Contents

Manual Ordering

Manual Order Number: ESD611-030WOU

This online help is applicable to ADABAS D Version 6.1.1 PE and to all subsequent releases, unless otherwise indicated in new editions or technical newsletters.

Specifications contained herein are subject to change and these changes will be reported in subsequent revisions or editions.

Readers' comments are welcomed. Comments may be addressed to the Documentation Department at the address on the back cover.

Introduction

This online help defines the syntax and semantics of the SQL statements of ADABAS D. An SQL statement performs an operation on an ADABAS database. The used parameters are host variables of a programming language in which the SQL statements are embedded.

The chapter 'Data Type' explains the principles upon which the ADABAS database system is based.
Then follows an explanation of the '<character>' which are used in the SQL statements.
The chapter '<create table statement>' describes the SQL statements for the definition of tables etc.
The chapter '<create user statement>' explains the protective mechanisms against illegal access and illegal modifications to the data.
The chapter 'Data Manipulation' describes the SQL statements for the insertion, update, and deletion of data.
The chapter 'Data Retrieval' deals with the SQL statements for data access.
The chapter 'Transactions' deals with the mechanisms for the maintenance of the consistency as well as for the synchronization of the ADABAS server.
The chapter 'System Tables' describes the view tables that contain information about the database objects and their relationships to each other and to programs.
The chapter 'Statistics' describes the possibilities that are available to a user for obtaining statistical information on the size of database objects as well as the frequency of specific events.
The chapter 'Restrictions' lists the restrictions which generally apply to data types, parameters, identifiers, etc.
The chapter 'Compatibility with Former Versions' specifies the SQL statements or parts of SQL statements that are still accepted for ensuring the compatibility with previous versions but which should no longer be used in new application programs. In older application programs, they should be replaced bit by bit by the corresponding new syntax.
The chapter 'Syntax' contains all syntax rules listed in alphabetical order.

The syntax notation used in this online help is BNF, with the following conventions:

Keywords are shown in uppercase letters for illustration purposes only. They can be specified in uppercase or lowercase letters.

```
<xyz>
```

Terms enclosed in angle brackets are syntactical units that are explained in this online help. The chapter 'Syntax' contains a list of the syntactical units in alphabetical order.

```
clause ::= rule
```

The SQL statements consist of clauses. The rules describe how simple clauses are assembled into more complex ones and their notation.

```
clause1  clause2
```

The two clauses are written one after the other, separated by at least one blank.

`[clause]`

Optional clause: may be omitted without substitution.

`clause1 | clause2 | ... | clausen`

Alternative clauses: only one can be used.

`clause,...`

The clause can be repeated as often as is desired. The individual repetitions must be written one after the other, separated from each other by a comma and any number of blanks.

`clause...`

The clause can be repeated as often as is desired. The individual repetitions must be written directly one after the other without a separating comma or blank.

Data Type

1. A data type is a set of values that can be represented.

2. A value is either a NULL value (undefined value), or the special NULL value, or a non-NULL value.

3. The NULL value is a special value. The comparison of the NULL value with all values is undefined.

4. A special NULL value is a special value which may occur in arithmetical operations when these lead to an overflow or a division by 0. The comparison of a special NULL value with any value is always undefined.

5. A non-NULL value is a character string, a number, a date value, a time value, a timestamp value, or a value of a LONG column.

See also

Character String

LONG Column

Number

Date Value

Time Value

Timestamp Value

Boolean

Character String

1. A character string is a series of alphanumeric characters. The maximum length of a character string is 254 characters.

2. Each character string has a code attribute (ASCII, EBCDIC, or BYTE). It defines the sort sequence to be used when comparing the values of this column.

3. All character strings with the same code attribute can be compared to each other. Character strings with the different code attributes ASCII and EBCDIC can be compared to each other. Character strings with the code attributes ASCII and EBCDIC can be compared to date, time, and timestamp values.

LONG Column

1. A LONG column contains a sequence of characters of any length to which no functions can be applied.

2. LONG columns cannot be compared to each other. The contents of LONG columns cannot be compared to character strings or other data types.

Number

1. There are fixed point and floating point numbers.

2. A fixed point number is described by the number of significant digits and the scale. The maximum number of significant digits is 18.

3. A floating point number consists of a mantissa and an exponent. The mantissa may have up to 18 significant digits. The valid range of values for floating point numbers consists of the intervals from -9.99999999999999999E+62 to -1E-64 and from +1E-64 to +9.99999999999999999E+62 and the value 0.0.

4. All numbers can be compared to each other.

Date Value

1. A date value is a special character string. A date value can be compared to other date values and to character strings with the code attributes ASCII and EBCDIC.

Time Value

1. A time value is a special character string. A time value can be compared to other time values and to character strings with the code attributes ASCII and EBCDIC.

Timestamp Value

1. A timestamp value is a special character string. A timestamp consists of a date and time value and a microsecond specification. A timestamp value can be compared to other timestamp values and to character strings with the code attributes ASCII and EBCDIC.

Boolean

1. A Boolean is a data type which can assume one of the states TRUE and FALSE and the NULL value. A Boolean value can only be compared to other Boolean values.

Parameter

1. SQL statements for ADABAS can be embedded in programming languages such as COBOL and C, thus allowing the database to be accessed from application programs. The values to be retrieved from or to be stored in the database can be passed within the SQL statements using parameters. The parameters are declared variables (the so-called host variables) within the embedding program.

2. The data type of the host variables is defined when declaring the variables in the programming language. Values of host variables are implicitly converted from the programming language data type to the ADABAS data type, and vice versa, if possible.

3. Each parameter can be combined with an indicator parameter that indicates irregularities (such as differing lengths of value and parameter, NULL value, special NULL value, etc.) that may have occurred during the assignment of values. For the transfer of NULL values and special NULL values, indicator parameters are indispensable. The indicator parameters are declared variables (the so-called indicator variables) within the embedding program.

4. More details about the embedding of SQL statements for ADABAS in programming languages are provided in the precompiler online help.

Table

1. A table is a set of rows.

2. A row is an ordered list of values. The row is the smallest unit of data which can be inserted into or deleted from a table.

3. Each row of a table has the same number of columns and contains a value for each column.

4. A base table is a table which usually has a permanent memory representation and description.
   It is also possible to create a base table which has only a temporary memory representation and description. This table and its description are implicitly dropped when a user's work with the database system is terminated (session end).

5. A result table is a temporary table which is generated from one or more base table(s) by executing a SELECT statement.

6. A view table is a table derived from base tables. A view table has a permanent description in the form of a SELECT statement.

7. A snapshot table is a table derived from base tables. A snapshot table has a permanent memory representation and description. To update the snapshot table with the values from the base tables, the REFRESH statement can be used.

8. Each table has a name that is unique within the whole database. To name result tables, names of existing tables can be used, but the original tables cannot be accessed as long as the result tables exist.

9. If the qualification of the user name is missing for a table name specification, first the partial catalog of the current user, then the partial catalog of the DBA who created the current user, and subsequently the partial catalog of the SYSDBA of the current user is scanned for the specified table name. Finally, the catalog part of the owner of the system tables is scanned, if required. A table of another user can only be used when the corresponding privileges have been granted.

Column

1. All values in a table column have the same data type. A value of a column in a row is the smallest unit of data that can be modified or selected from a table or to which functions can be applied.

2. All character strings in an alphanumeric column have the same length.

3. A numeric column is either a floating point column or a fixed point column. All numbers in a fixed point column have the same format; i.e., the same number of digits before and after the decimal point. All numbers in a floating point column have the same mantissa length.

4. Each column in a base table has a name that is unique within the table.

Domain

1. Domain definitions allow ranges of values to be defined and named for table columns.

2. Each domain definition has a name that is unique within the whole database.

3. If the qualification of the user name is missing for a domain specification, first the catalog part of the current user, then the catalog part of the DBA who created the current user, and at last the catalog part of the SYSDBA of the current user is scanned for the specified domain.

Index

1. Indexes serve to speed up the access to rows of a table. They can be created for a single column or for a sequence of columns. When defining indexes, it is necessary to specify whether the column values of different rows in the indexed columns must be unique or not.

2. A given index name, along with the table name, must be unique.

Synonym

1. A synonym is another name for a table.

2. Every synonym has a name that is unique within the whole database and differs from all the other table names.

User and Usergroup

1. When installing the system, user name/password combinations are defined.

   a) The CONTROLUSER
   controls and monitors the system. He is responsible for backing up the database. For these tasks, the ADABAS component CONTROL has been provided.

   b) The SYSDBA (system database administrator)
   installs the system; i.e., his tasks include creating user accounts. The position of the SYSDBA within the hierarchy of user classes is described in 2d below.

   c) The DOMAINUSER
   maintains the system tables. His name is always DOMAIN. Any password can be chosen.

   For the installation of the system, see the CONTROL online help.

2. There are four hierarchical classes of users in WARM database mode:

   a) STANDARD users
   can only access existing tables for which they have received privileges. For these tables, they can create synonyms and view tables. A STANDARD user can only create temporary tables.

   b) RESOURCE users
   have all the rights of a STANDARD user. In addition, they can create private tables and grant privileges for them.

   c) Database administrators (DBA)
   are responsible for the organization of the database system. The DBA has all the rights of a RESOURCE user. Database administrators can create RESOURCE users and STANDARD users.

   d) The system database administrator (SYSDBA)
   installs the system. The system database administrator has all the rights of a DBA. In addition, he can create users with DBA status.
   In a non-distributed database, there is only one SYSDBA.

3. It is possible to create usergroups. All members of a usergroup have the same rights on the data that is assigned to the usergroup.

4. Users can only be defined in the SQLMODEs ADABAS and ORACLE; usergroups can only be defined in SQLMODE ADABAS.

Privilege

1. A privilege is used for imposing restrictions on operations on certain objects.

2. Every user can grant privileges to other users for objects owned by him. Privileges on view tables may only be granted to other users when the user is the owner of the tables on which the view table is based, or when the user has the right to grant the privileges for the base tables to other users. Generally, a user is the owner of an object when he has created it.

3. Users with DBA or RESOURCE status can perform all operations on database objects that they own. The set of possible operations may be restricted for view tables, because not all view tables are updatable. If the user is the owner of a view table but not of all tables on which the view table is based, the set of operations allowed on this view table depends on the set of privileges granted to the user for the tables on which the view table is based. Moreover, users with DBA or RESOURCE status can perform operations on all objects for which they have received the corresponding privileges.

4. STANDARD users can only perform operations on objects if they have received the privileges to do so.

Database

1. A database consists of the catalog and the user data.

2. The catalog consists of metadata. The definitions of database objects such as base tables, view tables, synonyms, domains, indexes, users and usergroups are stored there.

3. The catalog consists of several parts. One part comprises information about the installation of the (distributed) database and the metadata with the definitions of users and usergroups. This part is not assigned to a user or usergroup.
   The catalog contains a part for each user or usergroup where the metadata for the objects, such as base tables, view tables, etc., created by the user or usergroup is stored.

4. A user can only access the metadata of another user or usergroup when he has received the privileges to do so.

5. All rows of all base tables are the user data of a database.

6. If a non-distributed database is concerned, SERVERDB designates the whole database.

Distributed Database

1. A distributed database consists of two or more SERVERDBs which have a common catalog and common user data.

2. There is one system database administrator (SYSDBA) on each SERVERDB. The SYSDBA may drop all users of this SERVERDB, even those not created by him.

3. Each user is assigned one of these SERVERDBs as a HOME SERVERDB. The user data in the base tables that are owned by the user, as well as the partial catalog of the user, are always stored on the HOME SERVERDB of this user.

4. The catalog consists of one part that is copied to all SERVERDBs and of another part that is only stored on one SERVERDB.

5. The partial catalog stored on all SERVERDBs contains the definitions of SERVERDBs, domains, users, usergroups, of base and view tables that where defined 'WITH REPLICATION' or that are based on base tables defined 'WITH REPLICATION'.

6. The partial catalog that is only stored on one SERVERDB comprises the partial catalogs of all users and usergroups for which this SERVERDB is the HOME SERVERDB. These partial catalogs describe all database objects defined by these users and usergroups, except for the set specified in item 5.

7. A table name specification does not contain any specification of the SERVERDB to which the table is assigned. SQL statements are independent of the SERVERDB to which a user or table is assigned. Each SQL statement can be executed from any SERVERDB as long as all SERVERDBs are in WARM mode and network communication between the SERVERDBs is possible. If one of these requirements is not met, the following conditions apply.

8. (Metadata) Data stored on one SERVERDB can only be modified when this SERVERDB is in WARM mode. If the session (see chapter 0) of the user who wants to make these modifications was not started on the SERVERDB where the data to be modified is stored, the two SERVERDBs must be connected to each other within the network.

9. (Metadata) Data stored on all SERVERDBs can be modified even if not all SERVERDBs are in WARM mode or if network communication to some SERVERDBs is interrupted. SERVERDBs that are shut down or not accessible within the network are informed about modifications to the database as soon as they are put into WARM mode by using the Operating / Restart / Warm menu function of the ADABAS component CONTROL or when network communication has been reestablished.

10. Special processing is done if the network of SERVERDBs has split into two subnetworks which can no longer communicate with each other within the network. (Metadata) Data stored on a SERVERDB contained in one of the subnetworks can be modified from any SERVERDB belonging to that subnetwork. (Metadata) Data stored within the other subnetwork cannot be modified.

11. To prevent the two subnetworks from contradictory modifications to the replicated (metadata) data, ADABAS determines the subnetwork with the greater number (the so-called majority) of SERVERDBs within the whole network. The subnetwork containing the majority is then allowed to modify the (metadata) data. This procedure is called the majority concept. For two subnetworks of equal size, ADABAS decides

the one that is to represent the majority. The minority subnetwork is not allowed to modify replicated data. In the case of read-accesses, it may happen that the minority subnetwork does not receive the latest state of (metadata) data updated by the majority. ADABAS displays warnings to inform the user about such a state.

12. Information about which SERVERDBs belong to the majority is contained in the corresponding system tables.

Transaction

1. A transaction is a sequence of database operations which form a unit with regard to data backup and synchronization. Transactions are closed with COMMIT or ROLLBACK. If a transaction is closed with COMMIT, all modifications made to the database within the transaction are kept. If a transaction is aborted with ROLLBACK, all modifications made to the database within this transaction are cancelled, even those terminated with SUBTRANS END (see 'Subtransaction'). Modifications closed with COMMIT cannot be cancelled with ROLLBACK.
   COMMIT and ROLLBACK implicitly open a new transaction.

2. ADABAS distinguishes between SHARE and EXCLUSIVE locks. SHARE locks prevent locked tables or table rows from being modified by other users, although read access is still possible. EXCLUSIVE locks prevent the locked data objects from being read or modified by other users, while the user who has specified the lock can modify the objects.

3. The locking of tables and table rows within a transaction is done with a lock mode determined when the user connects to ADABAS.

Subtransaction

1. The purpose of closed, nested transactions (subtransactions) is to let a series of database operations within a transaction appear as a unit with regard to modifications to the database.

2. Subtransactions are preceded by SUBTRANS BEGIN and closed by SUBTRANS END or SUBTRANS ROLLBACK.
   If a subtransaction is concluded with SUBTRANS END, the performed modifications are kept.
   If a subtransaction is closed with SUBTRANS ROLLBACK, all modifications made to the database are cancelled. Modifications made by subtransactions contained in this subtransaction are cancelled as well, even if they have been concluded with SUBTRANS END.

3. SUBTRANS END and SUBTRANS ROLLBACK have no influence on locks. These are only released by COMMIT or ROLLBACK. COMMIT or ROLLBACK implicitly close all subtransactions.

Session

1. When a user is defined, a password is assigned to him. To be able to work with a database, a combination of user name and password known to the database must be specified.

2. The user is given access to the database if the combination of user name and password is valid. The user opens a session and the first transaction.
   A user can only work with the database within a session. A session is terminated explicitly by the user.

3. The user name specified in order to get access to the database is called the 'current user' if the user is not a member of a usergroup. If the user is a member of a usergroup, then the name of the usergroup is called the 'current user'.

Data Integrity

1. ADABAS provides a rich choice of declarative integrity rules, thus simplifying the programming of applications.

2. A key consisting of one or more columns can be defined for each table. ADABAS ensures that keys in a table are unique. A key can be composed of columns of different data types.

3. In addition, uniqueness can be enforced for the values of other columns or column combinations (UNIQUE definition for 'alternate keys').

4. For single columns, values other than the NULL value can be enforced by specifying NOT NULL.

5. For each column, a value can be predefined (DEFAULT definition).

6. The specification of declarative integrity rules with regard to one table is possible.

7. Declarations of referential integrity constraints for delete and existence conditions between the rows of two tables can be made as well.

8. Complex integrity rules requiring access to more tables can be formulated using triggers or DB procedures.

DB Procedure

1. In a well structured ADABAS application, the SQL statements are typically not distributed over the entire application but are concentrated in a single access layer. This access layer has a procedural interface with the rest of the application at which the operations for application objects are made available in form of abstract data types.

2. In client server configurations, there is an interaction between client and server when executing any SQL statement in the access layer.

3. The number of these interactions can be drastically reduced when the SQL access layer is no longer run on the client but on the server.
   ADABAS provides a language for this purpose which allows an SQL access layer to be formulated on the server side.

4. This has three main advantages:

   - The number of interactions between client and server is reduced by several factors. Client-server communication is only required for each operation on the application object, not for each SQL statement. This enhances the performance of client-server configurations considerably.

   - The second advantage has to do with software engineering. The SQL access layer contains the procedurally formulated integrity and business rules. Their concentration on the server side and their elimination from the ADABAS applications have the effect that modifications to these rules can be made at a central place, immediately becoming valid for all ADABAS applications. In this way, the integrity and decision rules become a part of the database catalog.

   - An SQL access layer in the form of DB procedures transferred to the server side is an essential customizing tool, because it allows customer-specific database functionality to be provided.

5. To be able to perform a DB procedure, a user must have the call privilege for it. This call privilege is independent of the privileges granted to the user for the tables and columns used within the DB procedure. Therefore, a user may be able to execute SQL statements using a DB procedure, but cannot do so outside the DB procedure.

6. DB procedures are called explicitly from the programming language of the application. DB procedures can contain parameters, except for LONG columns. In a DB procedure, all SQL statements (DDL and DML) are available without any restrictions. The extent to which LONG columns can be used within DB procedures depends on the length of the LONG columns and the storage space available.
   The call of further DB procedures is supported.

7. For the call of a DB procedure, as for any SQL statement, it must be ensured that there are the desired effects in case of success and that there remain no effects in the database if errors occur. ADABAS provides nested transactions for this purpose. Each call of a DB procedure can be executed within a subtransaction which can be reset without interfering with the transaction control of the ADABAS application.

8. For the syntax and semantics of DB procedures, refer to the manual of the corresponding ADABAS component.

Trigger

1. While DB procedures are called explicitly from the programming language of an application, triggers are specialized procedures that run implicitly on a base table or a view table built on this base table after executing a DML statement.

2. The conditions under which a trigger is to be executed can be restricted further.

3. The trigger is executed for each row to which the SQL statement refers. The trigger code can access both the old values of the row (values before update or deletion) and the new values (values after update or insertion).

4. A trigger can call other triggers implicitly and DB procedures explicitly.

5. Triggers can be used to check complicated integrity rules, to initiate derived database modifications for this or other rows or to implement complicated rules for access protection.

6. For the programming of triggers, refer to the manuals of the corresponding ADABAS components.

DB Function

1. DB functions are specialized procedures having any number of input parameters but just one output parameter. The output parameter is the result of the function, thus also defining the data type of the function's result.

2. In SQL statements, DB functions can be used like predefined functions. DB functions can be used to transfer functionality from the application programming to the ADABAS server. If DB functions are used in search conditions, the size of the result, if any, can be decreased considerably. This reduces both the storage space required by the result and the overhead to transfer the result into the application program.

3. DB functions can be used in all SQLMODEs, except ANSI. They can be nested with predefined functions and DB functions.

4. Names of DB functions should differ from the names of predefined functions in any of the SQLMODEs. If a predefined function is available in an SQLMODE, the predefined function is used, not the DB function.

5. No SQL statements are valid within a DB function.

6. For the programming of DB functions, refer to the manuals of the corresponding ADABAS components.

Snapshot Table

1. Database modifications initiated by triggers following modifications to other table rows are performed synchronously. To create asynchronous replications of partial data, snapshot tables can be created and the data to be contained therein can be described in a way similar to that when defining view tables.

2. While a view table is a logical view to physically stored data, the snapshot table contains data that is stored physically. To update the contents of the snapshot table, the REFRESH statement must be issued. If a snapshot table only contains data from a base table and if there is a snapshot log, i.e., a protocol of the modifying operations performed between the last REFRESH statement and the current point in time, then only these modifications are made to the snapshot table. Otherwise, the complete content of the snapshot table is rebuilt.

3. Snapshot tables can only be selected. INSERT, UPDATE, or DELETE statements are not possible on snapshot tables.

Backup and Recovery Concept

1. In error situations that do not involve storage medium failures, ADABAS automatically restores the last consistent state of the database on restart. This means that all effects of committed transactions are preserved, while the effects of transactions open at the time of error occurrence are cancelled.

2. Storage medium failures require the loading of a previously backed up version of the database. They may also require the loading of several incremental data backups (see Backup / Save / Updated Pages menu function in the CONTROL online help) to restore the database to a state upon which the last log versions may be re-applied. When these actions are concluded, the last consistent database state has been restored.

3. ADABAS does not support the exchange of storage media. Instead, individual tables can be explicitly unloaded. This function is supported by the ADABAS component LOAD.

4. The ADABAS component CONTROL (see the CONTROL online help) which serves to perform the above-mentioned backup and recovery operations of the database can only be used by the CONTROLUSER. CONTROL can usually only be used once for each SERVERDB at any given time, parallel to normal database operation.

SQLMODE

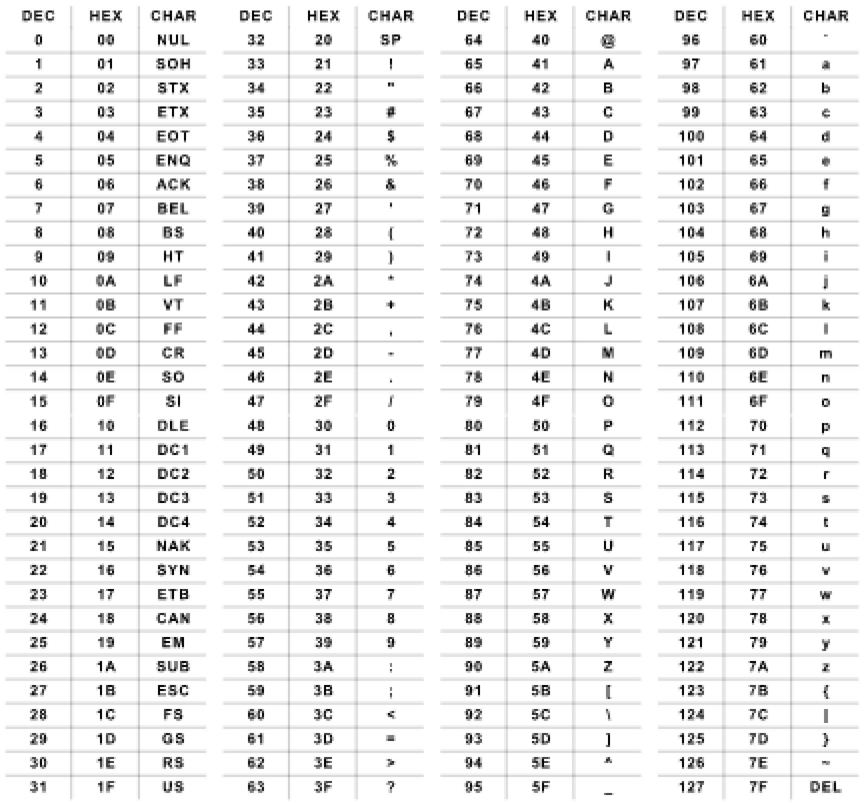
1. The database system ADABAS is able to perform correct ADABAS applications, as well as applications that are written according to the ANSI standard (ANSI X3.135-1992, Entry SQL), the definition of DB2 Version 3, or the definition of ORACLE7. ADABAS is able to check whether ADABAS applications conform to the above-mentioned definitions. This means in particular that any extension beyond the chosen definition is considered incorrect. However, the support of other SQLMODEs with regard to DDL statements is restricted.
   When connecting to ADABAS, one of the above-mentioned definitions or the SQLMODE ADABAS can be selected. The default is the SQLMODE ADABAS.

2. This online help describes the functionality of the database system ADABAS provided for the SQLMODE ADABAS. Only those effects of commands are described which refer to database objects that can be created in the selected SQLMODE. If database objects, e.g. tables, are created in one SQLMODE and addressed in another SQLMODE, these tables may contain columns of data types that are unknown in the current SQLMODE and that are therefore not described.

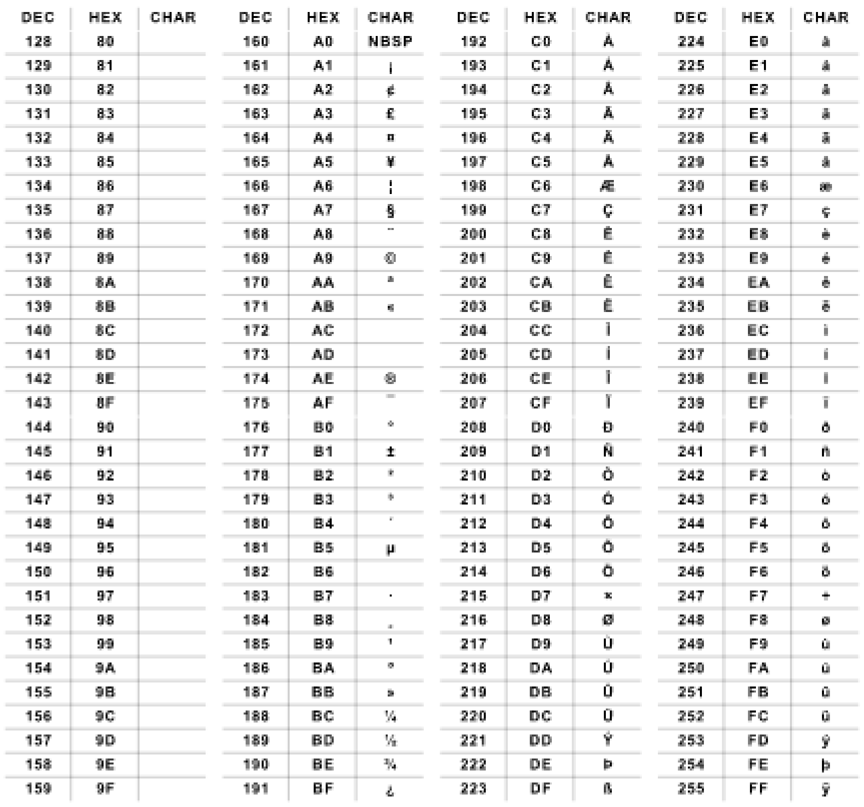
Code Tables

1. The database system ADABAS internally works either with the ASCII code according to ISO 8859/1.2 or with the EBCDIC code CCSID 500, Codepage 500.

2. The ASCII code according to ISO 8859/1.2 uses the following assignments:



DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR
0	00	NUL	32	20	SP	64	40	@	96	60	`
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	'	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL



DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR
128	80		160	A0	NBSP	192	C0	À	224	E0	à
129	81		161	A1	¡	193	C1	Á	225	E1	á
130	82		162	A2	¢	194	C2	Â	226	E2	â
131	83		163	A3	£	195	C3	Ã	227	E3	ã
132	84		164	A4	¤	196	C4	Ä	228	E4	ä
133	85		165	A5	¥	197	C5	Å	229	E5	å
134	86		166	A6	¦	198	C6	Æ	230	E6	æ
135	87		167	A7	§	199	C7	Ç	231	E7	ç
136	88		168	A8	¨	200	C8	È	232	E8	è
137	89		169	A9	©	201	C9	É	233	E9	é
138	8A		170	AA	ª	202	CA	Ê	234	EA	ê
139	8B		171	AB	«	203	CB	Ë	235	EB	ë
140	8C		172	AC		204	CC	Ì	236	EC	ì
141	8D		173	AD		205	CD	Í	237	ED	í
142	8E		174	AE	®	206	CE	Î	238	EE	î
143	8F		175	AF	¯	207	CF	Ï	239	EF	ï
144	90		176	B0	°	208	D0	Ð	240	F0	ð
145	91		177	B1	±	209	D1	Ñ	241	F1	ñ
146	92		178	B2	²	210	D2	Ò	242	F2	ò
147	93		179	B3	³	211	D3	Ó	243	F3	ó
148	94		180	B4	´	212	D4	Ô	244	F4	ô
149	95		181	B5	µ	213	D5	Õ	245	F5	õ
150	96		182	B6		214	D6	Ö	246	F6	ö
151	97		183	B7	·	215	D7	×	247	F7	÷
152	98		184	B8	¸	216	D8	Ø	248	F8	ø
153	99		185	B9	¹	217	D9	Ù	249	F9	ù
154	9A		186	BA	º	218	DA	Ú	250	FA	ú
155	9B		187	BB	»	219	DB	Û	251	FB	û
156	9C		188	BC	¼	220	DC	Ü	252	FC	ü
157	9D		189	BD	½	221	DD	Ý	253	FD	ý
158	9E		190	BE	¾	222	DE	Þ	254	FE	þ
159	9F		191	BF	¿	223	DF	ß	255	FF	ÿ

▭ possibly set by the operating system

3. The EBCDIC code CCSID 500, Codepage 500 uses the following assignments:

DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR
0	00	NUL	32	20	DS	64	40	SP	96	60	-
1	01	SOH	33	21	SOS	65	41	RSP	97	61	/
2	02	STX	34	22	FS	66	42	â	98	62	Â
3	03	ETX	35	23		67	43	ä	99	63	Ä
4	04	PF	36	24	BYP	68	44	à	100	64	À
5	05	HT	37	25	LF	69	45	á	101	65	Á
6	06	LC	38	26	ETB	70	46	ã	102	66	Ã
7	07	DEL	39	27	ESC	71	47	å	103	67	Å
8	08	GE	40	28		72	48	ç	104	68	Ç
9	09	RLF	41	29		73	49	ñ	105	69	Ñ
10	0A	SMM	42	2A	SM	74	4A	[	106	6A	¦
11	0B	VT	43	2B	CU2	75	4B	.	107	6B	,
12	0C	FF	44	2C		76	4C	<	108	6C	%
13	0D	CR	45	2D	ENQ	77	4D	(	109	6D	_
14	0E	SO	46	2E	ACK	78	4E	+	110	6E	>
15	0F	SI	47	2F	BEL	79	4F	!	111	6F	?
16	10	DLE	48	30		80	50	&	112	70	ø
17	11	DC1	49	31		81	51	é	113	71	É
18	12	DC2	50	32	SYN	82	52	ê	114	72	Ê
19	13	TM	51	33		83	53	ë	115	73	Ë
20	14	RES	52	34	PN	84	54	è	116	74	È
21	15	NL	53	35	RS	85	55	í	117	75	Í
22	16	BS	54	36	UC	86	56	î	118	76	Î
23	17	IL	55	37	EOT	87	57	ï	119	77	Ï
24	18	CAN	56	38		88	58	ì	120	78	Ì
25	19	EM	57	39		89	59	ß	121	79	`
26	1A	CC	58	3A		90	5A	]	122	7A	:
27	1B	CU1	59	3B	CU3	91	5B	$	123	7B	#
28	1C	IFS	60	3C	DC4	92	5C	*	124	7C	@
29	1D	IGS	61	3D	NAK	93	5D	)	125	7D	'
30	1E	IRS	62	3E		94	5E	;	126	7E	=
31	1F	IUS	63	3F	SUB	95	5F	^	127	7F	"

DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR
128	80	Ø	160	A0	µ	192	C0	{	224	E0	\
129	81	a	161	A1	~	193	C1	A	225	E1	÷
130	82	b	162	A2	s	194	C2	B	226	E2	S
131	83	c	163	A3	t	195	C3	C	227	E3	T
132	84	d	164	A4	u	196	C4	D	228	E4	U
133	85	e	165	A5	v	197	C5	E	229	E5	V
134	86	f	166	A6	w	198	C6	F	230	E6	W
135	87	g	167	A7	x	199	C7	G	231	E7	X
136	88	h	168	A8	y	200	C8	H	232	E8	Y
137	89	i	169	A9	z	201	C9	I	233	E9	Z
138	8A	«	170	AA	¡	202	CA	(SHY)	234	EA	²
139	8B	»	171	AB	¿	203	CB	ô	235	EB	Ô
140	8C	ð	172	AC	Ð	204	CC	ö	236	EC	Ö
141	8D	ý	173	AD	Ý	205	CD	ò	237	ED	Ò
142	8E	þ	174	AE	Þ	206	CE	ó	238	EE	Ó
143	8F	±	175	AF	®	207	CF	õ	239	EF	Õ
144	90	°	176	B0	¢	208	D0	}	240	F0	0
145	91	j	177	B1	£	209	D1	J	241	F1	1
146	92	k	178	B2	¥	210	D2	K	242	F2	2
147	93	l	179	B3	·	211	D3	L	243	F3	3
148	94	m	180	B4	©	212	D4	M	244	F4	4
149	95	n	181	B5	§	213	D5	N	245	F5	5
150	96	o	182	B6		214	D6	O	246	F6	6
151	97	p	183	B7	¼	215	D7	P	247	F7	7
152	98	q	184	B8	½	216	D8	Q	248	F8	8
153	99	r	185	B9	¾	217	D9	R	249	F9	9
154	9A	ª	186	BA		218	DA	¹	250	FA	³
155	9B	º	187	BB	|	219	DB	û	251	FB	Û
156	9C	æ	188	BC	¯	220	DC	ü	252	FC	Ü
157	9D	¸	189	BD	¨	221	DD	ù	253	FD	Ù
158	9E	Æ	190	BE	´	222	DE	ú	254	FE	Ú
159	9F	¤	191	BF	×	223	DF	ÿ	255	FF	EO

<character>

Function

defines the elements of character strings and of key words.

Format

```
<character> ::=
    <digit>
  | <letter>
  | <extended letter>
  | <hex digit>
  | <language specific character>
  | <special character>

<digit> ::=
    0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<letter> ::=
    A | B | C | D | E | F | G | H | I | J | K | L | M
  | N | O | P | Q | R | S | T | U | V | W | X | Y | Z
  | a | b | c | d | e | f | g | h | i | j | k | l | m
  | n | o | p | q | r | s | t | u | v | w | x | y | z

<extended letter> ::=
    # | @ | $

<hex digit> ::=
    0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
  | A | B | C | D | E | F
  | a | b | c | d | e | f

<language specific character> ::=
    Every letter that occurs in a North, Central or South
    European language, but is not contained in <letter>
    (e.g. the German umlauts, French grave accent, etc.).


<special character> ::=
    Every character except <digit>, <letter>, <extended letter>,
    <hex digit>,<language specific character>, and the character
    for the line end in a file.
```

Syntax Rules

none

General Rules

none

<literal>

Function

specifies a non-NULL value.

