUMNS	Owned columns referenced from UPDATE trigger column lists.
TRIGGER_COLUMN_USAGE	Owned columns referenced from a trigger action.
TRIGGER_TABLE_USAGE	Tables on which owned triggers depend.
TRIGGERS	Owned triggers.
USAGE_PRIVILEGES	USAGE privilege held on accessible objects.
VIEW_COLUMN_USAGE	Columns on which owned views depend.
VIEW_TABLE_USAGE	Tables on which owned views depend.
VIEWS	Accessible views.

ASSERTIONS

The ASSERTIONS system view shows all assertions owned by the current ident.

Column name	Data type	Description
CONSTRAINT _CATALOG	VARCHAR(128)	The name of the catalog containing the assertion.
CONSTRAINT _SCHEMA	VARCHAR(128)	The name of the schema containing the assertion.
CONSTRAINT_NAME	VARCHAR(128)	The name of the assertion.
IS_DEFERRABLE	VARCHAR(3)	One of: “YES” = the assertion is deferrable “NO” = the assertion is not deferrable
INITIALLY_DEFERRED	VARCHAR(3)	One of: “YES” = the assertion is immediate “NO” = the assertion is deferred.

CHARACTER_SETS

The CHARACTER_SETS system view describes each character set to which the current ident, or PUBLIC, has USAGE privilege.

Column name	Data type	Description
CHARACTER_SET _CATALOG	VARCHAR(128)	The name of the catalog containing the character set.
CHARACTER_SET _SCHEMA	VARCHAR(128)	The name of the schema containing the character set.
CHARACTER_SET _NAME	VARCHAR(128)	The name of the character set.
FORM_OF_USE	VARCHAR(128)	A user-defined name that indicates the form-of-use of the character set.
NUMBER_OF _CHARACTERS	INTEGER	The number of characters in the character set.
DEFAULT_COLLATE _CATALOG	VARCHAR(128)	The name of the catalog containing the default collation for the character set.
DEFAULT_COLLATE _SCHEMA	VARCHAR(128)	The name of the schema containing the default collation for the character set.
DEFAULT_COLLATE _NAME	VARCHAR(128)	The name of the default collation for the character set.

CHECK_CONSTRAINTS

The CHECK_CONSTRAINTS system view lists the check constraints that are owned by the current ident.

Column name	Data type	Description
CONSTRAINT _CATALOG	VARCHAR(128)	The name of the catalog containing the check constraint.
CONSTRAINT_SCHEMA	VARCHAR(128)	The name of the schema containing the check constraint.
CONSTRAINT_NAME	VARCHAR(128)	The name of the check constraint.
CHECK_CLAUSE	VARCHAR(2000)	The character representation of the search condition used in the check clause.

COLLATIONS

The COLLATIONS system view describes each collation to which the current ident, or PUBLIC, has access.

Column name	Data type	Description
COLLATION_CATALOG	VARCHAR(128)	The name of the catalog containing the collation.
COLLATION_SCHEMA	VARCHAR(128)	The name of the schema containing the collation.
COLLATION_NAME	VARCHAR(128)	Name of the collation.
CHARACTER_SET _CATALOG	VARCHAR(128)	The name of the catalog containing the character set on which the collation is defined.
CHARACTER_SET _SCHEMA	VARCHAR(128)	The name of the schema containing the character set on which the collation is defined.
CHARACTER_SET _NAME	VARCHAR(128)	The name of the character set on which the collation is defined.
PAD_ATTRIBUTE	VARCHAR(20)	One of the following values: “NO PAD” = the collation has the *no pad* attribute “PAD SPACE” = the collation has the *pad space* attribute.

COLUMN_DOMAIN_USAGE

The COLUMN_DOMAIN_USAGE system view lists the table columns which depend on domains owned by the current ident.

Column name	Data type	Description
DOMAIN_CATALOG	VARCHAR(128)	The name of the catalog containing the domain.
DOMAIN_SCHEMA	VARCHAR(128)	The name of the schema containing the domain.
DOMAIN_NAME	VARCHAR(128)	The name of the domain.
TABLE_CATALOG	VARCHAR(128)	The name of the catalog containing the table.
TABLE_SCHEMA	VARCHAR(128)	The name of the schema containing the table.
TABLE_NAME	VARCHAR(128)	The name of the table.
COLUMN_NAME	VARCHAR(128)	The name of the column.

COLUMN_PRIVILEGES

The COLUMN_PRIVILEGES system view lists privileges on table columns that were granted by the current ident and privileges on table columns that were granted to the current ident or to PUBLIC.

Column name	Data type	Description
GRANTOR	VARCHAR(128)	The name of the ident who granted the privilege.
GRANTEE	VARCHAR(128)	The name of the ident to whom the privilege was granted. Granting a privilege to PUBLIC will result in only one row (per privilege granted) in this view and the name “PUBLIC” will be shown.
TABLE_CATALOG	VARCHAR(128)	The name of the catalog containing the table.
TABLE_SCHEMA	VARCHAR(128)	The name of the schema containing the table.
TABLE_NAME	VARCHAR(128)	The name of the table containing the column on which the column privilege has been granted.
COLUMN_NAME	VARCHAR(128)	The name of the column on which the privilege has been granted.
PRIVILEGE_TYPE	VARCHAR(20)	A value describing the type of the column privilege that was granted. One of: “INSERT” “REFERENCES” “SELECT” “UPDATE”. Note that when multiple table column privileges are granted to the same user at the same time (e.g. when the keyword “ALL” is used), multiple rows appear in this view (one for each privilege granted).
IS_GRANTABLE	VARCHAR(3)	One of: “YES” = the privilege is held with the WITH GRANT OPTION “NO” = the privilege is held without the WITH GRANT OPTION.

COLUMNS

The COLUMNS system view lists the table columns to which the current ident, or PUBLIC, has access.

Column name	Data type	Description
TABLE_CATALOG	VARCHAR(128)	The name of the catalog containing the table or view.
TABLE_SCHEMA	VARCHAR(128)	The name of the schema containing the table or view.
TABLE_NAME	VARCHAR(128)	The name of the table or view.
COLUMN_NAME	VARCHAR(128)	The name of the column of the table or view.
ORDINAL_POSITION	INTEGER	The ordinal position of the column in the table. The first column in the table is number 1.
COLUMN_DEFAULT	VARCHAR(2000)	This shows the default value for the column. If the default value is a character string, the value shown is the string enclosed in single quotes. If the default value is a numeric literal, the value is shown in its original character representation without enclosing quotes. If the default value is a DATE, TIME or TIMESTAMP, the value shown is the appropriate keyword (e.g. DATE) followed by the literal representation of the value enclosed in single quotes (see Section 4.4.5 for a description of DATE, TIME and TIMESTAMP literals). If the default value is a pseudo-literal, the value shown is the appropriate keyword (e.g. CURRENT_DATE) without enclosing quotes. If the default value is the NULL value, the value shown is the keyword NULL without enclosing quotes. If the default value cannot be represented without truncation, then TRUNCATED is shown without enclosing quotes. If no default value was specified then its value is the NULL value. The value of COLUMN_DEF is syntactically suitable for use in specifying *default-value* in a CREATE TABLE or ALTER TABLE statement (except when TRUNCATED is shown).
IS_NULLABLE	VARCHAR(3)	One of: “NO” = the column is not nullable, according to the rules in the international standard “YES” = the NULL value is allowed in the column.

DATA_TYPE	VARCHAR(30)	Identifies the data type of the column. Can be one of the following: BIGINT BINARY BINARY VARYING CHARACTER CHARACTER VARYING DATE DECIMAL DOUBLE PRECISION FLOAT INTEGER INTERVAL NUMERIC REAL SMALLINT TIME TIMESTAMP.
CHARACTER _MAXIMUM_LENGTH	INTEGER	For a character data type, this shows the maximum length in characters. For all other data types it is the NULL value.
CHARACTER _OCTET_LENGTH	INTEGER	For a character data type, this shows the maximum length in octets. For all other data types it is the NULL value. (For single octet character sets, this is the same as CHARACTER_MAX_LENGTH).
NUMERIC_PRECISION	INTEGER	For numeric data types, this shows the total number of significant digits allowed in the column. For all other data types it is the NULL value.
NUMERIC_PRECISION _RADIX	INTEGER	This shows whether the NUMERIC_PRECISION is given in a binary or decimal radix. The numeric radix is always decimal in Mimer SQL, therefore the value 10 is always shown for numeric data types. For all other data types it is the NULL value.
NUMERIC_SCALE	INTEGER	This defines the total number of significant digits to the right of the decimal point. For BIGINT, INTEGER and SMALLINT, this is 0. For BINARY, BINARY VARYING, CHARACTER, CHARACTER VARYING, DATETIME, FLOAT, INTERVAL, REAL and DOUBLE PRECISION data types, it is the NULL value.
DATETIME_PRECISION	INTEGER	For DATE, TIME, TIMESTAMP and interval data types, this column contains the number of digits of precision for the fractional seconds component. For other data types it is the NULL value.

INTERVAL_TYPE	VARCHAR(30)	For interval data types, this is a character string specifying the interval qualifier. For other data types it is the NULL value. Can be one of: “YEAR” “YEAR TO MONTH” “DAY” “HOUR” “MINUTE” “SECOND” “DAY TO HOUR” “DAY TO MINUTE” “DAY TO SECOND” “HOUR TO MINUTE” “HOUR TO SECOND” “MINUTE TO SECOND”.
INTERVAL_PRECISION	INTEGER	For interval data types, this is the number of significant digits for the interval leading precision (see Section 4.3.3.2). For other data types it is the NULL value.
CHARACTER_SET _CATALOG	VARCHAR(128)	The name of the catalog containing the character set used by the column.
CHARACTER_SET _SCHEMA	VARCHAR(128)	The name of the schema containing the character set used by the column.
CHARACTER_SET _NAME	VARCHAR(128)	The name of the character set used by the column.
COLLATION_CATALOG	VARCHAR(128)	The name of the catalog containing the collation used by the column.
COLLATION_SCHEMA	VARCHAR(128)	The name of the schema containing the collation used by the column.
COLLATION_NAME	VARCHAR(128)	The name of the collation used by the column.
DOMAIN_CATALOG	VARCHAR(128)	The name of the catalog containing the domain used by the column.
DOMAIN_SCHEMA	VARCHAR(128)	The name of the schema containing the domain used by the column.
DOMAIN_NAME	VARCHAR(128)	The name of the domain used by the column.

CONSTRAINT_COLUMN_USAGE

The CONSTRAINT_COLUMN_USAGE system view lists the table columns on which constraints (referential constraints, unique constraints, check constraints and assertions) that are owned by the current ident are defined.

Column name	Data type	Description
TABLE_CATALOG	VARCHAR(128)	The name of the catalog containing the table.
TABLE_SCHEMA	VARCHAR(128)	The name of the schema containing the table.
TABLE_NAME	VARCHAR(128)	The name of the table.
COLUMN_NAME	VARCHAR(128)	The name of the table column.
CONSTRAINT_CATALOG	VARCHAR(128)	The name of the catalog containing the constraint.
CONSTRAINT_SCHEMA	VARCHAR(128)	The name of the schema containing the constraint.
CONSTRAINT_NAME	VARCHAR(128)	The name of the constraint.

CONSTRAINT_TABLE_USAGE

The CONSTRAINT_TABLE_USAGE system view lists the tables on which constraints (referential constraints, unique constraints, check constraints and assertions) that are owned by the current ident are defined.

Column name	Data type	Description
TABLE_CATALOG	VARCHAR(128)	The name of the catalog containing the table.
TABLE_SCHEMA	VARCHAR(128)	The name of the schema containing the table.
TABLE_NAME	VARCHAR(128)	The name of the table.
CONSTRAINT _CATALOG	VARCHAR(128)	The name of the catalog containing the constraint.
CONSTRAINT_SCHEMA	VARCHAR(128)	The name of the schema containing the constraint.
CONSTRAINT_NAME	VARCHAR(128)	The name of the constraint.

DOMAIN_CONSTRAINTS

The DOMAIN_CONSTRAINTS system view lists the domain constraints of domains to which the current ident has access.

Column name	**Data type**	**Description**
CONSTRAINT_CATALOG	VARCHAR(128)	The name of the catalog containing the constraint.
CONSTRAINT_SCHEMA	VARCHAR(128)	The name of the schema containing the constraint.
CONSTRAINT_NAME	VARCHAR(128)	Name of the constraint.
DOMAIN_CATALOG	VARCHAR(128)	The name of the catalog containing the domain on which the constraint is defined.
DOMAIN_SCHEMA	VARCHAR(128)	The name of the schema containing the domain on which the constraint is defined.
DOMAIN_NAME	VARCHAR(128)	The name of the domain on which the constraint is defined.
IS_DEFERRABLE	VARCHAR(3)	One of: “YES” = the constraint is deferrable “NO” = the constraint is not deferrable.
INITIALLY_DEFERRED	VARCHAR(3)	One of: “YES” = the constraint is immediate “NO” = the constraint is deferred.

DOMAINS

The DOMAINS system view describes each domain to which the current ident, or PUBLIC, has USAGE privilege.

Column name	Data type	Description
DOMAIN_CATALOG	VARCHAR(128)	The name of the catalog containing the domain.
DOMAIN_SCHEMA	VARCHAR(128)	The name of the schema containing the domain.
DOMAIN_NAME	VARCHAR(128)	Name of the domain.
DATA_TYPE	VARCHAR(30)	Identifies the data type of the domain. Can be one of the following: BIGINT BINARY BINARY VARYING CHARACTER CHARACTER VARYING DATE DECIMAL DOUBLE PRECISION FLOAT INTEGER INTERVAL NUMERIC REAL SMALLINT TIME TIMESTAMP.
CHARACTER _MAXIMUM_LENGTH	INTEGER	For a character data type, this shows the maximum length in characters. For all other data types it is the NULL value.
CHARACTER _OCTET_LENGTH	INTEGER	For a character data type, this shows the maximum length in octets. For all other data types it is the NULL value. (For single octet character sets, this is the same as CHARACTER_MAX_LENGTH).
CHARACTER_SET _CATALOG	VARCHAR(128)	The name of the catalog containing the character set used by the domain.
CHARACTER_SET _SCHEMA	VARCHAR(128)	The name of the schema containing the character set used by the domain.
CHARACTER_SET _NAME	VARCHAR(128)	The name of the character set used by the domain.
COLLATION_CATALOG	VARCHAR(128)	The name of the catalog containing the default collation for the character set.
COLLATION_SCHEMA	VARCHAR(128)	The name of the schema containing the default collation for the character set.
COLLATION_NAME	VARCHAR(128)	The name of the default collation for the character set.
NUMERIC_PRECISION	INTEGER	For numeric data types, this shows the total number of significant digits allowed in the column. For all other data types it is the NULL value.

NUMERIC_PRECISION _RADIX	INTEGER	This shows whether the NUMERIC_PRECISION is given in a binary or decimal radix. The numeric radix is always decimal in Mimer SQL, therefore the value 10 is always shown for numeric data types. For all other data types it is the NULL value.
NUMERIC_SCALE	INTEGER	This defines the total number of significant digits to the right of the decimal point. For BIGINT, INTEGER and SMALLINT, this is 0. For BINARY, BINARY VARYING, CHARACTER, CHARACTER VARYING, DATETIME, FLOAT, INTERVAL, REAL and DOUBLE PRECISION data types, it is the NULL value.
DATETIME_PRECISION	INTEGER	For DATE, TIME, TIMESTAMP and interval data types, this column contains the number of digits of precision for the fractional seconds component. For other data types it is the NULL value.
INTERVAL_TYPE	VARCHAR(30)	For interval data types, this is a character string specifying the interval qualifier for the named interval data type (see Section 4.3.3.2). For other data types it is the NULL value.
INTERVAL_PRECISION	INTEGER	For interval data types, this is the number of significant digits for the interval leading precision (see Section 4.3.3.2). For other data types it is the NULL value.
DOMAIN_DEFAULT	VARCHAR(2000)	This shows the default value for the domain. If the default value is a character string, the value shown is the string enclosed in single quotes. If the default value is a numeric literal, the value is shown in its original character representation without enclosing quotes. If the default value is a DATE, TIME or TIMESTAMP, the value shown is the appropriate keyword (e.g. DATE) followed by the literal representation of the value enclosed in single quotes (see Section 4.4.5 for a description of DATE, TIME and TIMESTAMP literals). If the default value is a pseudo-literal, the value shown is the appropriate keyword (e.g. CURRENT_DATE) without enclosing quotes. If the default value is the NULL value, the value shown is the keyword NULL without enclosing quotes. If the default value cannot be represented without truncation, then TRUNCATED is shown without enclosing quotes. If no default value was specified then its value is the NULL value. The value of DOMAIN_DEFAULT is syntactically suitable for use in specifying *default-value* in a CREATE TABLE or ALTER TABLE statement (except when TRUNCATED is shown).

EXT_COLUMN_REMARKS

The EXT_COLUMN_REMARKS system view shows remarks for columns that are accessible by the current ident.

Column name	Data type	Description
TABLE_CATALOG	VARCHAR(128)	The name of the catalog containing the table.
TABLE_SCHEMA	VARCHAR(128)	The name of the schema containing the table.
TABLE_NAME	VARCHAR(128)	The name of the table.
COLUMN_NAME	VARCHAR(128)	Name of the column.
REMARKS	VARCHAR(254)	Remark for column.

EXT_DATABANKS

The EXT_DATABANKS system view shows databanks on which the current ident has table privilege. An ident with the system privileges BACKUP or SHADOW may see all databanks in the system.

Column name	Data type	Description
DATABANK_CREATOR	VARCHAR(128)	The name of the authorization ident that created the databank.
DATABANK_NAME	VARCHAR(128)	The name of the databank.
DATABANK_TYPE	VARCHAR(20)	One of: “LOG” “TRANS” “NULL” “WORK”.
IS_MASTER	VARCHAR(3)	One of: “YES” = the databank is a master databank “NO” = the databank is not a master databank.
IS_ONLINE	VARCHAR(3)	One of: “YES” = the databank is online “NO” = the databank is offline.
FILE_NUMBER	INTEGER	Ordinal number for databank file.
FILE_NAME	VARCHAR(256)	File name for databank.

EXT_IDENTS

The EXT_IDENTS system view shows authorization idents either created by the current ident or those on which the current ident has execute or member privilege.

Column name	Data type	Description
IDENT_CREATOR	VARCHAR(128)	The name of the authorization ident that created the ident.
IDENT_NAME	VARCHAR(128)	The name of the ident.
IDENT_TYPE	VARCHAR(20)	One of: “GROUP” “OS_USER” “PROGRAM” “USER”.

EXT_INDEX_COLUMN_USAGE

The EXT_INDEX_COLUMN_USAGE system view shows on which table columns an secondary index is defined. Only indexes defined on table accessible by the current ident is shown.

Column name	Data type	Description
INDEX_CATALOG	VARCHAR(128)	Catalog name for the secondary index.
INDEX_SCHEMA	VARCHAR(128)	Schema name for the secondary index.
INDEX_NAME	VARCHAR(128)	Name of the index.
IS_UNIQUE	VARCHAR(3)	One of: “YES” = the index may not contain duplicates “NO” = the index may contain duplicates.
TABLE_NAME	VARCHAR(128)	Name of the table on which the index is defined.
COLUMN_NAME	VARCHAR(128)	Name of column present in index.
IS_ASCENDING	VARCHAR(3)	One of: “YES” = the sort order of the index is ascending “NO” = the sort order of the index is descending.
ORDINAL_POSITION	INTEGER	Ordinal position for the column within the index.

EXT_INDEXES

The EXT_INDEXES system view shows secondary indexes defined on tables that are accessible by the current ident.

Column name	**Data type**	**Description**
INDEX_CATALOG	VARCHAR(128)	Catalog name for the secondary index.
INDEX_SCHEMA	VARCHAR(128)	Schema name for the secondary index.
INDEX_NAME	VARCHAR(128)	Name of the secondary index.
TABLE_NAME	VARCHAR(128)	Name of the table on which the index is defined.
IS_UNIQUE	VARCHAR(3)	One of: “YES” = the index may not contain duplicates “NO” = the index may contain duplicates.

EXT_OBJECT_IDENT_USAGE

The EXT_OBJECT_IDENT system view shows objects accessible by the current ident.

Column name	**Data type**	**Description**
CREATOR_NAME	VARCHAR(128)	Name of authorization ident that created the object.
OBJECT_CATALOG	VARCHAR(128)	Catalog name for the object.
OBJECT_SCHEMA	VARCHAR(128)	Schema name for the object.
OBJECT_NAME	VARCHAR(128)	Name of the object.
OBJECT_TYPE	VARCHAR(20)	One of: “ASSERTION” “BASE TABLE” “CHARACTER SET” “COLLATION” “CONSTRAINT” “DATABANK” “DOMAIN” “FUNCTION” “IDENT” “INDEX” “MODULE” “PROCEDURE” “SCHEMA” “SEQUENCE” “SHADOW” “SYNONYM” “TRIGGER” “VIEW”.
CREATION_DATE	TIMESTAMP	Time when object was created.
ALTERATION_DATE	TIMESTAMP	Time when object was last altered.
REMARKS	VARCHAR(254)	Remark for the object.

EXT_OBJECT_OBJECT_USED

The EXT_OBJECT_OBJECT_USED system view shows which object that are used by an object. The used and using objects must be accessible to the current ident.

Column name	Data type	Description
USED_OBJECT_TYPE	VARCHAR(20)	Object type for used object. One of: “ASSERTION” “BASE TABLE” “CHARACTER SET” “COLLATION” “CONSTRAINT” “DATABANK” “DOMAIN” “FUNCTION” “IDENT” “INDEX” “MODULE” “PROCEDURE” “SCHEMA” “SEQUENCE” “SHADOW” “SYNONYM” “TRIGGER” “VIEW”.
USED_OBJECT _CATALOG	VARCHAR(128)	Catalog name for used object.
USED_OBJECT _SCHEMA	VARCHAR(128)	Schema name for used object.
USED_OBJECT_NAME	VARCHAR(128)	Name of used object
USED_SPECIFIC_NAME	VARCHAR(128)	Specific name for used object
USING_OBJECT_TYPE	VARCHAR(20)	Object type for using object. One of: “ASSERTION” “BASE TABLE” “CHARACTER SET” “COLLATION” “CONSTRAINT” “DATABANK” “DOMAIN” “FUNCTION” “IDENT” “INDEX” “MODULE” “PROCEDURE” “SCHEMA” “SEQUENCE” “SHADOW” “SYNONYM” “TRIGGER” “VIEW”.

USING_OBJECT _CATALOG	VARCHAR(128)	Catalog name for using object
USING_OBJECT _SCHEMA	VARCHAR(128)	Schema name for using object
USING_OBJECT_NAME	VARCHAR(128)	Name of using object
USING_SPECIFIC_NAME	VARCHAR(128)	Specific name for using object

EXT_OBJECT_OBJECT_USING

The EXT_OBJECT_OBJECT_USING system view shows which object that are using another object. The used and using objects must be accessible to the current ident.

Column name	Data type	Description
USING_OBJECT_TYPE	VARCHAR(20)	Object type for using object. One of: “ASSERTION” “BASE TABLE” “CHARACTER SET” “COLLATION” “CONSTRAINT” “DATABANK” “DOMAIN” “FUNCTION” “IDENT” “INDEX” “MODULE” “PROCEDURE” “SCHEMA” “SEQUENCE” “SHADOW” “SYNONYM” “TRIGGER” “VIEW”.
USING_OBJECT _CATALOG	VARCHAR(128)	Catalog name for using object.
USING_OBJECT _SCHEMA	VARCHAR(128)	Schema name for using object.
USING_OBJECT_NAME	VARCHAR(128)	Name of using object
USING_SPECIFIC_NAME	VARCHAR(128)	Specific name for using object

USED_OBJECT_TYPE	VARCHAR(20)	Object type for used object. One of: “ASSERTION” “BASE TABLE” “CHARACTER SET” “COLLATION” “CONSTRAINT” “DATABANK” “DOMAIN” “FUNCTION” “IDENT” “INDEX” “MODULE” “PROCEDURE” “SCHEMA” “SEQUENCE” “SHADOW” “SYNONYM” “TRIGGER” “VIEW”.
USED_OBJECT _CATALOG	VARCHAR(128)	Catalog name for used object
USED_OBJECT _SCHEMA	VARCHAR(128)	Schema name for used object
USED_OBJECT_NAME	VARCHAR(128)	Name of used object
USED_SPECIFIC_NAME	VARCHAR(128)	Specific name for used object

EXT_OBJECT_PRIVILEGES

The EXT_OBJECT_PRIVILEGES system view shows which object privileges that are granted to an authorization ident. Either GRANTEE or GRANTOR should be equal to current ident.

Column name	**Data type**	**Description**
GRANTEE	VARCHAR(128)	Name of authorization ident that has received the privilege.
OBJECT_CATALOG	VARCHAR(128)	Catalog name for object.
OBJECT_SCHEMA	VARCHAR(128)	Schema name for object.
OBJECT_NAME	VARCHAR(128)	Name of object on which privilege is granted.
OBJECT_TYPE	VARCHAR(20)	One of: “CHARACTER SET” “DOMAIN” “DATABANK” “FUNCTION” “IDENT” “PROCEDURE” “SEQUENCE”.
PRIVILEGE_TYPE	VARCHAR(20)	One of: “EXECUTE” “FUNCTION” “MEMBER” “PROCEDURE” “TABLE” “USAGE”.
GRANTOR	VARCHAR(128)	Name of authorization ident that granted the privilege.
IS_GRANTABLE	VARCHAR(3)	One of: “YES” = the grantee may grant the privilege “NO” = the grantee may not grant the privilege.