Format

```
<literal> ::=
    <string literal>
  | <numeric literal>

<string literal> ::=
    ''
  | '<character>...'
  | <hex literal>

<hex literal> ::=
    x''
  | X''
  | x'<hex digit seq>'
  | X'<hex digit seq>'

<hex digit seq> ::=
    <hex digit> <hex digit>
  | <hex digit seq> <hex digit> <hex digit>

<numeric literal> ::=
    <fixed point literal>
  | <floating point literal>

<fixed point literal> ::=
    [<sign>] <unsigned integer>[.<unsigned integer>]
  | [<sign>] <unsigned integer>.
  | [<sign>] .<unsigned integer>

<sign> ::=
    +
  | -

<unsigned integer> ::=
    <digit>


<floating point literal> ::=
    <mantissa>E<exponent>
  | <mantissa>e<exponent>

<mantissa> ::=
    <fixed point literal>

<exponent> ::=
    [<sign>] [ [<digit>] <digit>] <digit>
```

Syntax Rules

1. An apostrophe within a character string is represented by two successive apostrophes.

2. A character string can have up to 254 characters.

3. A hexadecimal character string may comprise up to 508 hexadecimal digits.

General Rules

1. A <string literal> of the type '<character>...' or '' is only valid for a value referring to an alphanumeric column with the code attribute ASCII or EBCDIC (see the chapter '<create table statement>, <column definition>').

2. A <hex literal> is only valid for a value referring to a column with the code attribute BYTE (see the chapter '<create table statement>, <column definition>').

3. A <string literal> of the type '', x'' and X'', and <string literal>s which only contain blanks are not the same value as the NULL value.

<token>

Function

specifies lexical units.

Format

```
<token> ::=
    <regular token>
  | <delimiter token>

<regular token> ::=
    <literal>
  | <key word>
  | <identifier>
  | <parameter name>

<key word> ::=
    <not restricted key word>
  | <restricted key word>
  | <reserved key word>


<not restricted key word> ::=
  ACCOUNTING    ACTIVATE     ADABAS       ADD_MONTHS   AFTER
  ANALYZE       ANSI

  BAD           BEGINLOAD    BLOCKSIZE    BUFFER

  CACHELIMIT    CACHES       CANCEL       CLEAR        COLD
  COMPLETE      CONFIG       CONSOLE      CONSTRAINTS  COPY
  COSTLIMIT     COSTWARNING  CURRVAL

  DATA          DAYS         DB2          DBA          DBFUNCTION
  DBPROC        DBPROCEDURE  DEGREE       DESTPOS      DEVICE
  DEVSPACE      DIAGNOSE     DISABLE      DIV          DOMAINDEF
  DSETPASS      DUPLICATES   DYNAMIC

  ENDLOAD       ENDPOS       EUR          EXPLAIN      EXPLICIT

  FIRSTPOS      FNULL        FORCE        FORMAT       FREAD
  FREEPAGE      FWRITE


  GATEWAY       GRANTED

  HEXTORAW      HOLD         HOURS

  IMPLICIT      INDEXNAME    INIT         INITRANS     INSTR
  INTERNAL      ISO

  JIS

  KEEP

  LABEL         LASTPOS      LAST_DAY     LOAD

  MAXTRANS      MDECLARE     MDELETE      MFETCH       MICROSECONDS
  MINSERT       MINUTES       MLOCK         MOD           MONITOR
  MONTHS     MONTHS_BETWEEN MSELECT       MUPDATE

  NEW_TIME      NEXTVAL       NEXT_DAY      NOLOG         NORMAL
  NOSORT        NVL

  OFF           OPTIMISTIC    ORACLE        OUT           OVERWRITE

  PAGES         PARAM         PARSE         PARSEID       PARTICIPANTS
  PASSWORD      PATTERN       PCTUSED       PERMLIMIT     POS
  PRIV          PROC          PSM

  QUICK

  RANGE         RAWTOHEX      RECONNECT     REFRESH       REPLICATION
  REST          RESTART       RESTORE       REUSE         RFETCH

  SAME          SAPR3         SAVE          SAVEPOINT     SEARCH
  SECONDS       SEGMENT       SELECTIVITY   SEQUENCE      SERVERDB
  SHUTDOWN      SNAPSHOT      SOUNDS        SOURCEPOS     SQLID
  SQLMODE       STANDARD      STARTPOS      STAT          STATE
  STORAGE       STORE         SUBPAGES      SUBTRANS

  TABID         TABLEDEF      TEMP          TEMPLIMIT     TERMCHAR
  TIMEOUT       TO_CHAR       TO_DATE       TO_NUMBER     TRANSFILE
  TRIGGERDEF

  UNLOAD        UNLOCK        UNTIL         USA           USERID

  VERIFY        VERSION       VSIZE         VTRACE

  WAIT

  YEARS


<restricted key word> ::=
  ACTION        ADD           AND           AS            ASC
  AT            AUDIT

  BEGIN         BETWEEN       BOTH          BUFFERPOOL    BY

  CASCADE       CAST          CATALOG       CHECK         CLOSE
  CLUSTER       COMMENT       COMMIT        CONCAT        CONNECT
  CONSTRAINT    CREATE        CURRENT_DATE  CURRENT_TIME  CURSOR

  DECLARE       DESC          DESCRIBE      DISCONNECT    DOMAIN
  DROP

  EDITPROC      END           ESCAPE        EXCLUSIVE     EXECUTE
  EXTRACT

  FALSE         FETCH         FOREIGN

  GET           GRANT

  IDENTIFIED    IN            INDICATOR     INNER         IS
  ISOLATION

  JOIN

  LANGUAGE      LEADING       LEVEL         LIKE          LOCAL
  LOCK
  MINUS        MODE         MODIFY

  NATURAL      NO           NOWAIT       NUMBER

  OBID         ON           ONLY         OPEN         OPTIMIZE
  OPTION       OR           OUTER

  PCTFREE      PRECISION    PRIVILEGES   PROCEDURE    PUBLIC

  RAW          READ         REFERENCES   RELEASE      RENAME
  RESOURCE     RESTRICT     REVOKE       ROLLBACK     ROW
  ROWNUM       ROWS

  SCHEMA       SHARE        SYNONYM      SYSDATE

  TABLESPACE   TRAILING     TRANSACTION  TRIGGER      TRUE

  UID          UNIQUE       UNKNOWN      USAGE        USING

  VALIDPROC    VARCHAR2     VARYING      VIEW


  WHENEVER     WORK         WRITE


<reserved key word> ::=
  ABS          ACOS         ADDDATE      ADDTIME      ALL
  ALPHA        ALTER        ANY          ASCII        ASIN
  ATAN         ATAN2        AVG

  BINARY       BIT          BOOLEAN      BYTE

  CEIL         CEILING      CHAR         CHARACTER    CHR
  COLUMN       CONNECTED    COS          COSH         COT
  COUNT        CURDATE      CURRENT      CURTIME

  DATABASE     DATE         DATEDIFF     DAY          DAYNAME
  DAYOFMONTH   DAYOFWEEK    DAYOFYEAR    DBYTE        DEC
  DECIMAL      DECODE       DEFAULT      DEGREES      DELETE
  DIGITS       DIRECT       DISTINCT     DOUBLE

  EBCDIC       ENTRY        ENTRYDEF     EXCEPT       EXISTS
  EXP          EXPAND

  FIRST        FIXED        FLOAT        FLOOR        FOR
  FROM

  GRAPHIC      GREATEST     GROUP

  HAVING       HEX          HOUR

  IFNULL       IGNORE       INDEX        INITCAP      INSERT
  INT          INTEGER      INTERSECT    INTO

  KEY

  LAST         LCASE        LEAST        LEFT         LENGTH
  LFILL        LINK         LIST         LN           LOCALSYSDBA
  LOG          LOG10        LONG         LOWER        LPAD
  LTRIM

  MAKEDATE     MAKETIME     MAPCHAR      MAX          MICROSECOND
  MIN           MINUTE        MONTH         MONTHNAME

  NEXT          NOROUND       NOT           NOW           NULL
  NUM           NUMERIC

  OBJECT        OF            ORDER

  PACKED        PI            POWER         PREV          PRIMARY


  RADIANS       REAL          REFERENCED    REJECT        REPLACE
  RFILL         RIGHT         ROUND         ROWID         ROWNO
  RPAD          RTRIM

  SECOND        SELECT        SELUPD        SET           SHOW
  SIGN          SIN           SINH          SMALLINT      SOME
  SOUNDEX       SQRT          STAMP         STATISTICS    STDDEV
  SUBDATE       SUBSTR        SUBTIME       SUM           SYSDBA

  TABLE         TAN           TANH          TIME          TIMEDIFF
  TIMESTAMP     TIMEZONE      TO            TOIDENTIFIER TRANSLATE
  TRIM          TRUNC         TRUNCATE

  UCASE         UNION         UPDATE        UPPER         USER
  USERGROUP

  VALUE         VALUES        VARCHAR       VARGRAPHIC    VARIANCE

  WEEKOFYEAR    WHERE         WITH

  YEAR

  ZONED
```

```
<identifier> ::=
    <simple identifier>
  | <double quotes><special identifier><double quotes>

<simple identifier> ::=
    <first character> [<identifier tail character>...]

<first character> ::=
    <letter>
  | <extended letter>
  | <language specific character>

<identifier tail character> ::=
    <letter>
  | <extended letter>
  | <language specific character>
  | <digit>
  | <underscore>

<underscore> ::=
    _


<delimiter token> ::=
    ( | ) | , | . | + | - | * | /
  | < | > | <> | != | = | <= | >=
  | ¬= | ¬< | ¬> for a computer with the code type EBCDIC
  | ~= | ~< | ~> for a computer with the code type ASCII
```

```
<double quotes> ::=
    "

<special identifier> ::=
    <special identifier character>...

<special identifier character> ::=
    Any character.
```

Syntax Rules

1. Each <token> can be followed by any number of blanks. Each <regular token> must be concluded by a <delimiter token> or a blank. Key words and identifiers can be entered in uppercase/lowercase characters.

2. <reserved key word>s must not be used as <simple identifier>s. These are only allowed for <special identifier>s.

3. <double quotes> within a <special identifier> are represented by two successive <double quotes>.

4. For databases to be operated in different SQLMODEs, it is recommended not to use <restricted key word>s as <simple identifier>s because these could cause problems when using another SQLMODE.

General Rules

1. <simple identifier>s are always converted into uppercase characters within the database. Therefore, <simple identifier>s are not case sensitive.

2. If the name of a database object is to contain lowercase characters, special characters or blanks, <special identifier>s must be used.

Names

Function

identify objects.

Format

```
<user name> ::=
    <identifier>

<usergroup name> ::=
    <identifier>

<owner> ::=
    <user name>
  | <usergroup name>
  | TEMP

<alias name> ::=
    <identifier>

<column name> ::=
    <identifier>

<constraint name> ::=
    <identifier>

<domain name> ::=
    [<owner>.]<identifier>

<index name> ::=
    <identifier>

<reference name> ::=
    <identifier>

<referential constraint name> ::=
    <identifier>

<result table name> ::=
    <identifier>


<synonym name> ::=
    <identifier>

<termchar set name> ::=
    <identifier>

<table name> ::=
    [<owner>.]<identifier>
  | <synonym name>

<db procedure> ::=
    [<owner>.]<program name>.<procedure name>

<program name> ::=
    <identifier>
```

```
<procedure name> ::=
    <identifier>

<trigger name> ::=
    <identifier>

<parameter name> ::=
    :<identifier>

<indicator name> ::=
    <parameter name>

<serverdb name> ::=
    <string literal>

<servernode name> ::=
    <string literal>

<password> ::=
    <identifier>
  | <first password character> [<identifier tail character>...]

<first password character> ::=
    <letter>
  | <extended letter>
  | <language specific character>
  | <digit>
```

Syntax Rules

1. <servernode name>s are truncated after the 64th character. All the other names are truncated after the 18th character.

2. For parameter names, the conventions of the programming language in which the SQL statements of ADABAS are embedded determine the number of significant characters.

3. The <identifier>s for parameter names may contain the characters '.' and '-', but not as the first character.
   Also valid are: <identifier>(<identifier>) and :<identifier> (.<identifier>.).

General Rules

1. A <user name> identifies a user. It is defined by a <create user statement>.

2. A <usergroup name> identifies a usergroup. It is defined by a <create usergroup statement>.

3. <owner> identifies the owner of an object. <owner> is the user name if the owner does not belong to a usergroup. <owner> is the usergroup name if the owner belongs to a usergroup. If TEMP is specified as <owner> in a <table name>, then a temporary table owned by the current user is concerned.

4. A new column name <alias name> defines the name of a column in a view table or in

a snapshot table. It is defined in a <create view statement> or <create snapshot statement>.

5. A <column name> identifies a column. An identifier is defined as <column name> by a <create table statement>, <create view statement>, <alter table statement>, <create snapshot statement>, or in a <query statement>.

6. The name of a condition on rows of a table, <constraint name>, is defined in the <constraint definition> of the <create table statement> or <alter table statement>.

7. The name of a range of values, <domain name>, identifies a domain in a table column. It is defined by a <create domain statement>. The specification TEMP as <owner> made in a <domain name> is not valid.

8. An <index name> identifies an index created by a <create index statement>.

9. An identifier is declared to be a <reference name> for a certain scope and is associated with exactly one table. The scope of this declaration is the entire SQL statement. The same reference name specified in various scopes can be associated with different tables or with the same table.

10. A <referential constraint name> identifies a referential integrity rule which is created by a <referential constraint definition> in the <create table statement> or in the <alter table statement> defining delete or existence conditions between two tables.

11. A <result table name> identifies a result table defined by a <query statement>.

12. A <synonym name> is a designation for a table. This designation is only known for one user or usergroup. A <synonym name> is defined by a <create synonym statement>.

13. A <termchar set name> identifies a TERMCHAR SET defined by the ADABAS component CONTROL.

14. A <table name> identifies a table. An identifier is defined as <table name> by a <create table statement>, <create view statement>, <create snapshot statement>, or <create synonym statement>. ADABAS uses some <table name>s for internal purposes. The <identifier>s of these <table name>s begin with 'SYS'. To prevent conflicting names, it is recommended not to use <table name>s beginning with 'SYS'. If the qualification of the user name is missing for a table name specification, first the partial catalog of the current user, then the partial catalog of the DBA who created the current user, and then the partial catalog of the SYSDBA of the current user is scanned for the specified table name. Finally, the partial catalog of the owner of the system tables is scanned, if required.

15. A <db procedure> identifies a DB procedure defined with the aid of an ADABAS component. The specification TEMP as <owner> made in a <db procedure> is not valid.

16. A <trigger name> identifies a trigger defined for a table with the aid of an ADABAS component.

17. A <parameter name> identifies a host variable in an application containing SQL statements of ADABAS.

18. An <indicator name> identifies an indicator variable in an application which can be specified together with a <parameter name> whose value indicates irregularities such

as the occurrence of a NULL value or of different lengths of value and parameter.

19. A <serverdb name> identifies either the whole, non-distributed database or a part of the distributed database which was defined with the aid of the ADABAS component CONTROL.

20. The <password> is needed to establish the connection to the ADABAS server. The <password> of a user is defined by a <create user statement>. It can be altered by an <alter password statement>.

<column spec>

Function

specifies a column in a table.

Format

```
<column spec> ::=
    <column name>
  | <table name>.<column name>
  | <reference name>.<column name>
  | <result table name>.<column name>
```

Syntax Rules

none

General Rules

none

<parameter spec>

Function

specifies a parameter.

Format

```
 <parameter spec> ::=
     <parameter name> [<indicator name>]
```

Syntax Rules

none

General Rules

1. A <parameter spec> specifies a parameter which can be followed by an indicator parameter. The indicator parameter must be declared as a variable in the embedding programming language. It must be possible to assign at least four-digit integers to such a variable.

2. Parameters which are to receive values retrieved from the database are called output parameters.

3. Parameters containing values that are to be passed to the database are called input parameters.

4. In the case of input parameters, an indicator parameter having a value greater than or equal to 0 indicates that the parameter value is the value to be passed to the database.

5. In the case of input parameters, an indicator parameter having a value less than 0 indicates that the value represented by the parameter is the NULL value.

6. In the case of output parameters, an indicator parameter having the value 0 indicates that the passed value is the parameter value, not the NULL value.

7. In the case of alphanumeric output parameters, an indicator parameter having a value greater than 0 indicates that the assigned character string was too long and has been truncated. The indicator parameter then indicates the untruncated length of the original output column.

8. In the case of numeric output parameters, an indicator parameter having a value greater than 0 indicates that the assigned value has too many significant digits and decimal positions have been truncated. The indicator parameter then indicates the number of digits of the original value.

9. In the case of output parameters, an indicator parameter having the value -1 indicates

that the value represented by the parameter is the NULL value.

10. In the case of numeric output parameters, an indicator parameter having the value -2 indicates that the value represented by the parameter is the special NULL value.

11. The special NULL value is generated by arithmetical operations when these lead to an overflow or to a division by 0. The special NULL value is only valid for output columns and for columns in the <order clause>. If an overflow occurs in an arithmetical operation or a division by 0 at another place, the SQL statement is abnormally terminated. For sorting, the special NULL value is greater than all non-NULL values, but less than the NULL value.

Specifying Values

Function

specifies a value.

Format

```
<extended value spec> ::=
    <value spec>
  | DEFAULT
  | STAMP

<value spec> ::=
    <literal>
  | <parameter spec>
  | NULL
  | USER
  | USERGROUP
  | LOCALSYSDBA
  | SYSDBA [(<user name>)]
  | SYSDBA [(<usergroup name>)]
  | DATE
  | TIME
  | TIMESTAMP
  | TIMEZONE
  | TRUE
  | FALSE

<string spec> ::=
    <expression>
```

Syntax Rules

none

General Rules

1. The key word DEFAULT denotes the value defined as default for the column in the <create table statement> or <alter table statement>.
   If such a value is not defined, the function DEFAULT is not allowed.

2. ADABAS is able to generate unique values. These consist of the SERVERDB number and of consecutive numbers counted for each SERVERDB. The consecutive numbers begin with X'000000000001'. The values are generated in ascending order. It cannot be ensured that a sequence of values is uninterrupted. The key word STAMP produces the next key which ADABAS generated on the HOME SERVERDB of the specified table. For a replicated table, STAMP produces the next key which ADABAS generated on the current SERVERDB. STAMP is allowed in an <insert statement> and in an <update statement> and can only be applied to columns of the data type CHAR(n) BYTE where n>=8.
   If the user needs to know the generated value before applying it to a column, the

<next stamp statement> must be used.

3. The key word NULL denotes the NULL value.

4. The key word USER denotes the name of the current user. If the user issuing the SQL statement belongs to a usergroup, then USERGROUP denotes the usergroup name, otherwise, the user name.

5. The key word LOCALSYSDBA denotes the SYSDBA of the SERVERDB where the current user has performed the <connect statement>.
   The key word SYSDBA denotes the SYSDBA of the SERVERDB which is the HOME SERVERDB of the user <user name> or usergroup <usergroup name>. If neither a <user name> nor a <usergroup name> is specified, SYSDBA denotes the SYSDBA of the SERVERDB which is the HOME SERVERDB of the current user.

6. The key word DATE denotes the current date.

7. The key word TIME denotes the current time.

8. The key word TIMESTAMP denotes the current timestanp value which consists of date and time and microseconds.

9. The key word TIMEZONE denotes the time zone of the current SERVERDB. This value is currently preset to the value 0 and cannot be changed yet.

10. The key words TRUE and FALSE denote the corresponding values of Boolean columns.

11. For a <string spec>, only <expression>s that denote an alphanumeric value as the result are valid.

See also

Date and Time Format

Date and Time Format

Function

specifies the format in which date, time, and timestamp values are represented.

Format

```
 <datetimeformat> ::=
     EUR
   | INTERNAL
   | ISO
   | JIS
   | USA
```

Syntax Rules

1. The representation of a date value depends on the current format. In the list,

```
'YYYY' stands for a four-digit identifier of a year,
'MM'   stands for a two-digit  identifier of a month (01-12),
'DD'   stands for a two-digit  identifier of a day   (01-31).
```

Format	General Form	Example
EUR	'DD.MM.YYYY'	'19.09.1996'
INTERNAL	'YYYYMMDD'	'19960923'
ISO	'YYYY-MM-DD'	'1996-09-23'
JIS	'YYYY-MM-DD'	'1996-09-23'
USA	'MM/DD/YYYY'	'09/23/1996'

In all formats, except INTERNAL, leading zeros may be omitted in the identifiers of the month and day.

2. The representation of a time value depends on the current format. In the list,

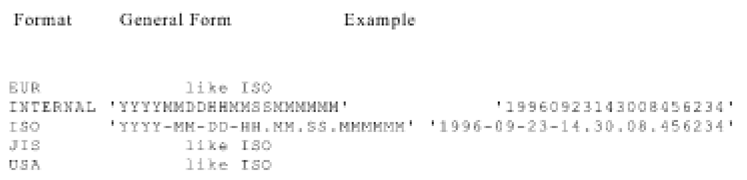
```
'HHHH' stands for a four-digit identifier of an hour, or
'HH'   stands for a two-digit  identifier of an hour,
'MM'   stands for a two-digit  identifier of minutes (00-59),
'SS'   stands for a two-digit  identifier of seconds (00-59).
```

Format	General Form	Example
EUR	'HH.MM.SS'	'14.30.08'
INTERNAL	'HHHHMMSS'	'00143008'
ISO	'HH.MM.SS'	'14.30.08'
JIS	'HH:MM:SS'	'14:30:08'
USA	'HH:MM AM (PM)'	'2:30 PM'

In all time formats, the identifier of the hour must consist of at least one digit. In the time format USA, the identifier of minutes can be omitted completely. In all the other formats, except INTERNAL, the identifiers of minutes and seconds must consist of at least one digit.

3. The representation of a timestamp value depends on the current format. In the list,

```
'YYYY'   stands for a four-digit identifier of a year,
'MM'     stands for a two-digit identifier of a month (01-12),
'DD'     stands for a two-digit identifier of a day   (01-31),
'HH'     stands for a two-digit identifier of an hour (00-24),
'MM'     stands for a two-digit identifier of minutes (00-59),
'SS'     stands for a two-digit identifier of seconds (00-59),
'MMMMMM' stands for a six-digit identifier of microseconds.
```



Format	General Form	Example
EUR	like ISO	
INTERNAL	'YYYYMMDDHHMMSSMMMMMM'	'19960923143008456234'
ISO	'YYYY-MM-DD-HH.MM.SS.MMMMMM'	'1996-09-23-14.30.08.456234'
JIS	like ISO	
USA	like ISO	

In all date and time formats, the identifier of microseconds may be omitted. In all formats, except INTERNAL, the identifiers of the month and day must consist of at least one digit.

General Rules

1. The date and time format determines the representation in which date, time and timestamp values may be included in SQL statements and the way in which results are to be represented.

2. The date and time format is determined during the installation of the database.

3. A user can change the date and time format for his session by setting the SET parameters of the ADABAS components or by specifying the corresponding parameters when using programs.

Specifying a Key

Function

specifies a location in a key-listed table.

Format

```
 <key spec> ::=
     <column name> = <value spec>
```

Syntax Rules

1. The <value spec> must not be the NULL value.

General Rules

1. The <column name> must denote a key column of the table.

2. The key specification must contain all key columns of a table. The <key spec>s are separated by a comma.

3. The key specification indicates a location in a key-listed table, without requiring the existence of a row of the specified key values.

4. For tables created without key columns, there is the implicitly created column SYSKEY CHAR(8) BYTE which contains a key generated by ADABAS. This column can only be used in a <key spec>.

<function spec>

Function

specifies a value which is obtained by applying a function to an argument.

Format

```
<function spec> ::=
    <arithmetic function>
  | <trigonometric function>
  | <string function>
  | <date function>
  | <time function>
  | <extraction function>
  | <special function>
  | <conversion function>
  | <userdefined function>

<userdefined function> ::=
    Each DB function defined by any user.
```

Syntax Rules

none

General Rules

1. The arguments and results of the functions are numeric, alphanumeric or Boolean values. The date, time and timestamp values are alphanumeric values which are subject to certain restrictions. LONG columns are not allowed as arguments.

2. A <userdefined function> is a DB function which was defined in SQLMODE ADABAS and is available in the other SQLMODEs except ANSI. The result of a <userdefined function> is a numeric, alphanumeric or Boolean value. If a DB function has a name that is the name of a known predefined function in the current SQLMODE, then this function is used and not the DB function.

See also

<arithmetic function>

<trigonometric function>

<string function>

<date function>

<time function>
<extraction function>
<special function>
<conversion function>

<arithmetic function>

Function

specifies a function which produces a numeric value as the result.

Format

```
<arithmetic function> ::=
    TRUNC   ( <expression>[, <expression>] )
  | ROUND   ( <expression>[, <expression>] )
  | NOROUND ( <expression> )
  | FIXED   ( <expression>[, <unsigned integer>
              [, <unsigned integer>] ] )
  | CEIL    ( <expression> )
  | FLOOR   ( <expression> )
  | SIGN    ( <expression> )
  | ABS     ( <expression> )
  | POWER   ( <expression>, <expression> )
  | EXP     ( <expression> )
  | SQRT    ( <expression> )
  | LN      ( <expression> )
  | LOG     ( <expression>, <expression> )
  | PI
  | LENGTH  ( <expression> )
  | INDEX   ( <string spec>, <string spec>
              [,<expression>[, <expression>] ] )
```

Syntax Rules

none

General Rules

1. TRUNC
   Let a and s be numbers.
   If s>0, then TRUNC(a,s) is the number a truncated s digits after the decimal point.
   If s=0, then TRUNC(a,s) is the integral part of a.
   If s<0, then TRUNC(a,s) is the number a truncated s digits before the decimal point.
   If s is not specified, then the value 0 is implicitly assumed for s.
   If s is not an integer value, then the integral part of s is used.
   If a is a floating point number, then the result is a floating point number. Otherwise, the result is a fixed point number. If a is the NULL value, then TRUNC(a,s) is the NULL value. It is true that TRUNC(a,s) is the special NULL value when a is the special NULL value.

2. ROUND
   Let a and s be numbers.
   If a>=0, then ROUND(a,s)=TRUNC(a+0.5*10E-s, s).
   If a<0, then ROUND(a,s)=TRUNC(a-0.5*10E-s, s).
   If s is not specified, then the value 0 is implicitly assumed for s.

If s is not an integer value, then the integral part of s is used.
If a is a floating point number, then the result is a floating point number. Otherwise, the result is a fixed point number. If a is the NULL value, then ROUND(a,s) is the NULL value. It is true that ROUND(a,s) is the special NULL value when a is the special NULL value.

3. NOROUND
The function NOROUND(a) prevents the result of the <expression> a from being rounded in the case of an <update statement> or an <insert statement>. Without a NOROUND specification the <expression> will be rounded when its data type differs from that of the target column. If the non-rounded number does not correspond to the data type of the target column, an error message is output.
If a is the NULL value, then the result is the NULL value. If a is the special NULL value, then the result is the special NULL value.

4. FIXED
The function FIXED(a,p,s) can be used to output the number a in a format of the data type FIXED(p,s). Digits after the decimal point are rounded to s digits, if necessary. If a is the NULL value, then the result is the NULL value. If a is the special NULL value, then the result is the special NULL value. If ABS(a)>10(p-s), then the result is the special NULL value. If s is not specified, then the value 0 is implicitly assumed for s. If p is not specified, then the value 18 is implicitly assumed for p.

5. CEIL
If a is a number, then CEIL(a) is the smallest integer value that is greater than or equal to a. The result is a fixed point number with 0 digits after the decimal point. If it is not possible to represent the result of CEIL(a) in a fixed point number, then an error message is output.
If a is the NULL value, then CEIL(a) is the NULL value. It is true that CEIL(a) is the special NULL value when a is the special NULL value.

6. FLOOR
If a is a number, then FLOOR(a) is the greatest integer value that is less than or equal to a. The result is a fixed point number with 0 digits after the decimal point. If it is not possible to represent the result of FLOOR(a) in a fixed point number, then an error message is output.
If a is the NULL value, then FLOOR(a) is the NULL value. It is true that FLOOR(a) is the special NULL value when a is the special NULL value.

7. SIGN
Let a be a number. Then the following applies:

```
If a < 0, then SIGN(a) = -1.
If a = 0, then SIGN(a) =  0.
If a > 0, then SIGN(a) =  1.
```

If a is the NULL value, then SIGN(a) is the NULL value. It is true that SIGN(a) is the special NULL value when a is the special NULL value.

8. ABS
If a is a number, then ABS(a) is the absolute value of a. If a is the NULL value, then ABS(a) is the NULL value. It is true that ABS(a) is the special NULL value when a is the special NULL value.

9. POWER
Let a and b be numbers, then POWER(a,b) = ab. If b is not an integer value, then an error message is output. If a or b is the NULL value, then the result is the NULL value. It is true that POWER(a,b) is the special NULL value when a is the special NULL value.

10. EXP
Let a be a number, then EXP(a) = ea, where e = 2.71828183. If a is the NULL value, then the result is the NULL value. It is true that EXP(a) is the special NULL value when a is the special NULL value.

11. SQRT
Let a be a number > 0, then SQRT(a) is the square root of a. If a is a number = 0, then the result of SQRT(a) is 0. If a is a number < 0 or a is the NULL value, then the result is the NULL value. It is true that SQRT(a) is the special NULL value when a is the special NULL value.

12. LN
Let a be a number, then LN(a) is the natural logarithm of a. If a is the NULL value, then the result is the NULL value. It is true that LN(a) is the special NULL value when a is the special NULL value.

13. LOG
Let a be a number, then LOG(a,b) is the logarithm b to the base of a. If a or b is the NULL value, then the result is the NULL value. It is true that LOG(a,b) is the special NULL value when b is the special NULL value.

14. PI
The result of the function PI is the value of the mathematical constant π.

15. LENGTH
LENGTH can be applied to any data type.
If a is a character string of length n, then LENGTH(a)=n. The length of a character string is determined without consideration of trailing blanks (code attribute ASCII or EBCDIC) or binary zeros (code attribute BYTE).
LENGTH indicates the number of bytes needed for the internal representation of the value. If a is the NULL value, then LENGTH(a) is the NULL value. If a is the special NULL value, then LENGTH(a) is the special NULL value.

16. INDEX
INDEX produces the position of the substring specified as the second parameter within the character string specified as the first parameter. The optional third parameter indicates the start position for the search for the substring. If it is omitted, the search starts at the beginning; i.e., at start position 1. The start position must be greater than or equal to 1. The optional fourth parameter indicates which occurrence of the substring is to be searched for. If it is omitted, the first occurrence of the substring will be searched for.
If a and b are character strings and b is not at least s times a substring of a, then INDEX(a,b,p,s) is equal to 0. If a is a character string and b is the empty character string, then INDEX(a,b,p,s) is equal to p. If a, b, p, or s is the NULL value, then INDEX(a,b,p,s) is the NULL value. If p or s is the special NULL value, then an error message is output.

<trigonometric function>

Function

specifies a trigonometric function which produces a numeric value as the result.

Format

```
<trigonometric function> ::=
    COS     ( <expression> )
  | SIN     ( <expression> )
  | TAN     ( <expression> )
  | COT     ( <expression> )
  | COSH    ( <expression> )
  | SINH    ( <expression> )
  | TANH    ( <expression> )
  | ACOS    ( <expression> )
  | ASIN    ( <expression> )
  | ATAN    ( <expression> )
  | ATAN2   ( <expression>, <expression> )
  | RADIANS ( <expression> )
  | DEGREES ( <expression> )
```

Syntax Rules

none

General Rules

1. All <trigonometric function>s produce the NULL value as the result if the <expression> or one of the <expression>s produces the NULL value. If the <expression> or one of the <expression>s produces the special NULL value, then the <trigonometric function> produces the special NULL value as the result.

2. The <expression> in all <trigonometric function>s, except RADIANS, denotes a specification of the angle in radians.

3. COS
   If a is a number, then COS(a) is the cosine of the number a.

4. SIN
   If a is a number, then SIN(a) is the sine of the number a.

5. TAN
   If a is a number, then TAN(a) is the tangent of the number a.

6. COT
   If a is a number, then COT(a) is the cotangent of the number a.

7. COSH
   If a is a number, then COSH(a) is the hyperbolic cosine of the number a.

8. SINH
If a is a number, then SINH(a) is the hyperbolic sine of the number a.

9. TANH
If a is a number, then TANH(a) is the hyperbolic tangent of the number a.

10. ACOS
If a is a number, then ACOS(a) is the arc cosine of the number a.

11. ASIN
If a is a number, then ASIN(a) is the arc sine of the number a.

12. ATAN
If a is a number, then ATAN(a) is the arc tangent of the number a.

13. ATAN2
If a and b are numbers in the range between $-\pi$ and $+\pi$, then ATAN2(a,b) is the arc tangent of the value a/b.

14. RADIANS
If a is a number, then RADIANS(a) is the angle in radians of the number a.

15. DEGREES
If a is a number, then DEGREES(a) is the measure of degree of the number a.

<string function>

Function

specifies a function which produces an alphanumeric value as the result.

Format

```
<string function> ::=
    <string spec> || <string spec>
  | <string spec> & <string spec>
  | SUBSTR    ( <string spec>, <expression>[, <expression>] )
  | LFILL     ( <string spec>, <string literal>
                [,<unsigned integer> ] )
  | RFILL     ( <string spec>, <string literal>
                [,<unsigned integer> ] )
  | LPAD      ( <string spec>, <expression>, <string literal>
                [,<unsigned integer> ] )
  | RPAD      ( <string spec>, <expression>, <string literal>
                [,<unsigned integer> ] )
  | TRIM      ( <string spec>[, <string spec> ] )
  | LTRIM     ( <string spec>[, <string spec> ] )
  | RTRIM     ( <string spec>[, <string spec> ] )
  | EXPAND    ( <string spec>, <unsigned integer> )
  | UPPER     ( <string spec> )
  | LOWER     ( <string spec> )
  | INITCAP   ( <string spec> )
  | REPLACE   ( <string spec>, <string spec>[, <string spec> ] )
  | TRANSLATE ( <string spec>, <string spec>, <string spec> )
  | MAPCHAR   ( <string spec>[, <unsigned integer> ]
                [, <mapchar set name> ] )
  | ALPHA     ( <string spec>[, <unsigned integer> ] )
  | ASCII     ( <string spec> )
  | EBCDIC    ( <string spec> )
  | SOUNDEX   ( <string spec> )

<mapchar set name> ::=
    <identifier>
```

Syntax Rules

none

General Rules

1. Concatenation, ||
   If x is a character string of length n and if y is a character string of length m, then x||y is the concatenation xy of length n+m. If a character string comes from a column, then its length is determined without consideration of trailing blanks (code attribute ASCII or EBCDIC) or binary zeros (code attribute BYTE). If an operand of the concatenation is the NULL value, then the result is the NULL value.
   Columns having the same code attribute can be concatenated. Columns having the

different code attributes ASCII and EBCDIC can be concatenated. Columns with the code attributes ASCII and EBCDIC can be concatenated with date, time, or timestamp values.