EXT_SEQUENCES

The EXT_SEQUENCES system view shows sequences that are accessible to the current ident.

Column name	Data type	Description
SEQUENCE_CATALOG	VARCHAR(128)	Catalog name for the sequence
SEQUENCE_SCHEMA	VARCHAR(128)	Schema name for the sequence.
SEQUENCE_NAME	VARCHAR(128)	Name of the sequence.
IS_UNIQUE	VARCHAR(3)	One of: “YES” = the sequence will yield unique values “NO” = the sequence may repeat itself.
INITIAL_VALUE	INTEGER	The initial value for the sequence..
INCREMENT	INTEGER	The increment for the sequence.
MAXIMUM_VALUE	INTEGER	The maximum value for the sequence.

EXT_SHADOWS

The EXT_SHADOWS system view shows shadows. A user with shadow privilege may see any shadow or shadows on databanks that are created by the user.

Column name	Data type	Description
SHADOW_CREATOR	VARCHAR(128)	Creator of the shadow.
SHADOW_NAME	VARCHAR(128)	Name of the shadow.
IS_ONLINE	VARCHAR(3)	One of: “YES” = the shadow is online “NO” = the shadow is offline.
FILE_NUMBER	INTEGER	Ordinal number for physical file.
FILE_NAME	VARCHAR(256)	Name of physical file for shadow.

EXT_SOURCE_DEFINITION

The EXT_SOURCE_DEFINITION system view shows a textual definition for owned objects.

Column name	Data type	Description
OBJECT_CATALOG	VARCHAR(128)	Catalog name for object.
OBJECT_SCHEMA	VARCHAR(128)	Schema name for object.
OBJECT_NAME	VARCHAR(128)	Name of object.
OBJECT_TYPE	VARCHAR(20)	One of: “CONSTRAINT” “FUNCTION” “MODULE” “PROCEDURE” “TRIGGER”.
SOURCE_DEFINITION	VARCHAR(5000)	Definition text for object.
SOURCE_LENGTH	INTEGER	Total length of source.
LINE_NUMBER	INTEGER	The “line number” within the source. 1 = this is the first 5000 character block of the source, 2 = this is the second 5000 character block of the source, etc.

EXT_STATISTICS

The EXT_STATISTICS system view shows when statistics for a base table was collected.

Column name	Data type	Description
TABLE_CATALOG	VARCHAR(128)	Catalog name for base table.
TABLE_SCHEMA	VARCHAR(128)	Schema name for base table.
TABLE_NAME	VARCHAR(128)	Name of base table.
STATISTICS _GATHERED	TIMESTAMP(2)	Time when statistics for this table was collected

EXT_SYNONYMS

The EXT_SYNONYMS system view shows synonyms on accessible tables.

Column name	Data type	Description
SYNONYM_CATALOG	VARCHAR(128)	Catalog name for synonym.
SYNONYM_SCHEMA	VARCHAR(128)	Schema name for synonym.
SYNONYM_NAME	VARCHAR(128)	Name of synonym.
TABLE_CATALOG	VARCHAR(128)	Catalog for table on which synonym is defined.
TABLE_SCHEMA	VARCHAR(128)	Schema name for table.
TABLE_NAME	VARCHAR(128)	Name of table.

EXT_SYSTEM_PRIVILEGES

The EXT_SYSTEM_PRIVILEGES system view shows granted to or by the current ident.

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Name of grantee.
PRIVILEGE_TYPE	VARCHAR(20)	One of: “BACKUP” “DATABANK” “IDENT” “SCHEMA” “SHADOW” “STATISTICS”.
GRANTOR	VARCHAR(128)	Name of grantor.
IS_GRANTABLE	VARCHAR(3)	One of: “YES” = grantee has grant option “NO” = grantee has not grant option.

EXT_TABLE_DATABANK_USAGE

The EXT_TABLE_DATABANK_USAGE system view shows in which databank a base table is located. Base tables accessible to the current ident are shown.

Column name	Data type	Description
TABLE_CATALOG	VARCHAR(128)	Catalog for table.
TABLE_SCHEMA	VARCHAR(128)	Schema name for table.
TABLE_NAME	VARCHAR(128)	Name of table.
DATABANK_CREATOR	VARCHAR(128)	Name of authorization ident that created databank.
DATABANK_NAME	VARCHAR(128)	Name of databank.

KEY_COLUMN_USAGE

The KEY_COLUMN_USAGE system view lists the table columns that are constrained as keys and owned by the current ident.

Column name	Data type	Description
CONSTRAINT _CATALOG	VARCHAR(128)	The name of the catalog containing the table constraint.
CONSTRAINT_SCHEMA	VARCHAR(128)	The name of the schema containing the table or view.
CONSTRAINT_NAME	VARCHAR(128)	The name of the table constraint.
TABLE_CATALOG	VARCHAR(128)	The name of the catalog containing the table or view.
TABLE_SCHEMA	VARCHAR(128)	The name of the schema containing the table or view.
TABLE_NAME	VARCHAR(128)	The name of the table or view.
COLUMN_NAME	VARCHAR(128)	The name of the column.
ORDINAL_POSITION	INTEGER	The ordinal position of the column within the key.

MODULES

The MODULES system views shows modules created by the current ident.

Column name	Data type	Description
MODULE_CATALOG	VARCHAR(128)	Catalog name for module.
MODULE_SCHEMA	VARCHAR(128)	Schema name for module.
MODULE_NAME	VARCHAR(128)	Name of module.
DEFAULT_CHARACTER _SET_CATALOG	VARCHAR(128)	Catalog name for default character set.
DEFAULT_CHARACTER _SET_SCHEMA	VARCHAR(128)	Schema name for default character set.
DEFAULT_CHARACTER _SET_NAME	VARCHAR(128)	The name for default character set.
DEFAULT_SCHEMA _CATALOG	VARCHAR(128)	Catalog name for default schema of the module.
DEFAULT_SCHEMA _NAME	VARCHAR(128)	Name of default schema of the module.
MODULE_DEFINITION	VARCHAR(2000)	Module definition; if text larger than 2000 the word TRUNCATED is stored. The complete text can always be found in the source definition view.
SQL_PATH	VARCHAR(2000)	The default path for the module

PARAMETERS

The PARAMETERS system view lists the parameters of routines on which the current ident, or PUBLIC, has EXECUTE privilege.

Column name	Data type	Description
SPECIFIC_CATALOG	VARCHAR(128)	The catalog name for the specific name of the routine.
SPECIFIC_SCHEMA	VARCHAR(128)	The schema name for the specific name of the routine.
SPECIFIC_NAME	VARCHAR(128)	The specific name of the routine.
ORDINAL_POSITION	INTEGER	The ordinal position of the parameter in the routine. The first parameter in the routine is number 1.
PARAMETER_MODE	VARCHAR(128)	Indicates whether the parameter is IN, OUT or INOUT.
PARAMETER_NAME	VARCHAR(128)	The name of the parameter.
DATA_TYPE	VARCHAR(128)	The data type of the parameter.
CHARACTER _MAXIMUM_LENGTH	INTEGER	For a character data type, this shows the maximum length in characters.
CHARACTER _OCTET_LENGTH	INTEGER	For a character data type, this shows the maximum length in octets. (For single octet character sets, this is the same as CHARACTER_MAX_LENGTH).
CHARACTER_SET _CATALOG	VARCHAR(128)	The name of the catalog containing the character set.
CHARACTER_SET _SCHEMA	VARCHAR(128)	The name of the schema containing the character set.
CHARACTER_SET _NAME	VARCHAR(128)	Name of the character set.
COLLATION_CATALOG	VARCHAR(128)	The name of the catalog containing the collation.
COLLATION_SCHEMA	VARCHAR(128)	The name of the schema containing the collation.
COLLATION_NAME	VARCHAR(128)	Name of the collation.
NUMERIC_PRECISION	INTEGER	For numeric data types, this shows the total number of significant digits allowed in the column. For all other data types it is the NULL value.
NUMERIC_PRECISION _RADIX	INTEGER	This shows whether the NUMERIC_PRECISION is given in a binary or decimal radix. The numeric radix is always decimal in Mimer SQL, therefore the value 10 is always shown for numeric data types. For all other data types it is the NULL value.

NUMERIC_SCALE	INTEGER	This defines the total number of significant digits to the right of the decimal point. For BIGINT, INTEGER and SMALLINT, this is 0. For BINARY, BINARY VARYING, CHARACTER, CHARACTER VARYING, DATETIME, FLOAT, INTERVAL, REAL and DOUBLE PRECISION data types, it is the NULL value.
DATETIME_PRECISION	INTEGER	For DATE, TIME, TIMESTAMP and interval data types, this column contains the number of digits of precision for the fractional seconds component. For other data types it is the NULL value.
INTERVAL_TYPE	VARCHAR(128)	For interval data types, this is a character string specifying the interval qualifier for the named interval data type (see Section 4.3.3.2).
INTERVAL_PRECISION	INTEGER	For interval data types, this is the number of significant digits for the interval leading precision (see Section 4.3.3.2).

REFERENTIAL_CONSTRAINTS

The REFERENTIAL_CONSTRAINTS system view lists the referential constraints that are accessible by the current ident.

Column name	Data type	Description
CONSTRAINT _CATALOG	VARCHAR(128)	The name of the catalog containing the referential constraint.
CONSTRAINT_SCHEMA	VARCHAR(128)	The name of the schema containing the referential constraint.
CONSTRAINT_NAME	VARCHAR(128)	The name of the referential constraint.
UNIQUE_CONSTRAINT _CATALOG	VARCHAR(128)	The name of the catalog containing the unique constraint being referenced.
UNIQUE_CONSTRAINT _SCHEMA	VARCHAR(128)	The name of the schema containing the unique constraint being referenced
UNIQUE_CONSTRAINT _NAME	VARCHAR(128)	The name of the unique constraint being referenced.
MATCH_OPTION	VARCHAR(20)	One of: NONE PARTIAL FULL.
UPDATE_RULE	VARCHAR(20)	One of: CASCADE SET NULL SET DEFAULT NO ACTION.
DELETE_RULE	VARCHAR(20)	One of: CASCADE SET NULL SET DEFAULT NO ACTION.

ROUTINE_COLUMN_USAGE

The ROUTINE_COLUMN_USAGE system view lists the table columns that are owned by the current ident which are referenced from within a routine.

Column name	**Data type**	**Description**
SPECIFIC_CATALOG	VARCHAR(128)	The catalog name for the specific name of the routine.
SPECIFIC_SCHEMA	VARCHAR(128)	The schema name for the specific name of the routine.
SPECIFIC_NAME	VARCHAR(128)	The specific name of the routine.
ROUTINE_CATALOG	VARCHAR(128)	The name of the catalog containing the routine.
ROUTINE_SCHEMA	VARCHAR(128)	The name of the schema containing the routine.
ROUTINE_NAME	VARCHAR(128)	The name of the routine.
TABLE_CATALOG	VARCHAR(128)	The name of the catalog containing the table.
TABLE_SCHEMA	VARCHAR(128)	The name of the schema containing the table.
TABLE_NAME	VARCHAR(128)	The name of the table.
COLUMN_NAME	VARCHAR(128)	The name of the column.

ROUTINE_PRIVILEGES

The ROUTINE_PRIVILEGES system view lists privileges that were granted by the current ident, and privileges that were granted to the current ident or to PUBLIC, on a routine.

Column name	Data type	Description
GRANTOR	VARCHAR(128)	The name of the ident who granted the privilege.
GRANTEE	VARCHAR(128)	The name of the ident to whom the privilege was granted. Granting a privilege to PUBLIC will result in only one row (per privilege granted) in this view and the name “PUBLIC” will be shown.
SPECIFIC_CATALOG	VARCHAR(128)	The catalog name for the specific name of the routine.
SPECIFIC_SCHEMA	VARCHAR(128)	The schema name for the specific name of the routine.
SPECIFIC_NAME	VARCHAR(128)	The specific name of the routine.
ROUTINE_CATALOG	VARCHAR(128)	The name of the catalog containing the routine.
ROUTINE_SCHEMA	VARCHAR(128)	The name of the schema containing the routine.
ROUTINE_NAME	VARCHAR(128)	The name of the routine.
PRIVILEGE_TYPE	VARCHAR(128)	The type of the privilege.
IS_GRANTABLE	VARCHAR(3)	One of: “YES” = the privilege is held with the WITH GRANT OPTION “NO” = the privilege is held without the WITH GRANT OPTION.