2. Concatenation, &
   The concatenation x&y produces the same result as the concatenation x||y.

3. SUBSTR
   If x is a character string of length n, then SUBSTR(x,a,b) is that part of the character string x which begins at the ath character and has a length of b characters.
   SUBSTR(x,a) corresponds to SUBSTR(x,a,n-a+1) and produces all characters of the character string x from the ath character to the last character (nth).
   If b is specified as <unsigned integer>, then a value greater than (n-a+1) is also valid for b. In all the other cases, the value of b must not exceed the value (n-a+1). If b > (n-a+1), then SUBSTR(x,a) is performed internally. As many blanks (code attribute ASCII or EBCDIC) or binary zeros (code attribute BYTE) are appended to the end of this result as are needed to give the result the length b.
   If x, a or b is the NULL value, then SUBSTR(x,a,b) is the NULL value.

4. LFILL
   At the beginning of the character string defined as the first parameter, LFILL inserts the character defined as the second parameter as often as is needed to give the character string the length specified in the third parameter. If the third parameter is missing, the first parameter must designate a CHAR or VARCHAR column, which is then filled with the specified character up to the column's maximum length. If the first parameter is a character string with the code attribute ASCII or EBCDIC, then the second parameter must be a <string literal> consisting of a single character. If the first parameter is a character string with the code attribute BYTE, then the second parameter must be a <hex literal> that designates a single character, therefore consisting of two <hex digit>s. If the first parameter is the NULL value, then LFILL produces the NULL value as the result. If the second or third parameter is the NULL value, then an error message is output.

5. RFILL
   At the end of the character string defined as the first parameter, RFILL inserts the character defined as the second parameter as often as is needed to give the character string the length specified in the third parameter. If the third parameter is missing, the first parameter must designate a CHAR or VARCHAR column, which is then filled with the specified character up to the column's maximum length. If the first parameter is a character string with the code attribute ASCII or EBCDIC, then the second parameter must be a <string literal> consisting of a single character. If the first parameter is a character string with the code attribute BYTE, the second parameter must be a <hex literal> designating a single character, therefore consisting of two <hex digit>s. If the first parameter is the NULL value, then RFILL produces the NULL value as the result. If the second or third parameter is the NULL value, then an error message is output.

6. LPAD
   The first and third parameters of LPAD must be character strings. If the first parameter is a character string with the code attribute ASCII or EBCDIC, then the third parameter must be a <string literal> consisting of a single character. If the first parameter is a character string with the code attribute BYTE, the third parameter must be a <hex literal> designating a single character, therefore consisting of two <hex digit>s. The result of the second parameter must be a non-negative integer. The optional fourth parameter must be greater than or equal to the sum of LENGTH(first parameter)+(second parameter). If no fourth parameter is specified, then the first parameter must

designate a CHAR or VARCHAR column.
At the beginning of the character string defined as the first parameter, LPAD inserts the character defined as the third parameter as often as is specified in the second parameter. In the character string specified as the first parameter, leading and trailing blanks are truncated. The optional fourth parameter defines the maximum total length of the character string thus created. If the fourth parameter is missing, the first parameter must designate a CHAR or VARCHAR column, the maximum length of which will then be applied. If the first or second parameter is the NULL value, LPAD produces the NULL value as the result. If the second parameter is the special NULL value, then an error message is output.

7. RPAD
   The first and third parameters of RPAD must be character strings. If the first parameter is a character string with the code attribute ASCII or EBCDIC, then the third parameter must be a <string literal> consisting of a single character. If the first parameter is a character string with the code attribute BYTE, then the third parameter must be a <hex literal> designating a single character, therefore consisting of two <hex digit>s. The result of the second parameter must be a non-negative integer. The optional fourth parameter must be greater than or equal to the sum of LENGTH(first parameter)+(second parameter). If no fourth parameter is specified, then the first parameter must designate a CHAR or VARCHAR column.
   At the end of the character string defined as the first parameter, RPAD inserts the character defined as the third parameter as often as is specified in the second parameter. In the character string specified as the first parameter, leading and trailing blanks are truncated. The optional fourth parameter defines the maximum total length of the character string thus created. If the fourth parameter is missing, the first parameter must designate a CHAR or VARCHAR column, the maximum length of which will be applied. If the first or second parameter is the NULL value, RPAD produces the NULL value as the result. If the second parameter is the special NULL value, then an error message is output.

8. TRIM
   TRIM removes all characters specified in the second parameter from the beginning of the first parameter, so that the result of TRIM begins with the first character that was not specified in the second parameter. At the same time, TRIM removes the blanks (code attribute ASCII or EBCDIC) or binary zeros (code attribute BYTE) from the end of the character string specified as the first parameter and then all characters specified in the second parameter, so that the result of TRIM ends with the last character that was not specified in the second parameter. If no second parameter is specified, then only the blanks (code attribute ASCII or EBCDIC) or binary zeros (code attribute BYTE) are removed. The length of the character string decreases accordingly. TRIM applied to the NULL value produces the NULL value as the result.

9. LTRIM
   LTRIM removes all characters specified in the second parameter from the beginning of the character string specified as first parameter, so that the result of LTRIM begins with the first character that was not specified in the second parameter. If no second parameter is specified, then a blank (code attribute ASCII or EBCDIC) or the binary zero (code attribute BYTE) is implicitly assumed. The length of the character string decreases accordingly. LTRIM applied to the NULL value produces the NULL value as the result.

10. RTRIM
    RTRIM first removes the blanks (code attribute ASCII or EBCDIC) or the binary zeros (code attribute BYTE) from the end of the character string specified as first parameter,

then all characters specified in the second parameter, so that the result of RTRIM ends with the last character that was not specified in the second parameter. If no second parameter is specified, then only the blanks (code attribute ASCII or EBCDIC) or the binary zeros (code attribute BYTE) are removed. The length of the character string decreases accordingly. RTRIM applied to the NULL value produces the NULL value as the result.

11. EXPAND
    At the end of the character string defined as first parameter, EXPAND inserts as many blanks (code attribute ASCII or EBCDIC) or binary zeros (code attribute BYTE) as are needed to give the character string the length specified in the second parameter. If the first parameter is the NULL value, then EXPAND produces the NULL value as the result.

12. UPPER
    LOWER
    UPPER and LOWER transform a character string into uppercase or lowercase characters. UPPER and LOWER applied to the NULL value produce the NULL value.

13. INITCAP
    INITCAP changes the character string in such a way that the first character of a word is an uppercase character and the rest of the word consists of lowercase characters. Words are separated by one or more characters which are neither letters nor digits. INITCAP applied to the NULL value produces the NULL value.

14. REPLACE
    In the character string specified as the first parameter, REPLACE replaces the character string specified as the second parameter with the character string specified as the third parameter. If no third parameter is specified or if the third parameter is the NULL value, then the character string specified as the second parameter is removed from the first character string. If the first parameter is the NULL value, then REPLACE produces the NULL value as the result. If the second parameter is the NULL value, then REPLACE produces the first parameter as the result without modifying it.

15. TRANSLATE
    In the character string specified as the first parameter, TRANSLATE replaces the ith character of the second character string with the ith character of the third character string. The lengths of the second and third character strings must be equal. If the first parameter is the NULL value, then the result produces the NULL value. If the second parameter is the NULL value, then TRANSLATE produces the first parameter as the result without modifying it.

16. MAPCHAR
    In almost every North, Central, and South European language, there are letters that do not occur in any other language and that cannot be entered or displayed on every terminal (e.g., the German umlauts, the French grave accent etc.). Within the ASCII code according to ISO 8859/1.2 and the EBCDIC code CCSID 500, Codepage 500, these letters are placed in positions which can hardly ever be used for sorting.
    To resolve these problems, MAPCHAR SETs were implemented which can be used to map individual country-specific letters to one or two non-country-specific letters. This allows, e.g., for transforming 'ü' into 'ue'.
    A mapping of country-specific letters is implicitly defined and stored under the name 'DEFAULTMAP' while configuring a SERVERDB. This default map can be changed. But it is also possible to define any number of additional MAPCHARSETs using the ADABAS component CONTROL.

MAPCHAR (a,p,i) maps the string a with the help of the MAPCHAR SET i. MAPCHAR (a) corresponds to MAPCHAR (a,DEFAULTMAP).
The optional second parameter indicates the maximum length of the result. If no second parameter is specified, then the length of the <string spec> is implicitly assumed as the second parameter. If the <string spec> designates a CHAR or VARCHAR column and no second parameter is specified, then the length of the column is implicitly assumed as the second parameter.
MAPCHAR applied to the NULL value produces the NULL value.
The function MAPCHAR enables an appropriate sort, e.g., if 'ü' is to be treated as 'ue' for sorting purposes.
An example is

```
SELECT..., MAPCHAR(<column name>) sort,...
FROM...ORDER BY sort
```

17. ALPHA
ALPHA (a,p) corresponds to UPPER (MAPCHAR (a,p,DEFAULTMAP)).
The function ALPHA enables an appropriate sort, e.g., if 'ü' is to be treated for sorting purposes as 'UE'.
An example is

```
SELECT..., ALPHA(<column name>) sort,...
FROM...ORDER BY sort
```

18. ASCII
EBCDIC
If the function ASCII is applied to a character string of the code attribute EBCDIC or ASCII, then the result is the character string in ASCII representation. If the function EBCDIC is applied to a character string with the code attribute EBCDIC or ASCII, then the result is the character string in EBCDIC representation. The functions ASCII or EBCDIC applied to the NULL value produce the NULL value.
The application of the functions ASCII and EBCDIC is useful when a specific code is to be used for a sort or a comparison.

19. SOUNDEX
SOUNDEX applies the soundex algorithm to the character string and produces a value of data type CHAR (4) as the result. SOUNDEX applied to the NULL value produces the NULL value as the result.
SOUNDEX is useful when the <sounds predicate> is to be applied frequently to a column c. As no indexes can be used in such a case, it is recommended for performance reasons to define an additional table column c1 of data type CHAR (4) into which the result of SOUNDEX (c) will be inserted. The requests should refer to c1. For performance reasons, c1 = SOUNDEX <string literal> should be used instead of the condition c SOUNDS LIKE <string literal>.

<date function>

Function

specifies a date function.

Format

```
<date function> ::=
    ADDDATE    ( <date or timestamp expression>, <expression> )
  | SUBDATE    ( <date or timestamp expression>, <expression> )
  | DATEDIFF   ( <date or timestamp expression>,
                 <date or timestamp expression> )
  | DAYOFWEEK  ( <date or timestamp expression> )
  | WEEKOFYEAR ( <date or timestamp expression> )
  | DAYOFMONTH ( <date or timestamp expression> )
  | DAYOFYEAR  ( <date or timestamp expression> )
  | MAKEDATE   ( <expression>, <expression> )
  | DAYNAME    ( <date or timestamp expression> )
  | MONTHNAME  ( <date or timestamp expression> )

<date or timestamp expression> ::=
    <expression>
```

Syntax Rules

none

General Rules

1. The <date or timestamp expression> must produce a date value, a timestamp value, or an alphanumeric value as the result. This value must correspond to the current date or timestamp format.

2. The <expression> in ADDDATE and SUBDATE must produce a numeric value.

3. The <expression>s in MAKEDATE must produce numeric values. The first <expression> must be greater than or equal to 0. The second <expression> must not be equal to 0.

4. Although the Gregorian calendar was only introduced in 1582, it can also be applied to date functions that use dates prior to that year. This means that every year is assumed to have either 365 or 366 days.

5. ADDDATE
   SUBDATE
   The <expression>s in ADDDATE and SUBDATE represent a number of days.
   The result of ADDDATE and SUBDATE is a date or timestamp value which is obtained either by adding the value of <expression> to the specified date or timestamp value <date or timestamp expression> or by subtracting the value of <expression> from the specified date or timestamp value <date or timestamp expression>. Fractional digits of <expression> are truncated.

If the first or second parameter is the NULL value, then ADDDATE and SUBDATE produce the NULL value as the result. If the second parameter is the special NULL value, then an error message is output.

6. DATEDIFF
The result of DATEDIFF is a numeric value indicating the positive difference (absolute amount) in days with respect to the two specified days. When timestamp values are specified, only the date specifications included there are considered. The time specifications contained in a timestamp value are not considered. If the first or second parameter is the NULL value, then DATEDIFF produces the NULL value as the result.

7. DAYOFWEEK
DAYOFWEEK produces a numeric value between 1 and 7 indicating the day of the week. The first day of a week is Monday, the second day is Tuesday, etc. DAYOFWEEK applied to the NULL value produces the NULL value as the result.

8. WEEKOFYEAR
WEEKOFYEAR produces a numeric value between 1 and 53 indicating the week of the year in which the specified day is located. WEEKOFYEAR applied to the NULL value produces the NULL value as the result.

9. DAYOFMONTH
DAYOFMONTH produces a numeric value between 1 and 31 indicating what day of the month the specified day is. DAYOFMONTH applied to the NULL value produces the NULL value as the result.

10. DAYOFYEAR
DAYOFYEAR produces a numeric value between 1 and 366 indicating what day of the year the specified day is. DAYOFYEAR applied to the NULL value produces the NULL value as the result.

11. MAKEDATE
The result of MAKEDATE is a date. The first <expression> represents a year, the second <expression> represents a day.
For example, MAKEDATE(1996,49) is equal to '19960218' in the date format INTERNAL. Fractional digits of the <expression>s are truncated.
If the first or second parameter is the NULL value, then MAKEDATE produces the NULL value as the result. If the first or second parameter is the special NULL value, then an error message is output.

12. DAYNAME
DAYNAME produces a character string which corresponds to the name of the weekday (from Sunday to Saturday) of the specified day. If the parameter is the NULL value, then DAYNAME produces the NULL value.

13. MONTHNAME
MONTHNAME produces a character string which corresponds to the month name (from January to December) of the specified day. If the parameter is the NULL value, then MONTHNAME produces the NULL value.

<time function>

Function

specifies a time function.

Format

```
<time function> ::=
    ADDTIME  ( <time or timestamp expression>, <time expression> )
  | SUBTIME  ( <time or timestamp expression>, <time expression> )
  | TIMEDIFF ( <time or timestamp expression>,
               <time or timestamp expression> )
  | MAKETIME ( <hours>, <minutes>, <seconds> )

<time or timestamp expression> ::=
    <expression>

<time expression> ::=
    <expression>

<hours> ::=
    <expression>

<minutes> ::=
    <expression>

<seconds> ::=
    <expression>
```

Syntax Rules

none

General Rules

1. The <time or timestamp expression> must produce a time value, a timestamp value or an alphanumeric value as the result. This value must correspond to the current time or timestamp format.

2. The <time expression> must produce a time value or an alphanumeric value as the result. This value must correspond to the current timeformat.

3. ADDTIME
   SUBTIME
   The result of ADDTIME and SUBTIME is a time value or a timestamp value obtained by adding or subtracting the time specified in the second parameter to or from the time value or timestamp value specified in the first parameter. If two time values are specified for SUBTIME, then the second argument must be less than the first argument. If the first or second parameter is the NULL value, then ADDTIME and SUBTIME produce the NULL value as the result.

4. TIMEDIFF
   The arguments must have the same data type, i.e., either be a time value or a timestamp value. The result of TIMEDIFF is a time value indicating the positive time difference between the two specified values. If both arguments are timestamp values or alphanumeric values corresponding to the current timestamp format, then the date specifications in timestamp values are considered for the calculation. For differences of more than 9999 hours, the number of hours modulo 10000 is produced as the result. If the first or second parameter is the NULL value, then TIMEDIFF produces the NULL value as the result.

5. MAKETIME
   The result of MAKETIME is a time value indicating the sum of the three arguments.
   If one of the parameters is the NULL value, then MAKETIME produces the NULL value as the result. If one of the parameters is the special NULL value, then an error message is output.
   <hours>, <minutes>, and <seconds> must be integer values and be greater than or equal to 0. If they are not integer numbers, the fractional digits are truncated.

<extraction function>

Function

specifies a function which either extracts portions from date, time or timestamp values or which forms a date, time, or timestamp value.

Format

```
<extraction function> ::=
    YEAR        ( <date or timestamp expression> )
  | MONTH       ( <date or timestamp expression> )
  | DAY         ( <date or timestamp expression> )
  | HOUR        ( <time or timestamp expression> )
  | MINUTE      ( <time or timestamp expression> )
  | SECOND      ( <time or timestamp expression> )
  | MICROSECOND ( <expression> )
  | TIMESTAMP   ( <expression>[, <expression> ] )
  | DATE        ( <expression> )
  | TIME        ( <expression> )

<date or timestamp expression> ::=
    <expression>

<time or timestamp expression> ::=
    <expression>
```

Syntax Rules

none

General Rules

1. YEAR
   MONTH
   DAY
   The <date or timestamp expression> in YEAR, MONTH, and DAY must be a date or timestamp value.
   The result of YEAR, MONTH or DAY is a numeric value which represents the year or month or day specification made in the <date or timestamp expression>.
   If the parameter is the NULL value, then the result is the NULL value.

2. HOUR
   MINUTE
   SECOND
   The <time or timestamp expression> in HOUR, MINUTE or SECOND must be a time or timestamp value.
   The result of HOUR, MINUTE or SECOND is a numeric value which represents the hour or minute or second specification made in the <time or timestamp expression>.
   If the parameter is the NULL value, then the result is the NULL value.

3. MICROSECOND
   The <expression> in MICROSECOND must be a timestamp value.
   The result of MICROSECOND is a numeric value which represents the microsecond specification made in the <expression>.
   If the parameter is the NULL value, then the result is the NULL value.

4. TIMESTAMP
   If only one <expression> is specified for TIMESTAMP, then this must be a timestamp value or it must produce an alphanumeric value as the result. This value must correspond to the current format of timestamp values. The result of TIMESTAMP then is the timestamp value.
   If two <expression>s are specified for TIMESTAMP, then the first one must be a date value and the second one a time value. Both <expression>s can produce an alphanumeric value as the result. This value must correspond to the current format of date or time values, respectively. The result of TIMESTAMP is a timestamp value formed from the date value, the time value and 0 microseconds.
   If one parameter is the NULL value, then TIMESTAMP produces the NULL value.

5. DATE
   If the <expression> in DATE is a date value or produces an alphanumeric value as the result which corresponds to the current date format, then the result of DATE is this date value.
   If this function is applied to an alphanumeric value, a check is made as to whether the specified value corresponds to the current format of date values.
   If the <expression> in DATE is a timestamp value or produces an alphanumeric value as the result which corresponds to the current format of timestamp values, then the result of DATE is the date value which forms part of the timestamp value.
   If the <expression> in DATE produces either a fixed point number or a floating point number as the result, then the result of DATE is a date value which corresponds to the xth day following the 12/31/0000, where x =TRUNC(<expression>).
   If the parameter is the NULL value, then DATE produces the NULL value. If the parameter is the special NULL value, then an error message is output.

6. TIME
   If the <expression> in TIME is a time value or produces an alphanumeric value as the result which corresponds to the current time format, then the result of TIME is this time value.
   If this function is applied to an alphanumeric value, a check is made as to whether the specified value corresponds to the current format of time values.
   If the <expression> in TIME is a timestamp value or produces an alphanumeric value as the result which corresponds to the current format of timestamp values, then the result of TIME is the time value which forms part of the timestamp value.
   If the parameter is the NULL value, then TIME produces the NULL value.

<special function>

Function

specifies a function which is not limited to specific data types.

Format

```
<special function> ::=
    VALUE    ( <expression>, <expression>,... )
  | GREATEST ( <expression>, <expression>,... )
  | LEAST    ( <expression>, <expression>,... )
  | DECODE   ( <check expression>,
               <search and result spec>,...
               [, <default expression> ] )

<search and result spec> ::=
    <search expression>, <result expression>

<search expression> ::=
    <expression>

<result expression> ::=
    <expression>

<check expression> ::=
    <expression>

<default expression> ::=
    <expression>
```

Syntax Rules

none

General Rules

1. VALUE
   The arguments of the VALUE function must be comparable.
   The arguments are evaluated one after the other in the specified order. If an argument is a non-NULL value, then the result of the VALUE function is the first occurring non-NULL value. If every argument is the special NULL value, then the result of the VALUE function is the special NULL value. Otherwise, the result is the NULL value.
   The VALUE function can be used for replacing a NULL value with a non-NULL value. An example be 'SALARY + VALUE(BONUS,0)' where SALARY and BONUS are assumed to be column names of one table.

2. GREATEST
   LEAST
   GREATEST and LEAST can be applied to any data type. The data types of the <expression>s must be comparable. The result of GREATEST or LEAST is the greatest or smallest value determined as the result of one of the <expression>s. If at

least one argument is the NULL value or the special NULL value, then the result of GREATEST or LEAST is the NULL value.

3. DECODE
The data types of the <check expression> and of the <search expression>s must be comparable. The data types of the <result expression>s and the optional <default expression> must be comparable. The data types of the <search expression>s and of the <result expression>s need not be comparable.
DECODE compares the result of the <check expression> with one <search expression> result after the other. If conformity is established, the result of DECODE is the result of the <result expression> which is included in the <search and result spec> containing the matching <search expression>. If the result of the <check expression> and the result of a <search expression> is the NULL value, then conformity is established. The comparison of the special NULL value with any other value never results in conformity.
If no conformity can be established, DECODE produces the result of the <default expression>. If no <default expression> is specified, then the result of DECODE is the NULL value.

<conversion function>

Function

specifies a function which converts a value of one data type into another data type.

Format

```
<conversion function> ::=
    NUM       ( <expression> )
  | CHR       ( <expression>[, <unsigned  integer> ] )
  | HEX       ( <expression> )
  | CHAR      ( <expression>[, <datetimeformat> ] )
```

Syntax Rules

none

General Rules

1. NUM
   NUM can be applied to character strings with the code attribute ASCII or EBCDIC, to date, time or timestamp values, to numeric and Boolean values. If a character string can be interpreted as a numeric value, then NUM transforms this character string into the corresponding numeric value. NUM applied to a numeric value has no effect. NUM applied to a Boolean value produces 1 for the Boolean value TRUE and 0 for the Boolean value FALSE.
   NUM applied to the NULL value produces the NULL value. NUM applied to the special NULL value produces the special NULL value. If NUM is applied either to a character string which cannot be interpreted as a numeric value or to an argument which is neither a character string with the code attribute ASCII or EBCDIC nor a numeric or Boolean value, then an error message is output. If NUM is applied to a character string which can be interpreted as a numeric value outside the interval -9.99999999999999999E+62 and 9.99999999999999999E+62, then NUM produces the special NULL value.

2. CHR
   CHR can only be applied to numeric values, character strings, and Boolean values. CHR transforms a numeric value into a character string which corresponds to the CHAR representation of the numeric value. CHR applied to a character string has no effect. CHR applied to a Boolean value produes 'T' for the Boolean value TRUE and 'F' for the Boolean value FALSE.
   CHR applied to the NULL value produces the NULL value. CHR applied to the special NULL value produces an error message. If CHR is applied to an argument which is neither a numeric value nor a character string, nor a Boolean value, then an error message is output. The code attribute of the resultant character string corresponds to the code type of the computer.
   CHR(a,k), where 1<=k<=254, defines an output with the length attribute k. If k is not specified, a value is determined for k according to the data type and length of a. If a denotes a column of data type FLOAT(p), then the following is true:
   if p=1, then k=6; if p>1, then k=p+6.

If a denotes a column of data type FIXED(p,s), then the following is true:
if p=s, then k=p+3; if p>s>0, then k=p+2; if s=0, then k=p+1.

3. HEX
   HEX produces the hexadecimal representation of the argument. HEX can be applied to any data type with the restriction that character strings may only have a maximum length of 127. HEX applied to the NULL value produces the NULL value as the result. HEX applied to the special NULL value produces an error message.

4. CHAR
   CHAR can only be applied to date, time or timestamp values. The result of CHAR is a character string which corresponds to the date, time or timestamp value in the format specified in the optional second parameter. If the second parameter is missing, the current date and time format is assumed for <datetimeformat>. The different presentation formats for date, time, and timestamp values are described in the chapter 'Date and Time Format'.
   If the first parameter is the NULL value, then CHAR produces the NULL value as the result.

<set function spec>

Function

specifies a function. The argument of the function is a set of values.

Format

```
<set function spec> ::=
    COUNT (*)
  | <distinct function>
  | <all function>

<distinct function> ::=
    <set function name> ( DISTINCT <expression> )

<all function> ::=
    <set function name> ( [ALL] <expression> )

<set function name> ::=
    COUNT
  | MAX
  | MIN
  | SUM
  | AVG
  | STDDEV
  | VARIANCE
```

Syntax Rules

1. The <expression> must not contain a <set function spec>.

General Rules

1. Each <query spec> contains a <table expression>. The <table expression> produces a temporary result table. This temporary result table can be grouped using a <group clause>. The argument of a <distinct function> or an <all function> is created on the basis of a temporary result table or group.

2. The argument of a <distinct function> is a set of values. This set is generated by applying the <expression> to each row of a temporary result table or of a group and by eliminating all NULL values and duplicate values. Special NULL values are not removed. Two special NULL values are assumed to be identical.
   If the set is empty and the <distinct function> is applied to the whole temporary result table, the result of AVG, MAX, MIN, STDDEV, SUM, and VARIANCE is the NULL value, and the result of COUNT is 0.
   If there is no group to which the <distinct function> could be applied, the result table is empty.
   If the set contains at least one special NULL value, the result of the <distinct function> is the special NULL value.

3. The argument of an <all function> is a set of values. This set is generated by applying the <expression> to each row of the temporary result table or of a group and by eliminating all NULL values from the result. Special NULL values are not removed. Two special NULL values are assumed to be identical.
   If the set is empty and the <all function> is applied to the whole temporary result table, the result of AVG, MAX, MIN, STDDEV, SUM, and VARIANCE is the NULL value, and the result of COUNT is 0.
   If there is no group to which the <all function> could be applied, the result table is empty.
   If the set contains at least one special NULL value, the result of the <all function> is the special NULL value.
   The result of an <all function> is independent of whether the key word ALL is specified or not.

4. The result of COUNT(*) is the number of rows in a temporary result table or of a group. The result of COUNT (DISTINCT <expression> is the number of values of the argument in the <distinct function>. The result of COUNT (ALL <expression>) is the number of values of the argument in the <all function>.

5. The result of MAX is the largest value of the argument. The result of MIN is the smallest value of the argument.

6. SUM can only be applied to numeric values. The result of SUM is the sum of the values of the argument. The result has the data type FLOAT(18).

7. AVG can only be applied to numeric values. The result of AVG is the arithmetical average of the values of the argument. The result has the data type FLOAT(18).

8. STDDEV can only be applied to numeric values. The result of STDDEV is the standard deviation of the values of the argument. The result has the data type FLOAT(18).

9. VARIANCE can only be applied to numeric values. The result of VARIANCE is the variance of the values of the argument. The result has the data type FLOAT(18).

10. Contrary to the usual locking mechanisms, no locks are set for some <set function spec>s, irrespective of the <isolation spec> specified when connecting to the database.

<expression>

Function

specifies a value which is generated, if required, by applying arithmetical operators to values.

Format

```
<expression> ::=
    <term>
  | <expression> + <term>
  | <expression> - <term>

<term> ::=
    <factor>
  | <term> * <factor>
  | <term> / <factor>
  | <term> DIV <factor>
  | <term> MOD <factor>

<factor> ::=
    [<sign>] <primary>

<sign> ::=
    +

  | -

<primary> ::=
    <value spec>
  | <column spec>
  | <function spec>
  | <set function spec>
  | (<expression>)

<expression list> ::=
  (<expression>,...)
```

Syntax Rules

none

General Rules

1. The arithmetical operators +, -, *, /, DIV, and MOD can only be applied to numeric data types.

2. The result of an <expression> is either a non-NULL value, the NULL value, or the special NULL value.

3. The result of an <expression> is the NULL value if any <primary> has the NULL value.

4. The result of an <expression> is the special NULL value if any <primary> has the special NULL value. The result of an <expression> is the special NULL value if this <expression> leads to a division by 0 or to an overflow of the internal temporary result.

5. If both operands of an operator are fixed point numbers, then the result is either a fixed point number or a floating point number. The data type of the result depends on the operation as well as on the precision and scale of the operands. Note that the data type of the specified column is used in case of a column name specification, not the precision and scale of the current column value.

   The result of addition, subtraction, and multiplication is generated from a temporary result which can have more than 18 valid digits. If the temporary result has no more than 18 valid digits, the final result is equal to the temporary result. Otherwise, a result is generated as a floating point number with a precision of 18 digits. Digits after the decimal point are truncated, if necessary.

   Let p and s represent the precision and scale of the first operand, p' and s' the corresponding values of the second operand.

   If max(p-s,p'-s') + max(s,s') + 1 <= 18, then addition and subtraction produce a valid result as a fixed point number. The precision of the result obtained by addition and subtraction is max(p-s,p'-s') + max(s,s') + 1, the scale is max(s,s').

   If (p+p') <= 18, then multiplication produces a valid result as a fixed point number. The precision of the result obtained by multiplication is p+p', the scale is s+s'.

   If (p-s+s') <= 18, then division produces a valid result as a fixed point number. The precision of the result obtained by division is 18 and the scale is 18-(p-s+s').
   If the second operand of the division has the value 0, the result is the special NULL value.

6. If a and b are integers and ABS(a)<1E18 and ABS(b)<1E18 and b is not 0, then (a DIV b) = TRUNC(a/b).
   If b=0, then the result of a DIV b is the special NULL value.
   If any of the specified conditions is not satisfied, an error message is issued.

7. If a and b are integers and ABS(a)<1E18 and 0<b<1E18, then the following is true:

   ```
   Let m = a-b*(a DIV b)
   If m>=0, then (a MOD b) = m
   If m<0 , then (a MOD b) = m+b
   ```

   If b=0, then the result of a MOD b is the special NULL value. If any of the specified conditions is not satisfied, an error message is issued.

8. If a floating point number occurs in an arithmetical expression, the result is a floating point number.

9. If no parentheses are used, the operators have the following precedence: <sign> has a higher precedence than the multiplicative operators *, /, DIV, and MOD, and the additive operators + and -. The multiplicative operators have a higher precedence than the additive operators. The multiplicative operators have the same precedence among each other, and the same applies to the additive operators. Operators with the same precedence are evaluated from left to right.

<predicate>

Function

specifies a condition which is 'true', 'false', or 'unknown'.

Format

```
 <predicate> ::=
     <between predicate>
   | <bool predicate>
   | <comparison predicate>
   | <default predicate>
   | <exists predicate>
   | <in predicate>
   | <join predicate>
   | <like predicate>
   | <null predicate>
   | <quantified predicate>
   | <rowno predicate>
   | <sounds predicate>
```

Syntax Rules

none

General Rules

1. A predicate specifies a condition which is either 'true' or 'false' or 'unknown'. The result is generated by applying the predicate either to a given table row or to a group of table rows that was formed by the <group clause>.

2. Columns with the same code attribute can be compared to each other. Columns with the different code attributes ASCII and EBCDIC can be compared to each other. Columns of the code attributes ASCII and EBCDIC can be compared to date, time or timestamp values.

3. LONG columns can only be used in the <null predicate>.

See also

<between predicate>

<bool predicate>

<comparison predicate>

<default predicate>

<exists predicate>

<in predicate>

<join predicate>

<like predicate>

<null predicate>

<quantified predicate>

<rowno predicate>

<sounds predicate>

<between predicate>

Function

checks whether a value lies within a given interval.

Format

```
 <between predicate> ::=
     <expression> [NOT] BETWEEN <expression> AND <expression>
```

Syntax Rules

none

General Rules

1. Let x, y, and z be the results of the first, second and third <expression>. The values x, y and z must be comparable with each other.
2. (x BETWEEN y AND z) has the same result as (x>=y AND x<=z).
3. (x NOT BETWEEN y AND z) has the same result as
   NOT(x BETWEEN y AND z).
4. If x, y or z are NULL values, then (x [NOT] BETWEEN y AND z) is unknown.

<bool predicate>

Function

specifies a comparison between two Boolean values.

Format

```
 <bool predicate> ::=
     <column spec> [ IS [NOT] <bool spec> ]

 <bool spec> ::=
     TRUE
   | FALSE
```

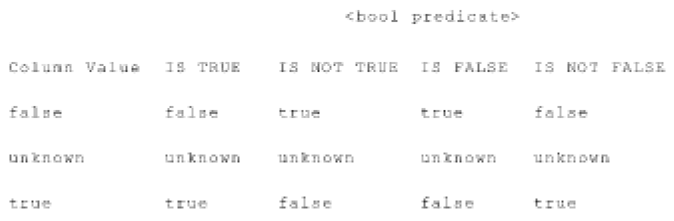
Syntax Rules

1. If only one <column spec> is specified, then this corresponds to the syntax <column spec> IS TRUE.

General Rules

1. The <column spec> must always denote a column of the data type BOOLEAN.

2. The following rules apply to the result of the <bool predicate>:



	<bool predicate>			
Column Value	IS TRUE	IS NOT TRUE	IS FALSE	IS NOT FALSE
false	false	true	true	false
unknown	unknown	unknown	unknown	unknown
true	true	false	false	true

<comparison predicate>

Function

specifies a comparison between two values or between lists of values.

Format

```
<comparison predicate> ::=
    <expression> <comp op> <expression>
  | <expression> <comp op> <subquery>
  | <expression list> <equal or not> (<expression list>)
  | <expression list> <equal or not> <subquery>

<comp op> ::=
    < | > | <> | != | = | <= | >=
  | ¬= | ¬< | ¬> for a computer with the code type EBCDIC
  | ~= | ~< | ~> for a computer with the code type ASCII

<equal or not> ::=
    =
  | <>
  | ¬= for a computer with the code type EBCDIC
  | ~= for a computer with the code type ASCII
```

Syntax Rules

1. The <subquery> must produce a result table which contains as many columns as <expression>s are specified at the left of the operator. The <subquery> may contain no more than one row.

2. The <expression list> specified to the right of <equal or not> must contain as many <expression>s as are specified in the <expression list> at the left of <equal or not>.

General Rules

1. Let x be the result of the first <expression> and y the result of the second <expression> or of the <subquery>. The values x and y must be comparable with each other.

2. Numbers are compared to each other according to their algebraic values.

3. Character strings are compared character by character. If the character strings have different lengths, the shorter one is padded with blanks (code attribute ASCII, EBCDIC) or with binary zeros (code attribute BYTE), so that they have the same length when being compared. If the character strings have the different code attributes ASCII and EBCDIC, one of these character strings is implicitly converted so that they have the same code attribute.

4. Two character strings are identical if they have the same characters in the same positions. If they are not identical, their relation is determined by the first differing

character found during comparison from left to right. This comparison is made according to the code attribute (ASCII, EBCDIC, or BYTE) chosen for this column.

5. If an <expression list> is specified to the left of <equal or not>, then x is the value list consisting of the results of the <expression>s x1, x2, ..., xn of this value list. y is the result of the <subquery> or the result of the second value list. A value list y consists of the results of the <expression>s y1, y2, ..., yn. A value xm must be comparable with the corresponding value ym.

6. x=y is true if xm=ym is valid for all m=1, ..., n. x<>y is true if there is at least one m for which xm<>ym is valid. (x <equal or not> y) is unknown if there is no m for which (xm <equal or not> ym) is false and if there is at least one m for which (xm <equal or not> ym) is unknown.

7. If x, xm, ym, or y are NULL values, or if the result of the <subquery> is empty, then (x <comp op> y) or (x <equal or not> y) is unknown.

8. The <join predicate> is a special case of the <comparison predicate>. The <join predicate> is described in a separate section.

<default predicate>

Function

checks whether a column contains the DEFAULT value defined for this column.

Format

```
 <default predicate> ::=
     <column spec> <comp op> DEFAULT
```

Syntax Rules

none

General Rules

1. A <default spec> must have been defined in the <create table statement> or <alter table statement> for the specified column.

2. If the column contains the NULL value, then <column spec> <comp op> DEFAULT is undefined.

3. The same rules apply that are listed for the <comparison predicate>.

<exists predicate>

Function

checks whether a result table contains at least one row.

Format

```
 <exists predicate> ::=
     EXISTS <subquery>
```

Syntax Rules

none

General Rules

1. The truth value of an <exists predicate> is either true or false.
2. Let T be the result table produced by <subquery>. (EXISTS T) is true if and only if T contains at least one row.

<in predicate>

Function

checks whether a value or value list is contained in a given set of values or set of value lists.

Format

```
 <in predicate> ::=
     <expression> [NOT] IN <subquery>
   | <expression> [NOT] IN (<expression>,...)
   | <expression list> [NOT] IN <subquery>
   | <expression list> [NOT] IN (<expression list>,...)
```

Syntax Rule

1. The <subquery> must produce a result table which contains as many columns as <expression>s are specified to the left of the operator IN.

2. Each <expression list> specified to the right of the operator IN must contain as many <expression>s as are specified in the <expression list> to the left of the operator IN.

General Rules

1. Let x be the result of the <expression> and S be either the result of the <subquery> or the values of the sequence of <expression>s. S is a set of values. The value x and the values in S must be comparable with each other.

2. If an <expression list> is specified to the left of the operator IN, then let x be the value list consisting of the result of the <expression>s x1, x2, ..., xn of this value list. Let S be either the result of the <subquery> that consists of a set of value lists s or a sequence of value lists s. A value list s consists of the results of the <expression>s s1, s2, ..., sn. A value xm must be comparable with all values sm.

3. x=s is true if xm=sm is valid for all m=1, ..., n. x=s is false if there is at least one m for which xm=sm is false. x=s is unknown if there is no m for which xm=sm is false and if there is at least one m for which xm=sm is unknown.

4. If x=s is true for at least one value or value list s of S, then (x IN S) is true.

5. If x=s is not true for any value or any value list s of S and x=s is unknown for at least one value or value list s of S, then (x IN S) is unknown.

6. If S is empty or if x=s is false for every value or value list s of S, then (x IN S) is false.

7. (x NOT IN S) has the same result as NOT(x IN S).

<join predicate>

Function

specifies a join.

Format

```
 <join predicate> ::=
     <expression> [<outer join indicator>]
             <comp op>
     <expression> [<outer join indicator>]

 <outer join indicator>  ::=
     (+)
```

Syntax Rules

1. A <join predicate> can be specified without, with one or with two <outer join indicator>s.

General Rules

1. Each <expression> must contain a <column spec>. There must be a <column spec> of the first <expression> and a <column spec> of the second <expression>, so that the <column spec>s refer to different table names or reference names.

2. Let x be the value of the first <expression> and y the value of the second <expression>. The values x and y must be comparable with each other.

3. The same rules apply that are listed for the <comparison predicate>.

4. If at least one <outer join indicator> is specified in a <join predicate> of a <search condition>, the corresponding <table expression> must have two underlying base tables or the following must apply:

   a) <outer join indicator>s are only specified for one of the tables specified in the <from clause>.
   b) Any <join predicate> of this table to just one other table contain the <outer join indicator>.
   c) All the other <join predicate>s contain no <outer join indicator>.

   If more than two underlying base tables are required for the <query spec> and if one of the above-mentioned rules cannot be satisfied, a <query expression> can be used in the <from clause>.
   The term of underlying base tables is explained in detail in the chapter '<from clause>'.

5. Usually, rows are only transferred to the result table if they have a counterpart corresponding to the <comp op> in the other table specified in the <join predicate>.
   If it must be ensured that every row of a table is contained in the result table at least once, the <outer join indicator> must be specified on the side of <comp op> where the

other table is specified.
If it is not possible to find at least one counterpart for a table row in the other table, this row is used to build a row for the result table. The NULL value is then used for the output columns which are usually formed from the other table's columns.
Since the <outer join indicator> can be specified on both sides of <comp op> if the <table expression> has just two underlying base tables, it can be ensured for both tables that every row is contained in the result table at least once.

6. The <join predicate> is a special case of the <comparison predicate>. The number of <join predicate>s in a <search condition> is limited to 64.

<like predicate>

Function

serves to search for character strings which have a particular pattern.

Format

```
<like predicate> ::=
    <expression> [NOT] LIKE <like expression>
                 [ESCAPE <expression>]

<like expression> ::=
    <expression>
  | '<pattern element>...'

<pattern element> ::=
    <match string>
  | <match set>

<match string> ::=
    %
  | *
  | X'1F'

<match set> ::=
    <underscore>
  | ?
  | X'1E'
  | <match char>
  | ([<complement sign>]<match class>...)

<match char> ::=
    Any character except
    %, *, X'1F', <underscore>, ?, X'1E', (.

<complement sign> ::=
    ^
  | ~
  | ¬


<match class> ::=
    <match range>
  | <match element>

<match range> ::=
    <match element>-<match element>

<match element> ::=
    Any character except )
```

Syntax Rules

none

General Rules

1. The <expression> of the <like expression> must produce an alphanumeric value, or a date or time value.

2. A <match string> stands for a sequence of n characters, where n >= 0.

3. A <match set> is a set of characters.
   Thereby <underscore>, '?', X'1E' stand for any character, <match char> for itself.
   A sequence of <match class>es consists of a list of characters (<match element>s) or the specification of ranges of characters (<match range>s) or a combination of these.
   A sequence of <match class>es can be negated by placing a <complement sign> in front of it. It is not possible to place a <complement sign> in front of each single <match class>.
   Note that the <complement sign> '~' can only be used in the case of a computer with the code type ASCII and the <complement sign> '^' can only be used in the case of a computer with the code type EBCDIC.

4. Let x be the value of the <expression> and y the value of the <like expression>.

5. If x or y are NULL values, then (x LIKE y) is unknown.

6. If x and y are non-NULL values, then (x LIKE y) is either true or false.

7. (x LIKE y) is true if x can be divided into substrings in such a way that the following is valid:

   a) A substring of x is a sequence of 0, 1, or more contiguous characters, and each character of x belongs to exactly one substring.

   b) If the nth <pattern element> of y is a <match set>, then the nth substring of x is a single character which is contained in the <match set>.

   c) If the nth <pattern element> of y is a <match string>, then the nth substring of x is a sequence of 0 or more characters.

   d) The number of substrings of x and y is identical.

8. If ESCAPE is specified, then the corresponding <expression> must produce an alphanumeric value which consists of just one character. If this escape character is contained in the <like expression>, the subsequent character is considered to be a <match char>; i.e., it stands for itself.
   The use of an escape character is required if <underscore>, '?', '%' or '*', or the hexadecimal value X'1E' or X'1F' is to be searched for.

   Example:

   LIKE '*_'
   Any character string having the minimum length of 1 is searched for.

   LIKE '*:_*' ESCAPE ':'
   A character string having any number of characters is searched for, where the character string must contain an <underscore>.

9. (x NOT LIKE y) has the same result as NOT(x LIKE y).

<null predicate>

Function

specifies a check for a NULL value.

Format

```
 <null predicate> ::=
     <expression> IS [NOT] NULL
```

Syntax Rules

none

General Rules

1. The truth value of a <null predicate> is either true or false.

2. Let x be the value of the <expression>. (x IS NULL) is true if and only if x is the NULL value.

3. If x is the special NULL value, then (x IS NULL) is false.

4. (x IS NOT NULL) has the same result as NOT(x IS NULL).

<quantified predicate>

Function

compares a value to a single-column result table.

Format

```
<quantified predicate> ::=
    <expression> <comp op> <quantifier> (<expression>,...)
  | <expression> <comp op> <quantifier> <subquery>
  | <expression list> <equal or not>
    <quantifier> (<expression list>,...)
  | <expression list> <equal or not> <quantifier> <subquery>

<quantifier> ::=
    ALL
  | <some>

<some> ::=
    SOME
  | ANY
```

Syntax Rules

1. The <subquery> must produce a result table which contains as many columns as <expression>s are specified to the left of the operator.

2. Each <expression list> specified to the right of <equal or not> must contain as many <expression>s as are specified in the <expression list> to the left of <equal or not>.

General Rules

1. Let x be the result of the <expression> and S the result of the <subquery> or sequence of <expression>s. S is a set of values. The value x and the values in S must be comparable with each other.

2. If S is empty or (x <comp op> s) is true for every value s of S, then (x <comp op> ALL S) is true.

3. If (x <comp op> s) is not false for any value s of S and (x <comp op> s) is unknown for at least one value s of S, then (x <comp op> ALL S) is unknown.

4. If (x <comp op> s) is false for at least one value s of S, then (x <comp op> ALL S) is false.

5. If (x <comp op> s) is true for at least one value s of S, then (x <comp op> <some> S) is true.

6. If (x <comp op> s) is not true for any value s of S and (x <comp op> s) is unknown for at least one value s of S, then (x <comp op> <some> S) is unknown.

7. If S is empty or (x <comp op> s) is false for every value s of S, then (x <comp op> <some> S) is false.

8. If an <expression list> is specified to the left of <equal or not>, then let x be the value list consisting of the results of the <expression>s x1, x2, ..., xn of this value list. Let S be either the result of the <subquery> consisting of a set of value lists s or a sequence of value lists s. A value list s consists of the results of the <expression>s s1, s2, ..., sn. A value xm must be comparable with all values sm.

9. x=s is true if xm=sm is valid for all m=1, ...n. x<>s is true if there is at least one m for which xm<>sm. (x <equal or not> s) is unknown if there is no m for which (xm <equal or not> sm) is false and if there is at least one m for which (xm <equal or not> sm) is unknown.

10. If S is empty or (x <equal or not> s) is true for each value list s of S, then (x <equal or not> ALL S) is true.

11. If (x <equal or not> s) is false for no value list s of S and (x <equal or not> s) is unknown for at least one value list s of S, then (x <equal or not> ALL S) is unknown.

12. If (x <equal or not> s) is false for at least one value list s of S, then (x <equal or not> ALL S) is false.

13. If (x <equal or not> s) is true for at least one value list s of S, then (x <equal or not><some> S) is true.

14. If (x <equal or not> s) is true for no value list s of S and (x <equal or not> s) is unknown for at least one value list s of S, then (x <equal or not> <some> S) is unknown.

15. If S is empty or (x <equal or not> s) is false for each value list s of S, then (x <equal or not> <some> S) is false.

<rowno predicate>

Function

limits the number of rows of a result table.

Format

```
<rowno predicate> ::=
    ROWNO <  <rowno spec>
  | ROWNO <= <rowno spec>

<rowno spec> ::=
    <unsigned integer>
  | <parameter spec>
```

Syntax Rules

1. The <rowno predicate> may only be used in a <where clause> of a <query spec>. In the <where clause>, it can be used like any other <predicate>. But there is the restriction that the <rowno predicate> must be logically combined with other predicates by AND, that it must not be negated by NOT, and that it may occur only once in the <where clause>. To guarantee that these rules are met, it is recommended to use the format
   WHERE (<search condition>) AND <rowno predicate>.

General Rules

1. The <rowno spec> specifies the maximum number of rows that the result table is to contain. It must specify a value which allows at least for a single-row result table.

2. If without a <rowno predicate> specification, more result rows might be found than are specified in the <rowno spec>, then for a <rowno predicate>, these result rows would not be considered and no error message would be output.

3. If a <rowno predicate> and an <order clause> are specified, then only the first n result rows are searched and sorted. The result usually differs from that which would have been obtained without a <rowno predicate> specification, only considering the first n result rows.

4. If a <rowno predicate> and a <set function spec> are specified, then the <set function spec> is only applied to the number of result rows limited by the <rowno spec>.

<sounds predicate>

Function

specifies a phonetic comparison.

Format

```
 <sounds predicate> ::=
     <expression> [NOT] SOUNDS [LIKE] <expression>
```

Syntax Rules

1. The specification of LIKE in the <sounds predicate> has no effect.

General Rules

1. The values of the <expression>s must be alphanumeric and have the code attribute ASCII or EBCIDC.
2. Let x be the value of the first <expression> and y the value of the second <expression>.
3. If x or y are NULL values, then (x SOUNDS y) is unknown.
4. If x and y are non-NULL values, then (x SOUNDS y) is either true or false.
5. If x and y are phonetically identical, then (x SOUNDS y) is true. The phonetic comparison is carried out according to the SOUNDEX algorithm. First, all vowels and some consonants are eliminated, then all consonants which are similar in sound are mapped to each other. See also the function <string function>.
6. (x NOT SOUNDS y) has the same result as NOT (x SOUNDS y).

<search condition>

Function

combines conditions which can be 'true', 'false', or 'unknown'.

Format

```
<search condition> ::=
    <boolean term>
  | <search condition> OR <boolean term>

<boolean term> ::=
    <boolean factor>
  | <boolean term> AND <boolean factor>

<boolean factor> ::=
    [NOT] <boolean primary>

<boolean primary> ::=
    <predicate>
  | (<search condition>)
```

Syntax Rules

none

General Rules

1. Each specified <predicate> is applied to a given table row or to a group of table rows that was formed by the <group clause>. The results are combined with the specified Boolean operators (AND, OR, NOT) in order to generate the result of the <search condition>.

2. If no parentheses are used, the precedence of the operators is as follows: NOT has a higher precedence than AND and OR, AND has a higher precedence than OR. Operators having the same precedence are evaluated from left to right.

3. The following rules apply to NOT:

   NOT(true) is false.
   NOT(false) is true.
   NOT(unknown) is unknown.

4. The following rules apply to AND:

AND	false	unknown	true
false	false	false	false
unknown	false	unknown	unknown
true	false	unknown	true

5. The following rules apply to OR:

OR	false	unknown	true
false	false	unknown	true
unknown	unknown	unknown	true
true	true	true	true

SQL Statement

Function

specifies any SQL statement.

Format

```
<sql statement> ::=
    <create table statement>
  | <drop table statement>
  | <alter table statement>
  | <rename table statement>
  | <rename column statement>
  | <exists table statement>
  | <create domain statement>
  | <drop domain statement>
  | <create synonym statement>
  | <drop synonym statement>
  | <rename synonym statement>
  | <create snapshot statement>
  | <drop snapshot statement>
  | <create snapshot log statement>
  | <drop snapshot log statement>
  | <create view statement>
  | <drop view statement>
  | <rename view statement>
  | <create index statement>
  | <drop index statement>
  | <comment on statement>

  | <create user statement>
  | <create usergroup statement>
  | <drop user statement>
  | <drop usergroup statement>
  | <alter user statement>
  | <alter usergroup statement>
  | <grant user statement>
  | <grant usergroup statement>
  | <alter password statement>
  | <grant statement>
  | <revoke statement>

  | <insert statement>
  | <update statement>
  | <delete statement>
  | <refresh statement>
  | <clear snapshot log statement>
  | <next stamp statement>

  | <query statement>
  | <open cursor statement>
  | <fetch statement>
  | <close statement>
  | <single select statement>
  | <select direct statement: searched>
  | <select direct statement: positioned>
  | <select ordered statement: searched>
  | <select ordered statement: positioned>
```

```
   | <explain statement>

   | <connect statement>
   | <commit statement>
   | <rollback statement>
   | <subtrans statement>
   | <lock statement>
   | <unlock statement>
   | <release statement>

   | <update statistics statement>
   | <monitor statement>
```

Syntax Rules

none

General Rules

1. The SQL statements of the 1st block are described in the chapter '<create table statement>'.
2. The SQL statements of the 2nd block are described in the chapter '<create user statement>'.
3. The SQL statements of the 3rd block are described in the chapter 'Data Manipulation'.
4. The SQL statements of the 4th block are described in the chapter 'Data Retrieval'.
5. The SQL statements of the 5th block are described in the chapter 'Transactions'.
6. The SQL statements of the 6th block are described in the chapter 'Statistics'.
7. All SQL statements can be embedded in programming languages. For a detailed description, refer to the online help on the precompilers.
8. All SQL statements, except those concerning the <next stamp statement>, can be specified interactively.

<create table statement>

Function

creates a base table.

Format

```
<create table statement> ::=
    CREATE TABLE <table name> [(<table description element>,...)]
          [<table option>]
          [AS <query expression> [<duplicates clause>] ]
  | CREATE TABLE <table name> LIKE <source table>
            [<table option>]

<table description element> ::=
    <column definition>
  | <constraint definition>
  | <referential constraint definition>
  | <key definition>
  | <unique definition>

<table option> ::=
    WITH REPLICATION
  | IGNORE ROLLBACK

<source table> ::=
    <table name>
```

Syntax Rules

1. If no <query expression> is specified, the <create table statement> must contain at least one <column definition>.

2. A table may contain up to 255 <column definition>s. If a table is defined without a key column, ADABAS implicitly creates a key column. In this case, up to 254 additional columns can be defined.

3. The <create table statement> may contain no more than one <key definition>.

General Rules

1. Omitting the <owner> in the <table name> has the same effect as specifying the current user as <owner>.
   If TEMP is specified as <owner>, then a temporary table is created which only exists for the duration of the session of the current user. At the end of the session, both the table as well as the rows contained in it are dropped.
   If the <owner> of the <table name> is not TEMP, then <owner> must be identical to the name of the current user.

2. As a result of a <create table statement>, data describing the table is stored in the

catalog. This data is called metadata. It is stored on the HOME SERVERDB of the current user. Any table rows inserted with the <insert statement> are stored in this SERVERDB as well. Tables generated using the <create table statement> are called base tables.

3. The <table name> must not be identical to the name of an existing table of the current user.

4. If the <owner> of the <table name> is not TEMP, then the current user must have DBA or RESOURCE status.

5. Tables for which IGNORE ROLLBACK is specified are not affected by the transaction mechanism; i.e., rolling back a transaction does not roll back any modifications pertaining to this table. IGNORE ROLLBACK can only be specified for temporary tables.

6. Users with DBA status can use the <table option> WITH REPLICATION to provide each SERVERDB in the distributed database with a replication of both the metadata and the table rows, so that access to the table can always be handled from the current SERVERDB without requiring network communication.
   While executing the <create table statement>, the SERVERDB must belong to the majority.
   WITH REPLICATION cannot be specified for temporary tables.

7. If a <query expression> is specified, a base table is created with the same structure as the result table defined by the <query expression>. If <column definition>s are specified, then each <column definition> may only consist of a <column name>, and the number of <column definition>s must equal the number of columns in the result table generated by the <query expression>. The <data type> of the ith column of the generated base table corresponds to that of the ith column in the result table generated by the <query expression>. The result table must not contain LONG columns. If the <create table statement> contains no <column definition>s, the column names are taken from the result table as well.
   The rows of the result table are implicitly inserted into the generated base table. The <duplicates clause> (see '<insert statement>') can be used to control the behavior of the statement in the event of key collisions.
   If the <duplicates clause> is omitted or REJECT DUPLICATES is specified, then the <create table statement> fails whenever key collisions occur.
   If IGNORE DUPLICATES is specified, then any rows causing key collisions upon insertion are ignored.
   If UPDATE DUPLICATES is specified, then any rows causing key collisions upon insertion overwrite the rows with which they collide.
   The same restrictions apply for the <query expression> here as for the <query expression> of an <insert statement>.

8. The current user becomes the owner of the created table. The user obtains the INSERT, UPDATE, DELETE, and SELECT privilege for this table. For nontemporary tables, the owner has the INDEX, REFERENCES, and ALTER privilege, in addition.

9. <source table> must denote a base table, a view table, a snapshot table, or a synonym. Specifying a synonym has the same effect as specifying the table for which the synonym was defined.
   The user must have at least one privilege for this table.
   If 'LIKE <source table>' is specified, an empty base table is created which, from the point of view of the current user, has the same structure as the table <source table>; i.e., it has all columns with the same column names and definitions as the <source

table> that are known to the user. This view need not be identical with the actual structure of the <source table>, since the user may not know all the columns because of privilege limitations.
If all key columns of the <source table> are contained in the newly created table, then these make up the key columns of this table. Otherwise, ADABAS implicitly inserts a key column SYSKEY CHAR(8) BYTE which makes up the key of the base table.
The <default spec>s of the accepted columns of the <source table>, as well as all <constraint definition>s of the <source table> whose referenced columns are accepted in the table, are also valid for the newly created table. The current user is the owner of the created base table.

10. Once a table has been created, the properties of a table can be changed. Under certain conditions, the <alter table statement> can be used to add further columns or to drop existing columns or to alter data types and the <constraint definition>. Columns can be renamed with the <rename column statement>. The table can be renamed with the <rename table statement>.

See also

<column definition>

<constraint definition>

<referential constraint definition>

<key definition>

<unique definition>

<column definition>

Function

defines a table column.

Format

```
<column definition> ::=
    <column name> <data type> <column attributes>
  | <column name> <domain name> [<key or not null spec>]

<data type> ::=
    CHAR[ACTER] (<unsigned integer>) [<code spec>]
  | VARCHAR (<unsigned integer>) [<code spec>]
  | LONG [VARCHAR] [<code spec>]
  | BOOLEAN
  | FIXED (<unsigned integer> [,<unsigned integer>])
  | FLOAT (<unsigned integer>)
  | DATE
  | TIME
  | TIMESTAMP

<code spec> ::=
    ASCII
  | EBCDIC
  | BYTE

<column attributes> ::=
    [<key or not null spec>]
    [<default spec>]
    [<constraint definition>]
    [REFERENCES <referenced table> [(referenced column)]]
    [UNIQUE]

<key or not null spec> ::=
    [PRIMARY] KEY
  | NOT NULL [WITH DEFAULT]

<default spec> ::=
    DEFAULT <default value>


<default value> ::=
    <literal>
  | NULL
  | USER
  | USERGROUP
  | DATE
  | TIME
  | TIMESTAMP
  | STAMP
  | TRUE
  | FALSE

<referenced table> ::=
    <table name>

<referenced column> ::=
<column name>
```

Syntax Rules

1. If [PRIMARY] KEY is specified, the table definition must not contain a <key definition>.

2. The <column attributes> [PRIMARY] KEY and UNIQUE must not be specified together in a <column definition>.

3. For columns of the data type LONG, only NOT NULL may be specified as <column attributes>.

4. Columns of the data type LONG must not occur in temporary tables.

5. If the <create table statement> contains a <query expression>, the <column definition> must only consist of the <column name>.

General Rules

1. The name and data type of each column are defined by <column name> and <data type>. The <column name>s must be unique within a base table.

2. CHAR[ACTER] (n) and VARCHAR (n) define an alphanumeric column with the length attribute n. The length attribute must be greater than 0 and less than or equal to 254. If the length attribute is omitted, n=1 is assumed. According to the code attribute ASCII or EBCDIC, the values of this column are stored in the ISO 8859/1.2 ASCII code or in the EBCDIC code CCSID 500, Codepage 500. In the case of the code attribute BYTE, the values in this column are treated as code-independent. If no code attribute is specified, the code attribute defined during the installation of the ADABAS system is used.

3. If CHAR[ACTER] (n) is specified, the value n determines whether ADABAS stores the values of this column in fixed length or in variable length. If the values are to be stored in variable length regardless of n, VARCHAR must be specified. Otherwise, specifying VARCHAR has the same effect as CHAR.

4. LONG defines an alphanumeric column of any length which can be used in the <insert statement>, in the <update columns and values> of the <update statement>, as <select column>, and in the <null predicate>. If no <code spec> is specified for the LONG column, the code attribute defined during the installation is assumed.

5. BOOLEAN defines a column which can only receive the NULL value or the value TRUE or FALSE.

6. FIXED(p,s) defines a fixed point column with the precision p and the scale s. The precision must be greater than 0 and less than or equal to 18. The scale must not be greater than the precision. If s is omitted, the scale is equal to 0.

7. FLOAT(p) defines a floating point column with the precision p. The precision must be greater than 0 and less than or equal to 18.

8. DATE defines an alphanumeric column where date values are stored. The function

DATE can be used to retrieve the current date.

9. TIME defines an alphanumeric column where time values are stored. The function TIME can be used to retrieve the current time.

10. TIMESTAMP defines an alphanumeric column where timestamp values are stored. The function TIMESTAMP can be used to retrieve the current timestamp value.

11. If a <domain name> is specified, it must identify an existing range of values. The data type and the length of the domain is assigned to the column <column name>. If the domain has a <constraint definition>, this has the same effect as specifying the corresponding <constraint definition> in the <column definition>.

12. Columns, which are part of the key, or for which NOT NULL or a <default spec> was defined, are called NOT NULL columns. The NULL value cannot be inserted into these columns.

13. NOT NULL columns without <default spec>s are called mandatory columns. Whenever rows are inserted, values must be specified for these columns.

14. Columns which are not mandatory are called optional columns. The insertion of a row does not require a value specification for these columns. If a <default spec> exists for the column, the <default value> is stored in the column. If there is no <default spec>, the NULL value is stored in the column.

15. If an index is created for a single optional column, this index contains no rows that have the NULL value in this column. Consequently, for certain requests, the search strategy that would be the best for performance cannot be applied when this index is used. NOT NULL should therefore be specified for all columns where the NULL value will not occur. For columns where the NULL value could occur, the definition of a <default spec> should be considered, because its value is used instead of the NULL value. Rows having the default value are contained in an index.

16. If KEY is specified, this column is part of the key of a table. This column is called key column. All key columns must be the first columns specified for a table. The order of the key columns affects the <select ordered statement>. ADABAS ensures that the key values of a table are unique. The sum of the internal lengths of the key columns must not exceed 255 characters. The number of key columns in a table must be less than 128. To improve performance, the key should start with key columns which can assume a great number of different values and which are to be used frequently in conditions with the operator '='.

17. If a table is defined without a key column, ADABAS implicitly generates the key column SYSKEY CHAR(8) BYTE. This column is not visible when SELECT * is performed; but it can be stated explicitly and has the same meaning as a key column. The SYSKEY column can be used to obtain unique keys generated by ADABAS. The keys are in ascending order, thus reflecting the order of insertion into the table. The key values in the column SYSKEY are only unique within a table; i.e., the SYSKEY column in two tables that are different from each other may contain the same values.

18. If a <default spec> has been made for a column, the <default value> must be a value which can be inserted into the column. If DEFAULT <literal> is specified, the <literal> must be comparable with the data type of the column. The maximum length of a <default value> is 254 characters. DEFAULT USER or DEFAULT USERGROUP can only be specified for columns of the data type [VAR]CHAR(n) where n >= 18. DEFAULT DATE can only be specified for columns of the data type DATE. DEFAULT

TIME can only be specified for columms of the data type TIME. DEFAULT TIMESTAMP can only be specified for columns of the data type TIMESTAMP. DEFAULT STAMP can only be specified for columns of the data type CHAR(n) BYTE where n>=8. DEFAULT TRUE or DEFAULT FALSE can only be specified for columns of the data type BOOLEAN.

19. NOT NULL WITH DEFAULT defines a <default value> which depends on the data type of the column:

```
[VAR]CHAR(n)        ==> <default value> = ' '
[VAR]CHAR(n) BYTE   ==> <default value> = x'00'
FIXED(p,s)          ==> <default value> = 0
FLOAT(p)            ==> <default value> = 0
DATE                ==> <default value> = DATE
TIME                ==> <default value> = TIME
TIMESTAMP           ==> <default value> = TIMESTAMP
BOOLEAN             ==> <default value> = FALSE
```

20. The specification of REFERENCES <referenced table> [(<referenced column>)] has the same effect as the specification of the <referential constraint definition> FOREIGN KEY (<column name>) REFERENCES <referenced table> [<referenced column>)].

21. A <constraint definition> defines a condition which must be satisfied by all values of the column defined in the <column definition>.

22. In addition to the data types listed above, the following data types are permitted in <column definition>s and are mapped to the above-mentioned types:

```
INT[TEGER]           is mapped to  FIXED(10)
SMALLINT             is mapped to  FIXED(5)
DEC[IMAL](p,s)       is mapped to  FIXED(p,s)
DEC[IMAL](p)         is mapped to  FIXED(p)
DEC[IMAL]            is mapped to  FIXED(5)
FLOAT                is mapped to  FLOAT(15)
FLOAT(19..64)        is mapped to  FLOAT(18)
DOUBLE PRECISION     is mapped to  FLOAT(18)
REAL(p)              is mapped to  FLOAT(p)
REAL                 is mapped to  FLOAT(15)
CHAR[ACTER]          is mapped to  CHAR(1)
LONG VARCHAR         is mapped to  LONG
```

23. The following table shows the memory requirements of a column value, in bytes, depending on the various data types:

```
CHAR(n)
          n <=  30    : n + 1
     30 < n <= 254    : n + 1  for key columns,
                        n + 2  otherwise
    254 < n           : n + 3
VARCHAR(n)
     30 < n <= 254    : n + 1  for key columns,
                        n + 2  otherwise
    254 < n           : n + 3
LONG                  :  9
FIXED (p,s)           : (p+1) DIV 2 + 2
FLOAT (p)             : (p+1) DIV 2 + 2
BOOLEAN               :  2
```

```
DATE                    :  9
TIME                    :  9
TIMESTAMP               : 21
```

The memory requirements of all columns in a table must not exceed 4047 bytes.

<constraint definition>

Function

defines a condition which must be satisfied by the rows of a table.

Format

```
 <constraint definition> ::=
     CHECK <search condition>
   | CONSTRAINT <search condition>
   | CONSTRAINT <constraint name> CHECK <search condition>
```

Syntax Rules

1. The <search condition> of the <constraint definition> must not contain a <subquery>.

2. Column names in the <search condition> of the <constraint definition> must only be in the form of <column name>.

General Rules

1. A <constraint definition> defines a condition which must be satisfied by all rows of the table.

2. If there is no <constraint name> specification, ADABAS assigns a name that is unique within the table.

3. If a <constraint name> is specified, then it must differ from all the other <constraint name>s of the table.

4. If the <search condition> contains only a single column name of the table, then it is possible at the time of table generation to check whether the <search condition> is true for an additionally specified <default value> of this column. If it is not true, the <create table statement> fails.

5. If the <search condition> contains more than one column name for the table, it is not possible to determine at the time of table generation whether the <search condition> is true for default values of the table. In this case, any attempt to insert default values into the table in the process of executing the <insert statement> or the <update statement> may fail.

6. Before inserting a row or updating a column occurring in the <constraint definition>, ADABAS checks the <constraint definition> of the column. If the <constraint definition> is violated, the <insert statement> or <update statement> fails.

<referential constraint definition>

Function

defines existence conditions between the rows of two tables.

Format

```
<referential constraint definition> ::=
    FOREIGN KEY  [<referential constraint name>]
    (<referencing column>,...)
    REFERENCES <referenced table> [(<referenced column>,...)]
    [<delete rule>]

<referencing column> ::=
    <column name>

<delete rule> ::=
    ON DELETE CASCADE
  | ON DELETE RESTRICT
  | ON DELETE SET DEFAULT
  | ON DELETE SET NULL
```

Syntax Rules

none

General Rules

1. The <referential constraint definition> is part of a <create table statement> or an <alter table statement>. In the following rules, the table defined by the <create table statement> or specified in the <alter table statement> is referred to as the referencing table.

2. The referencing table and the <referenced table> must not be temporary base tables.

3. The current user must have the ALTER privilege for the referencing table and the REFERENCES privilege for the <referenced table>.

4. If a <referential constraint name> is specified, it must differ from all existing <referential constraint name>s of the referencing table.

5. If no <referential constraint name> is specified, ADABAS assigns a <referential constraint name> which is unique with respect to the referencing table.

6. The <referencing column>s must denote columns of the referencing table and must be different from each other. They are called foreign key columns.

7. Omitting the <referenced column>s has the same effect as specifying the key columns of the <referenced table> in the defined order.

8. If the <referenced column>s do not identify the key of the <referenced table>, then the

<referenced table> must have a <unique definition> whose <column name>s match the <referenced column>s.

9. The number of columns of the <referencing column>s must correspond to the number of <referenced column>s. The nth <referencing column> corresponds to the nth <referenced column>. The data type and the length of each <referencing column> must match the data type and length of the corresponding <referenced column>.

10. If SET NULL is defined as the <delete rule>, then none of the <referencing column>s can be a NOT NULL column.

11. If SET DEFAULT is defined as the <delete rule>, then a <default spec> must have been defined for each <referencing column>.

12. A table T' is called CASCADE dependent on a table T, if there is a sequence of <referential constraint>s R1,R2,...,Rn with n>=1, so that

    a) T' is the referencing table of R1 and

    b) T is the <referenced table> of Rn and

    c) all <referential constraint definition>s specify CASCADE and

    d) for i=1,...,n-1 and n>1, the <referenced table> of Ri is equal to the referencing table of Ri+1.

    The following graph illustrates an example where n=3:

```
         R1              R2              R3
T' <__________ T1 <__________ T2 <__________ T
        CASCADE         CASCADE         CASCADE
```

13. Let R1 and R2 be two different <referential constraint definition>s with the same referencing table S. T1 denotes the <referenced table> of R1, T2 denotes the <referenced table> of R2.
    If T1 equals T2, or if there is a table T, so that T1 and T2 are CASCADE dependent on T, then R1 and R2 must both specify either CASCADE or RESTRICT.

    Graphic illustration:

```
         R1             CASCADE-dependent
  <__________ T1 <_______________________
S                                          T
  <__________ T2 <_______________________
         R2             CASCADE-dependent
```

    Remark: There are two different sequences of <referential constraint definition>s associating S with T. A <delete statement> on T is followed by an action in S. The above-mentioned restriction for R1 and R2 was chosen so that the result of the <delete statement> is not dependent on which of the two different sequences of <referential constraint definition>s has been processed first.

14. A reference cycle is a sequence of <referential constraint definition>s R1,R2,...,Rn with n>1, so that

    a) for i=1,...,n-1 the <referenced table> of Ri is equal to the referencing table of

R_{i+1}, and

b) the <referenced table> of R_n is equal to the referencing table of R_1.

15. Reference cycles where all <referential constraint definition>s specify CASCADE are not allowed.
Reference cycles where one <referential constraint definition> does not specify CASCADE and all the other <referential constraint definition>s specify CASCADE are not allowed.

16. A row of the referencing table is called the matching row of a <referenced table> row when the values of the corresponding <referencing column>s and of the <referenced column>s are the same.

17. A <referential constraint definition> defines a 1:n relationship between two tables. This means that more than one matching row can exist for each row of the <referenced table>.

18. Any attempt to update a row of the <referenced table> in a <referenced column> fails whenever at least one matching row exists.