ROUTINE_TABLE_USAGE

The ROUTINE_TABLE_USAGE system view lists the tables that are owned by the current ident on which SQL-invoked routines depend.

Column name	Data type	Description
SPECIFIC_CATALOG	VARCHAR(128)	The catalog name for the specific name of the routine.
SPECIFIC_SCHEMA	VARCHAR(128)	The schema name for the specific name of the routine.
SPECIFIC_NAME	VARCHAR(128)	The specific name of the routine.
ROUTINE_CATALOG	VARCHAR(128)	The name of the catalog containing the routine.
ROUTINE_SCHEMA	VARCHAR(128)	The name of the schema containing the routine.
ROUTINE_NAME	VARCHAR(128)	The name of the routine.
TABLE_CATALOG	VARCHAR(128)	The name of the catalog containing the table.
TABLE_SCHEMA	VARCHAR(128)	The name of the schema containing the table.
TABLE_NAME	VARCHAR(128)	The name of the table.

ROUTINES

The ROUTINES system view lists the routines on which the current ident, or PUBLIC, has EXECUTE privilege.

Column name	Data type	Description
SPECIFIC_CATALOG	VARCHAR(128)	The catalog name for the specific name of the routine.
SPECIFIC_SCHEMA	VARCHAR(128)	The schema name for the specific name of the routine.
SPECIFIC_NAME	VARCHAR(128)	The specific name of the routine.
ROUTINE_CATALOG	VARCHAR(128)	The name of the catalog containing the routine.
ROUTINE_SCHEMA	VARCHAR(128)	The name of the schema containing the routine.
ROUTINE_NAME	VARCHAR(128)	The name of the routine.
MODULE_CATALOG	VARCHAR(128)	The name of the catalog containing the module to which the routine belongs.
MODULE_SCHEMA	VARCHAR(128)	The name of the schema containing the module to which the routine belongs.
MODULE_NAME	VARCHAR(128)	The name of the module to which the routine belongs.
ROUTINE_TYPE	VARCHAR(20)	Indicates whether the routine is a function or a procedure.
DATA_TYPE	VARCHAR(30)	The data type returned by the function.
CHARACTER _MAXIMUM_LENGTH	INTEGER	For a character data type, this shows the maximum length in characters.
CHARACTER _OCTET_LENGTH	INTEGER	For a character data type, this shows the maximum length in octets. (For single octet character sets, this is the same as CHARACTER_MAX_LENGTH).
CHARACTER_SET _CATALOG	VARCHAR(128)	The name of the catalog containing the character set.
CHARACTER_SET _SCHEMA	VARCHAR(128)	The name of the schema containing the character set.
CHARACTER_SET _NAME	VARCHAR(128)	Name of the character set.
COLLATION_CATALOG	VARCHAR(128)	The name of the catalog containing the collation.
COLLATION_SCHEMA	VARCHAR(128)	The name of the schema containing the collation.
COLLATION_NAME	VARCHAR(128)	Name of the collation.
NUMERIC_PRECISION	INTEGER	For numeric data types, this shows the total number of significant digits allowed in the column. For all other data types it is the NULL value.

NUMERIC_PRECISION _RADIX	INTEGER	This shows whether the NUMERIC_PRECISION is given in a binary or decimal radix. The numeric radix is always decimal in Mimer SQL, therefore the value 10 is always shown for numeric data types. For all other data types it is the NULL value.
NUMERIC_SCALE	INTEGER	This defines the total number of significant digits to the right of the decimal point. For BIGINT, INTEGER and SMALLINT, this is 0. For BINARY, BINARY VARYING, CHARACTER, CHARACTER VARYING, DATETIME, FLOAT, INTERVAL, REAL and DOUBLE PRECISION data types, it is the NULL value.
DATETIME_PRECISION	INTEGER	For DATE, TIME, TIMESTAMP and interval data types, this column contains the number of digits of precision for the fractional seconds component. For other data types it is the NULL value.
INTERVAL_TYPE	VARCHAR(30)	For interval data types, this is a character string specifying the interval qualifier for the named interval data type (see Section 4.3.3.2).
INTERVAL_PRECISION	INTEGER	For interval data types, this is the number of significant digits for the interval leading precision (see Section 4.3.3.2).
EXTERNAL_LANGUAGE	VARCHAR(20)	The language for the routine if it is an external routine, otherwise the NULL value is shown
IS_DETERMINISTIC	VARCHAR(3)	One of: “YES” = the function was declared as DETERMINISTIC when it was created “NO” = the function was not declared as DETERMINISTIC when it was created.
SQL_DATA_ACCESS	VARCHAR(128)	One of: “CONTAINS SQL” “READS SQL” “MODIFIES SQL”.
ROUTINE_BODY	CHAR(20)	One of: “SQL” = the routine is an SQL routine “EXTERNAL” = the routine is an external routine.
ROUTINE_DEFINITION	VARCHAR(2000)	The text of the routine definition. If the actual definition would not fit into the maximum length of this column, the NULL value will be shown and the definition text will appear in the view EXT_SOURCE_DEFINITION.
EXTERNAL_NAME	VARCHAR(128)	The external name of the routine if it is an external routine, otherwise the NULL value is shown.
PARAMETER_STYLE	CHAR(20)	The parameter passing style of the routine if it is an external routine, otherwise the NULL value is shown.
SQL_PATH	VARCHAR(2000)	The path for the routine. The NULL value is shown if no path is defined.

SCHEMA_LEVEL _ROUTINE	CHAR(3)	One of: “YES” = the routine was created on its own “NO” = the routine was created in a module.
IS_RESULT	CHAR(3)	One of: “YES” = the routine returns a result-set “NO” = the routine does not return a result-set.

SCHEMATA

The SCHEMATA system view lists the schemas that are owned by the current ident.

Column name	Data type	Description
CATALOG_NAME	VARCHAR(128)	The name of the catalog containing the schema.
SCHEMA_NAME	VARCHAR(128)	The name of the schema.
SCHEMA_OWNER	VARCHAR(128)	The name of the ident who created the schema.
DEFAULT_CHARACTER _SET_CATALOG	VARCHAR(128)	The name of the catalog that contains the default character set for the schema.
DEFAULT_CHARACTER _SET_SCHEMA	VARCHAR(128)	The name of the schema that contains the default character set for the schema.
DEFAULT_CHARACTER _SET_NAME	VARCHAR(128)	The name of the default character set for the schema.
SQL_PATH	VARCHAR(2000)	The default path for the schema

SQL_LANGUAGES

The SQL_LANGUAGES system view lists the conformance levels, options and dialects supported by the SQL implementation.

Column name	Data type	Description
SQL_LANGUAGE _SOURCE	VARCHAR(254)	The organization that defined the SQL version.
SQL_LANGUAGE_YEAR	VARCHAR(254)	The year the relevant source document was approved.
SQL_LANGUAGE _CONFORMANCE	VARCHAR(254)	The conformance level to the relevant document that the implementation claims.
SQL_LANGUAGE _INTEGRITY	VARCHAR(254)	(Meaning no longer defined).
SQL_LANGUAGE _IMPLEMENTATION	VARCHAR(254)	A character string, defined by the vendor, that uniquely defines the vendor’s SQL product.
SQL_LANGUAGE _BINDING_STYLE	VARCHAR(254)	Included to envisage future adoption of direct, module or other binding styles.
SQL_LANGUAGE _PROGRAMMING _LANGUAGE	VARCHAR(254)	The host language for which the binding style is supported.

TABLE_CONSTRAINTS

The TABLE_CONSTRAINTS system view lists the table constraints that are accessible by the current ident.

Column name	**Data type**	**Description**
CONSTRAINT _CATALOG	VARCHAR(128)	The name of the catalog containing the table constraint.
CONSTRAINT_SCHEMA	VARCHAR(128)	The name of the schema containing the table or view.
CONSTRAINT_NAME	VARCHAR(128)	The name of the table constraint.
TABLE_CATALOG	VARCHAR(128)	The name of the catalog containing the table or view.
TABLE_SCHEMA	VARCHAR(128)	The name of the schema containing the table or view.
TABLE_NAME	VARCHAR(128)	The name of the table or view.
CONSTRAINT_TYPE	VARCHAR(20)	One of: “CHECK” “FOREIGN KEY” “PRIMARY KEY” “UNIQUE”.
IS_DEFERRABLE	VARCHAR(3)	One of: “YES” = the constraint is deferrable “NO” = the constraint is not deferrable.
INITIALLY_DEFERRED	VARCHAR(3)	One of: “YES” = the constraint is immediate “NO” = the constraint is deferred.

TABLE_PRIVILEGES

The TABLE_PRIVILEGES system view lists privileges that were granted by the current ident, and privileges that were granted to the current ident or to PUBLIC, on an entire table.

Column name	**Data type**	**Description**
GRANTOR	VARCHAR(128)	The name of the ident who granted the privilege.
GRANTEE	VARCHAR(128)	The name of the ident to whom the privilege was granted. Granting a privilege to PUBLIC will result in only one row (per privilege granted) in this view and the name “PUBLIC” will be shown.
TABLE_CATALOG	VARCHAR(128)	The name of the catalog containing the table.
TABLE_SCHEMA	VARCHAR(128)	The name of the schema containing the table.
TABLE_NAME	VARCHAR(128)	The name of the table in question.
PRIVILEGE_TYPE	VARCHAR(20)	A value describing the type of the privilege. One of: DELETE INSERT REFERENCES SELECT UPDATE. Rows where privilege type is REFERENCES or UPDATE only describe cases where the grantee was granted the privilege on the **entire** table. Where the GRANT statement granted REFERENCES or UPDATE privilege to specified columns of a table, no rows appear in TABLE_PRIVILEGES but there are rows in COLUMN_PRIVILEGES. Note that when multiple table privileges are granted to the same user at the same time (e.g. when the keyword “ALL” is used), multiple rows appear in this view (one for each privilege granted).
IS_GRANTABLE	VARCHAR(3)	One of: “YES” = the privilege is held with the WITH GRANT OPTION “NO” = the privilege is held without the WITH GRANT OPTION.

TABLES

The TABLES system view lists tables to which the current ident, or PUBLIC, has access.

Column name	Data type	Description
TABLE_CATALOG	VARCHAR(128)	The name of the catalog containing the table or view.
TABLE_SCHEMA	VARCHAR(128)	The name of the schema containing the table or view.
TABLE_NAME	VARCHAR(128)	The name of the table or view.
TABLE_TYPE	VARCHAR(20)	One of: “BASE TABLE” = the row describes a table “VIEW” = the row describes a view.

TRANSLATIONS

The TRANSLATIONS system view lists the character translations on which the current ident, or PUBLIC, has USAGE privilege.

The source character set is the character set to which the characters that are to be translated by the translation belong. The target character set is the character set to which the characters that are the result of the translation belong.

Column name	Data type	Description
TRANSLATION _CATALOG	VARCHAR(128)	The name of the catalog containing the translation.
TRANSLATION _SCHEMA	VARCHAR(128)	The name of the schema containing the translation.
TRANSLATION_NAME	VARCHAR(128)	The name of the translation.
SOURCE_CHARACTER _SET_CATALOG	VARCHAR(128)	The name of the catalog containing the source character set.
SOURCE_CHARACTER _SET_SCHEMA	VARCHAR(128)	The name of the schema containing the source character set.
SOURCE_CHARACTER _SET_NAME	VARCHAR(128)	The name of the source character set.
TARGET_CHARACTER _SET_CATALOG	VARCHAR(128)	The name of the catalog containing the target character set.
TARGET_CHARACTER _SET_SCHEMA	VARCHAR(128)	The name of the schema containing the target character set.
TARGET_CHARACTER _SET_NAME	VARCHAR(128)	The name of the target character set.

TRIGGERED_UPDATE_COLUMNS

The TRIGGERED_UPDATE_COLUMNS system view lists the columns owned by the current ident that are referenced from the explicit column list in the trigger event of an UPDATE trigger.

Column name	Data type	Description
TRIGGER_CATALOG	VARCHAR(128)	The name of the catalog containing the trigger.
TRIGGER_SCHEMA	VARCHAR(128)	The name of the schema containing the trigger.
TRIGGER_NAME	VARCHAR(128)	The name of the trigger.
EVENT_OBJECT _CATALOG	VARCHAR(128)	The name of the catalog containing the table or view on which the trigger is created.
EVENT_OBJECT _SCHEMA	VARCHAR(128)	The name of the schema containing the table or view on which the UPDATE trigger is created.
EVENT_OBJECT _TABLE	VARCHAR(128)	The name of the table or view on which the UPDATE trigger is created.
EVENT_OBJECT _COLUMN	VARCHAR(128)	The name of the table or view column referenced in the column list of the UPDATE trigger event.

TRIGGER_COLUMN_USAGE

The TRIGGER_COLUMN_USAGE view lists the table columns on which triggers, that owned by the current ident, depend because they are referenced in the search condition of the trigger or in one of the statements in the body of the trigger.

Column name	Data type	Description
TRIGGER_CATALOG	VARCHAR(128)	The name of the catalog containing the trigger.
TRIGGER_SCHEMA	VARCHAR(128)	The name of the schema containing the trigger.
TRIGGER_NAME	VARCHAR(128)	The name of the trigger.
TABLE_CATALOG	VARCHAR(128)	The name of the catalog containing the referenced table or view.
TABLE_SCHEMA	VARCHAR(128)	The name of the schema containing the referenced table or view.
TABLE_NAME	VARCHAR(128)	The name of the referenced table or view.
COLUMN_NAME	VARCHAR(128)	The name of the referenced column.

TRIGGER_TABLE_USAGE

The TRIGGER_TABLE_USAGE view lists the tables on which triggers, that owned by the current ident, depend.

Column name	Data type	Description
TRIGGER_CATALOG	VARCHAR(128)	The name of the catalog containing the trigger.
TRIGGER_SCHEMA	VARCHAR(128)	The name of the schema containing the trigger.
TRIGGER_NAME	VARCHAR(128)	The name of the trigger.
TABLE_CATALOG	VARCHAR(128)	The name of the catalog containing the referenced table or view.
TABLE_SCHEMA	VARCHAR(128)	The name of the schema containing the referenced table or view.
TABLE_NAME	VARCHAR(128)	The name of the referenced table or view.

TRIGGERS

The TRIGGERS system view lists the triggers owned by the current ident.

Column name	Data type	Description
TRIGGER_CATALOG	VARCHAR(128)	The name of the catalog containing the trigger.
TRIGGER_SCHEMA	VARCHAR(128)	The name of the schema containing the trigger.
TRIGGER_NAME	VARCHAR(128)	The name of the trigger.
EVENT _MANIPULATION	CHAR(20)	The data manipulation event triggering execution of the trigger (the “trigger event”). One of: “INSERT” “DELETE” “UPDATE”.
EVENT_OBJECT _CATALOG	VARCHAR(128)	The name of the catalog containing the table or view on which the trigger is created.
EVENT_OBJECT _SCHEMA	VARCHAR(128)	The name of the schema containing the table or view on which the trigger is created.
EVENT_OBJECT_TABLE	VARCHAR(128)	The name of the table or view on which the trigger is created.
ACTION_ORDER	INTEGER	Ordinal number for trigger execution. This number will define the execution order of triggers on the same table and with the same value for EVENT_MANIPULATION, ACTION_CONDITION, CONDITION_TIMING and ACTION_ORIENTATION. The trigger with 1 in this column will be executed first, followed by the trigger with 2 etc.
ACTION_CONDITION	VARCHAR(2000)	The character representation of the search condition in the WHEN clause of the trigger. If the length of the text exceeds 2000 characters, the NULL value will be shown.

ACTION_STATEMENT	VARCHAR(2000)	The character representation of the body of the trigger. If the length of the text exceeds 2000 characters, the NULL value will be shown.
ACTION_ORIENTATION	CHAR(20)	One of: “ROW” = the trigger is a row trigger “STATEMENT” = the trigger is a statement trigger.
CONDITION_TIMING	CHAR(20)	One of: “BEFORE” = the trigger is executed before the triggering data manipulation operation “INSTEAD OF” = the trigger is executed instead of the triggering data manipulation operation “AFTER” = the trigger is executed after the triggering data manipulation operation.
CONDITION _REFERENCE_OLD _TABLE	VARCHAR(128)	The identifier specified in the OLD TABLE clause.
CONDITION _REFERENCE_NEW _TABLE	VARCHAR(128)	The identifier specified in the NEW TABLE clause.
CONDITION _REFERENCE_OLD _ROW	VARCHAR(128)	The identifier specified in the OLD ROW clause.
CONDITION _REFERENCE_NEW _ROW	VARCHAR(128)	The identifier specified in the NEW ROW clause.
COLUMN_LIST_IS _IMPLICIT	CHAR(3)	One of: “YES” = the trigger will be executed on update of any column in the table “NO” = the trigger will only be executed on update of those columns specified in the UPDATE OF clause in the trigger definition.
CREATED	TIMESTAMP(2)	The date when the trigger was created.

USAGE_PRIVILEGES

The USAGE_PRIVILEGES system view lists the USAGE privileges that were granted by the current ident, and granted to the current ident or to PUBLIC.

Column name	**Data type**	**Description**
GRANTOR	VARCHAR(128)	The name of the ident who granted the privilege.
GRANTEE	VARCHAR(128)	The name of the ident to whom the privilege was granted. Granting a privilege to PUBLIC will result in only one row (per privilege granted) in this view and the name “PUBLIC” will be shown.
OBJECT_CATALOG	VARCHAR(128)	The name of the catalog that contains the object character set or collation.
OBJECT_SCHEMA	VARCHAR(128)	The name of the schema that contains the object character set or collation.
OBJECT_NAME	VARCHAR(128)	The name of the character set or collation.
OBJECT_TYPE	VARCHAR(128)	One of: “CHARACTER SET” = the privilege is held on a character set “COLLATION” = the privilege is held on a collation “DOMAIN” = the privilege is held on a domain.
PRIVILEGE_TYPE	VARCHAR(20)	This will always be “USAGE”.
IS_GRANTABLE	VARCHAR(3)	One of: “YES” = the privilege is held with the WITH GRANT OPTION “NO” = the privilege is held without the WITH GRANT OPTION.

VIEW_COLUMN_USAGE

The VIEW_COLUMN_USAGE system view lists the table columns on which views that are owned by the current ident depend.

Column name	**Data type**	**Description**
VIEW_CATALOG	VARCHAR(128)	The name of the catalog containing the view.
VIEW_SCHEMA	VARCHAR(128)	The name of the schema containing the view.
VIEW_NAME	VARCHAR(128)	The name of the view.
TABLE_CATALOG	VARCHAR(128)	The name of the catalog containing the table.
TABLE_SCHEMA	VARCHAR(128)	The name of the schema containing the table.
TABLE_NAME	VARCHAR(128)	The name of the table.
COLUMN_NAME	VARCHAR(128)	The name of the column.

VIEW_TABLE_USAGE

The VIEW_TABLE_USAGE system view lists the tables on which views that are owned by the current ident depend.

Column name	Data type	Description
VIEW_CATALOG	VARCHAR(128)	The name of the catalog containing the view.
VIEW_SCHEMA	VARCHAR(128)	The name of the schema containing the view.
VIEW_NAME	VARCHAR(128)	The name of the view.
TABLE_CATALOG	VARCHAR(128)	The name of the catalog containing the table.
TABLE_SCHEMA	VARCHAR(128)	The name of the schema containing the table.
TABLE_NAME	VARCHAR(128)	The name of the table.

VIEWS

The VIEWS system view lists the views to which the current ident as access.

Column name	Data type	Description
TABLE_CATALOG	VARCHAR(128)	The name of the catalog containing the table or view.
TABLE_SCHEMA	VARCHAR(128)	The name of the schema containing the table or view.
TABLE_NAME	VARCHAR(128)	The name of the table or view.
VIEW_DEFINITION	VARCHAR(2000)	The definition of the view as it would appear in a CREATE VIEW statement. If the actual definition would not fit into the maximum length of this column, the NULL value will be shown.
CHECK_OPTION	VARCHAR(20)	The value “CASCADED” is shown if WITH CHECK OPTION was specified in the CREATE VIEW statement that created the view, and the value “NONE” is shown otherwise.
IS_UPDATABLE	VARCHAR(3)	One of: “YES” = the view is updatable “NO” = the view is not updatable.

Standard compliance

The table below summarizes standards compliance concerning the views in INFORMATION_SCHEMA.

Standard	Compliance	Comments
X/Open-95 SQL92	EXTENDED	The views with names beginning “EXT_” are a Mimer SQL extension.
SQL/PSM	YES	All views with names beginning “ROUTINE_” or “MODULE_” are included in the SQL/PSM standard.
SQL99	YES	All views with “TRIGGER” in the name are included in the SQL99 standard.

7.2 INFO_SCHEM

The table below summarizes the system views that are part of the schema INFO_SCHEM (view names appear in their unqualified form):

View name	Description
CHARACTER_SETS	Accessible character sets and their default collations.
COLLATIONS	Accessible collations.
COLUMN_PRIVILEGES	Privileges on table columns.
COLUMNS	Accessible table columns.
INDEXES	Accessible indexes.
SCHEMATA	All schemas in the database.
SERVER_INFO	Attributes of the current database server.
SQL_LANGUAGES	All supported SQL standards and dialects.
TABLE_PRIVILEGES	Privileges on tables.
TABLES	Accessible tables.
TRANSLATIONS	Accessible character set translations.
USAGE_PRIVILEGES	Privileges on character sets and collations.
VIEWS	Accessible views.

CHARACTER_SETS

The CHARACTER_SETS system view describes each character set to which the current ident, or PUBLIC, has USAGE privilege.

The view includes one row describing the character set named SQL_TEXT.

Column name	Data type	Description
CHARSET_CAT	VARCHAR(128)	The name of the catalog containing the character set.
CHARSET_SCHEM	VARCHAR(128)	The name of the schema containing the character set.
CHARSET_NAME	VARCHAR(128)	Name of the character set.
FORM_OF_USE	VARCHAR(128)	A user-defined name that indicates the form-of-use of the character set.
NUM_CHARS	INTEGER	The number of characters in the character set.
DEF_COLLATE_CAT	VARCHAR(128)	The name of the catalog containing the default collation for the character set.
DEF_COLLATE_SCHEM	VARCHAR(128)	The name of the schema containing the default collation for the character set.
DEF_COLLATE_NAME	VARCHAR(128)	The name of the default collation for the character set.
REMARKS	VARCHAR(254)	May contain descriptive information about the character set.

COLLATIONS

The COLLATIONS system view describes each collation to which the current ident, or PUBLIC, has USAGE privilege.

The view includes one row describing the collation named SQL_TEXT which is defined on the character set named SQL_TEXT and is the default collation for that character set.

Column name	**Data type**	**Description**
COLLATION_CAT	VARCHAR(128)	The name of the catalog containing the collation.
COLLATION_SCHEM	VARCHAR(128)	The name of the schema containing the collation.
COLLATION_NAME	VARCHAR(128)	Name of the collation.
CHARSET_CAT	VARCHAR(128)	The name of the catalog containing the character set on which the collation is defined.
CHARSET_SCHEM	VARCHAR(128)	The name of the schema containing the character set on which the collation is defined.
CHARSET_NAME	VARCHAR(128)	The name of the character set on which the collation is defined.
PAD_ATTRIBUTE	VARCHAR(20)	One of the following values: “NO PAD” = the collation has the *no pad* attribute “PAD SPACE” = the collation has the *pad space* attribute.
REMARKS	VARCHAR(254)	May contain descriptive information about the collation.

COLUMN_PRIVILEGES

The COLUMN_PRIVILEGES system view lists privileges on table columns that were granted by the current ident and privileges on table columns that were granted to the current ident or to PUBLIC.

Granting a privilege on an entire table implicitly grants privileges on each of the table columns (as well as on future columns in the table), each of which is shown in this view.

The view includes rows that represent the implicit privileges that the creator of a table has on the table.

A row will be shown for each instance of a privilege granted on each table column (whether or not the specified privilege can be granted or revoked on individual columns), the exception is DELETE which can never apply to an individual column.

Column name	**Data type**	**Description**
GRANTOR	VARCHAR(128)	The name of the ident who granted the privilege. For rows that describe the implicit privileges that a table creator has on each column in a table, the name "_SYSTEM" is shown.
GRANTEE	VARCHAR(128)	The name of the ident to whom the privilege was granted. Granting a privilege to PUBLIC will result in only one row (per privilege granted) in this view and the name "PUBLIC" will be shown.
TABLE_CAT	VARCHAR(128)	The name of the catalog containing the table.
TABLE_SCHEM	VARCHAR(128)	The name of the schema containing the table.
TABLE_NAME	VARCHAR(128)	The name of the table containing the column on which the column privilege has been granted.
COLUMN_NAME	VARCHAR(128)	The name of the column on which the privilege has been granted.
PRIVILEGE_TYPE	VARCHAR(20)	A value describing the type of the column privilege that was granted. One of: "INSERT" "REFERENCES" "SELECT" "UPDATE". Note that when multiple table column privileges are granted to the same user at the same time (e.g. when the keyword "ALL" is used), multiple rows appear in this view (one for each privilege granted).
IS_GRANTABLE	VARCHAR(3)	One of: "YES" = the privilege is held with the WITH GRANT OPTION "NO" = the privilege is held without the WITH GRANT OPTION.
REMARKS	VARCHAR(254)	May contain descriptive information about the privilege.