19. The <delete rule> defines the effects of the deletion of a row from the <referenced table> on the referencing table.

    Whenever RESTRICT was specified or the <delete rule> was omitted, then the deletion of a row from the <referenced table> fails whenever there are matching rows.

    Whenever CASCADE was specified and a row is deleted from the <referenced table>, all matching rows are deleted.

    Whenever SET NULL was specified and a row is deleted from the <referenced table>, all columns in the <referencing column> are assigned the NULL value for each matching row.

    Whenever SET DEFAULT was specified and a row is deleted from the <referenced table>, each <referencing column> is assigned the DEFAULT value for each matching row.

20. The following restrictions apply for the insertion or update of rows in the referencing table:

    Let R be a row to be inserted or updated. Insertion and update are only possible if one of the following conditions is true for each pertinent <referenced table>:

    a) R is a matching row.

    b) R contains a NULL value in one of the <referencing column>s.

    c) The <referential constraint definition> defines SET DEFAULT, and R contains the DEFAULT value in all <referencing column>s.

21. A <referential constraint definition> is termed self-referencing if the <referenced table> matches the referencing table.

22. In self-referencing <referential constraint definition>s, the processing sequence of a <delete statement> can be significant. This case is illustrated in the description below.

The following is a basic description and, therefore, may deviate from the actual implementation.

If CASCADE was specified, all rows affected by the <delete statement> are deleted first, while the <referential constraint definition> is ignored. Then ADABAS deletes all matching rows of the rows just deleted. This is followed by the deletion of all matching rows related to the immediately preceding deletion procedure, etc.

If SET NULL or SET DEFAULT is specified, all rows affected by the <delete statement> are deleted first, while the <referential constraint definition> is ignored. Then SET NULL or SET DEFAULT is applied to all matching rows.

23. When rows are deleted from a <referenced table>, the third entry of SQLERRD in the SQLCA (for further details, see the online help on the precompilers) is set to the number of rows deleted from the <referenced table>.

24. In the case of <insert statement>s and <update statement>s issued on referencing tables, the ADABAS lock behavior on the <referenced table> is equivalent to ISOLATION LEVEL 1, independent of the ISOLATION LEVEL selected for the current session.
    In the case of <delete statement>s issued on <referenced table>s, the ADABAS lock behavior is equivalent to ISOLATION LEVEL 3.

<key definition>

Function

defines the key of a table.

Format

```
<key definition> ::=
    PRIMARY KEY (<column name>,...)
```

Syntax Rules

none

General Rules

1. The <key definition> is part of a <create table statement> or <alter table statement>; i.e., it refers to a base table. <column name> must always identify a column of this table.

2. The <key definition> defines the key of a table. The <column name>s of the <key definition> are the key columns of the table.

3. <column name> must not identify any column of the data type LONG.

4. The sum of the internal lengths of the key columns must not exceed 255 characters.

5. Key columns are NOT NULL columns.

6. ADABAS ensures that no key column has the NULL value and that no two rows of the table have the same values in all key columns.

<unique definition>

Function

defines the uniqueness of column value combinations.

Format

```
 <unique definition> ::=
     UNIQUE (<column name>,...)
```

Syntax Rules

none

General Rules

1. Including a <unique definition> in the <create table statement> has the same effect as the corresponding <create table statement> without the <unique definition> followed by a <create index statement> with UNIQUE specification. The same rules apply as are described under <create index statement>.

2. If more than one <column name> is specified, ADABAS assigns the index a unique <index name>.

3. ADABAS ensures that no two rows of the table have the same values in the indexed columns.

<drop table statement>

Function

drops a base table.

Format

```
 <drop table statement> ::=
     DROP TABLE <table name> [<cascade option>]

 <cascade option> ::=
      CASCADE
    | RESTRICT
```

Syntax Rules

none

General Rules

1. The <table name> must be the name of an existing base table.

2. The current user must be the owner of the base table.

3. All metadata and rows of the base table are dropped. All view definitions, indexes, privileges, synonyms, triggers, and <referential constraint definition>s derived from this base table are dropped. All snapshot tables derived from the base table to be dropped remain unaffected. ADABAS marks them in such a way that the <query expression> defining the snapshot tables must be performed again when the <refresh statement> is executed the next time. This means that the <refresh statement> fails if the dropped table has not been recreated in the meantime.

4. If the <cascade option> RESTRICT is specified and view tables or synonyms are based on the table identified by <table name>, then the <drop table statement> fails. If no <cascade option> is specified, CASCADE is assumed.

5. If a table dropped in the course of a <drop table statement> is addressed in a DB procedure, this procedure is marked as not executable.

6. To apply the specified <delete rule> to all data linked to the base table by a <referential constraint definition> with corresponding <delete rule>, first a <delete statement> and then the <drop table statement> must be performed for the base table.

<alter table statement>

Function

alters properties of a table.

Format

```
<alter table statement> ::=
    ALTER TABLE <table name> <add definition>
  | ALTER TABLE <table name> <drop definition>
  | ALTER TABLE <table name> <alter definition>
  | ALTER TABLE <table name> <referential constraint definition>
  | ALTER TABLE <table name> DROP FOREIGN KEY
    <referential constraint name>
```

Syntax Rules

none

General Rules

1. The <table name> must be the name of an existing base table.

2. The table must not be a temporary base table.

3. The current user must have the ALTER privilege for the table identified by <table name>.

4. If a <referential constraint definition> was specified, a new <referential constraint> is defined for the base table.

5. If DROP FOREIGN KEY was specified, the <referential constraint name> identified by the <referential constraint definition> is dropped.

See also

<add definition>

<drop definition>

<alter definition>

<add definition>

Function

defines additional properties for a table.

Format

```
<add definition> ::=
    ADD  <column definition>,...
  | ADD (<column definition>,...)
  | ADD <constraint definition>
  | ADD <key definition>
  | ADD REPLICATION
```

Syntax Rules

1. The specification of a <domain name> in a <column definition> is only allowed if the domain was defined without a <default spec>.

General Rules

1. The table specified in the <alter table statement> is extended by the columns specified in <column definition>s.
   These specifications must not exceed the maximum number of columns allowed and the maximum length of a row. For the computation of the row length, it must be taken into account that, deviating from the description in the section <column definition>, the space requirement of each column with a length less than 31 characters and of a data type other than VARCHAR is increased by 1 character.

2. The <column name>s specified in the <column definition>s must differ from each other and must not be identical to any names of columns existing in the table.

3. The columns contain the NULL value in all rows. If the NULL value violates a <constraint definition> of the table, the <alter table statement> fails.

4. In every other respect, specifying a <column definition> in an <alter table statement> has the same effect as including the <column definition> in the <create table statement>.

5. If view tables are defined on the specified table, and these view tables use '*' to make reference to the columns of the table, the <alter table statement> fails if <alias name>s are defined for any one of these view tables. The reason is that the number of view table columns defined by the <alias name>s does not match the number of columns fetched by '*' after performing the <add definition>.
   If '*' but no <alias name> was specified when defining a view table, then this view table contains the columns which were added to the base table with the <add definition>.

6. If a <constraint definition> is specified, the condition defined by the <search condition> of the <constraint definition> must be true for all rows of the table.

7. If ADD PRIMARY KEY is specified, a key is defined for the table identified in the <alter table statement>. At execution time, the table must only contain the key column SYSKEY generated by ADABAS. The columns specified in the <key definition> must identify columns of the table and meet the properties of the key; i.e., none of the columns may contain the NULL value and no two rows in the table may have the same values in all columns of the <key definition>. The new key is stored in the metadata of the table. The key column SYSKEY is omitted.

8. The specification of ADD REPLICATION has the effect that the metadata, rows and indexes of the table identified in the <alter table statement> are provided on every SERVERDB of the distributed database, so that read access to this data is always possible without network communication.
   The current user as well as the owner of the table must have DBA status.
   The SERVERDB where the <alter table statement> is to be executed must belong to the majority at execution time. After executing the <alter table statement>, the metadata, rows and indexes of the table are available on all SERVERDBs belonging to the majority. SERVERDBs which could not be accessed during the execution of the <alter table statement> copy the data as soon as they belong to the majority again.

9. ADD PRIMARY KEY and ADD REPLICATION require extensive copy operations which may take a long time especially for tables with many rows.

<drop definition>

Function

removes properties of a table.

Format

```
 <drop definition> ::=
     DROP  <column name>,...  [<cascade option>]
   | DROP (<column name>,...) [<cascade option>]
   | DROP CONSTRAINT <constraint name>
   | DROP PRIMARY KEY
   | DROP REPLICATION
```

Syntax Rules

none

General Rules

1. Each <column name> must be a column of the table identified by the <alter table statement>. The column must be neither a key column nor a foreign key column of a <referential constraint definition> of the table.

2. In the metadata of the table, the columns are marked as dropped. A <drop definition> does not reduce the memory requirements of the underlying table.

3. Any privileges existing for these columns are dropped as well.

4. If one of the columns to be dropped occurs in a <select column> of a view definition, then the column of the view table defined by the <select column> is dropped.
   If this view table is used in the <from clause> of another view table, the procedure described is applied recursively to this view table.

5. If one of the columns to be dropped occurs in the <table expression> of a view definition, then the view definition and all related view tables, privileges and synonyms are dropped if none of the <cascade option>s or the <cascade option> CASCADE is specified.
   If RESTRICT is specified, the <alter table statement> fails.

6. Existing indexes referring to columns to be dropped are also dropped. The storage locations for the dropped indexes are released.

7. All <constraint definition>s containing one of the dropped columns are dropped.

8. If DROP CONSTRAINT is specified, the <constraint name> must identify a <constraint definition> of the table. The latter is then removed from the metadata of the table.

9. If DROP PRIMARY KEY is specified, the table identified by the <alter table statement> must contain a key. The key is replaced by the key column SYSKEY generated by

ADABAS. A prerequisite is that the table has no more than 254 columns, the maximum row length of 4047 bytes is not exceeded, and no key column is a <referenced column> of a <referential constraint definition>.

10. The specification of DROP REPLICATION has the effect that the metadata, rows and indexes of the table identified by the <alter table statement> are no longer available on each SERVERDB of the distributed database, but are only provided on the HOME SERVERDB of the table owner.
    Both the SERVERDB where the <alter table statement> is to be executed and the HOME SERVERDB of the owner must belong to the majority at execution time.

11. DROP PRIMARY KEY and DROP REPLICATION require extensive copy operations which may take a long time especially for tables with many rows.

<alter definition>

Function

alters the properties of a column or of a <constraint definition>.

Format

```
<alter definition> ::=
   COLUMN <column name> <alter data type>
 | COLUMN <column name> NOT     NULL
 | COLUMN <column name> DEFAULT NULL
 | COLUMN <column name> ADD   <default spec>
 | COLUMN <column name> ALTER <default spec>
 | COLUMN <column name> DROP DEFAULT
 | ALTER CONSTRAINT <constraint name> CHECK <search condition>
 | ALTER <key definition>

<alter data type> ::=
    <data type>
  | <domain name>
```

Syntax Rules

none

General Rules

1. The data type of a key column or foreign key column cannot be altered.

2. A specified <alter data type> replaces the existing <data type>. The new data type must be compatible with the former data type, or, more precisely:

   a) [VAR]CHAR(n) can be changed to [VAR]CHAR(m) with m>=n.

   b) The code attribute ASCII can be changed to EBCDIC and vice versa.

   c) FIXED(p,s) can be changed to FIXED(m,n) with m>=p and n>=s and m-n>=p-s.

   d) FIXED(p,s) can be changed to FLOAT(m) with m>=p.

   e) FLOAT(p) can be changed to FLOAT(m) with m>=p.

3. If the <domain name> identifies a domain that has a <constraint definition>, then this <constraint definition> is assigned to the identified table. ADABAS attempts to assign the <domain name> as the <constraint name>. If this fails because there is a <constraint name> with this name, then a unique name is created.

4. If the <domain name> identifies a domain that has a <default spec>, then this <default spec> is assigned to the column identified by the <column name>.

5. In some cases, the specification of an <alter data type> has the effect that a new table column is defined implicitly. This column is not visible to the user. If the addition of the new column could have the effect that the maximum number of columns would be exceeded, the <alter table statement> fails.

6. The expansion of a column of the base table can have the effect that the maximum length of a row is exceeded. In this case, the <alter table statement> fails.

7. The expansion of a column of the base table can have the effect that the column of a view table defined on this base table becomes too long. In this case, the <alter table statement> fails.

8. Changing the data type of a column can have the effect that indexes defined across the column are implicity recreated. Expanding a column can have the effect that an index consisting of several columns becomes too wide. In this case, the <alter table statement> fails.

9. NOT NULL can only be specified if the column contains no NULL values.

10. DEFAULT NULL allows the NULL value for the column. If the column has a <default spec>, the <alter table statement> fails. ADABAS does not check whether the NULL value violates existing <constraint definition>s of the table; i.e., the insertion of the NULL value can fail while executing the <insert statement> or <update statement>.

11. ADD <default spec> assigns a default value to the column. In any rows having the NULL value in the column, the NULL value is replaced by the default value.

12. ALTER <default spec> assigns a new default value to the column. All rows having the old default value in the column remain unaltered.

13. DROP DEFAULT drops the <default spec> of the column. If the column is the foreign key column of a <referential constraint> with the <delete rule> ON DELETE SET DEFAULT, the <alter table statement> fails.

14. If CONSTRAINT is specified, the <constraint name> must identify a <constraint definition> of the table. If the specified <search condition> is not violated by any row of the table, then this <search condition> replaces the existing <search condition> of the <constraint definition>; otherwise, the <alter table statement> fails.

15. If PRIMARY KEY is specified, the key defined by the <key definition> replaces the key of the table identified by the <alter table statement>. The columns specified in the <key definition> must identify columns of the table and meet the properties of the key; i.e., none of the columns may contain the NULL value and no two rows in the table may have the same values in all columns of the <key definition>.
If a column of the key to be replaced is a <referenced column> of a <referential constraint>, the <alter table statement> fails.
The alteration of the key of a table requires extensive copy operations which may take a long time especially for tables with many rows.

<rename table statement>

Function

changes the name of a base table.

Format

```
<rename table statement> ::=
    RENAME TABLE <old table name> TO <new table name>

<old table name> ::=
    <table name>

<new table name> ::=
    <identifier>
```

Syntax Rules

none

General Rules

1. The table identified by <old table name> must be a base table.

2. The table identified by <old table name> must not be a temporary table.

3. The table may only be renamed by its owner.

4. The name <new table name> must not yet be used for a base table, view table, snapshot table or synonym of the current user.

5. The table identified by <old table name> is given the <new table name>. All its various properties, e.g., privileges and indexes, remain unchanged. The definitions of snapshot tables and view tables based on the <old table name> are adapted to the new name. For snapshot tables, these adaptations are only visible after executing a <refresh statement>.

<rename column statement>

Function

changes the name of a table column.

Format

```
 <rename column statement> ::=
     RENAME COLUMN <table name>.<column name> TO <column name>
```

Syntax Rules

none

General Rules

1. The specified table must be a base table, a view table or a snapshot table.

2. The column may be only renamed by the owner of the table.

3. The specified table column is given a new name.
   If the column name of a view table or snapshot table defined on this table was derived from the column name of the base table, the old column name in the view table is replaced by the new name. If the new column name is identical to an existing column name of the view table, the <rename column statement> fails. For snapshot tables, the renaming is only visible after reexecuting the <refresh statement>.

<exists table statement>

Function

indicates the existence or non-existence of a table.

Format

```
<exists table statement> ::=
    EXISTS TABLE <table name>
```

Syntax Rules

none

General Rules

1. The specified table must be a base table, a view table, a snapshot table or a synonym.
2. The existence or non-existence of the specified table is indicated by the return code 0 or by the error message -4004 UNKNOWN TABLE NAME.
3. A table only exists for a user if the user has a privilege on this table.

<create domain statement>

Function

defines a domain.

Format

```
<create domain statement> ::=
    CREATE DOMAIN <domain name> <data type>
    [<default spec>] [<constraint definition>]
```

Syntax Rules

1. The <constraint definition> must not contain a <constraint name>.

General Rules

1. The <create domain statement> can be issued by all users with DBA status.
2. A domain is defined, which can be used by any user in the <create table statement> and in the <alter table statement> to define a column.
3. If <domain name> has no <owner>, then the current user is assumed as <owner>. Otherwise, <owner> must be identical to the name of the current user. The current user becomes the owner of the domain.
4. The name of the domain must differ from any existing domain names of the current user.
5. If a domain is created with a <constraint definition>, then the <domain name> in the <search condition> functions as the column name.

<drop domain statement>

Function

drops the definition of a domain.

Format

```
<drop domain statement> ::=
    DROP DOMAIN <domain name>
```

Syntax Rules

none

General Rules

1. The metadata of the domain is dropped from the catalog.
2. <domain name> must identify an existing domain.
3. The current user must be owner of the domain.
4. Dropping a domain has no effect on tables in which this domain was used to define columns.

<create synonym statement>

Function

defines a synonym for a table name.

Format

```
<create synonym statement> ::=
    CREATE SYNONYM <synonym name> FOR <table name>
```

Syntax Rules

none

General Rules

1. The <table name> must not denote a temporary base table.
2. The user must have a privilege on the specified table <table name>.
3. The <synonym name> must not be identical to the name of an existing base table, or the name of a synonym of the current user.
4. The synonym definition expands the set of table synonyms available to this user.
5. The synonym name can be specified anywhere instead of the table name. This has the same effect as specifying the table name for which the synonym was defined.

<drop synonym statement>

Function

drops a synonym for a table name.

Format

```
<drop synonym statement> ::=
    DROP SYNONYM <synonym name>
```

Syntax Rules

none

General Rules

1. The specified <synonym name> must identify an existing synonym.
2. The synonym definition is removed from the set of table name synonyms available to the user.

<rename synonym statement>

Function

changes the name of a synonym.

Format

```
<rename synonym statement> ::=
    RENAME SYNONYM <old synonym name> TO <new synonym name>

<old synonym name> ::=
    <synonym name>

<new synonym name> ::=
    <synonym name>
```

Syntax Rules

none

General Rules

1. The synonym identified by <old synonym name> must have been created by the current user.
2. There must not be a table with the <new synonym name> available to the current user.
3. The specified synonym is given a new name.

<create snapshot statement>

Function

creates a snapshot table.

Format

```
 <create snapshot statement> ::=
     CREATE SNAPSHOT <table name> [(<alias name>,...)]
     AS <query expression>
```

Syntax Rules

1. The <query expression> must not contain a parameter specification.

General Rules

1. A table generated by the <create snapshot table> is called a snapshot table. Structure and contents of the snapshot table are equivalent to the result table defined by the <query expression>. In contrast to a corresponding view table, the data of the snapshot table is physically stored on the medium and the contents of the snapshot table are not always identical to the result of the <query expression>.

2. The metadata and the contents of the snapshot table are stored on the SERVERDB where the current user has opened his session.

3. The rows of a snapshot table cannot be changed by the <insert statement>, <update statement> or <delete statement>.

4. The current user must have the privilege to execute the <query expression>.

5. The <query expression> must not make reference to a snapshot table, temporary table or <result table name>.

6. The <table name> must not be identical to the name of an existing table of the current user.

7. The <alias name>s define the column names of the snapshot table. They must differ from each other, and their number must be identical to the number of the result table defined by the <query expression>.
   If no <alias name>s are specified, then the column names of the result table defined by the <query expression> are applied.

8. The current user is the owner of the snapshot table. The current user must have the SELECT privilege for all columns of the snapshot table which are derived from columns for which he has the right to grant the SELECT privilege. Furthermore, he can only grant the INDEX privilege.

9. ADABAS distinguishes between simple and complex snapshot tables. Simple

snapshot tables have the following properties:

a) The <query expression> contains up to one <from clause> which contains up to one <table name>; i.e., the <query expression> contains no <subquery> and no join.

b) The <query expression> contains no DISTINCT, UNION, EXCEPT, INTERSECT, or GROUP BY.

c) The <query expression> contains no <set function spec>.

d) The snapshot table is not based on a replicated base table.

e) The snapshot table is not based on a view table for which one of the conditions a) to d) is not valid.

Each snapshot table which does not satisfy one of these rules is a complex snapshot table.

10. To tally the contents of the snapshot table with the contents of the result table defined by the <query expression>, the <refresh statement> can be used in SQLMODE ADABAS. ADABAS distinguishes between two methods of executing the <refresh statement>:

    a) If the snapshot table is a simple snapshot table and the base table on which the snapshot table is based has a snapshot log, then this snapshot log can be used to determine the differences between the contents of the snapshot table and the result table of the <query expression>. Only these differences are transferred to update the snapshot table. In many cases, this is more convenient than to transfer the complete result table into the snapshot table.

    b) All rows of the snapshot table are deleted. Then all rows of the result table defined by the <query expression> are inserted.

<drop snapshot statement>

Function

drops a snapshot table.

Format

```
 <drop snapshot statement> ::=
     DROP SNAPSHOT <table name>
```

Syntax Rules

none

General Rules

1. <table name> must identify a snapshot table.
2. The current user must be the owner of the snapshot table.
3. The metadata and all rows of the snapshot table are dropped.
4. All indexes, synonyms and view tables defined on the snapshot table are dropped.
5. If <table name> identifies a simple snapshot table and the underlying base table has a snapshot log, then any information of the snapshot log is dropped that is only relevant for refresh operations on the snapshot table to be dropped. If the snapshot table to be dropped is the only simple snapshot table based on the base table, then the corresponding snapshot log is not written until the next simple snapshot table is created on this base table.

<create snapshot log statement>

Function

creates a snapshot log.

Format

```
 <create snapshot log statement> ::=
     CREATE SNAPSHOT LOG ON <table name>
```

Syntax Rules

none

General Rules

1. <table name> must identify a non-temporary base table.

2. <table name> must not identify a non-replicated base table.

3. The current user must be the owner of the base table.

4. The <create snapshot log statement> creates a snapshot log for the base table identified by <table name>. In a snapshot log, ADABAS stores information about the modified rows of the table. This information can be used later with a <refresh statement> to update a snapshot table without having to execute the complete <query expression>, because only the modifications made since the last execution of the <refresh statement> are performed. In many cases, this is convenient because the data transfer between the SERVERDBs is reduced considerably.

5. ADABAS only writes the snapshot log if there is at least one simple snapshot table based on the table <table name>. Otherwise, the snapshot log is created but not filled when rows of the table are modified.

<drop snapshot log statement>

Function

drops a snapshot log.

Format

```
<drop snapshot log statement> ::=
    DROP SNAPSHOT LOG ON <table name>
```

Syntax Rules

none

General Rules

1. The base table identified by <table name> must have a snapshot log.

2. The current user must be the owner of the base table.

3. The snapshot log and the information contained in it are dropped. If rows of the base table are modified, these modifications are no longer recorded in the snapshot log.

4. After dropping the snapshot log, the <query expression> must be executed completely to update snapshot tables that are based on the base table <table name>.

<create view statement>

Function

creates a view table.

Format

```
 <create view statement> ::=
     CREATE [OR REPLACE] VIEW <table name> [(<alias name>,...)]
     AS <query expression>
     [WITH CHECK OPTION]
```

Syntax Rules

1. The <query expression> must not contain a parameter specification.

2. The <query expression> must not refer to a temporary table or a <result table name>.

3. The number of <alias name>s must be equal to the number of columns in the result table generated by the <query expression>.

4. If a <select column> of the <query expression> identifies a column of the data type LONG, then the <from clause> must contain just one table identifier with just one underlying base table.

General Rules

1. A table generated by the <create view statement> is called a view table. The execution of the <create view statement> has the effect that metadata describing the view table is stored in the catalog. The metadata is stored on the HOME SERVERDB of the user. If one of the base tables underlying the view table is defined with the <table option> WITH REPLICATION, the metadata of the view table is stored on all SERVERDBs of the distributed database.
   A view table never exists physically but is formed from the rows of the underlying base table(s) when this view table is specified in an <sql statement>.

2. If the specification of REPLACE is omitted, the <table name> must not be identical to the name of an existing table.

3. If REPLACE is specified, then <table name> may be identical to the name of an existing view table. In this case, the definition of the existing view table is replaced by the new definition. ADABAS then attempts to adapt privileges granted for the existing view table to the new view definition; usually, the privileges for the view table are kept in this way. Privileges are only removed implicitly if conflicts occur that cannot be resolved by ADABAS. Should there be large differences between the two view definitions, then the <create view statement> can fail in the following cases:

   a) The <create view statement> of a view table based on the existing view table cannot be executed free of errors on the new view definition.

b) The old view table is replicated and the new view table is not replicated, or vice versa.

4. The user must have the SELECT privilege for all columns which occur in the view definition. The user is the owner of the view table and has at least the SELECT privilege for it. The user may grant the SELECT privilege for any columns in the view table derived from columns for which the user is authorized to grant the SELECT privilege to others. The user has the INSERT, UPDATE, or DELETE privilege when he has the corresponding privileges for the tables on which the view table is based, and when the view table is updatable. The user may grant any of these privileges to other users when he is authorized to grant the corresponding privilege for all tables on which the view table is based.

5. The <alias name>s define the column names of the view table. If no <alias name>s are specified, then the column names of the result table generated by the <query expression> are applied to the view table. The column names of the view table must be unique. Otherwise, <alias name>s must be specified for the result table generated by the <query expression>. The column descriptions for the view table are taken from the corresponding columns in the <query expression>. The <from clause> of the <query expression> may contain one or more tables.

6. The view table is always identical to the table that would be obtained as the result of the <query expression>.

7. A view table is a complex view table if one of the following conditions is satisfied:

   a) The definition of the view table contains DISTINCT or GROUP BY or HAVING.

   b) The <create view statement> contains EXCEPT, INTERSECT, or UNION.

   c) The <search condition> of the <query expression> in the <create view statement> contains a <subquery>.

   d) The <create view statement> contains an outer join, that is, an <outer join indicator> in a <join predicate> of the <search condition>.

8. A view table is called updatable if it is not a complex view table, and if it is not based on a complex view table.

   For join view tables; i.e., view tables whose <from clause> contains more than one table or join view table, the following additional conditions must be satisfied:

   a) Each base table on which the view table is based has a key defined by the user.

   b) <referential constraint definition>s must exist between the base tables on which the view table is based.

   c) There is just one base table on which the view table is based. The base table is not the <referenced table> of a <referential constraint definition> for another base table underlying the view table. This table is the key table of the view table.

   d) For each base table on which the view table is based, there is a sequence of <referential constraint definition>s so that the respective base table can be accessed from the key table.

e) The <referential constraint definition>s must be specified in the form of <join predicate>s in the <search condition> of the <create view statement>; i.e., the condition 'key column = foreign key column' must be specified for each column of each <referential constraint definition>.

f) The <create view statement> must contain either the primary key column or the foreign key column of each <referential constraint definition> as <select column>. It must not contain both key columns.

g) The view table must be defined WITH CHECK OPTION.

This brief description serves as a concise summary of the conditions for join view tables. For a formal description of these conditions, please refer to the end of this section

9. The owner of the view table has the INSERT privilege; i.e., the user may specify a view table in the <insert statement> as the table into which insertion is to be made if the following conditions are satisfied:

   a) The view table is updatable.

   b) The owner of the view table has the INSERT privilege for all tables in the <from clause> of the <create view statement>.

   c) The <select column>s in the <create view statement> consist of <table columns> or <column name>s, not of <expression>s with more than one <column name>.

   d) The <create view statement> contains all mandatory columns of all tables of the <from clause> as <select column>.

10. The owner of the view table has the UPDATE privilege for a column of the view table; i.e., the user may specify a column in the <update statement> as column to be updated if the following conditions are satisfied:

   a) The view table is updatable.

   b) The owner of the view table has the UPDATE privilege for the <table columns> or the <column name> defining the column.

   c) The column is defined by a specification of <table columns> or by a <column name>, but not by an <expression> with more than one <column name>.

11. The owner of the view table has the DELETE privilege for the view table; i.e., the user may specify a view table in the <delete statement> as the table from which a column or row is to be deleted if the following conditions are satisfied:

   a) The view table is updatable.

   b) The owner of the view table has the DELETE privilege for all tables of the <from clause> of the <create view statement>.

12. If the <create view statement> contains the WITH CHECK OPTION, then the owner of the view table must have the INSERT, UPDATE, or DELETE privilege for the view table.
The specification of WITH CHECK OPTION has the effect that the <insert statement> or <update statement> issued on the view table does not create any rows which

subsequently could not be selected via the view table; i.e., the <search condition> of the view table must be true for any resulting rows.
The CHECK OPTION is inherited; i.e., if a view table V was defined WITH CHECK OPTION and V occurs in the <from clause> of an updatable view table V1, then only those rows can be inserted or altered using V1 which can be selected using V.

13. If DISTINCT is specified, then it is not possible to execute a <select ordered statement: searched> on the defined view table.

14. If a complex view table or a join view table is concerned, then it is not possible to execute a <select direct statement> or <select ordered statement>.

15. The following paragraphs provide a formal description of the conditions which must be satisfied before a join view table can be updated. The basic premise is that the <from clause> in the definition of the join view table V contains the base tables T1 .. Tn (n > 1).

16. Let Ti and Tj be two base tables selected by V. Let Rij be a <referential constraint definition> between Ti and Tj, in which Ti is the referencing table and Tj the <referenced table>. Let PKj1 .. PKjm be the key columns of Tj and FKi1 .. FKim the corresponding foreign key columns of Ti. The <referential constraint definition> is relevant to V if the join predicate (PKj1 = FKi1 AND .. AND PKjm = FKim) is part of the <search condition> of V.

17. Let Ti and Tj be two base tables selected by V and Rij be a <referential constraint definition> between Ti and Tj, which is relevant to V. Ti is the predecessor of Tj (Ti < Tj) if Rij is the only <referential constraint definition> between Ti und Tj, which is relevant to V.

18. Let Rij be a <referential constraint definition> which is relevant to V. Rij defines a 1 : 1 relationship between Ti and Tj if the foreign key columns of Rij make up the key columns of Ti.

19. Let Rij be a <referential constraint definition> which is relevant to V and s a key column of Tj or a foreign key column of this <referential constraint definition> of Ti. The column c can be derived from V if exactly one of the following conditions is satisfied.

    a) c is an element of a <select column> of V.

    b) There is a key column or a foreign key column c' of a <referential constraint definition> relevant to V, which can be derived from V, and the join predicate c = c' is part of the <search condition> of V.

20. A column v of V corresponds to a column c of an underlying base table T if

    a) v is the ith column of V and c is the ith <select column> of V, or

    b) v corresponds to a key column PK in Tj, belonging to a <referential constraint definition> Rij relevant to V, and c is the foreign key column of Ti assigned to PK, or

    c) v corresponds to a foreign key column FK in Ti, belonging to a <referential constraint definition> Rij relevant to V, and c is the key column of Tj assigned to FK.

21. V is updatable if the following conditions are satisfied:

a) Each base table Ti (1 <= i <= n) has a key defined by the user.

b) ADABAS must be able to determine a processing sequence for the underlying base tables; i.e., an order Ti1 .. Tin of the tables T1 .. Tn must exist, such that j < k follows from Tij < Tik. The columns of V from which the key columns of Ti1 can be derived make up the key of V. Ti1 is called the key table of V. The order of the tables need not be unique.

c) Starting with a row in the key table of V, it must be possible to assign each underlying base table exactly one row; i.e., there is a sequence of tables Ti1 .. Tij for each table Tij (1 <= j <= n), such that Ti1 < .. < Tij is true.
This sequence is unique for each base table referred to by V.

d) It must be possible to derive the key columns and foreign key columns of all <referential constraint definition>s relevant to V from the columns of V.

e) The join predicates needed for the recognition of the relevance of a <referential constraint definition> must be specified in parts of the <search condition> defined WITH CHECK OPTION. If the view definition only contains base tables, this means that the view table must be defined WITH CHECK OPTION. If a view table V is derived from a view table V' and if V' was defined WITH CHECK OPTION, then V inherits the CHECK OPTION for the part of the qualification passed on by V'.

<drop view statement>

Function

drops a view table.

Format

```
 <drop view statement> ::=
     DROP VIEW <table name> [<cascade option>]
```

Syntax Rules

none

General Rules

1. The table name must denote an existing view table.

2. The user must be the owner of the specified view table.

3. The metadata of the view table and all dependent synonyms, view tables and privileges are dropped. The tables on which the view table was created remain unaffected. All snapshot tables derived from the view table to be dropped remain unaffected. ADABAS marks them in such a way that the <query expression> defining the snapshot tables must be performed again when the <refresh statement> is executed the next time. This means that the <refresh statement> fails if the dropped table has not been recreated in the meantime.

4. If the <cascade option> RESTRICT is specified and view tables or synonyms exist on the view table, then the <drop view statement> fails.

5. If a view table dropped in the course of the <drop view statement> is addressed in a DB procedure, this procedure is marked as not executable.

<rename view statement>

Function

changes the name of a view table.

Format

```
<rename view statement> ::=
    RENAME VIEW <old table name> TO <new table name>

<old table name> ::=
    <table name>

<new table name> ::=
    <identifier>
```

Syntax Rules

none

General Rules

1. The table identified by <old table name> must be a view table.
2. The current user must be the owner of the view table.
3. The <new table name> must not yet be used for a table of the current user.
4. The view table identified by <old table name> is given the <new table name>.
5. The <create view statement> of the view table identified by <old table name> is adapted to the new name. The result of this adaptation can be retrieved from the table DOMAIN.VIEWS.
6. The definitions of snapshot tables and view tables based on the view table <old table name> are adapted to the new name. For snapshot tables, these adaptations are only visible after executing a <refresh statement>.

<create index statement>

Function

creates an index for a base table or a snapshot table.

Format

```
<create index statement> ::=
    CREATE [UNIQUE] INDEX <index spec>

<index spec> ::=
    <unnamed index spec>
  | <named   index spec>

<unnamed index spec> ::=
    <table name>.<column name> [<order spec>]

<named index spec> ::=
    <index name> ON <table name> (<index clause>,...)

<index clause> ::=
    <column name> [<order spec>]

<order spec> ::=
    ASC
  | DESC
```

Syntax Rules

1. The <named index spec> must not contain more than 16 <column name>s.

General Rules

1. The table identified by <table name> must be an existing base table or snapshot table.
2. The table denoted by <table name> must not be a temporary table.
3. The <index name> of a named index must not be identical to an existing <index name> of an index for the table.
4. Up to 256 named indexes may be created per table.
5. If an index was created on exactly one column, then it is not possible to create another one-column index on this column.
6. If the <index name> is the only difference between the index defined by the <create index statement> and an existing index for the table, then the <create index statement> fails.
7. The sum of the internal lengths of the columns to be indexed must not exceed 255 characters.

8. The current user must be the owner of the table identified by <table name> or have the INDEX privilege for the table.

9. The index is created across the specified table columns. The secondary key consists of the specified columns of the table, in the specified order. The specification of ASC or DESC has the effect that the index values are stored in ascending or descending order. If the specification of ASC or DESC is omitted, ASC is implicitly assumed.

10. The index and the metadata describing the index are stored on the HOME SERVERDB of the table identified by <table name>. If the table has the <table option> WITH REPLICATION, the data is stored on all SERVERDBs of the distributed database. In this case, the SERVERDB where the <create index statement> is to be executed must belong to the majority.