COLUMNS

The COLUMNS system view lists the table columns that the current ident is privileged to access.

If the current ident, or PUBLIC, holds one or more of INSERT, DELETE or SELECT privilege on a table, then one row is shown for each column in the table.

If the current ident, or PUBLIC, holds REFERENCES or UPDATE privilege on one or more table columns, then one row is shown for each of the columns.

Column name	Data type	Description
TABLE_CAT	VARCHAR(128)	The name of the catalog containing the table or view.
TABLE_SCHEM	VARCHAR(128)	The name of the schema containing the table or view.
TABLE_NAME	VARCHAR(128)	The name of the table or view.
COLUMN_NAME	VARCHAR(128)	The name of the column of the table or view.
ORDINAL_POSITION	INTEGER	The ordinal position of the column in the table. The first column in the table is number 1.
COLUMN_DEF	VARCHAR(2000)	This shows the default value for the column. If the default value is a character string, the value shown is the string enclosed in single quotes. If the default value is a numeric literal, the value is shown in its original character representation without enclosing quotes. If the default value is a DATE, TIME or TIMESTAMP, the value shown is the appropriate keyword (e.g. DATE) followed by the literal representation of the value enclosed in single quotes (see Section 4.4.5 for a description of DATE, TIME and TIMESTAMP literals). If the default value is a pseudo-literal, the value shown is the appropriate keyword (e.g. CURRENT_DATE) without enclosing quotes. If the default value is the NULL value, the value shown is the keyword NULL without enclosing quotes. If the default value cannot be represented without truncation, then TRUNCATED is shown without enclosing quotes. If no default value was specified then its value is the NULL value. The value of COLUMN_DEF is syntactically suitable for use in specifying *default-value* in a CREATE TABLE or ALTER TABLE statement (except when TRUNCATED is shown).
IS_NULLABLE	VARCHAR(3)	One of: “NO” = the column is not nullable, according to the rules in the international standard “YES” = the NULL value is allowed in the column.

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

INTERVAL_PREC	INTEGER	For interval data types, this is the number of significant digits for the interval leading precision (see Section 4.3.3.2). For other data types it is the NULL value.
CHAR_SET_CAT	VARCHAR(128)	The name of the catalog containing the character set used by the column.
CHAR_SET_SCHEM	VARCHAR(128)	The name of the schema containing the character set used by the column.
CHAR_SET_NAME	VARCHAR(128)	The name of the character set used by the column.
COLLATION_CAT	VARCHAR(128)	The name of the catalog containing the collation used by the column.
COLLATION_SCHEM	VARCHAR(128)	The name of the schema containing the collation used by the column.
COLLATION_NAME	VARCHAR(128)	The name of the collation used by the column.
DOMAIN_CAT	VARCHAR(128)	The name of the catalog containing the domain used by the column.
DOMAIN_SCHEM	VARCHAR(128)	The name of the schema containing the domain used by the column.
DOMAIN_NAME	VARCHAR(128)	The name of the domain used by the column.
REMARKS	VARCHAR(254)	May contain descriptive information about the column.

INDEXES

The INDEXES system view lists the index columns created on tables or views to which the current ident, or PUBLIC, has SELECT access.

Column name	Data type	Description
TABLE_CAT	VARCHAR(128)	The name of the catalog containing the table or view.
TABLE_SCHEM	VARCHAR(128)	The name of the schema containing the table or view.
TABLE_NAME	VARCHAR(128)	The name of the base table on which the index column exists.
COLUMN_NAME	VARCHAR(128)	The name of the column of the base table and index.
INDEX_NAME	VARCHAR(128)	The unique name of the index.
ORDINAL_POSITION	INTEGER	The ordinal number of the column in the index. The ordering is determined by ordering of the columns in the CREATE INDEX statement. The numbering of the columns of the index starts at 1 and increases contiguously.
NON_UNIQUE	VARCHAR(3)	One of: “NO” = only one row is allowed in the table identified by TABLE_NAME for each combination of values in the columns of the index “YES” = the index is not a unique index.
ASC_OR_DESC	CHAR(1)	One of: “A” = the order of the referenced index column is ascending “D” = the order of the referenced index column is descending.
REMARKS	VARCHAR(254)	May contain descriptive information about the index.

SCHEMATA

The SCHEMATA system view lists all the schemas in the database.

Column name	**Data type**	**Description**
CAT_NAME	VARCHAR(128)	The name of the catalog containing the schema.
SCHEM_NAME	VARCHAR(128)	The name of the schema.
SCHEM_OWNER	VARCHAR(128)	The name of the ident who created the schema.
DEF_CHAR_SET_CAT	VARCHAR(128)	The name of the catalog that contains the default character set for the schema.
DEF_CHAR_SET_SCHEM	VARCHAR(128)	The name of the schema that contains the default character set for the schema.
DEF_CHAR_SET_NAME	VARCHAR(128)	The name of the default character set for the schema.
REMARKS	VARCHAR(254)	May contain descriptive information about the table.

SERVER_INFO

The SERVER_INFO system view lists the attributes of the database server to which the application is currently connected. Each row provides information about one attribute. The attribute is identified in the SERVER_ATTRIBUTE column and the value of the attribute for the current server is shown in the ATTRIBUTE_VALUE column.

Column name	**Data type**	**Description**
SERVER_ATTRIBUTE	VARCHAR(254)	The name of the server attribute. One of: CATALOG_NAME COLLATION_SEQ IDENTIFIER_LENGTH INTERVAL_FRACTIONAL_PRECISION INTERVAL_LEADING_PRECISION ROW_LENGTH TIME_PRECISION TIMESTAMP_PRECISION TXN_ISOLATION USERID_LENGTH
ATTRIBUTE_VALUE	VARCHAR(254)	The value of the corresponding server attribute. For CATALOG_NAME: “YES” = the server supports catalog names, otherwise “NO”. For COLLATION_SEQ: either “ISO 8859-1” or “EBCDIC” to indicate the assumed ordering of the character set for the server. For IDENTIFIER_LENGTH: the maximum number of characters allowed for a user-defined name, shown as the character string representation of the decimal value. For INTERVAL_FRACTIONAL_PRECISION: the default seconds precision for all interval data types with a seconds component. For INTERVAL_LEADING_PRECISION: the default leading precision for all intervals. For ROW_LENGTH: the maximum size of a row, shown as the character string representation of the decimal value. For TIME_PRECISION: the default seconds precision for all objects of data type TIME. For TIMESTAMP_PRECISION: the default seconds precision for all objects of data type TIMESTAMP. For TXN_ISOLATION: the initial transaction isolation level assumed by the server. One of: READ UNCOMMITTED READ COMMITTED REPEATABLE READ SERIALIZABLE. For USERID_LENGTH: the maximum number of characters of a user name (or “authorization identifier”), shown as the character string representation of the decimal value.

SQL_LANGUAGES

The SQL_LANGUAGES system view lists all the SQL standards and SQL dialects to which the SQL product claims conformance (including subsets defined by ISO and vendor-specific versions).

All versions can be shown in a single table. Each row has at least the columns described below, however additional columns may be specified for an individual vendor or specific international standard.

Column name	**Data type**	**Description**
SOURCE	VARCHAR(254)	The organization that defined the SQL version.
SOURCE_YEAR	VARCHAR(254)	The year the relevant source document was approved.
CONFORMANCE	VARCHAR(254)	The conformance level to the relevant document that the implementation claims.
INTEGRITY	VARCHAR(254)	(Meaning no longer defined).
IMPLEMENTATION	VARCHAR(254)	A character string, defined by the vendor, that uniquely defines the vendor's SQL product.
BINDING_STYLE	VARCHAR(254)	Included to envisage future adoption of direct, module or other binding styles.
PROGRAMMING_LANG	VARCHAR(254)	The host language for which the binding style is supported.

TABLE_PRIVILEGES

The TABLE_PRIVILEGES system view lists privileges that were granted by the current ident, and privileges that were granted to the current ident or to PUBLIC, on an entire table.

An entry in this view does not assert that the grantee holds the privilege on any specific column of the table, because privileges can be granted on a table and then revoked on individual columns. Information about privileges on specific columns is presented in the COLUMN_PRIVILEGES view.

Column name	**Data type**	**Description**
GRANTOR	VARCHAR(128)	The name of the ident who granted the privilege. For rows that describe the implicit privileges that a table creator has on each column in a table, the name "_SYSTEM" is shown.
GRANTEE	VARCHAR(128)	The name of the ident to whom the privilege was granted. Granting a privilege to PUBLIC will result in only one row (per privilege granted) in this view and the name "PUBLIC" will be shown.
TABLE_CAT	VARCHAR(128)	The name of the catalog containing the table.
TABLE_SCHEM	VARCHAR(128)	The name of the schema containing the table.
TABLE_NAME	VARCHAR(128)	The name of the table in question.
PRIVILEGE_TYPE	VARCHAR(20)	A value describing the type of the column privilege that was granted. One of: DELETE INSERT REFERENCES SELECT UPDATE. Rows where privilege type is REFERENCES or UPDATE only describe cases where the grantee was granted the privilege on the **entire** table. Where the GRANT statement granted REFERENCES or UPDATE privilege to specified columns of a table, no rows appear in TABLE_PRIVILEGES but there are rows in COLUMN_PRIVILEGES. Note that when multiple table privileges are granted to the same user at the same time (e.g. when the keyword "ALL" is used), multiple rows appear in this view (one for each privilege granted).
IS_GRANTABLE	VARCHAR(3)	One of: "YES" = the privilege is held with the WITH GRANT OPTION "NO" = the privilege is held without the WITH GRANT OPTION.
REMARKS	VARCHAR(254)	May contain descriptive information about the privilege.

TABLES

The TABLES system view lists the tables and views on which the current ident, or PUBLIC, holds one or more privileges (INSERT, DELETE or SELECT; or REFERENCES or UPDATE on one or more columns).

Column name	Data type	Description
TABLE_CAT	VARCHAR(128)	The name of the catalog containing the table or view.
TABLE_SCHEM	VARCHAR(128)	The name of the schema containing the table or view.
TABLE_NAME	VARCHAR(128)	The name of the table or view.
TABLE_TYPE	VARCHAR(20)	One of: “BASE TABLE” = the row describes a table “VIEW” = the row describes a view.
REMARKS	VARCHAR(254)	May contain descriptive information about the table.

TRANSLATIONS

The TRANSLATIONS system view lists the translations on which the current ident, or PUBLIC, has USAGE privilege.

The source character set is the character set to which the characters that are to be translated by the translation belong. The target character set is the character set to which the characters that are the result of the translation belong.

Column name	Data type	Description
TRANSLATION_CAT	VARCHAR(128)	The name of the catalog containing the translation.
TRANSLATION_SCHEM	VARCHAR(128)	The name of the schema containing the translation.
TRANSLATION_NAME	VARCHAR(128)	The name of the translation.
SRC_CHARSET_CAT	VARCHAR(128)	The name of the catalog containing the source character set.
SRC_CHARSET_SCHEM	VARCHAR(128)	The name of the schema containing the source character set.
SRC_CHARSET_NAME	VARCHAR(128)	The name of the source character set.
TGT_CHARSET_CAT	VARCHAR(128)	The name of the catalog containing the target character set.
TGT_CHARSET_SCHEM	VARCHAR(128)	The name of the schema containing the target character set.
TGT_CHARSET_NAME	VARCHAR(128)	The name of the target character set.
REMARKS	VARCHAR(254)	May contain descriptive information about the table.

USAGE_PRIVILEGES

The USAGE_PRIVILEGES system view lists privileges that were granted by the current ident, and privileges that were granted to the current ident or to PUBLIC, on character sets and collations.

Column name	Data type	Description
GRANTOR	VARCHAR(128)	The name of the ident who granted the privilege.
GRANTEE	VARCHAR(128)	The name of the ident to whom the privilege was granted. Granting a privilege to PUBLIC will result in only one row (per privilege granted) in this view and the name “PUBLIC” will be shown.
OBJECT_CAT	VARCHAR(128)	The name of the catalog that contains the object character set or collation.
OBJECT_SCHEM	VARCHAR(128)	The name of the schema that contains the object character set or collation.
OBJECT_NAME	VARCHAR(128)	The name of the character set or collation.
OBJECT_TYPE	VARCHAR(20)	One of: “CHARACTER SET” = the privilege is held on a character set “COLLATION” = the privilege is held on a collation.
PRIVILEGE_TYPE	VARCHAR(20)	This will always be “USAGE”.
IS_GRANTABLE	VARCHAR(3)	One of: “YES” = the privilege is held with the WITH GRANT OPTION “NO” = the privilege is held without the WITH GRANT OPTION.
REMARKS	VARCHAR(254)	May contain descriptive information about the privilege.

VIEWS

The VIEWS system view lists the viewed tables on which the current ident, or PUBLIC, holds one or more privileges (INSERT, DELETE or SELECT; or REFERENCES or UPDATE on one or more columns).

The views listed in this system view also appear in the TABLES system view.

Column name	Data type	Description
TABLE_CAT	VARCHAR(128)	The name of the catalog containing the table or view.
TABLE_SCHEM	VARCHAR(128)	The name of the schema containing the table or view.
TABLE_NAME	VARCHAR(128)	The name of the table or view.
VIEW_DEFINITION	VARCHAR(2000)	The definition of the view as it would appear in a CREATE VIEW statement. If the actual definition would not fit into the maximum length of this column, the NULL value will be shown.
CHECK_OPTION	VARCHAR(20)	The value “CASCADED” is shown if WITH CHECK OPTION was specified in the CREATE VIEW statement that created the view, and the value “NONE” is shown otherwise.
IS_UPDATABLE	VARCHAR(3)	One of: “YES” = the view is updatable “NO” = the view is not updatable.
REMARKS	VARCHAR(254)	May contain descriptive information about the view.

Standard compliance

The table below summarizes standards compliance concerning the views in INFO_SCHEM.

Standard	Compliance	Comments
X/Open-95 **SQL92**	YES	Fully compliant.

7.3 FIPS_DOCUMENTATION

The table below summarizes the system views that are part of the schema FIPS_DOCUMENTATION (view names appear in their unqualified form):

View name	Description
SQL_FEATURES	Lists each FIPS feature with details about Mimer SQL compliance.
SQL_SIZING	For each FIPS database construct, lists the minimum limits requirements for Entry level and Intermediate level compliance and the Mimer SQL limit.

SQL_FEATURES

The SQL_FEATURES system view describes Mimer SQL compliance with each FIPS feature.

Column name	Data type	Description
FEATURE_ID	SMALLINT	The FIPS identifier for the feature.
FEATURE_NAME	CHAR(50)	The name of the FIPS feature.
CLASSIFICATION	CHAR(12)	Shows the FIPS conformance level at which the feature first becomes required. One of: “TRANSITIONAL” “INTERMEDIATE” “FULL” “RDA”.
IS_SUPPORTED	CHAR(3)	Shows whether the FIPS feature is supported in Mimer SQL or not. One of: “YES” “NO”.
IS_VERIFIED	CHAR(3)	Shows whether Mimer SQL compliance has been externally verified or not. One of: “YES” “NO”.
FEATURE_COMMENTS	VARCHAR(500)	May contain descriptive information about Mimer SQL support for the FIPS feature.

SQL_SIZING

The SQL_SIZING system view describes the minimum limits requirements for FIPS Entry and Intermediate level support and shows the Mimer SQL limits.

Column name	Data type	Description
SIZING_ID	SMALLINT	The FIPS identifier for the sizing limit.
DESCRIPTION	CHAR(50)	The description of the database construct sizing limit.
ENTRY_VALUE	INTEGER	The default Entry level sizing limit.
INTERMEDIATE_VALUE	INTEGER	The default Intermediate level sizing limit.
VALUE_SUPPORTED	INTEGER	The sizing limit supported by Mimer SQL.
SIZING_COMMENT	VARCHAR(500)	May contain comments on the sizing limit.

Standard compliance

The table below summarizes standards compliance concerning the views in FIPS_DOCUMENTATION.

Standard	Compliance	Comments
X/Open-95 SQL92	YES	Fully compliant.

A RESERVED WORDS

The following keywords are reserved in Mimer SQL statements. They must be enclosed in quotation marks if they are to be used as SQL identifiers.

- ALL
- ALLOCATE
- ALTER
- AND
- ANY
- ARE
- AS
- AT
- AUTHORIZATION
- BEGIN
- BETWEEN
- BOTH
- BY
- CALL
- CASCADED
- CASE
- CAST
- CHARACTER
- CHECK
- CLOSE
- COLLATE
- COLUMN
- COMMIT
- CONNECT
- CONSTRAINT
- CORRESPONDING
- CREATE
- CROSS
- CURRENT
- CURRENT_DATE
- CURRENT_PATH
- CURRENT_TIME
- CURRENT_TIMESTAMP
- CURRENT_USER
- CURSOR
- DAY
- DEALLOCATE
- DECLARE
- DEFAULT
- DELETE
- DESCRIBE
- DISCONNECT
- DISTINCT
- DO
- DROP
- EACH[1]
- ELSE
- ELSEIF
- END
- EXCEPT
- EXECUTE
- EXISTS
- FALSE
- FETCH
- FOR
- FOREIGN
- FROM
- FULL
- FUNCTION
- GET
- GLOBAL
- GRANT
- GROUP
- HAVING
- HOLD[1]
- HOUR
- IDENTITY
- IF
- IN
- INDICATOR
- INNER
- INOUT
- INSERT
- INTERSECT
- INTERVAL
- INTO
- IS
- JOIN
- LARGE[1]
- LEADING
- LEAVE
- LEFT
- LIKE
- LOCAL
- LOCALTIME[1]
- LOCALTIMESTAMP[1]
- LOOP
- MATCH
- MINUTE
- MODULE
- MONTH
- NATIONAL
- NATURAL
- NEW[1]
- NOT
- NULL
- OF
- OLD[1]
- ON
- ONLY
- OPEN
- OR
- ORDER
- OUT
- OUTER
- OVERLAPS
- PRECISION
- PREPARE
- PRIMARY
- PROCEDURE
- REFERENCES
- REFERENCING[1]
- RELEASE[1]
- REPEAT
- RESIGNAL
- RETURN
- REVOKE
- RIGHT
- ROLLBACK
- ROW[1]
- SECOND
- SELECT
- SESSION_USER
- SET
- SIGNAL
- SOME
- SPECIFIC
- SQLEXCEPTION
- SQLSTATE
- SQLWARNING
- START[1]
- SYSTEM_USER
- TABLE
- THEN
- TIMEZONE_HOUR
- TIMEZONE_MINUTE
- TO
- TRAILING
- TRANSLATION
- TRIGGER[1]
- TRUE
- UNION
- UNIQUE
- UNKNOWN
- UNTIL
- UPDATE
- USER
- USING
- VALUE
- VALUES
- VARYING
- WHEN
- WHERE
- WHILE
- WITH
- WITHOUT[1]
- YEAR

Note: The keywords marked with “[1]” are reserved in the SQL99 standard.

In addition, the keyword END-EXEC is reserved specifically in statements embedded in COBOL programs.

To avoid keywords that may become reserved in future versions of Mimer SQL, the following list of potential reserved words should be noted:

ARRAY	FREE	REF	SPECIFICTYPE
ASYMMETRIC	GROUPING	ROLLUP	SYMMETRIC
CUBE	ITERATE	SAVEPOINT	TREAT
CURRENT_ROLE	LATERAL	SCOPE	UNNEST
CYCLE	METHOD	SEARCH	
DEREF	NONE	SIMILAR	

There is no guarantee, however, that all of these keywords will, in fact, become reserved words in a future version of Mimer SQL and it is also almost certain that additional words will be added to the list as future versions emerge.

An identifier in Mimer SQL can be protected against conflicting with potential reserved words by including a digit or an underscore in the identifier and by ensuring that it does not begin with "CURRENT_", "SESSION_", "SYSTEM_" or "TIMEZONE_".

The following keywords are reserved in SQL92 and SQL/PSM, but not in Mimer SQL.

ABSOLUTE	DESCRIPTOR	LOWER	SCHEMA
ACTION	DETERMINISTIC	MAX	SCROLL
ADD	DIAGNOSTICS	MIN	SECTION
ASC	DOMAIN	NAMES	SESSION
ASSERTION	DOUBLE	NCHAR	SIZE
AVG	ESCAPE	NEXT	SMALLINT
BIT	EXCEPTION	NO	SPACE
BIT_LENGTH	EXEC	NULLIF	SQL
CASCADE	EXIT	NUMERIC	SQLCODE
CATALOG	EXTERNAL	OCTET_LENGTH	SQLERROR
CHAR	EXTRACT	OPTION	SUBSTRING
CHAR_LENGTH	FIRST	OUTPUT	SUM
CHARACTER_LENGTH	FLOAT	PAD	TEMPORARY
COALESCE	FOUND	PARTIAL	TIME
COLLATION	GO	PARAMETER	TIMESTAMP
CONDITION	GOTO	PATH	TRANSACTION
CONNECTION	HANDLER	POSITION	TRANSLATE
CONSTRAINTS	IMMEDIATE	PRESERVE	TRIM
CONTAINS	INITIALLY	PRIOR	UNDO
CONTINUE	INPUT	PRIVILEGES	UPPER
CONVERT	INSENSITIVE	PUBLIC	USAGE
COUNT	INT	READ	VARCHAR
DATE	INTEGER	REAL	VIEW
DEC	ISOLATION	RELATIVE	WHENEVER
DECIMAL	KEY	RESTRICT	WORK
DEFERRABLE	LANGUAGE	RETURNS	WRITE
DEFERRED	LAST	ROWS	ZONE
DESC	LEVEL	ROUTINE	

B CHARACTER SET

The character set used by Mimer SQL is ISO 8859-1. Characters are sorted in numerical order of their codes.

	HIGH															
LOW	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	▨	▨	SP	0	@	P	`	p	▨	▨	NS	°	À	Ð	à	ð
1	▨	▨	!	1	A	Q	a	q	▨	▨	¡	±	Á	Ñ	á	ñ
2	▨	▨	"	2	B	R	b	r	▨	▨	¢	²	Â	Ò	â	ò
3	▨	▨	#	3	C	S	c	s	▨	▨	£	³	Ã	Ó	ã	ó
4	▨	▨	$	4	D	T	d	t	▨	▨	¤	´	Ä	Ô	ä	ô
5	▨	▨	%	5	E	U	e	u	▨	▨	¥	µ	Å	Õ	å	õ
6	▨	▨	&	6	F	V	f	v	▨	▨	¦	¶	Æ	Ö	æ	ö
7	▨	▨	'	7	G	W	g	w	▨	▨	§	·	Ç	×	ç	÷
8	▨	▨	(	8	H	X	h	x	▨	▨	¨	¸	È	Ø	è	ø
9	▨	▨	)	9	I	Y	i	y	▨	▨	©	¹	É	Ù	é	ù
A	▨	▨	*	:	J	Z	j	z	▨	▨	ª	º	Ê	Ú	ê	ú
B	▨	▨	+	;	K	[	k	{	▨	▨	«	»	Ë	Û	ë	û
C	▨	▨	,	<	L	\	l	|	▨	▨	¬	¼	Ì	Ü	ì	ü
D	▨	▨	-	=	M	]	m	}	▨	▨	SH	½	Í	Ý	í	ý
E	▨	▨	.	>	N	^	n	~	▨	▨	®	¾	Î	Þ	î	þ
F	▨	▨	/	?	O	_	o	▨	▨	▨	¯	¿	Ï	ß	ï	ÿ

SP	=	space
NS	=	non-breaking space
SH	=	soft hyphen

C LIMITS

The limits specified for X/Open-95 are lower bounds. Compliant systems should support at least the specified number. If a box is empty this means the standard does not specify a limit.

LIMIT	**Mimer SQL**	**X/Open-95**
Character string length	15000	254
Variable length character string	15000	254
BINARY string length	15000	254
Variable BINARY string length	15000	254
DECIMAL precision in digits	45	15
FLOAT precision in digits	45	47 bits
SMALLINT precision in digits	5	5
INTEGER precision in digits	10	10
BIGINT precision in digits	19	
REAL precision in digits	7	21 bits
DOUBLE PRECISION precision in digits	16	47 bits
SQLCODE precision in digits	10	9
Fractional precision, in decimal digits, of TIME seconds component	9	6
Fractional precision, in decimal digits, of TIMESTAMP seconds component	9	6
Fractional precision, in decimal digits, of INTERVAL seconds component	9	6
Leading precision, in decimal digits, for INTERVAL data types	7-12	7
Fractional precision for INTERVAL data types that have a SECOND field	9	6
Number of columns in a table	252	100
Number of columns in an index	32	6
Number of tables referenced in a statement	unlimited *)	10

LIMIT	Mimer SQL	X/Open-95
Number of cursors open simultaneously	unlimited **)	10
Number of columns a single update statement can update	unlimited *)	20
Number of nested subqueries	15	9
SQL statement length in bytes	32767	4000
Total length of a row, the sum of • Two bytes for column identification • Lengths of all character fields • Precisions of all numeric fields	16000	2000
Identifier length	128	18

*) There is a limit on the total complexity of an SQL statement. The specified X/Open limits can safely be handled by Mimer SQL.