11. If UNIQUE is specified, ADABAS ensures that no two rows of the specified table have the same values in the indexed columns. NULL values in one-column indexes are considered to be non-identical.

12. Indexes facilitate the access via non-key columns. But the maintenance of indexes means additional overhead in connection with <insert statement>s, <update statement>s and <delete statement>s. ASC or DESC can be specified to support the processing in a specific sort sequence that corresponds to the index definition.

<drop index statement>

Function

drops an index and its description.

Format

```
 <drop index statement> ::=
     DROP INDEX <index name> [ON <table name>]
   | DROP INDEX <table name>.<column name>
```

Syntax Rules

none

General Rules

1. The specified <table name> must be the name of an existing base table or snapshot table.

2. The specified index must exist.

3. If the <index name> clearly denotes an index, the specification 'ON <table name>' can be omitted.

4. The current user must be the owner of the table identified by <table name> or have the INDEX privilege for the table <table name>.

5. The metadata of the specified index is deleted from the catalog. The storage space occupied by the index is released.

6. If the table identified by <table name> has the <table option> WITH REPLICATION, then the SERVERDB where the current user has opened his session must belong to the majority.

<comment on statement>

Function

creates, alters, or drops a comment for a database object.

Format

```
<comment on statement> ::=
    COMMENT ON <object spec> IS <comment>

<object spec> ::=
    COLUMN  <table name>.<column name>
  | DBPROC  <db procedure>
  | DOMAIN  <domain name>
  | INDEX   <index name> ON <table name>
  | INDEX   <table name>.<column name>
  | TABLE   <table name>
  | TRIGGER <trigger name> ON <table name>
  | USER    <user name>
  | <parameter name>

<comment> ::=
    <string literal>
  | <parameter name>
```

Syntax Rules

none

General Rules

1. COMMENT ON can be used to store comments for database objects in the catalog.

2. If COLUMN is specified, then <column name> must be a column of the table identified by <table name>. The current user must be the owner of the table. A comment is stored for the column. The comment can be retrieved by selecting the system table DOMAIN.COLUMNS.

3. If DBPROC is specified, then <db procedure> must identify an existing DB procedure which is owned by the current user. A comment is stored for the DB procedure. The comment can be retrieved by selecting the system table DOMAIN.DBPROCEDURES.

4. If DOMAIN is specified, then <domain name> must identify a domain of the current user. A comment is stored for the domain. The comment can be retrieved by selecting the system table DOMAIN.DOMAINS.

5. If INDEX is specified, then <index name> or <column name> must be an index of the table identified by <table name>. The current user must be the owner of the table. A comment is stored for the index. The comment can be retrieved by selecting the system table DOMAIN.INDEXES.

6. If TABLE is specified, then <table name> must identify a non-temporary base table, view table or snapshot table of the current user. A comment is stored for the table. The comment can be retrieved by selecting the system table DOMAIN.TABLES.

7. If TRIGGER is specified, then <trigger name> must be a trigger of the table identified by <table name>. The current user must be the owner of the table. A comment is stored for the trigger. The comment can be retrieved by selecting the system table DOMAIN.TRIGGERS.

8. If USER is specified, then <user name> must identify an existing user who is owned by the current user. A comment is stored for the user. The comment can be retrieved by selecting the system table DOMAIN.USERS.

9. If a <parameter name> is specified as <object spec>, then the corresponding variable must contain one of the following values:

```
'COLUMN  <table name>.<column name>'
'DBPROC  <db procedure>'
'DOMAIN  <domain name>.<column name>'
'INDEX   <index name> ON <table name>'
'INDEX   <table name>.<column name>'
'TABLE   <table name>'
'TRIGGER <trigger name> ON <table name>'
'USER    <user name>'
```

<create user statement>

Function

defines a user.

Format

```
<create user statement> ::=
    CREATE USER <user name> PASSWORD <password>
           [<user mode>]
           [PERMLIMIT   <unsigned integer>]
           [TEMPLIMIT   <unsigned integer>]
           [TIMEOUT     <unsigned integer>]
           [COSTWARNING <unsigned integer>]
           [COSTLIMIT   <unsigned integer>]
           [CACHELIMIT  <unsigned integer>]
           [[NOT] EXCLUSIVE]
           [AT <serverdb name>]
  | CREATE USER <like user> PASSWORD <password>
           LIKE <source user>
  | CREATE USER <user name> PASSWORD <password>
           USERGROUP <usergroup name>

<user mode> ::=
    DBA
  | RESOURCE
  | STANDARD

<like user> ::=
    <user name>

<source user> ::=
    <user name>
```

Syntax Rules

1. If no <user mode> is specified, STANDARD is assumed implicitly.
2. If no <user mode> or if the <user mode> STANDARD is specified, PERMLIMIT must not be specified.
3. <unsigned integer> must be greater than 0.
4. The TIMEOUT value is specified in seconds and must lie between 30 and 86400.
5. The COSTLIMIT value must be greater than the COSTWARNING value.
6. If the EXCLUSIVE clause is omitted, ADABAS implicitly assumes EXCLUSIVE (without NOT).

General Rules

1. The <create user statement> defines a user. The existence and the properties of the user are recorded in the catalog in the form of metadata.

2. The current user must have DBA status. The user is the owner of the generated user.

3. The <user name> or <like user> must not be identical with the name of an existing user or usergroup.

4. The <password> must be specified when an ADABAS session is opened. It ensures that only authorized users obtain access to ADABAS.

5. The <user mode> specifies the user class or the status of the defined user. The user class establishes the operations on the database that may be carried out by the defined user.

6. If the user status DBA is specified, the specified user obtains the right to define private data and DB procedures, and to grant privileges for this data to other users. By specifying the <table option> WITH REPLICATION, the user can make the metadata of a table available in the catalog of each SERVERDB of the distributed database. The user can define additional users. DBA status may only be conferred by the SYSDBA created during ADABAS installation.

7. If RESOURCE is specified as the user status, the specified user obtains the right to define private data and DB procedures, and to grant the related privileges to other users.

8. If STANDARD is specified as the user status, then, aside from defining view tables, synonyms, and temporary tables, the user can only access private data created by other users for which the appropriate privileges have been granted to him.

9. The user classes are hierarchically ordered as follows:

   a) The user status RESOURCE encompasses all rights exercised by users with STANDARD status.

   b) The user status DBA encompasses all rights exercised by users with RESOURCE status.

   c) The SYSDBA, implicitly created during the installation of a SERVERDB, has the privilege to create users with DBA status on this SERVERDB. The SYSDBA is the owner of all users who were created by him or by a DBA owned by him, or whose HOME SERVERDB corresponds to his own HOME SERVERDB. Otherwise, the SYSDBA has the same function and the same rights as a DBA; i.e., whenever a DBA is allowed to execute an SQL statement, a SYSDBA can do this as well.

10. Including a PERMLIMIT in the definition of a DBA or RESOURCE user limits the disk space available for this user's private tables. This specification is made in 4 KB units. If PERMLIMIT is omitted, the user has unlimited space (within the limits of the sizes of the data devspaces specified during the installation) for private table storage.

11. Including a TEMPLIMIT in a user definition limits the disk space available to this user for the generation of temporary result tables, temporary base tables, and for execution plans. This specification is made in 4 KB units. If TEMPLIMIT is omitted, the user has unlimited space (within the limits of the sizes of the data devspaces defined during the

installation).

12. The TIMEOUT value establishes the maximum value which can be specified in the CONNECT statement as TIMEOUT value. The TIMEOUT value defines the maximum time that may pass between the completion of an <sql statement> and the issuing of the next <sql statement>.

13. COSTWARNING and COSTLIMIT specifications limit costs by preventing a user from executing <query statement>s or <insert statement>s in the form of INSERT...SELECT... beyond a specified degree of complexity.
Prior to the execution of such an SQL statement, the costs expected to result from this statement are estimated. This estimated SELECT cost value can be output using an <explain statement>. In interactive mode, it is compared with the COSTWARNING and COSTLIMIT values specified for the user. For <query statement>s or <insert statement>s having the form INSERT...SELECT... and which are embedded in a programming language, the specified COSTWARNING and COSTLIMIT values are not taken into account.

14. COSTWARNING specifies the minimum estimated SELECT cost value beyond which the user receives a warning. When this happens, the user is asked whether the relatively expensive SQL statement should actually be executed.

15. COSTLIMIT specifies the estimated SELECT cost value beyond which the SELECT statement is not executed.

16. CACHELIMIT specifies, in units of 4 KB, the maximum cache size, which the user may specify in the <connect statement> for result tables, temporary base tables, and execution plans.

17. If EXCLUSIVE is specified, then it is not possible to open two different ADABAS sessions of the user at the same time. With NOT EXCLUSIVE, this is possible.

18. AT <serverdb name> assigns a HOME SERVERDB to the user. <serverdb name> must identify a known SERVERDB in the distributed database. The HOME SERVERDB of the user is the storage location of any table rows created by the user; i.e., it is the HOME SERVERDB of all tables generated by the user. If the specification of HOME SERVERDB is omitted, the user is assigned the HOME SERVERDB of the user's owner. The only HOME SERVERDB which can be assigned to users with DBA status is the HOME SERVERDB of the generating SYSDBA.

19. If LIKE is specified, the current user must have owner authorization for the <source user>.

20. If LIKE is specified and the <source user> is not a member of a usergroup, the <user mode> and the values for PERMLIMIT, TEMPLIMIT, TIMEOUT, COSTWARNING, COSTLIMIT, CACHELIMIT, and EXCLUSIVE are assigned to the newly defined <like user> who were specified for the <source user>. In addition, the <like user> receives any privileges that other users granted the <source user>.

21. If LIKE is specified and <source user> is a member of a usergroup, then a new group member is defined with the name <like user>.

22. If USERGROUP is specified, the user issuing the SQL statement must be the owner of the usergroup. The user <user name> must be a member of the usergroup <usergroup name>.

<create usergroup statement>

Function

defines a usergroup.

Format

```
<create usergroup statement> ::=
    CREATE USERGROUP <usergroup name>
           [<usergroup mode>]
           [PERMLIMIT   <unsigned integer>]
           [TEMPLIMIT   <unsigned integer>]
           [TIMEOUT     <unsigned integer>]
           [COSTWARNING <unsigned integer>]
           [COSTLIMIT   <unsigned integer>]
           [CACHELIMIT  <unsigned integer>]
           [[NOT] EXCLUSIVE]
           [AT <serverdb name>]

<usergroup mode> ::=
    RESOURCE
  | STANDARD
```

Syntax Rules

1. If no <usergroup mode> is specified, ADABAS implicitly assumes STANDARD.
2. If no <usergroup mode> or if STANDARD is specified, PERMLIMIT must not be specified.
3. The TIMEOUT value is specified in seconds and must lie between 0 and 32400.
4. The COSTLIMIT value must be greater than the COSTWARNING value.
5. If the EXCLUSIVE clause is omitted, ADABAS implicitly assumes EXCLUSIVE (without NOT).

General Rules

1. The current user must have DBA status.
2. The <usergroup name> must not be identical with the name of an existing user or usergroup.
3. A usergroup is defined. Several users who are members of this usergroup can be defined using a <create user statement>. All private objects created by members of the usergroup are identified by the usergroup name. The owner of a private object is the group, not the user who created the object. Each user can work with any private object of the group, as if this user were the owner of the object. Privileges can only be granted or revoked from the group. A privilege cannot be granted or revoked from a single member of the group.

4. The properties of a member of a usergroup are equivalent to those of a user who is not a member of a group. These properties are described in the <create user statement>.

<drop user statement>

Function

drops the definition of a user.

Format

```
<drop user statement> ::=
    DROP USER <user name> [<cascade option>]
```

Syntax Rules

none

General Rules

1. The current user must have owner authorization over the user to be dropped.
2. At the time when the <drop user statement> is executed, the user identified by <user name> must not be connected to any SERVERDB of the database.
3. The SERVERDB where the <drop user statement> is to be executed must belong to the majority.
4. If the user to be dropped does not belong to a usergroup and is the owner of DB procedures, synonyms or tables, and the <cascade option> RESTRICT is specified, the <drop user statement> fails.
   If no <cascade option> or the <cascade option> CASCADE is specified, all DB procedures, synonyms and tables of the user to be dropped, as well as indexes, privileges, triggers, view tables, etc. based on these objects are dropped.
5. If a user with DBA status is dropped, any users generated by him remain untouched. The SYSDBA of the HOME SERVERDB of the dropped DBA becomes the new owner of this user.
6. The metadata of the user to be dropped is dropped from the catalog.

<drop usergroup statement>

Function

drops the definition of a usergroup.

Format

```
 <drop usergroup statement> ::=
     DROP USERGROUP <usergroup name> [<cascade option>]
```

Syntax Rules

none

General Rules

1. The current user must have owner authorization over the usergroup to be dropped.

2. At the time when the <drop usergroup statement> is issued, no member of the usergroup must be connected to any SERVERDB of the database.

3. The SERVERDB where the <drop usergroup statement> is to be executed must belong to the majority.

4. If the usergroup to be dropped is the owner of DB procedures, synonyms, or tables, and the <cascade option> RESTRICT is specified, then the <drop usergroup statement> fails.
   If no <cascade option> or the <cascade option> CASCADE is specified, then all DB procedures, synonyms, and tables of the usergroup to be dropped, as well as all indexes, privileges, triggers, view tables, etc. based on these objects are dropped.

5. The metadata of the usergroup to be dropped is dropped from the catalog.

<alter user statement>

Function

alters the properties assigned to a user.

Format

```
<alter user statement> ::=
    ALTER USER <user name> [<user mode>]
          [PERMLIMIT   <altered value>]
          [TEMPLIMIT   <altered value>]
          [TIMEOUT     <altered value>]
          [COSTWARNING <altered value>]
          [COSTLIMIT   <altered value>]
          [CACHELIMIT  <altered value>]
          [[NOT] EXCLUSIVE]

<altered value> ::=
    <unsigned integer>
  | NULL
```

Syntax Rules

1. At least one of the optional clauses must be specified.

General Rules

1. The specified <user name> must denote a defined user, who is not a member of a usergroup.

2. The current user must have owner authorization over the user whose properties are to be altered.

3. At the time when the <alter user statement> is issued, the user identified by <user name> must not be connected to any SERVERDB of the database.

4. The SERVERDB where the <alter user statement> is to be executed must belong to the majority.

5. If the new <user mode> is DBA, then DBA status is granted to the user specified by <user name>. DBA status can only be granted by the SYSDBA.

6. If the new <user mode> is RESOURCE, then RESOURCE status is granted to the user specified by <user name>. If the user had DBA status before, owner authorization is revoked from him for all users created by him. The new owner will be the SYSDBA who created the user identified by <user name>.

7. If the new <user mode> is STANDARD, the current status is revoked from the user, and the user loses the right to create own base tables, snapshot tables, and DB procedures. All the user's base tables, snapshot tables, and DB procedures are dropped.

8. If no <user mode> is specified, then the status of the user is not altered.

9. PERMLIMIT and TEMPLIMIT specifications for the specified user may be altered. The PERLIMIT specification may only be altered if the new value is greater than the current space requirement of all private tables.

10. If the NULL value is specified for <altered value>, then any previously defined value is cancelled.

<alter usergroup statement>

Function

alters the properties assigned to a usergroup.

Format

```
 <alter usergroup statement> ::=
     ALTER USERGROUP <usergroup name> [<usergroup mode>]
           [PERMLIMIT   <altered value>]
           [TEMPLIMIT   <altered value>]
           [TIMEOUT     <altered value>]
           [COSTWARNING <altered value>]
           [COSTLIMIT   <altered value>]
           [CACHELIMIT  <altered value>]
           [[NOT] EXCLUSIVE]
```

Syntax Rules

1. At least one of the optional clauses must be specified.

General Rules

1. The specified usergroup <usergroup name> must identify a defined usergroup.

2. The current user must have owner authorization over the usergroup whose properties are to be altered.

3. At the time when the <alter usergroup statement> is issued, no member of the usergroup must be connected to any SERVERDB of the database.

4. The SERVERDB where the <alter user statement> is to be executed must belong to the majority.

5. If the new <usergroup mode> is RESOURCE, then the specified usergroup <usergroup name> is granted the status RESOURCE.

6. If the new <usergroup mode> is STANDARD, then the usergroup loses its current status and the right to hold own data. All base tables and DB procedures of the usergroup are dropped.

7. If no <usergroup mode> is specified, the status of the usergroup remains unaltered.

8. PERMLIMIT and TEMPLIMIT specifications may be altered for the specified usergroup. The PERMLIMIT specification may only be altered if the new value is greater than the current space requirement of all private tables.

9. If the NULL value is specified for <altered value>, then any previously defined value is cancelled.

<grant user statement>

Function

grants another user the owner authorization of a SYSDBA or a DBA over a user.

Format

```
<grant user statement> ::=
    GRANT USER <user name>,...
    [FROM <user name>] TO <user name>
```

Syntax Rules

none

General Rules

1. The current user must be a DBA.
2. The <user name>s specified to the right of the keywords FROM and TO must be different from each other and must identify DBAs. If 'FROM <user name>' is not specified, ADABAS implicitly assumes the current user.
3. The <user name>s specified to the right of the keywords GRANT USER must identify existing users with RESOURCE or STANDARD status for which the user specified to the right of the keyword FROM has owner authorization. These users must not be members of a usergroup.
4. The FROM user grants the TO user the owner authorization which the FROM user has over the specified users. These rights are revoked from the FROM user. In particular, the TO user is granted the right to drop any specified user and to alter the status and other properties of this user.

<grant usergroup statement>

Function

grants another user the owner authorization of a SYSDBA or DBA over a usergroup.

Format

```
 <grant usergroup statement> ::=
     GRANT USERGROUP <usergroup name>,...
     [FROM <user name>] TO <user name>
```

Syntax Rules

none

General Rules

1. The current user must be a DBA.
2. The <user name>s specified to the right of the keywords FROM and TO must be different from each other and must identify DBAs. If 'FROM <user name>' is not specified, ADABAS implicitly assumes the current user.
3. The <usergroup name> must identify a usergroup for which the user specified to the right of the keyword FROM has the owner authorization.
4. The FROM user grants the TO user the owner authorization which the FROM user has over the specified usergroup. These rights are revoked from the FROM user. In particular, the TO user is granted the right to drop any usergroup <usergroup name>, to alter the status and properties of this usergroup, as well as to drop or create group members.

<alter password statement>

Function

alters the password of a user.

Format

```
<alter password statement> ::=
    ALTER PASSWORD <old password> TO <new password>
  | ALTER PASSWORD <user name> <new password>

<old password> ::=
    <password>

<new password> ::=
    <password>
```

Syntax Rules

none

General Rules

1. <old password> must match the password entered in the catalog for the current user.
2. If <user name> is specified, then the current user must be the SYSDBA of the <user name>'s HOME SERVERDB or the SYSDBA of the HOME SERVERDB of the <user name>'s owner.
3. The <new password> must be specified in the <connect statement> when the next session of the user is opened.

<grant statement>

Function

grants privileges for tables and single columns, as well as for the execution of DB procedures.

Format

```
<grant statement> ::=
    GRANT <priv spec>,... TO <grantee>,... [WITH GRANT OPTION]
  | GRANT EXECUTE ON <db procedure> TO <grantee>,...

<priv spec> ::=
    <table privileges> ON [TABLE] <table name>,...

<table privileges> ::=
    ALL [PRIV[ILEGES]]
  | <privilege>,...

<privilege> ::=
    INSERT
  | UPDATE [(<column name>,...)]
  | SELECT [(<column name>,...)]
  | SELUPD [(<column name>,...)]
  | DELETE
  | INDEX
  | ALTER
  | REFERENCES [(<column name>,...)]

<grantee> ::=
    PUBLIC
  | <user name>
  | <usergroup name>
```

Syntax Rules

none

General Rules

1. A <priv spec> defines a set of privileges for each table identified by <table name>. None of these tables must be a temporary base table.
   The user must have the authorization to grant privileges for the specified tables. For base tables, the owner of the table has this authorization.
   For view tables and snapshot tables, it may happen that not even the owner is authorized to grant all privileges. Which privileges a user may grant for a view table or snapshot table is determined by ADABAS upon generation of the table. The result depends on the type of the table, as well as on the user's privileges for the tables selected in the view table or snapshot table. The owner of a table can retrieve the privileges he is allowed to grant by selecting the system table DOMAIN.PRIVILEGES.

2. The INSERT privilege allows the user identified by <grantee> to insert rows into the

specified tables. The current user must have the authorization to grant the INSERT privilege.

3. The UPDATE privilege allows the user identified by <grantee> to update rows in the specified tables. If <column name>s are specified, the rows may only be updated in the columns identified by these names. The current user must have the authorization to grant the UPDATE privilege.

4. The SELECT privilege allows the user identified by <grantee> to select rows from the specified tables. If <column name>s are specified, then only the columns defined by these names can be selected. The current user must have the authorization to grant the SELECT privilege.

5. SELUPD grants the privileges SELECT and UPDATE. If <column name>s are specified, then the rows may only be altered and selected in the columns identified by these names. The current user must have the authorization to grant both the SELECT and the UPDATE privilege.

6. The DELETE privilege allows the user identified by <grantee> to delete rows from the specified tables. The current user must have the authorization to grant the DELETE privilege.

7. The INDEX privilege allows the user identified by <grantee> to execute the <create index statement> and the <drop index statement> for the specified tables. The INDEX privilege can only be granted for base tables and snapshot tables, and the current user must have the authorization to grant the INDEX privilege.

8. The ALTER privilege allows the user identified by <grantee> to execute the <alter table statement> for the specified tables. The ALTER privilege can only be granted for base tables, and the current user must have the authorization to grant the ALTER privilege.

9. The REFERENCES privilege allows the user identified by <grantee> to specify the table <table name> as <referenced table> in a <column definition> or <referential constraint definition>. The current user must have the authorization to grant the REFERENCES privilege. If <column name>s are specified, columns identified by these names can only be specified as <referenced column>s.

10. All privileges which the user is authorized to grant for the tables using ALL [PRIV[ILEGES]] are granted to the users identified by the sequence of <grantee>s.

11. <grantee> must not be identical with the <user name> of the current user and the name of the table owner.

12. <grantee> must not denote a member of a usergroup.

13. If PUBLIC is specified, the listed privileges are granted to all users, both to current ones and to any created later.

14. The specification of WITH GRANT OPTION allows the user identified by <grantee> to grant other users the received privileges. The current user must have the authorization to grant the privileges to be passed on.

15. GRANT EXECUTE allows the user identified by <grantee> to execute the DB procedure <db procedure>.
    The current user must be the owner of the DB procedure.

During the translation of a DB procedure, ADABAS checks whether the owner of this DB procedure has the authorization to grant all privileges that are required for the execution of the DB procedure. If this is not the case, the <grant statement> fails. Otherwise, the users identified by the sequence of <grantee>s implicitly receive all privileges that are required for the execution of the DB procedure. These privileges only remain in effect for the execution of the DB procedure; i.e., these privileges do not exist for the users in programs or sessions with interactive ADABAS tools, unless they have been granted explicitly.

<revoke statement>

Function

revokes privileges.

Format

```
 <revoke statement> ::=
     REVOKE <priv spec>,... FROM <grantee>,... [<cascade option>]
   | REVOKE EXECUTE ON <db procedure> FROM <grantee>,...
```

Syntax Rules

none

General Rules

1. The owner of a table can revoke the privileges granted for this table from any user. By specifying ALL, the owner of the table revokes all privileges granted for the table from the user.

2. If a user is not the owner of the table, he may only revoke the privileges he has granted. If a user who is not the owner of the table specifies ALL, he revokes all privileges he has granted for this table from the user identified by <grantee>.

3. If the SELECT privilege was granted for a table without the specification of <column name>s, REVOKE SELECT (<column name>,...) can be used to revoke the SELECT privilege for the specified columns; the SELECT privilege for table columns that have not been specified remains unaffected. The same is true for the UPDATE and SELUPD privileges.

4. The <revoke statement> can cascade; i.e., revoking a privilege from one user can have the effect that this privilege is revoked from other users who may have received this privilege from the user specified in the <revoke statement>. More precisely:
   Let U_1, U_2, and U_3 be users. U_1 grants U_2 the privilege set P WITH GRANT OPTION, and U_2 grants U_3 the privilege set P', P' <= P. If U_1 revokes the privilege set P'', P'' <= P from the user U_2, then the privilege set (P' * P'') is implicitly revoked from U_3.

5. Whenever the SELECT privilege is revoked from the owner of a view table for a column which is a <select column> but does not occur in the <table expression> of the view definition, then the column defined by <select column> is dropped from the view table.
   If this view table is used in the <from clause> of another view table, then the described procedure is recursively applied to this view table.

6. If the SELECT privilege is revoked from the owner of a view table for a column or table occurring in the <table expression> of the view definition, the view table is dropped, along with all view tables, privileges, and synonyms based on this view table, if no <cascade option> or the <cascade option> CASCADE is specified. If RESTRICT is

specified, the <revoke statement> fails in this case.

7. If REVOKE EXECUTE is specified, the authorization to execute the DB procedure <db procedure> is revoked from the user identified by <grantee>. The authorization for execution can only be revoked by the owner of the DB procedure.

Data Manipulation

Every SQL statement for data manipulation implicitly sets an EXCLUSIVE lock for each inserted, updated, or deleted row.
Whenever a user holds too many row locks on a table within a transaction, ADABAS tries to convert these row locks into a table lock. If this causes collisions with other locks, ADABAS continues to request row locks. This means that table locks are obtained without waiting periods. The limit beyond which ADABAS tries to transform row locks into table locks depends on the installation parameter MAXLOCKS that indicates the maximum number of possible lock entries.

<insert statement>

Function

inserts rows into a table.

Format

```
<insert statement> ::=
    INSERT [INTO] <table name> <insert columns and values>
    [<duplicates clause>]

<insert columns and values> ::=
    [(<column name>,...)] VALUES (<extended expression>,...)
  | [(<column name>,...)] <query expression>
  | SET <set insert clause>,...

<extended expression> ::=
    <expression>
  | DEFAULT
  | STAMP

<duplicates clause> ::=
    REJECT DUPLICATES
  | IGNORE DUPLICATES
  | UPDATE DUPLICATES

<set insert clause> ::=
    <column name> = <extended value spec>
```

Syntax Rules

1. A column specified in the optional sequence of <column name>s or a column of a <set insert clause> identified by <column name> is a target column. Target columns can be specified in any order.

2. If neither a sequence of <column name>s nor a <set insert clause> is specified, this has the same effect as the specification of a sequence of <column name>s containing all columns of the table in the order in which they were defined in the <create table statement> or <create view statement>. In this case, every table column defined by the user is a target column.

3. The number of specified <extended expression>s must equal the number of target columns. The ith <extended expression> is assigned the ith <column name>.

4. The number of <select column>s specified in the <query expression> must equal the number of target columns.

General Rules

1. <table name> must identify an existing base table or view table or a synonym.

2. If a <set insert clause> or <column name>s are specified, all specified column names must identify columns of the table <table name>.
   If the table <table name> was defined without a key; i.e., if the column SYSKEY was implicitly created by ADABAS, the column SYSKEY must not occur in the sequence of <column name>s or in a <set insert clause>.
   A column must not occur more than once in a sequence of <column name>s or in more than one <set insert clause>.

3. The user must have the INSERT privilege for the table identified by <table name>.
   If <table name> identifies a view table, it may happen that not even the owner of the view table has the INSERT privilege because the view table is not updatable.

4. All mandatory columns of the table identified by <table name> must be target columns.

5. If <table name> identifies a view table, rows are inserted into the base table(s), on which the view table is based. In this case, the target columns of <table name> correspond to the columns of base tables, on which the view table is based. In the following paragraphs, the term target column always refers to the corresponding column of the base tables.

6. If there is no <query expression> in the <insert statement>, exactly one row is inserted into the table <table name>. The effects this has on join view tables are described below. The inserted row has the following contents:

   a) All columns of the base table which are target columns of the <insert statement> contain the value assigned to the respective target column.

   b) All columns of the base table which are not target columns of the <insert statement> and for which a <default spec> exists contain the <default value>.

   c) All columns of the base table which are not target columns of the <insert statement> and for which no <default spec> exists contain the NULL value.

7. If <table name> does not identify a join view table and if there is already a row with the key specified for the row to be inserted, the result depends on the <duplicates clause> (see below). If the <duplicates clause> is omitted, the <insert statement> fails.

8. If <table name> identifies a join view table, a row is inserted into each base table on which the view table is based. If there is already a row in the key table of the view table with the key of the row to be inserted, the <insert statement> fails. If any row in a base table, which is not the key table of the view table, already has the key of the row to be inserted, then the <insert statement> fails if the row to be inserted does not match the existing row.

9. If the <insert statement> contains a <query expression>, <table name> must not identify a join view table.

10. A <query expression> in the <insert statement> defines a result table whose ith column is assigned to the ith target column. out of each result table row, a row is formed for the table <table name> and inserted into the base table on which <table name> is based. Each of these rows has the following contents:

    a) Each base table column which is the target column of the <insert statement>

contains the value of the column in the current result table row assigned to it.

b) All columns of the base table which are not target columns of the <insert statement> and for which a <default spec> exists contain the <default value>.

c) All columns of the base table which are not target columns of the <insert statement> and for which no <default spec> exists contain the NULL value.

11. If there is already a row in the base table with the key of the row to be inserted, the following cases must be distinguished:

a) If IGNORE DUPLICATES is specified, the new row is not inserted and ADABAS continues to process the <insert statement>.

b) If UPDATE DUPLICATES is specified, the new row overwrites the existing row and ADABAS continues to process the <insert statement>.

c) If no <duplicates clause> or if REJECT DUPLICATES is specified, the <insert statement> fails.

12. If there is more than one key collision for the same key for an <insert statement> with UPDATE DUPLICATES and <query expression> specification, then it is impossible to predict what content the respective base table row will have once the <insert statement> is completed.

13. If for an <insert statement> with IGNORE DUPLICATES and <query expression> specification, more than one row of the result table produce the same base table key, and if this key has not yet existed in the base table, then it is impossible to predict which row will be inserted into the table.

14. If <table name> identifies a table without user-defined key, then the <duplicates clause> has no effect.

15. If there are <constraint definition>s for the base tables into which rows are to be inserted by using the <insert statement>, ADABAS checks for each row to be inserted whether it satisfies the <constraint definition>s. If this is not the case for at least one row, the <insert statement> fails.

16. If at least one of the base tables into which rows are to be inserted using the <insert statement> is the referencing table of a <referential constraint definition>, ADABAS checks for each row to be inserted, whether the foreign key resulting from the row exists as a key or as a value of an index defined with UNIQUE in the corresponding <referenced table>. If this is not the case for at least one row, the <insert statement> fails.

17. Let C be a target column and v a non-NULL value to be stored in C.

18. If C is a numeric column, v must be a number within the permitted range of values of C. If v is the result of a <query expression>, fractional digits are rounded, if necessary.

19. If C is an alphanumeric column with the code attribute ASCII or EBCDIC, then v must be a character string with a length not exceeding the length attribute of C. Trailing blanks are disregarded in determining the length of v. If the length of v is shorter than the length attribute of C, then v is lengthened by the appropriate number of blanks. If an alphanumeric value with the code attribute ASCII (EBCDIC) is assigned to a column with the code attribute EBCDIC (ASCII), the value is implicitly converted prior

to its assignment.

20. If C is an alphanumeric column with the code attribute BYTE, then v must be a hexadecimal character string with a length not exceeding the length attribute of C. Trailing binary zeros are disregarded in determining the length of v.
If the length of v is shorter than the length attribute of C, then v is lengthened by the corresponding number of binary zeros.

21. If C is a column of the data type DATE, then v must be a date value in the current date format.

22. If C is a column of the data type TIME, then v must be a time value in the current time format.

23. If C is a column of the data type TIMESTAMP, then v must be a timestamp value in the current timestamp format.

24. If C is a column of the data type BOOLEAN, then v must denote one of the values TRUE, FALSE, or the NULL value.

25. The value specified by a <parameter spec> of an <expression> is the value of the parameter identified by this <parameter spec>. If an indicator parameter is specified with a negative value, then the value defined by the <parameter spec> is the NULL value.

26. The <insert statement> can only be used to assign a value to columns of the data type LONG if it contains a parameter or NULL specification. The assignment of values to LONG columns is therefore only possible with some ADABAS tools. For details, refer to the corresponding manuals.

27. An <insert statement> sets the third entry of SQLERRD in the SQLCA (see the Precompiler online help) to the number of inserted rows.

28. If errors occur in the process of inserting rows, the <insert statement> fails, leaving the table unmodified.

<update statement>

Function

updates column values in table rows.

Format

```
<update statement> ::=
    UPDATE [OF] <table name> [<reference name>]
           <update columns and values>
           [KEY <key spec>,...]
           [WHERE <search condition>]
  | UPDATE [OF] <table name> [<reference name>]
           <update columns and values>
           WHERE CURRENT OF <result table name>

<update columns and values> ::=
    SET <set update clause>,...
  | (<column>,...) VALUES (<extended value spec>,...)

<set update clause> ::=
    <column name> = <extended expression>
  | <column name> = <subquery>
```

Syntax Rules

1. Columns whose values are to be updated are called target columns.

2. The number of the specified <extended value spec>s must equal the number of target columns. The ith <extended value spec> is assigned to the ith target column.

3. The <expression> in a <set update clause> must not contain a <set function spec>.

4. The <subquery> must produce a single-column result table with up to one row.

General Rules

1. <table name> must identify an existing base table, view table, or a synonym.

2. All target columns must identify columns of the table <table name>, and each target column may only be listed once.

3. The current user must have the UPDATE privilege for each target column in <table name>.
   If <table name> identifies a view table, it may happen that not even the owner of the view table is able to update column values because the view table is not updatable.

4. If <table name> identifies a view table, column values are only updated in rows which belong to the base tables on which the view table is based. In this case, the target columns of <table name> correspond to columns of the base tables, on which the view

table is based. In the following paragraphs, the term target column always refers to the corresponding column in the base tables.

5. Values of key columns defined by a user for a <create table statement> or <alter table statement> can be updated. The implicit key column SYSKEY, if created, cannot be updated.

6. If <table name> identifies a join view table, columns may exist which can only be updated in combination with other columns. This is true of all target columns, which are

   a) located in a base table which is not a key table of the join view table and which does not have a 1 : 1 relationship with the key table of the join view table, or which are

   b) foreign key columns of a <referential constraint definition> which is relevant to the join view table.

   To determine the combination of columns for a given column v in the join view table V, use the following procedure:

   a) Determine the base table Tj containing the column which corresponds to v.

   b) Determine the unique sequence of tables Ti1 .. Tik containing Tj.

   c) Determine Til, the last table of this sequence, which is in a 1 : 1 relationship with the key table.

   d) The columns of V which correspond to the foreign key columns in Til of the V-relevant <referential constraint definition> between Til and Til+1 are elements of the column combination.

   e) All columns of V which correspond to columns of the tables Til+1..Tik are elements of the column combination.

   To update the column value of the column v, a value must be specified for each of the columns of the column combination.