**) The limit is only dependent on the amount of virtual memory available to the application.

D DEPRECATED FEATURES

Some non-standard features of earlier versions of Mimer SQL are retained for backward compatibility. Where these features have equivalents in the standard implementation, only the standard form is documented in the main body of this manual. Non-standard forms are documented below. Use of the standard forms is strongly recommended in new applications.

Non-standard Mimer SQL features which do not have equivalents in the standards are documented in the main body of the manual.

D.1 Host variables

D.1.1 NULL values

Host variables with negative indicator variables are assigned NULL. In previous versions of Mimer SQL, the indicator variable could be given any negative value. The accepted standards however define the value of -1 as an indicator for NULL and reserve other negative values for future use. Use of values other than -1 is therefore not recommended.

D.2 Operators

Operator	**Standard form**	**Alternative form(s)**
string concatenation	||	&, !!
not equal to	<>	/=, !=, ¬=
less than or equal to	<=	¬>
greater than or equal to	>=	¬<

Note: The character used as the “not” sign (¬) is usually a circumflex (^) in standard ASCII character sets.

D.3 Statements

D.3.1 CONNECT

The syntax of the standard CONNECT statement differs from that in earlier versions of Mimer SQL. The previous form is still supported for backward compatibility.

Backward compatibility syntax:

```
►── CONNECT ident-name IDENTIFIED BY password ──────────────────────►
```

Note: The CONNECT statement and the standard-compliant CONNECT TO statement have different default modes for SET TRANSACTION START. CONNECT uses SET TRANSACTION START EXPLICIT, while CONNECT TO uses SET TRANSACTION START IMPLICIT.

D.3.2 SELECT NULL

The use of the keyword NULL in a select list is still supported for backward compatibility but should be regarded as a deprecated feature in Mimer SQL.

D.3.3 INCLUDE SQLCA

Prior to version 7.2 of Mimer SQL, an INCLUDE SQLCA statement was required in an embedded SQL program, to include the declaration of the SQL communication area. The INCLUDE SQLCA statement is now no longer required. Instead applications can use the SQLSTATE variable and the GET DIAGNOSTICS statement to get the information that was contained in SQLCA in the earlier Mimer SQL versions. See Appendix E for a description of SQLSTATE. For compatibility reasons the use of SQLCA is still supported.

D.3.4 SET TRANSACTION

Following the introduction of the SET TRANSACTION READ and SET TRANSACTION ISOLATION LEVEL options, the SET TRANSACTION CHANGES options no longer apply.

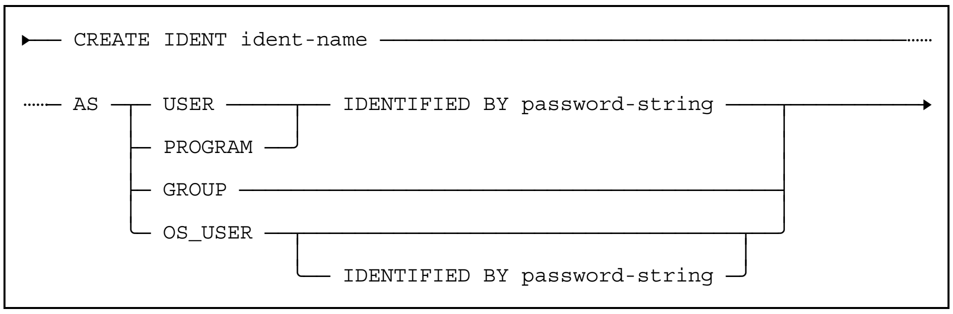
The following options are supported for backward compatibility only:

```
►── SET TRANSACTION ──── CHANGES ─┬─ INVISIBLE ─┬───────────────────────►
                                  └─ VISIBLE ───┘
```

D.3.5 CREATE IDENT

The keyword USING is now used when specifying the *password-string* for the ident being created. The use of IDENTIFIED BY is deprecated.

The following syntax is supported for backward compatibility only:



```
►── CREATE IDENT ident-name ──────────────────────────────────────······

······─ AS ─┬─ USER ─────┬─ IDENTIFIED BY password-string ─┬──────────►
            ├─ PROGRAM ──┘                                 │
            ├─ GROUP ──────────────────────────────────────┤
            └─ OS_USER ─┬─────────────────────────────────┬┘
                        └─ IDENTIFIED BY password-string ─┘
```

D.3.6 ALTER IDENT

The keyword USING is now used when specifying the *password-string* for the ident being altered. The use of IDENTIFIED BY is deprecated.

The following syntax is supported for backward compatibility only:

```
►── ALTER IDENT ident-name IDENTIFIED BY password-string ──────────►
```

D.3.7 ENTER

The keyword USING is now used when specifying the *password* for the program ident being entered. The use of IDENTIFIED BY is deprecated.

The following syntax is supported for backward compatibility only:

```
►── ENTER ident-name IDENTIFIED BY password ──────────────────────►
```

D.3.8 SELECT

The option of specifying a non-zero integer in the ORDER BY clause of a SELECT statement which represents the ordinal number of a column in the result table of a SELECT statement is a deprecated feature.

The deprecated syntax is still supported for backward compatibility. Its continued use, however, is discouraged because a future version of Mimer SQL will support a syntax allowing the specification of an expression in the same place as the integer, which will have incompatible semantics.

D.4 Communication areas

D.4.1 SQLDA

The SQL descriptor area SQLDA, which was used in earlier versions of Mimer SQL, has now been replaced by a standardized SQL descriptor area. The SQLDA area was allocated and maintained by constructions in the host language. The new SQL descriptor area is allocated and maintained by standardized embedded SQL statements.

The former SQL descriptor area SQLDA is still supported in Mimer SQL for compatibility reasons.

D.4.2 SQLCODE

The use of SQLCODE to retrieve status information has been replaced by the use of SQLSTATE and GET DIAGNOSTICS in Mimer SQL. For compatibility reasons, return codes can still be retrieved in SQLCODE, as in earlier versions of Mimer SQL. SQLCODE can now either be a field in the SQLCA, or a 4-byte integer variable in the application. Mimer SQL will assume the existence of an SQLCODE variable in the application if no INCLUDE SQLCA statement is found and neither SQLSTATE nor SQLCODE is declared between BEGIN DECLARE SECTION and END DECLARE SECTION.

The values of SQLCODE are the same as the values for the internal Mimer SQL return codes described in Appendix B of the Mimer SQL Programmer's Manual.

D.5 Program idents

D.5.1 MIMER_SW

The right to create and manage databank shadows was previously granted by having EXECUTE on the program ident MIMER_SW. This has now been replaced by the SHADOW system privilege.

D.5.2 MIMER_BR

The right to perform backup and restore operations was previously granted by having EXECUTE on the program ident MIMER_BR. This has now been replaced by the BACKUP system privilege.

D.5.3 MIMER_SC

The right to execute the UPDATE STATISTICS statement was previously granted by having EXECUTE on the program ident MIMER_SC. This has now been replaced by the STATISTICS system privilege.

D.6 Datetime scalar functions

The following scalar functions have been deprecated in the current version of Mimer SQL. They will be re-introduced in a future version with changed functionality.

D.6.1 CURRENT_TIME

This function, in its present form, has been deprecated. The same functionality is now provided by the LOCALTIME function. Continued use of CURRENT_TIME is not recommended as the function will be re-introduced, with changed functionality, in a future version of Mimer SQL and thus existing usage will not be forward-compatible.

D.6.2 CURRENT_TIMESTAMP

This function, in its present form, has been deprecated. The same functionality is now provided by the LOCALTIMESTAMP function. Continued use of CURRENT_TIMESTAMP is not recommended as the function will be re-introduced, with changed functionality, in a future version of Mimer SQL and thus existing usage will not be forward-compatible.

D.7 Data dictionary views

The predefined system views on the data dictionary tables, owned by the system ident called **MIMER**, that were supported in previous versions of Mimer SQL have now been replaced by the standard system views belonging to the schemas INFORMATION_SCHEMA and INFO_SCHEM which are owned by ident SYSTEM.

E RETURN STATUS AND CONDITIONS

In Mimer SQL, status information is returned in the SQLSTATE variable. This variable must be declared as a 5-character long string (excluding any terminating null byte), and the declaration of SQLSTATE must be made between BEGIN DECLARE SECTION and END DECLARE SECTION. The SQLSTATE variable together with the GET DIAGNOSTICS statement will provide the application with information about the most recently executed SQL statement.

For compatibility reasons, error codes can also be returned in SQLCODE, as in earlier versions of Mimer SQL. However, SQLCODE in Mimer SQL can be either a field in the SQLCA, or a 4-byte integer variable in the application. Mimer SQL will assume the existence of an SQLCODE variable if no INCLUDE SQLCA statement is found and neither SQLSTATE nor SQLCODE is declared between BEGIN DECLARE SECTION and END DECLARE SECTION.

There are three different types of conditions in Mimer SQL: NOT FOUND, SQLERROR and SQLWARNING (see the WHENEVER statement in Chapter 6).

SQLERRORs are returned using an error code in SQLSTATE or a negative value in the SQLCODE field. This is referred to as “an error code is returned” in this manual.

SQLWARNINGs are returned by setting either a “Success with warning”-code in SQLSTATE, or by setting the SQLWARN-fields in the SQLCA to the character “W”. This is referred to as “a warning flag is set” in this manual.

The NOT FOUND condition is returned by setting SQLSTATE to “No data” ('02000'), or by setting SQLCODE to +100. This is referred to as “a NOT FOUND condition code is returned” in this manual.

More detailed information about certain operations can be obtained by the GET DIAGNOSTICS statement described in Chapter 6 in this manual.

E.1 SQLSTATE return codes

SQLSTATE contains a 5 character long return code string that indicates status of an SQL statement. These return codes follows the established standards. Observe that not all standardized SQLSTATE return codes are used by Mimer SQL.

The SQLSTATE values consists of two fields: the class, which is the first two characters of the string, and the subclass, which is the terminating three characters of the string.

List of SQLSTATE values:

Class	Subclass	Meaning
00	000	**Successful completion**
01	000	**Warning**
01	002	- Disconnect error
01	003	- Null value eliminated in set function
01	004	- String data, right truncation
01	005	- Insufficient item descriptor areas
01	006	- Privilege not revoked
01	007	- Privilege not granted
01	008	- Implicit zero-bit padding
01	009	- Search condition too long for information schema
01	00A	- Query expression too long for information schema
01	00B	- Default option too long for information schema
02	000	**No data**
07	000	**Dynamic SQL error**
07	001	- *using clause* does not match dynamic parameter specifications
07	002	- *using clause* does not match target specification
07	003	- Cursor specification cannot be executed
07	004	- *using clause* required for dynamic parameters
07	005	- Prepared statement is not a cursor specification
07	006	- Restricted data type attribute violation
07	007	- *using clause* required for result fields
07	008	- Invalid descriptor count
07	009	- Invalid descriptor index
07	00F	- Undefined DATETIME_INTERVAL_CODE

Class	Subclass	Meaning
08	000	**Connection exception**
08	001	- Client unable to establish connection
08	002	- Connection name in use
08	003	- Connection does not exist
08	004	- Server rejected the connection
08	006	- Connection failure
09	000	**Trigger action exception**
0A	000	**Feature not supported**
0K	000	**Resignal when handler not active**
20	000	**Case not found for a case statement**
21	000	**Cardinality violation**
22	000	**Data exception**
22	001	- String data, right truncation
22	002	- Null value, no indicator parameter
22	003	- Numeric value out of range
22	005	- Error in assignment
22	006	- Invalid interval format
22	007	- Invalid datetime format
22	008	- Datetime field overflow
22	011	- Substring error
22	012	- Division by zero
22	015	- Interval field overflow
22	018	- Invalid character value for cast
22	019	- Invalid escape character
22	023	- Invalid parameter value
22	024	- Unterminated C string
22	025	- Invalid escape sequence
22	026	- String data, length mismatch
22	027	- Trim error
23	000	**Integrity constraint violation**
24	000	**Invalid cursor state**
25	000	**Invalid transaction state**
26	000	**Invalid SQL statement name**
27	000	**Triggered data change violation**
28	000	**Invalid authorization specification**
2E	000	**Invalid connection name**

Class	Subclass	Meaning
2F	000	**SQL routine exception**
2F	003	- Prohibited SQL-statement attempted
2F	005	- Function executed no return statement
33	000	**Invalid SQL descriptor name**
34	000	**Invalid cursor name**
35	000	**Invalid condition number**
3C	000	**Ambiguous cursor name**
40	000	**Transaction rollback**
40	001	- Serialization failure
40	003	- Statement completion unknown
42	000	**Syntax error or access rule violation**
44	000	**WITH CHECK OPTION violation**
45	000	**Unhandled user-defined exception**
S1	000	**General error**
S1	001	- Memory allocation failure

INDEX