7. <update columns and values> identifies one or more target columns and new values for these columns. The optional sequence of <key spec>s and the optional <search condition> or, in case of CURRENT OF, the cursor position within the result table <result table name> determine the rows of the specified table to be updated

8. If neither a sequence of <key spec>s nor a <search condition> nor CURRENT OF <result table name> is specified, all rows of the specified table are updated.

9. If a sequence of <key spec>s but no <search condition> is specified and a row with the specified key values exists, the corresponding values are assigned to the target columns of this row.

10. If a sequence of <key spec>s and a <search condition> are specified and a row with the specified key values exists, the <search condition> is applied to this row. If the <search condition> is satisfied, the corresponding values are assigned to the target columns of this row.

11. If no sequence of <key spec>s but a <search condition> is specified, the <search condition> is applied to each row of the specified table. The corresponding values are assigned to the target columns of all rows that satisfy the <search condition>.

12. If CURRENT OF <result table name> is specified, the <table name> in the <from clause> of the <query statement> that generated the result table <result table name> must be the same as the <table name> in the <update statement>.

13. If CURRENT OF <result table name> is specified and the cursor is positioned on a row of the result table, the corresponding values are assigned to the target columns of the corresponding row. The corresponding row is the row of the table specified in the <from clause> of the <query statement>, from which the particular result table row was formed. This procedure only works if the result table was specified with FOR UPDATE. It is impossible to predict whether the updated values in the corresponding row are visible the next time the same row of the result table is accessed.

14. If a sequence of <key spec>s is specified and none of the rows has the specified key values, then no row is updated. If a <search condition> applied to a row is not satisfied, then the row concerned is not updated.

15. If CURRENT OF <result table name> is specified and the cursor is not positioned on a row of the result table, no row is updated.

16. If no row is found for which the conditions defined by the optional clauses are satisfied, the message 100 - ROW NOT FOUND - is set.

17. If there are <constraint definition>s for the base tables in which rows have been updated using the <update statement>, ADABAS checks for each updated row whether it satisfies the <constraint definition>s. If this is not the case for at least one row, the <update statement> fails.

18. For each row in which the values of foreign key columns have been updated using the <update statement>, ADABAS checks whether the respective resulting foreign key exists as a key or as a value of an index defined with UNIQUE in the corresponding <referenced table>. If this is not the case for at least one row, the <update statement> fails.

19. For each row in which the value of a <referenced column> of a <referential constraint definition> is to be updated using the <update statement>, ADABAS checks whether there are rows in the corresponding <referencing table> that contain the old column values as foreign keys. If this is the case for at least one row, the <update statement> fails.

20. The <subquery> must produce a result table containing up to one row.

21. Let C be a target column and v a non-NULL value for the modification of C.

22. If C is a numeric column, then v must be a number within the permitted range of values for C. If v is the result of an <expression> that is not made up of a single <numeric literal>, then fractional digits are rounded whenever necessary.

23. If C is an alphanumeric column with the code attribute ASCII or EBCDIC, then v must be a character string with a length that does not exceed the length attribute of C. Trailing blanks are disregarded in determining the length of v. If the length of v is shorter than the length attribute of C, then v is lengthened by the corresponding number of blanks. When assigning an alphanumeric value with the code attribute

ASCII (EBCDIC) to a column with the code attribute EBCDIC (ASCII), the value is implicitly converted prior to its assignment.

24. If C is an alphanumeric column with the code attribute BYTE, then v must be a hexadecimal character string with a length that does not exceed the length attribute of C. Trailing binary zeros are disregarded in determining the length of v.
If the length of v is shorter than the length attribute of C, then v is lengthened by the corresponding number of binary zeros.

25. If C is a column of the data type DATE, then v must be a date value in the current date format.

26. If C is a column of the data type TIME, then v must be a time value in the current time format.

27. If C is a column of the data type TIMESTAMP, then v must be a timestamp value in the current timestamp format.

28. If C is a column of the data type BOOLEAN, then v must denote one of the values TRUE, FALSE, or the NULL value.

29. The <update statement> can only be used to assign a new value to columns of the data type LONG if it contains a parameter or NULL specification. The assignment of values to LONG columns is therefore only possible with some ADABAS tools. For details, refer to the corresponding manuals.

30. An <update statement> sets the third entry of SQLERRD in the SQLCA (see the Precompiler online help) to the number of updated rows. Rows are also counted as updated when the old value was overwritten with a new but identical value.

31. Should errors occur in the process of updating a row, the <update statement> fails, leaving the table unmodified.

<delete statement>

Function

deletes rows from a table.

Format

```
 <delete statement> ::=
     DELETE [FROM] <table name> [<reference name>]
            [KEY <key spec>,...]
            [WHERE <search condition>]
   | DELETE [FROM] <table name> [<reference name>]
            WHERE CURRENT OF <result table name>
```

Syntax Rules

none

General Rules

1. <table name> must identify an existing base table, view table, or a synonym.

2. The current user must have the DELETE privilege for the table identified by <table name>.
   f <table name> identifies a view table, it may happen that not even the owner of the view table has the DELETE privilege because the view table is not updatable.

3. If <table name> identifies a view table, rows are deleted from the base tables, on which the view table is based.
   If <table name> identifies a join view table, then rows are only deleted in the key table of the join view table and in base tables on which the view table is based and which have a 1 : 1 relationship with the key table.

4. The optional sequence of <key spec>s and the optional <search condition> or, in case of CURRENT OF <result table name>, the cursor position determines the rows of the specified table to be deleted.

5. If neither a sequence of <key spec>s nor a <search condition> nor CURRENT OF <result table name> is specified, all rows of the specified table are deleted.

6. If a sequence of <key spec>s but no <search condition> is specified and a row with the specified key values exists, then the row is deleted.

7. If a sequence of <key spec>s and a <search condition> are specified and a row with the specified key values exists, then the <search condition> is applied to this row. If the <search condition> is satisfied, then the row is deleted.

8. If no sequence of <key spec>s but a <search condition> is specified, the <search condition> is applied to each row of the specified table. All rows for which the <search

condition> is satisfied are deleted.

9. If CURRENT OF <result table name> is specified, the <table name> in the <from clause> of the <query statement> which generated the result table must be the same as the <table name> in the <delete statement>.

10. If CURRENT OF <result table name> is specified and the cursor is positioned on a row of the result table, the corresponding row is deleted. The corresponding row is the row of the table specified in the <from clause> of the <query statement>, from which the result table row was formed. This procedure requires that the result table was specified with FOR UPDATE. Afterwards, the cursor is positioned behind the result table row.
    It is impossible to predict whether the deletion of the corresponding row is visible the next time the same row of the result table is accessed.

11. If a sequence of <key spec>s is specified and none of the rows has the specified key values, no row is deleted. If a <search condition> applied to a row is not satisfied, this row is not deleted. If CURRENT OF <result table name> is specified and the cursor is not positioned on a row of the result table, no row is deleted.

12. If no row is found which satisfies the conditions defined by the optional clauses, the message 100 - ROW NOT FOUND - is set.

13. For each row deleted in the course of the <delete statement> which comes from a <referenced table> of at least one <referential constraint definition>, one of the following actions is taken - depending on the <delete rule> of the <referential constraint definition>:

    a) <delete rule> = DELETE CASCADE
    All matching rows in the corresponding foreign key table are deleted.

    b) <delete rule> = DELETE RESTRICT
    If there are matching rows in the corresponding foreign key table, the <delete statement> fails.

    c) <delete rule> = DELETE SET NULL
    The NULL value is assigned to the respective foreign key columns of all matching rows in the corresponding foreign key table.

    d) <delete rule> = DELETE SET DEFAULT
    The <default value> is assigned to the respective foreign key columns of all matching rows in the corresponding foreign key table.

14. A <delete statement> sets the third entry of SQLERRD in the SQLCA (see the Precompiler online help) to the number of deleted rows. If this counter has the value -1, either a great part of the table or the complete table was deleted by the <delete statement>.

15. If errors occur in the course of the <delete statement>, the statement fails, leaving the table unmodified.

<refresh statement>

Function

updates a snapshot table.

Format

```
 <refresh statement> ::=
     REFRESH SNAPSHOT <table name> [COMPLETE]
```

Syntax Rules

none

General Rules

1. <table name> must identify an existing snapshot table.

2. The current user must be the owner of the snapshot table identified by <table name>.

3. The contents of the snapshot table are updated; i.e., after execution of the <refresh statement>, the snapshot table contains the result of the <query expression> defined for the <create snapshot statement>. If indexes were defined for the snapshot table, these are updated as well.

4. If COMPLETE is specified, the existing contents of the snapshot table are deleted and completely recreated. If COMPLETE is not specified, then it depends on the definition of the <query expression> and on the definition of a snapshot log whether only the modifications on an underlying table need to be executed in the snapshot table or the contents of the snapshot table are completely to be recreated.

5. If there is a snapshot log for the only table underlying the snapshot table, the snapshot log is deleted after executing the <refresh statement>. Deletion starts at the beginning of the snapshot log and stops at the first entry required for the refresh of the oldest snapshot table that needs to be refreshed.

6. If data definition SQL statements were performed on the table(s) underlying a snapshot table between the <create snapshot statement> or the last <refresh statement> for the specified snapshot table and the current <refresh statement>, then the snapshot table is updated completely. Indexes defined on the snapshot table are implicitly dropped. If they are needed, they must be recreated using a new <create index statement>.

7. If data definition SQL statements performed on the underlying table(s) in the meantime have the effect that the <query expression> specified for the <create snapshot statement> can no longer be executed free of errors, then an error message is output for the next <refresh statement>, not for the data definition SQL statement on the underlying table.

8. If errors occur with the <refresh statement>, this statement fails, leaving the snapshot table unmodified.

<clear snapshot log statement>

Function

deletes the contents of the snapshot log of the specified table.

Format

```
 <clear snapshot log statement> ::=
     CLEAR SNAPSHOT LOG ON <table name>
```

Syntax Rules

none

General Rules

1. <table name> must identify an existing base table.

2. The current user must be the owner of the snapshot table identified by <table name>.

3. The contents of the snapshot log are completely deleted. The next <refresh statement> for snapshot tables based on the specified table has the effect that the snapshot table is deleted and recreated although the <refresh statement> was specified without COMPLETE.

4. The <clear snapshot log statement> can be used to release storage space in the database. The <clear snapshot log statement> makes sense if no <refresh statement> has been performed for some snapshot tables that are based on the specified table and that would use the snapshot log for the <refresh statement> for a very long time. On the one hand, the number of modifications which had to be made to the snapshot table can become so large that recreating the complete contents of the snapshot table could be more advantageous than performing each single modification. On the other hand, the storage space required for the snapshot log of a table that is frequently modified can become very large.

<next stamp statement>

Function

produces a unique key generated by ADABAS.

Format

```
 <next stamp statement> ::=
     NEXT STAMP [FOR <tablename>] [INTO] <parameter name>
```

Syntax Rules

none

General Rules

1. ADABAS is able to generate unique values. These values consist of the number of the SERVERDB and a number, which is consecutive for the specific SERVERDB and begins with X'000000000001'. The values are assigned in ascending order. It cannot be ensured that a sequence of values is uninterrupted. These values can be stored in a column of the data type CHAR(n) BYTE with n>=8.

2. NEXT STAMP assigns the next key generated by ADABAS to the variable denoted by <parameter name>. If the <next stamp statement> is specified without a <table name>, the value is generated on the SERVERDB to which the current user is connected. If a <table name> is specified in the <next stamp statement>, the value is generated on the HOME SERVERDB of the table. In this way, an ascending order can be ensured, which would not be possible if values were generated on different SERVERDBs.

3. The <next stamp statement> cannot be used in interactive mode; it can only be embedded in a programming language.

4. The keyword STAMP can be used in <insert statement>s and <update statement>s if the next value is to be generated by ADABAS and to be stored in a column without the user knowing the value.

Data Retrieval

A network failure in a distributed database can have the effect that not all SERVERDBs of the database can communicate with each other. Data stored on a SERVERDB to which network communication is no longer possible cannot be read nor modified.

If the network of SERVERDBs has divided into two subnetworks because of the failure of network communication, the majority concept is applied. This means that SERVERDBs belonging to the larger subnetwork, the majority, can still modify the replicated (metadata) data. SERVERDBs that could not be accessed when these modifications were made are informed about the modifications after reestablishing the network communication.

As a result, SERVERDBs that do not belong to the majority may only be able to modify local (metadata) data. Data retrieval of local and replicated data is possible. It can happen that data of replicated tables has been modified in the majority and these modifications could not be made in the local copy of data because of the missing network communication, so that the data is no longer up to date. A warning informs the user about such a situation (cf. SQLWARNA in the Precompiler online help).

<query statement>

Function

specifies a result table that can be ordered.

Format

```
<query statement> ::=
    <declare cursor statement>
  | <named select statement>
  | <select statement>

<declare cursor statement> ::=
    DECLARE <result table name> CURSOR FOR <select statement>

<named select statement> ::=
    <named query expression>
    [<order clause>]
    [<update clause>]
    [<lock option>]
    [FOR REUSE]

<select statement> ::=
    <query expression>
    [<order clause>]
    [<update clause>]
    [<lock option>]
    [FOR REUSE]
```

Syntax Rules

none

General Rules

1. The <declare cursor statement> defines a result table with the <result table name>. To generate this result table, an <open cursor statement> specifying the name of the result table is needed.

2. The <named select statement> defines and generates a result table with the <result table name>. An <open cursor statement> is not allowed for such a result table.

3. The <select statement> defines and generates an unnamed result table. An <open cursor statement> is not allowed for such a result table. The difference between a named result table and an unnamed result table is that the unnamed result table cannot be specified in the <from clause> or in CURRENT OF <result table name> of a subsequent SQL statement. Moreover, the column names of a result table generated by a <named select statement> must be unique; this is not necessary for a result table generated by a <select statement> or defined by a <declare cursor statement>.

4. The rules that in the present and following chapters are specified for the <declare cursor statement>, as well as the rules for the <open cursor statement> apply for the <named select statement> and the <select statement>.

5. If the result table is to be specified in the <from clause> of a subsequent <query statement>, it should be specified with FOR REUSE. If FOR REUSE is not specified, the reusability of the result table depends on internal system strategies.
As the specification of FOR REUSE deteriorates the response times of some <query statement>s, FOR REUSE should only be specified if such a specification is required for the reusability of the result table.

6. The order of rows in the result table depends on the internal search strategies of the system and is arbitrary. The only way to obtain a particular ordering of the result rows is by specifying an <order clause>.

7. A result table or, more precisely, the underlying base tables, are updatable if the <query statement> satisfies the following conditions:

   a) The <query expression> or the <named query expression> may only consist of one <query spec> or <named query spec>.

   b) One base table or one updatable view table may only be specified in the <from clause> of the <query spec> or <named query spec>.

   c) DISTINCT, GROUP BY or HAVING must not be specified.

   d) <expression>s must not contain a <set function spec>.

   e) The result table is a named result table; i.e. it was not generated by using a <select statement>.

8. An <update clause> can only be specified for updatable result tables. For updatable result tables, a position within a particular result table always corresponds to a position in the underlying tables and thus, ultimately, to a position in one or more base tables. If an <update clause> was specified, the position in the result table (specification of CURRENT OF <result table name>) can be used to modify the base table by an <update statement> or <delete statement>. The position in a base table can be used to issue a <select direct statement> or a <select ordered statement>; or a <lock statement> can be used to request a lock for the row concerned in each base table involved.

9. According to the search strategy either all rows of the result table are searched for a <named select statement>, <select statement> or <open cursor statement>, the result table being physically generated; or each next result table row is searched for a <fetch statement>, without being physically stored. This must be considered for the FETCH time behavior.

See also

<query expression>, <named query expression>

<query spec>, <named query spec>

<table expression>

<subquery>

<order clause>

<update clause>

<lock option>

<query expression>, <named query expression>

Function

specifies an unordered result table.

Format

```
<query expression> ::=
    <query term>
  | <query expression> UNION  [ALL] <query term>
  | <query expression> EXCEPT [ALL] <query term>

<query term> ::=
    <query primary>
  | <query term> INTERSECT [ALL] <query primary>

<query primary> ::=
    <query spec>
  | (<query expression>)

<named query expression> ::=
    <named query term>
  | <named query expression> UNION [ALL] <query term>
  | <named query expression> EXCEPT [ALL] <query term>

<named query term> ::=
    <named query primary>
  | <named query term> INTERSECT [ALL] <query primary>

<named query primary> ::=
    <named query spec>
  | (<named query expression>)
```

Syntax Rules

1. If a <named query expression> consists of more than one <query spec>, then only the first <query spec> of the <named query expression> may be a <named query spec>.

General Rules

1. A <named query expression> corresponds almost entirely to a <query expression>. Therefore only the <query expression> is described. Only if there is a significant difference between the <named query expression> and the <query expression>, the <named query expression> is described, too. The same is true for the <named query term>, <named query primary>, and <named query spec>.

2. A <query expression> specifies a result table. If the <query expression> only consists of one <query spec>, the result of the <query expression> is the unmodified result of the <query spec>.

3. If the <query expression> consists of more than one <query spec>, the number of <select column>s must be the same in all <query spec>s of the <query expression>. The particular ith <select column>s of the <query spec>s must be comparable.

Numeric columns can be compared to each other. If all ith <select column>s are numeric columns, the ith column of the result table is a numeric column.
Alphanumeric columns with the code attribute BYTE can be compared to each other.
Alphanumeric columns with the code attribute ASCII or EBCDIC can be compared to each other and to date, time, and timestamp values.
If all ith <select column>s are date values, the ith column of the result table is a date value.
If all ith <select column>s are time values, the ith column of the result table is a time value.
If all ith <select column>s are timestamp values, the ith column of the result table is a timestamp value.
Columns of the data type BOOLEAN can be compared to each other. If all ith <select column>s are values of the data type BOOLEAN, the ith column of the result table is of the data type BOOLEAN.
In all the other cases, the ith column of the result table is an alphanumeric column.
Comparable columns with differing code attributes are converted.
If columns are comparable but have different lengths, the corresponding column of the result table has the maximum length of the underlying columns.

4. The names of the result table columns are formed from the names of the <select column>s of the first <query spec>.

5. Let T1 be the left operand of UNION, EXCEPT or INTERSECT. Let T2 be the right operand. Let R be the result of the operation on T1 and T2.
   A row is a duplicate of another row if both rows have identical values in each column. NULL values are assumed to be identical. Special NULL values are assumed to be identical.

6. If UNION is specified, R contains all rows of T1 and T2.

7. If EXCEPT is specified, then R contains all rows of T1 which have no duplicate rows in T2.

8. If INTERSECT is specified, then R contains all rows of T1 which have a duplicate row in T2. One row of T2 can only be a duplicate row of just one row of T1. More than one row of T1 cannot have the same duplicate row in T2.

9. DISTINCT is implicitly assumed for the <query expression>s belonging to T1 and T2 if ALL is not specified. All duplicate rows are removed from R.

10. If parentheses are missing, then INTERSECT will be evaluated before UNION and EXCEPT. UNION and EXCEPT have the same precedence and will be evaluated from left to right in the case that parentheses are missing.

<query spec>, <named query spec>

Function

specifies an unordered result table.

Format

```
<query spec> ::=
    SELECT [<distinct spec>] <select column>,...
    <table expression>

<named query spec> ::=
    SELECT [<distinct spec>]
    <result table name> (<select column>,...)
    <table expression>

<distinct spec> ::=
    DISTINCT
  | ALL

<select column> ::=
    <table columns>
  | <derived column>
  | <rowno column>
  | <stamp column>

<table columns> ::=
    *
  | <table name>.*
  | <reference name>.*

<derived column> ::=
    <expression> [<result column name>]
  | <result column name> = <expression>

<rowno column> ::=
    ROWNO [<result column name>]
  | <result column name> = ROWNO

<stamp column> ::=
    STAMP [<result column name>]
  | <result column name> = STAMP


<result column name> ::=
    <identifier>
```

Syntax Rules

1. The specification of a column of the data type LONG in a <select column> is only valid in the uppermost sequence of <select column>s in a <query statement>, <single select statement>, <select direct statement> or <select ordered statement> if the <distinct spec> DISTINCT has not been used there.
   For restrictions to these options refer to the Precompiler online help, as well as to the

manuals of the other components.

2. The specification of a column of the data type LONG in a <select column> is only valid in the uppermost sequence of <select column>s in a <create view statement> which is based on exactly one base table.

3. If a <select column> contains a <set function spec>, the sequence of <select column>s to which the <select column> belongs must not contain any <table columns>, and every column name occurring in an <expression> must denote a grouping column, or the <expression> must consist of grouping columns.

4. A <rowno column> may only be specified in a <select column> which belongs to a <query statement>.

5. A <stamp column> may only be specified in a <select column> which belongs to a <query expression> of an <insert statement>.

General Rules

1. A <named query spec> corresponds almost entirely to a <query spec>. Therefore only the <query spec> is described in detail. Only if there is a significant difference between the <named query spec> and the <query spec>, the <named query spec> is described, too.

2. A <query spec> specifies a result table. The result table is generated from a temporary result table. The temporary result table is the result of the <table expression>.

3. If DISTINCT is specified as <distinct spec>, all duplicate rows are removed from the result table. If no <distinct spec> or if ALL is specified, duplicate rows are not removed. A row is a duplicate of another row if both have identical values in each column. NULL values are assumed to be identical. Special NULL values are assumed to be identcial.

4. The sequence of <select column>s defines the columns of the result table. The columns of the result table are produced from the columns of the temporary result table, completed by <rowno column>s or <stamp column>s, if any.
   The columns of the temporary result table are determined by the <from clause> of the <table expression>. The order of the column names of the temporary result table is determined by the order of the table names in the <from clause>.

5. The specification of <table columns> in a <select column> is an abbreviation of the specification of the result table columns.

6. If a <select column> of the format '*' is specified, this is an abbreviation of the specification of all temporary result table columns. In this case, the result table contains all columns of the temporary result table in an unmodified order.
   Columns for which the user has not the SELECT privilege and the implicitly generated column SYSKEY are not passed.

7. The specification of <table name>.* or <reference name>.* is an abbreviation of the specification of all columns of the underlying table. The first column name of the result table is taken from the first column name of the underlying table, the second column name of the result table corresponds to the second column name of the underlying table, etc. The order of the column names of the underlying table corresponds to the order determined when the underlying table is defined.
   Columns for which the user has not the SELECT privilege and the implicitly generated

column SYSKEY are not passed.

8. The specification of a <derived column> in a <select column> defines a column of the result table. If a column of the result table has the form '<expression> <result column name>' or the form '<result column name> = <expression>', then this result column gets the name <result column name>. If no <result column name> is specified and the <expression> is a <column spec> which denotes a column of the temporary result table, then the column of the result table gets the column name of the temporary result table. If no <result column name> is specified and the <expression> is no <column spec>, then the column gets the name 'EXPRESSION_', where '_' denotes a number with up to three digits, starting with 'EXPRESSION1', 'EXPRESSION2', etc.

9. If a <rowno column> is specified, a column of data type FIXED(10) is generated having the name ROWNO. It contains the values 1, 2, 3,... which represent a numbering of the result table rows. If the <rowno column> was specified either in the form 'ROWNO <result column name>' or in the form '<result column name> = ROWNO', then this result column is given the name <result column name>.
A <rowno column> must not be ordered by using ORDER BY.

10. ADABAS is able to generate unique values. These consist of the SERVERDB number and of consecutive numbers counted for each SERVERDB. The consecutive numbers begin with X'000000000001'. The values are generated in ascending order. It cannot be ensured that a sequence of values is uninterrupted.
The specification of a <stamp column> produces the next key generated by ADABAS for each row of the temporary result table. This key value is of the data type CHAR(8) BYTE.

11. Each column of a result table has exactly the same data type, the same length, the same precision, and the same scale as the <derived column> or the column underlying the <table columns>.
This does not apply to the data types DATE and TIMESTAMP. To enable the representation of any date and time format, the length of the result table column is set to the maximum length required for the representation of a date value (length 10) or a timestamp value (length 26).

12. Every column name specified in a <select column> must uniquely identify a column of one of the tables underlying the <query spec>. If need be, the column name must be qualified by the table identifier.

<table expression>

Function

specifies a simple or a grouped result table.

Format

```
 <table expression> ::=
     <from clause>
     [<where clause>]
     [<group clause>]
     [<having clause>]
```

Syntax Rules

1. The order of the <group clause> and <having clause> can be inverted.

General Rules

1. A <table expression> produces a temporary result table. If there are no optional clauses, this temporary result table is the result of the <from clause>. Otherwise, each specified clause is applied to the result of the previous clause and the table is the result of the last specified clause. The temporary result table contains all columns of all tables listed in the <from clause>.

See also

<from clause>

<where clause>

<group clause>

<having clause>

<from clause>

Function

specifies a table that is made up of one or more tables.

Format

```
<from clause> ::=
    FROM <table spec>,...

<table spec> ::=
    <table name> [<reference name>]
  | <result table name> [<reference name>]
  | (<query expression>) [<reference name>]
```

Syntax Rules

none

General Rules

1. Each <table spec> specifies a table identifier. A <table spec> that contains a <query expression> specifies a table identifier only if a <reference name> is specified.

2. If a <table spec> specifies no <reference name>, the <table name> or <result table name> is the table identifier. If a <table spec> specifies a <reference name>, the <reference name> is the table identifier.

3. Each <reference name> must differ from each <identifier> of each <table name> being a table identifier. If a <result table name> is a table identifier, there must not be any table identifier of the form <table name> equal to [<owner>.]<result table name>, where <owner> is the current user. Each table identifier must differ from any other table identifier.

4. The scope of validity of the table identifier is the entire <query spec>. If column names are to be qualified within the <query spec>, table identifiers must be used for this purpose.

5. The user must have the SELECT privilege for each specified table or for at least one column of the specified table.

6. The number of tables underlying a <from clause> is the sum of the tables underlying each <table spec>.
If a <table spec> denotes a base table, a snapshot table, a result table or the result of a <query expression>, the number of tables underlying this <table spec> is equal to 1.
If a <table spec> denotes a complex view table, the number of tables underlying this <table spec> is equal to 1.
If a <table spec> denotes a view table which is not a complex view table, the number of underlying tables is equal to the number of tables underlying the <from clause> of

the view table.
The number of tables underlying a <from clause> must not exceed 16.

7. The <from clause> specifies a table. This table can be derived from several base, view, snapshot, and result tables.

8. If a <table spec> contains a <query expression>, a result table matching this <query expression> is built. This result table gets a system-internal name which collides neither with an unnamed nor with a named result table. While the <from clause> is processed, the result of the <query expression> is used like a named result table; after the processing, it is implicitly deleted.

9. As a <table expression> which contains at least one <outer join indicator> specification may only have two underlying tables, it is necessary to use a <query expression> for the formulation of a <query spec> with at least three underlying tables and at least one <outer join indicator> in a <join predicate>.

10. The result of a <from clause> is a table which, in principle, is generated from the specified tables in the following way: If the <from clause> consists of a single <table spec>, the result is the specified table. If the <from clause> contains more than one <table spec>, a result table is built that includes all possible combinations of all rows of the first table with all rows of the second table, etc. Speaking in mathematical terms, the Cartesian product of all tables is formed. This rule describes the effect of the <from clause>, not its actual implementation.

11. <reference name>s are indispensable for the formulation of conditions to join a table to itself. For example, 'FROM HOTEL, HOTEL X' defines the <reference name> 'X' for the second occurrence of the table 'HOTEL'. Furthermore, <reference name>s are sometimes indispensable for the formulation of certain correlated subqueries. A <reference name> is also needed if a column of the <query expression> result can be only uniquely denoted by a <reference name> specification.

<where clause>

Function

specifies conditions for the result table.

Format

```
 <where clause> ::=
     WHERE <search condition>
```

Syntax Rules

1. An <expression> included in the <search condition> must not contain a <set function spec>.

General Rules

1. Each <column spec> directly contained in the <search condition> must uniquely denote a column from the tables specified in the <from clause> of the <table expression>. If necessary, the column name must be qualified with the table identifier. If <reference name>s were defined for table names in the <from clause>, these <reference name>s must be used as table identifiers in the <search condition>.

2. In the case of a correlated subquery (see chapter 0), a <column spec> can denote a column of a table which was specified in a <from clause> of another <table expression> of the <query spec>.

3. The <search condition> must only contain <column spec>s for which the user has the SELECT privilege.

4. The <search condition> is applied to every row of the temporary result table formed by the <from clause>. The result of the <where clause> is a table that only contains those rows of the result table for which the <search condition> is satisfied.

5. Usually, each <subquery> in the <search condition> is evaluated once. In the case of a correlated subquery, the <subquery> is executed for each row of the result table generated by the <from clause>.

<group clause>

Function

specifies a grouping for the result table.

Format

```
 <group clause> ::=
     GROUP BY <expression>,...
```

Syntax Rules

none

General Rules

1. Each column name specified in the <group clause> must uniquely denote a column of the tables underlying the <query spec>. If necessary, the column name must be qualified with the table identifier.

2. The <group clause> allows the functions SUM, AVG, MIN, MAX, COUNT, STDDEV, and VARIANCE to be applied not only to entire result tables but also to groups of rows within a result table. A group is defined by the grouping columns specified in GROUP BY. All rows of a group have the same values in the grouping columns. Rows containing the NULL value in a grouping column are combined to form a group. The same is true for the special NULL value.

3. GROUP BY generates one row for each group in the result table. Therefore, the <select column>s in the <query spec> may only contain those grouping columns and operations on grouping columns, as well as those <expression>s that use the functions SUM, AVG, MIN, MAX, COUNT, STDDEV, and VARIANCE.

4. If there is no row that satisfies the conditions indicated in the <where clause> and a <group clause> was specified, then the result table is empty.

<having clause>

Function

specifies the characteristics of a group.

Format

```
 <having clause> ::=
     HAVING <search condition>
```

Syntax Rules

none

General Rules

1. Each <expression> that is not specified in the argument of a <set function spec> but occurs in the <search condition> must denote a grouping column.

2. If the <having clause> is used without a preceding <group clause>, the result table built so far is regarded as a group.

3. The <search condition> is applied to each group of the result table. The result of the <having clause> is a table that only contains those groups for which the <search condition> is satisfied.

<subquery>

Function

specifies a result table that can be used in certain predicates and for the update of column values.

Format

```
<subquery> ::=
    (<query expression>)
```

Syntax Rules

1. A <subquery> used in a <set update clause> of an <update statement> must only form a single-column result table.

General Rules

1. The result of a <subquery> is a result table.
2. Subqueries can be used in certain predicates such as the <comparison predicate>, <exists predicate>, <in predicate>, and <quantified predicate>.
3. Subqueries can only be used in the <set update clause> of the <update statement>.

See also

Correlated Subquery

Correlated Subquery

Certain predicates can contain subqueries. These subqueries, in turn, can contain other subqueries, etc. A <subquery> containing subqueries is at a higher level than the subqueries included.
Within the <search condition> of a <subquery>, column names may occur that belong to tables contained in the <from clause> of higher-level subqueries. A <subquery> of this kind is called a correlated subquery. Tables that are used in subqueries in such a way are called correlated tables. No more than 16 correlated tables are allowed within an SQL statement. Columns that are used in subqueries in such a way are called correlated columns. Their number in an SQL statement is limited to 64.
If the qualifying table name or reference name does not clearly identify a table of a higher level, the table at the lowest level is taken from these non-unique tables.
If the column name is not qualified by the table name or reference name, the tables at higher levels are scanned for it. The column name must be unique in all tables of the <from clause> to which the table found belongs.
If a correlated subquery is used, the values of one or more columns of a temporary result row at a higher level are included in the <search condition> of a <subquery> at a lower level, whereby the result of the subquery is used for the definite qualification of the higher-level temporary result row.

Example:

We look at a table HOTEL which contains the column names NAME, CITY, HNO, and a table ROOM which contains the column names HNO and PRICE. For every city, the names of all hotels are searched which have prices less than the average price of the city concerned.

```
SELECT name, city
FROM   hotel X, room
WHERE  X.hno = room.hno
  AND  room.price < ( SELECT AVG(room.price)
                      FROM   hotel, room
                      WHERE  hotel.hno  = room.hno
                        AND  hotel.city = X.city )
```

<order clause>

Function

specifies a sorting sequence for a result table.

Format

```
<order clause> ::=
    ORDER BY <sort spec>,...

<sort spec> ::=
    <unsigned integer> [<sort option>]
  | <expression> [<sort option>]

<sort option> ::=
    ASC
  | DESC
```

Syntax Rules

1. The maximum number of <sort spec>s that form the sort criterion is 16.
2. If the <query expression> consists of more than one <query spec>, the specification of a <sort spec> is only allowed in the form <unsigned integer> [<sort option>].

General Rules

1. If a <query spec> is specified with DISTINCT, the total of the internal lengths of all sorting columns must not exceed 246 characters; otherwise, 250 characters.
2. Column names in the <sort spec>s must be columns of the tables specified in the <from clause> or denote a <result column name>.
3. If DISTINCT or a <set function spec> in a <select column> was used, the <sort spec> must denote a column of the result table.
4. A number n specified in the <sort spec> identifies the nth column in the result table. n must be less than or equal to the number of columns in the result table.
5. The specification of an <order clause> defines a sort for the result table.
6. The sort columns specified in the <order clause> determine the sequence of the sort criteria.
7. If ASC is specified, a sort is carried out putting the values in ascending order; if DESC is specified, in descending order. If no specification has been made, ASC is assumed.
8. Values are compared to each other according to the rules for the <comparison predicate> For sorting purposes, NULL values are greater than non-NULL values, and special NULL values are greater than non-NULL values but less than NULL values.

<update clause>

Function

specifies that a result table is to become updatable.

Format

```
 <update clause> ::=
     FOR UPDATE [OF <column name>,...]
```

Syntax Rules

none

General Rules

1. The specified column names must denote columns in the tables underlying the <query spec>. They need not occur in a <select column>.

2. The <query statement> containing the <update clause> must generate an updatable result table.

3. The <update clause> is prerequisite that the result table <result table name> can be used in an <update statement>, <delete statement>, <lock statement>, <select direct statement> or <select ordered statement> by means of CURRENT OF <result table name>. For other formats of the above mentioned SQL statements as well as in interactive mode, the <update clause> has no significance.

4. All columns of the underlying base tables are updatable if the user has the corresponding privileges, regardless of whether they were specified as <column name> or not.

5. For performance reasons, it is recommended to specify <column name>s only if the cursor is to be used in an <update statement>.

   f a column x is contained

   - in an index and
   - in the <search condition> of the <query statement> and
   - in a <set update clause> of the <update statement> in the form 'x = <expression>', where <expression> contains the column x,

   then it is strongly recommended to specify the column x as <column name> in the <update clause>.

   If at least one of these conditions is not satisfied, the column should not be specified.

<lock option>

Function

requests a lock for each selected row.

Format

```
 <lock option> ::=
     WITH LOCK <with lock info>

 <with lock info> ::=
     [(NOWAIT)] [EXCLUSIVE] [ISOLATION LEVEL <unsigned integer>]
   | [(NOWAIT)]  OPTIMISTIC [ISOLATION LEVEL <unsigned integer>]
```

Syntax Rules

1. <unsigned integer> may only assume the values 0, 1, 2, 3, 10, 15, 20 or 30.

General Rules

1. The <lock option> determines which locks are to be set on the read rows.

2. EXCLUSIVE defines an EXCLUSIVE lock. As long as the locked row has not been updated or deleted, the EXCLUSIVE lock can be cancelled using an <unlock statement>.

3. OPTIMISTIC defines an optimistic lock on rows. This lock makes only sense together with the ISOLATION LEVELs 0, 1, 10, and 15. An update operation of the current user on a row locked by this user using an optimistic lock is performed only if this row has not been updated in the meantime by a concurrent transaction. If this row has been changed in the meantime by a concurrent transaction, the update operation of the current user is rejected. The optimistic lock is released in both cases. If the update operation was successful, an EXCLUSIVE lock is set for this row. If the update operation was not successful, it should be repeated after reading the row again with or without optimistic lock. In this way, it can be ensured that the update is done to the current state and that no modifications are lost that have been made in the meantime. The request of an optimistic lock only collides with an EXCLUSIVE lock. Concurrent transactions do not collide with an optimistic lock.

4. Setting the locks is done irrespective of the <isolation spec> of the <connect statement>. The ISOLATION LEVEL of the <lock option> can denote a greater or smaller value than that of the <connect statement>. The <connect statement> rules apply for the different ISOLATION LEVELs.

5. The ISOLATION LEVEL specified by the <lock option> is only valid for the duration of the SQL statement which contains the <lock option> specification. Afterwards, the ISOLATION LEVEL which was specified in the <connect statement> is valid again.

6. If (NOWAIT) is specified, ADABAS does not wait for the release of a data object locked by another user, but it returns a message in the case that a collision occurs. If

no collision exists, the desired lock is set. If (NOWAIT) is not specified and a collision occurs, the release of the locked data object is waited for (but only as long as is specified by the installation parameter REQUEST TIMEOUT).

7. If neither EXCLUSIVE nor OPTIMISTIC is specified, a SHARE lock on rows is thus defined. If a SHARE lock was set on a row, no concurrent transaction can modify this row.

<open cursor statement>

Function

generates the result table previously defined with the specified name.

Format

```
 <open cursor statement> ::=
     OPEN <result table name>
```

Syntax Rules

none

General Rules

1. Existing result tables are implicitly deleted when a result table is generated with the same name.
2. All result tables which were generated within the current transaction are implicitly closed at the end of the transaction using the <rollback statement>.
3. All result tables are implicitly closed at the end of the session using the <release statement>. A <close statement> can be used to close them explicitly beforehand.
4. If the name of a result table is identical to that of a base table, view table, snapshot table or a synonym, these tables cannot be accessed during the existence of the result table.
5. At any given time during the processing of a result table, there is a position which may be before the first row, on a row, after the last row or between two rows. After generating the result table, this position is before the first row of the result table.
6. According to the search strategy, either all rows of the result table are searched when the <open cursor statement> is executed, the result table being physically generated; or each next result table row is searched when a <fetch statement> is executed, without being physically stored. This must be considered for the time behavior of <open cursor statement>s and <fetch statement>s.
7. If the result table is empty, the return code 100 - ROW NOT FOUND - is set.
8. The number of the result table rows is returned in the third entry of SQLERRD in the SQLCA (see the Precompiler online help). If this counter has the value -1, there is at least one result row.

<fetch statement>

Function

assigns the values of the current result table row to parameters.

Format

```
<fetch statement> ::=
    FETCH [<dir or position>] [<result table name>]
    INTO <parameter spec>,...

<dir or position> ::=
    <dir spec>
  | <position>
  | SAME

<dir spec> ::=
    FIRST
  | LAST
  | NEXT
  | PREV

<position> ::=
    POS (<unsigned integer>)
  | POS (<parameter spec>)
```

Syntax Rules

1. The <parameter spec> must denote a positive integer.

General Rules

1. If no result table name is specified, the <fetch statement> refers to the last unnamed result table that was generated.

2. Let C be the position in the result table. The return code 100 - ROW NOT FOUND - is output and no values are assigned to the parameters if any of the following conditions is satisfied:

   a) The result table is empty.

   b) C is positioned on or after the last result table row, and FETCH or FETCH NEXT is specified.

   c) C is positioned on or before the first row of the result table and FETCH PREV is specified.

   d) FETCH is specified with a <position> which does not lie within the result table.

3. If FETCH FIRST or FETCH LAST is specified and the result table is not empty, then C is positioned to the first or last row of the result table and the values of this row will be assigned to the parameters.

4. If FETCH or FETCH NEXT is specified and C is positioned before a row of the result table, then C will be located on this row and the values of this row will be assigned to the parameters.

5. If FETCH or FETCH NEXT is specified and C is positioned on a row which is not the last row of the result table, then C will be located on the next following row and the values in this row will be assigned to the parameters.

6. If FETCH PREV is specified and C is positioned after a row of the result table, then C will be located on this row and the values of this row will be assigned to the parameters.

7. If FETCH PREV is specified and C is positioned on a row which is not the first row of the result table, then C will be located on the preceding row and the values in this previous row will be assigned to the parameters.

8. Regardless of an <order clause> specification, there is an implicit order of the rows in a result table. This order enables an internal numbering which can be displayed with a <rowno column> specified as <select column>. <position> refers to this internal numbering.
   If a <position> less than or equal to the number of rows in the result table has been specified, then C will be positioned to the corresponding row and the values of this row will be assigned to the parameters. If a <position> greater than the number of rows in the result table has been specified, the return code 100 - ROW NOT FOUND - is output.
   If FOR REUSE has not been specified in the <query statement>, subsequent <insert statement>s, <update statement>s or <delete statement>s which refer to the underlying base table and which are issued by the current user or by another user may have the effect that a <fetch statement> issued repeatedly denotes different rows of the result table inspite of the same <position> specification.
   Other users can be prevented from modifying a table by issuing a <lock statement> for the whole table or by using the ISOLATION LEVEL 2, 3, 15, 20 or 30 for the <connect statement> or the <lock option> of the <query statement>.
   If this is not possible or if the user himself modifies the table, the specification FOR REUSE is necessary. Modifications made in the meantime are not visible then.

9. If FETCH SAME is specified, the last issued row of the result table is issued again.

10. The parameters specified by <parameter spec>s are output parameters. The parameter identified by the nth <parameter spec> corresponds to the nth value in the current result table row. If the number of columns in this row exceeds the number of specified parameters, the column values for which no corresponding parameters exist are ignored. If the number of columns in the row is less than the number of specified parameters, no values are assigned to the remaining parameters. An indicator parameter must be specified to assign NULL values or special NULL values.

11. Numbers are converted and character strings are truncated or lengthened, if necessary, to suit the corresponding parameters. If an error occurs when assigning a value to a parameter, the value is not assigned and no further values are assigned to the corresponding parameters for this <fetch statement>. Any values that have already been assigned to parameters remain unaffected.

12. Let p be a parameter and v the corresponding value in the current row of the result table. If v is a number, p must be a numeric parameter and v must lie within the permitted range of values for p. If v is a character string, p must be an alphanumeric

parameter.

13. According to the search strategy, either all rows of the result table are searched when the <open cursor statement>or <select statement> or the <named select statements> are executed, the result table being physically generated; or each next result table row is searched when a <fetch statement> is executed, without being physically stored. This must be considered for the time behavior of <fetch statement>s. Depending on the ISOLATION LEVEL selected, this can also be the reason for locking problems occurring with a FETCH, e.g., return code 500 - LOCK REQUEST TIMEOUT.

14. If a result table that was physically created contains LONG columns and if the ISOLATION LEVELs 0, 1, and 15 are used, then it is not sure that the contents of the LONG columns are consistent with the other columns. If the result table was not physically created, consistency is not ensured in ISOLATION LEVEL 0. For this reason, it is recommended to ensure consistency by using a <lock statement> or the ISOLATION LEVELs 2, 3, 20 or 30.

<close statement>

Function

closes a result table.

Format

```
 <close statement> ::=
     CLOSE [<result table name>]
```

Syntax Rules

none

General Rules

1. If the name of a result table is specified, this result table is closed. Its name can be used to denote another result table.

2. If no result table name is specified, an existing unnamed result table is closed, if any.

3. An unnamed result table is implicitly closed by the next <select statement>.

4. Result tables are implicitly closed when a result table with the same name is generated.

5. All result tables generated within the current transaction are implicitly closed at the end of the transaction using the <rollback statement>.

6. All result tables are implicitly closed at the end of the session using the <release statement>.

<single select statement>

Function

specifies a single-row result table and assigns the values of this result table to parameters.

Format

```
 <single select statement> ::=
     SELECT [<distinct spec>] <select column>,...
     INTO <parameter spec>,...
     FROM <table spec>,...
     [<where clause>]
     [<group clause>]
     [<having clause>]
     [<lock option>]
```

Syntax Rules

1. The order of the <group clause> and <having clause> can also be inverted.

General Rules

1. The specification of a column of the data type LONG in a <select column> is only valid in the uppermost sequence of <select column>s in a <single select statement> if the <distinct spec> DISTINCT was not used there.
   For restrictions to these options refer to the Precompiler online help as well as to the manuals of the other components.

2. The number of rows in the result table must not be greater than one. If the result table is empty or contains more than one row, corresponding messages or error codes are issued and no values are assigned to the parameters specified in the <parameter spec>s. For an empty result table, the return code 100 - ROW NOT FOUND - is set.

3. If the result table contains just one row, the values of this row are assigned to the corresponding parameters. The <fetch statement> rules apply for assigning the values to the parameters.

<select direct statement: searched>

Function

selects a table row. A specified key value is used for the selection.

Format

```
 <select direct statement: searched> ::=
     SELECT DIRECT <select column>,...
     INTO <parameter spec>,...
     FROM <table name>
     KEY <key spec>,...
     [<where clause>]
     [<lock option>]
```

Syntax Rules

1. The clause 'INTO <parameter spec>,...' may be omitted in interactive mode.

General Rules

1. The specification of a column of the data type LONG in a <select column> is only valid in the uppermost sequence of <select column>s in a <select direct statement: searched>.
   For restrictions to these options refer to the Precompiler online help, as well as to the manuals of the other components.

2. The user must have the SELECT privilege for the selected columns or for the entire table.

3. The <select direct statement: searched> is used to directly access a particular row of a table by specifying the key columns.
   For tables defined without key columns, there is the implicitly created column SYSKEY CHAR(8) BYTE which contains a key generated by ADABAS. The table column SYSKEY can therefore be used in the <select direct statement: searched> to access a specific table row.

4. If a row with the specified key values is found and the <search condition> for this row, if any, is satisfied, the corresponding column values are assigned to the parameters. The <fetch statement> rules apply for assigning the values to the parameters.

5. If there is no row with the specified key values, or if a row with the specified key values does exist but a <search condition> defined for this row is not satisfied, the return code 100 - ROW NOT FOUND - is issued and no values are assigned to the parameters specified in the <parameter spec>s.

<select direct statement: positioned>

Function

selects a table row. A cursor position is used for the selection.

Format

```
<select direct statement: positioned> ::=
    SELECT DIRECT <select column>,...
    INTO <parameter spec>,...
    FROM <table name>
    WHERE CURRENT OF <result table name>
    [<lock option>]
```

Syntax Rules

1. The clause 'INTO <parameter spec>,...' may be omitted in interactive mode.

2. The result table <result table name> must have been specified with FOR UPDATE.

General Rules

1. The specification of a column of the data type LONG in a <select column> is only valid in the uppermost sequence of <select column>s in a <select direct statement: positioned>.
   For restrictions to these options refer to the Precompiler online help, as well as to the manuals of the other components.

2. The <table name> of the <select direct statement: positioned> must be identical to the <table name> in the <from clause> of the <query statement> that generated the result table <result table name>.

3. If the cursor is positioned on a row of the result table, then column values are selected from the corresponding row and are assigned to parameters. The corresponding row is the row from the table which is specified in the <from clause> of the <query statement> and from which the row of the result table was formed. The <fetch statement> rules apply for assigning the values to the parameters.

4. If the cursor is not positioned on a row of the result table, an error message is issued and no values are assigned to the parameters.

<select ordered statement: searched>

Function

selects the first or last row, or, in relation to a position, the next or previous row in an ordered table. The order is defined by a key or by an index. The position is defined by the specification of key values and index values.

Format

```
<select ordered statement: searched> ::=
    <select ordered format1: searched>
  | <select ordered format2: searched>

<select ordered format1: searched> ::=
    SELECT <dir1 spec> <select column>,...
    INTO <parameter spec>,...
    FROM <table name>
    [<pos1 spec>]
    [<where clause>]
    [<lock option>]

<select ordered format2: searched> ::=
    SELECT <dir2 spec> <select column>,...
    INTO <parameter spec>,...
    FROM <table name>
    <pos2 spec>
    [<where clause>]
    [<lock option>]

<dir1 spec> ::=
    FIRST
  | LAST

<dir2 spec> ::=
    NEXT
  | PREV

<pos1 spec> ::=
    <index name spec>
  | <index pos spec> [KEY <key spec>,...]
  | KEY <key spec>,...


<pos2 spec> ::=
    [<index pos spec>] KEY <key spec>,...

<index name spec> ::=
    INDEX <column name>
  | INDEXNAME <index name>

<index pos spec> ::=
    INDEX <column name> = <value spec>
  | INDEXNAME <index name> VALUES (<value spec>,...)
```

Syntax Rules

1. The clause 'INTO <parameter spec>,...' may be omitted in interactive mode.

General Rules

1. The specification of a column of the data type LONG in a <select column> is only valid in the uppermost sequence of <select column>s in a <select ordered statement: searched>.
   For restrictions to these options refer to the Precompiler online help, as well as to the manuals of the other components.

2. The <column name> in the <index name spec> and in the <index pos spec> must denote an indexed column.

3. The user must have the SELECT privilege for the selected columns or for the entire table.

4. The <select ordered statement: searched> cannot be used for view tables which have been defined by SELECT DISTINCT or which have more than one underlying base table.

5. The <select ordered statement: searched> is used to access the first or last row of an order defined by the key or a secondary key, or to access the previous or next row starting at a specified position. For tables defined without key columns, there is the implicitly generated column SYSKEY CHAR(8) BYTE which contains a key generated by ADABAS. The table column SYSKEY can therefore be used in the <select ordered statement: searched> for positional access to a specific table row. In a table stored without replication, the order defined by the ascending values of SYSKEY corresponds to the order of insertions made to the table.

6. If no <index name spec> and no <index pos spec> is specified, the order is defined by the key. If an <index name spec> or an <index pos spec> is specified, then the order is defined by the secondary key and by the key. The ascending key order is then the second sort criterion. The position within the table can be explicitly specified by using the <index pos spec> and the <key spec>s. There is no need for any table row to contain the position values.

7. FIRST (LAST) produces a search for the first (last) row in the ordered table which satisfies the specified WHERE clause and which, in relation to the order, is greater (less) than or equal to the position.

8. NEXT (PREV) produces a search in ascending (descending) order for the next row which satisfies the specified WHERE clause, starting at the specified position. If no WHERE clause is specified, the result is the row which is next according to order and position.

9. If an <index name spec> or an <index pos spec> is specified and the corresponding index is a single-column index, the rows which contain NULL values in the indexed column are not taken into account for the <select ordered statement: searched>. In such a case, the result of the <select ordered statement: searched> can, by no means, be a row having a NULL value in the indexed column. A warning indicates this state.

10. If a row was found that satisfies the specified conditions, then the corresponding column values are assigned to the parameters. The <fetch statement> rules apply for

assigning the values to the parameters.

11. If the specified table does not contain a row that satisfies the specified conditions, the return code 100 - ROW NOT FOUND - is issued and no values are assigned to the parameters specified in the <parameter spec>s.

<select ordered statement: positioned>

Function

selects the first or last row, or, in relation to a position, the next or previous row in an ordered table. The order is defined by a key or by an index. The position is defined by a cursor position.

Format

```
<select ordered statement: positioned> ::=
    <select ordered format1: positioned>
  | <select ordered format2: positioned>

<select ordered format1: positioned> ::=
    SELECT <dir1 spec> <select column>,...
          INTO <parameter spec>,...
          FROM <table name>
          [<index name spec>]
          WHERE CURRENT OF <result table name>
          [<lock option>]
  | SELECT <dir1 spec> <select column>,...
          INTO <parameter spec>,....
          FROM <table name>
          [<index pos spec>]
          WHERE CURRENT OF <result table name>
          [<lock option>]

<select ordered format2: positioned> ::=
    SELECT <dir2 spec> <select column>,...
    INTO <parameter spec>,...
    FROM <table name>
    [<index pos spec>]
    WHERE CURRENT OF <result table name>
    [<lock option>]
```

Syntax Rules

1. The clause 'INTO <parameter spec>,...' may be omitted in interactive mode.

2. The result table <result table name> must have been specified with FOR UPDATE.

General Rules

1. The specification of a column of the data type LONG in a <select column> is only valid in the uppermost sequence of <select column>s in a <select direct statement: positioned>.
   For restrictions to these options refer to the Precompiler online help, as well as to the manuals of the other components.

2. The <column name> in the <index name spec> and in the <index pos spec> must denote an indexed column.

3. The user must have the SELECT privilege for the selected columns or for the entire table.

4. The <table name> of the <select direct statement: positioned> must be identical to the <table name> in the <from clause> of the <query statement> that generated the result table <result table name>.

5. The <select ordered statement: positioned> is used to access the first or last row of an order defined by the key or a secondary key, or to access the previous or next row starting at a specified position.

6. If no <index name spec> and no <index pos spec> is specified, the order is defined by the key. If an <index name spec> or an <index pos spec> is specified, then the order is defined by the secondary key and by the key. The ascending key order then is the second sort criterion. The position within the table is defined by the optional <index pos spec> and by a key value, whereby the key value is determined by the cursor position.

7. FIRST (LAST) produces a search for the first (last) row which, in relation to the order, is greater (less) than or equal to the position.

8. NEXT (PREV) produces a search in ascending (descending) order for the next row, starting at the specified position.

9. If an <index name spec> or an <index pos spec> is specified and the corresponding index is a single-column index, the rows which contain NULL values in the indexed column are not taken into account for the <select ordered statement: positioned>. In such a case, the result of the <select ordered statement: positioned> can, by no means, be a row having a NULL value in the indexed column.

10. If the cursor is positioned on a row of the result table and a row was found which satisfies the specified conditions, then the corresponding column values are assigned to the parameters. The <fetch statement> rules apply for assigning the values to the parameters.

11. If the cursor is not positioned on a row of the result table, then an error message is issued and no values are assigned to the parameters.

<explain statement>

Function

describes the search strategy applicable for a <query statement> or <single select statement>.

Format

```
 <explain statement> ::=
     EXPLAIN [(<result table name>)] <query statement>
   | EXPLAIN [(<result table name>)] <single select statement>
```

Syntax Rules

none

General Rules

1. A <query statement> or <single select statement> involves a search for particular rows of specified tables. The <explain statement> describes the internal search strategy used by ADABAS. This statement indicates in particular whether and in which form key columns or indexes are used for the search. The <explain statement> can be used to check which effects the creation or deletion of indexes will have for the selection of the search strategy for the specified SQL statement. It is also possible to estimate the time which ADABAS needs to process the specified SQL statement. The specified <query statement> or <single select statement> is not performed during the execution of the <explain statement>.

2. A result table is generated. It may be named. If the optional name specification is missing, the result table is given the name SHOW. The result table has the following structure:

```
 OWNER             CHAR(18)
 TABLENAME         CHAR(18)
 COLUMN_OR_INDEX   CHAR(18)
 STRATEGY          CHAR(40)
 PAGECOUNT         CHAR(10)
 O                 CHAR( 1)
 D                 CHAR( 1)
 T                 CHAR( 1)
 M                 CHAR( 1)
```

3. The sequence in which the SELECT is processed is described by the order of the rows in the result table.

4. The column 'STRATEGY' shows which search strategy(ies) is/are used and whether a result table is generated. A result table is physically generated if the column 'STRATEGY' contains 'RESULT IS COPIED' in the last result row.
The column 'COLUMN_OR_INDEX' shows which key column or indexed column or

which index is utilized for the strategy.
The column 'PAGECOUNT' shows which sizes are assumed for the tables or, in the case of certain strategies, for the indexes. These sizes influence the choice of the search strategy.
The assumed sizes are updated using the <update statistics statement> and can be requested by selecting the system table OPTIMIZERSTATISTICS. The current sizes of tables or indexes can be checked by selecting the system tables TABLESTATISTICS and INDEXSTATISTICS.If there are greater differences between the values contained in OPTIMIZERSTATISTICS and TABLESTATISTICS, the <update statistics statement> should be performed for this table.
The <update statistics statement> is implicitly performed for a table when during a search in this table the system finds out that the values determined by the last <update statistics statement> are much too small.

The last row contains the estimated SELECT cost value in the column 'PAGECOUNT'. The COSTLIMIT and COSTWARNING specifications in the <create user statement>, <create usergroup statement>, <alter user statement>, and <alter usergroup statement> refer to this estimated SELECT cost value.
The columns 'O', 'D', 'T', and 'M' serve support purposes and are therefore not explained.

5. For a more detailed description of the possible search strategies refer to the Precompiler online help.

Transactions

A transaction is a sequence of <sql statement>s that are handled by ADABAS as an atomic unit, in the sense that any modifications made to the database by the <sql statement>s are either all reflected in the state of the database, or else none of the database modifications are retained.
When a session is opened using the <connect statement>, this opens the first transaction. A <commit statement> or a <rollback statement> is used to conclude a transaction. When a transaction is successfully concluded using a <commit statement>, all database modifications are retained. When, on the other hand, a transaction is aborted using a <rollback statement>, or if it is aborted in another way, all database modifications performed within the given transaction are rolled back.
The <commit statement> and the <rollback statement> both implicitly open a new transaction.
Since ADABAS permits concurrent transactions on the same database objects, locks on rows, tables and the catalog are necessary to isolate individual transactions. Locks are either implicitly set by ADABAS in the course of processing an <sql statement> or explicitly set using the <lock statement>. These locks are assigned to the transaction that contains the <sql statement> or <lock statement>. ADABAS distinguishes between SHARE locks and EXCLUSIVE locks which either refer to rows or tables and optimistic row locks. In addition, there are special locks for the metadata of the catalog. These locks, however, are always set implicitly.
Once a SHARE lock is assigned to a transaction for a particular data object, other transactions can access the object but not modify it.
Once an EXCLUSIVE lock is assigned to a transaction for a particular data object, other transactions cannot modify this object. The object can only be accessed by transactions which do not use SHARE locks (see ISOLATION LEVEL 0).
EXCLUSIVE locks for rows which have not yet been modified and SHARE locks on rows can be released by the <unlock statement> before the end of the transaction.
The locks assigned to a transaction are usually released at the end of the transaction, making the respective database objects accessible again to other transactions.
The SQL statements SUBTRANS BEGIN, SUBTRANS END and SUBTRANS ROLLBACK subdivide a transaction into additional atomic units. These can be nested as often as necessary and in whatever form is necessary. Unlike transactions, however, modifications performed by subtransactions can be undone by a <rollback statement> or the SUBTRANS ROLLBACK of an enclosing subtransaction, even once the subtransaction has been closed with SUBTRANS END.

The following table gives a schematic overview on the possible parallel locks. EXCL means EXCLUSIVE.

Let a transaction have a(n)

Can another transaction	EXCL lock on a table	SHARE lock on a table	EXCL lock on a row	SHARE lock on a row	EXCL lock on the system catalog	SHARE lock on the system catalog
lock the table in EXCLUSIVE mode?	No	No	No	No	No	Yes
lock the table in SHARE mode?	No	Yes	No	Yes	No	Yes
lock any row of the table in EXCLUSIVE mode?	No	No	---	---	No	Yes
lock the locked row in EXCLUSIVE mode?	---	---	No	No	---	---
lock another row in EXCLUSIVE mode?	---	---	Yes	Yes	---	---
lock any row of the table in SHARE mode?	No	Yes	---	---	No	Yes
lock the locked row in SHARE mode?	---	---	No	Yes	---	---
lock another row in SHARE mode?	---	---	Yes	Yes	---	---
change the definition of the table in the system catalog?	No	No	No	No	No	No
read the definition of the table in the system catalog?	Yes	Yes	Yes	Yes	No	Yes

<connect statement>

Function

opens an ADABAS session and a transaction for a user.

Format

```
<connect statement> ::=
    CONNECT <user spec>
    IDENTIFIED BY <password spec>
    [SQLMODE <sqlmode spec>]
    [<isolation spec>]
    [TIMEOUT <unsigned integer>]
    [CACHELIMIT <unsigned integer>]
    [TERMCHAR SET <termchar set name>]

<user spec> ::=
    <parameter name>
  | <user name>

<password spec> ::=
    <parameter name>

<sqlmode spec> ::=
    ADABAS
  | ANSI
  | DB2
  | ORACLE

<isolation spec> ::=
    ISOLATION LEVEL <unsigned integer>
```

Syntax Rules

1. The <unsigned integer> after ISOLATION LEVEL may only have the values 0, 1, 2, 3, 10, 15, 20 and 30.

General Rules

1. If a valid combination of the <user spec> and <password spec> values is specified, the user opens a session, obtaining access to the database. Thus he is the current user in this session.

2. The database system ADABAS is able to execute correct ADABAS applications and applications which are written according to the ANSI standard (ANSI X3.135-1992, Entry SQL), according to the definition of DB2 Version 3 or according to the definition of ORACLE7. ADABAS is able to check whether new programs comply with one of the definitions specified above. This means in particular that any extension beyond the chosen definition is considered incorrect. The support of DDL statements in other SQLMODEs is, however, limited.
   The specification SQLMODE <sqlmode spec> allows the user to select one of the definitions specified above. The default specification is SQLMODE ADABAS.

This online help describes the functionality of the database system ADABAS which is available in the SQLMODE ADABAS.

3. A transaction is implicitly opened.

4. The <commit statement> or the <rollback statement> ends a transaction, implicitly opening a new one. At the end of each transaction, all locks assigned to the transaction are released, providing they are not maintained by a KEEP LOCK. The <isolation spec> specified in the <connect statement> is applied to each newly opened transaction.

5. Locks can be requested implicitly or explicitly. Locks are requested explicitly using the <lock statement>. Whether a lock must be requested implicitly or explicitly depends on the <isolation spec> in the <connect statement>. How long an implicit SHARE lock is maintained also depends on the <isolation spec>. Implicitly set EXCLUSIVE locks cannot be released within a transaction. Explicit lock requests are always possible, regardless of the <isolation spec>.

6. ISOLATION LEVEL 0 means that rows can be read without requesting SHARE locks; i.e., no SHARE locks are implicitly requested. For this reason, there is no guarantee that a given row will still be in the same state when it is read again within the same transaction as when it was accessed earlier, since it may have been modified in the meantime by a concurrent transaction.
   Furthermore, there is no guarantee that the state of a read row has already been recorded in the database using COMMIT WORK.
   When rows are inserted, updated or deleted, implicit EXCLUSIVE locks are assigned to the transaction for the rows concerned. These cannot be released until the end of the transaction.

7. ISOLATION LEVEL 1 or 10 means that a SHARE lock is assigned to the transaction for each read row R1 in a table. When the next row R2 in the same table is read, the lock on R1 is released and a SHARE lock is assigned to the transaction for the row R2. For data retrieval by using a <query statement>, ADABAS makes sure that, at the time each row is read, no EXCLUSIVE lock has been assigned to other transactions for the given row. It is, however, impossible to predict whether a <query statement> causes a SHARE lock for a row of the specified table or not and for which row this may occur.
   When rows are inserted, updated or deleted, implicit EXCLUSIVE locks are assigned to the transaction for the rows concerned. These cannot be released until the end of the transaction.

8. For all <sql statement>s which read exactly one table row using the key, ISOLATION LEVEL 15 is equivalent to ISOLATION LEVEL 1 or 10.
   For all other <sql statement>s, the behavior at ISOLATION LEVEL 15 is the same as that described for ISOLATION LEVEL 1, the one difference being that all the tables addressed by the <sql statement> are locked in SHARE mode prior to processing. When the <sql statement> generates a result table which is not physically stored, these locks are not released until the end of the transaction or until the result table is closed. Otherwise, these locks are released immediately once the <sql statement> has been processed.
   When rows are inserted, updated or deleted, ADABAS assigns implicit EXCLUSIVE locks to the transaction for the relevant rows. These EXCLUSIVE locks cannot be released until the end of the transaction.

9. ISOLATION LEVEL 2 or 20 means that all the tables addressed by the <sql statement> are locked in SHARE mode prior to processing. When the <sql statement>

generates a result table which is not physically stored, these locks are only released at the end of the transaction or when the result table is closed. Otherwise, these locks are released immediately once the related <sql statement> has been processed.
In addition, an implicit SHARE lock is assigned to the transaction for each row read during the processing of an <sql statement>. These SHARE locks can only be released by using the <unlock statement> or by ending the transaction.
When rows are inserted, updated or deleted, implicit EXCLUSIVE locks are assigned to the transaction for the rows concerned. These cannot be released until the end of the transaction.

10. ISOLATION LEVEL 3 or 30 means that an implicit table SHARE lock is assigned to the transaction for each table addressed by an <sql statement>. These table SHARE locks cannot be released until the end of the transaction.
When rows are inserted, updated or deleted, implicit EXCLUSIVE locks are assigned to the transaction for the rows concerned. These cannot be released until the end of the transaction.

11. If the <isolation spec> is omitted, ISOLATION LEVEL 1 is assumed.

12. Which <isolation spec> is selected affects both the degree of concurrency and the guaranteed consistency. A high degree of concurrency is characterized by a state in which a maximum number of concurrent transactions can process a database without long waiting periods for locks to be released. As for consistency considerations, there are three different phenomena to be considered, which can arise through concurrent access to the same database:

Phenomenon 1 :
A row is modified in the course of a transaction T1, and a transaction T2 reads this row before T1 has been concluded with a <commit statement>. T1 then executes the <rollback statement>; i.e., T2 has read a row, which never actually existed. This phenomenon is known as the "dirty read" phenomenon.

Phenomenon 2 :
A transaction T1 reads a row. A transaction T2 then modifies or deletes this row, concluding with the <commit statement>. If T1 subsequently reads the row again, T1 either receives the modified row or a message saying that the row no longer exists. This phenomenon is known as the "non-repeatable read" phenomenon.

Phenomenon 3 :
A transaction T1 executes an <sql statement> S, which reads a set of rows SR which satisfies a <search condition>. A transaction T2 then uses the <insert statement> or the <update statement> to create at least one additional row which also satisfies the <search condition>. If S is subsequently re-executed within T1, the set of read rows will differ from SR. This phenomenon is known as the "phantom" phenomenon.

The following table specifies which phenomena are possible for which <isolation spec>s :

	ISO 0	ISO 1	ISO 2	ISO 3
Dirty Read	+	-	-	-
Non Repeatable Read	+	+	-	-
Phantom	+	+	+	-

The lower the value of the <isolation spec>, the higher the degree of concurrency and the lower the guaranteed consistency. This makes it always necessary to find the compromise between concurrency and consistency that best suits the requirements of an application.

13. The TIMEOUT value defines the maximum period of inactivity during an ADABAS session. A period of inactivity is considered to be the time interval between the completion of one <sql statement> and the issuing of the next <sql statement>. As soon as the specified maximum TIMEOUT is exceeded, the session is implicitly aborted by using a ROLLBACK WORK RELEASE.

14. TIMEOUT values are specified in seconds. A TIMEOUT value can be specified for every user. The specified TIMEOUT value must be less than or equal to the defined maximum TIMEOUT value.

    a) For any user who was created with a TIMEOUT value, this value is the maximum TIMEOUT value.

    b) For any user who is a member of a usergroup created with a TIMEOUT value, this value is the maximum TIMEOUT value.

    c) For all other users, the installation parameter SESSION TIMEOUT represents the maximum TIMEOUT value.

15. If no TIMEOUT value is specified, ADABAS assumes the maximum TIMEOUT value or the SESSION TIMEOUT value, depending on which is smaller. The value of the SESSION TIMEOUT is defined during the installation of ADABAS by using the ADABAS component CONTROL.

16. If 0 is specified as the TIMEOUT value, no check is made for the period of inactivity, the result being that database resources might not be available again, although the corresponding application has finished already, possibly by an abnormal termination; without performing a <release statement>.

17. Users defined with the attribute NOT EXCLUSIVE can open several sessions at the same time. Whenever this is the case, or whenever two users of the same usergroup open a session at the same time, the sessions are considered to be distinct. This means that lock requests of the sessions concerned can collide.

18. The CACHELIMIT value is specified in 4KB units. A CACHELIMIT value can be specified for each user. The specified CACHELIMIT value must be less than or equal to the value of the defined maximum CACHELIMIT value.

    a) For any user created with a CACHELIMIT value, this value is the maximum CACHELIMIT value.

    b) For any user who is a member of a usergroup created with a CACHELIMIT value, this value is the maximum CACHELIMIT value.

    c) For all other users, the maximum CACHELIMIT value is predefined by the installation parameter MAX_TEMP_CACHE (see the CONTROL online help).

    When sessions are started involving the physical creation of large result tables or large temporary base tables, it is a good idea to create a session-specific cache, so that these temporary, session-specific result tables will not take up the data cache space concurrently used by all users.

19. ADABAS uses either the ASCII code according to ISO 8859/1.2 or the EBCDIC code CCSID 500, Codepage 500. Since these codes include characters that have a different hexadecimal representation on certain terminals, it is possible to define TERMCHAR SETs (see the CONTROL online help). For input and output, these TERMCHAR SETs enable the conversion between the terminal representation of characters and the code used within ADABAS. The <connect statement> can be used to select one of the defined TERMCHAR SETs which is then used for conversion during the session. If no or an unsuitable TERMCHAR SET is selected, it can happen that characters which are contained in the database and which are to be output are not correctly displayed on the terminal.

20. For more detailed information about the call parameters or mechanisms for the assignment of parameter values, refer to the online help on the precompilers, as well as to the manuals of the other components.

<commit statement>

Function

closes the current transaction and starts a new one.

Format

```
 <commit statement> ::=
     COMMIT [WORK] [KEEP <lock statement>]
```

Syntax Rules

1. The <lock statement> must not specify a <wait option>.

General Rules

1. The <commit statement> closes the current transaction. This means that the modifications executed within the transaction are recorded, making them visible to concurrent users as well.
   The <commit statement> implicitly opens a new transaction. Any locks set, either implicitly or explicitly, within this new transaction are assigned to this transaction.

2. If the <lock statement> is omitted, any locks assigned to the transaction are released.

3. If a <lock statement> is specified, the locks specified in it are kept beyond the end of the transaction and then assigned to the implicitly opened new transaction - provided, however, that the locks specified in the <lock statement> are assigned to the transaction being ended. Any locks assigned to the transaction being ended that are not specified in the <lock statement> are released.

4. The <isolation spec> declared in the <connect statement> controls the setting of locks in the new transaction.

<rollback statement>

Function

aborts the current transaction and starts a new one.

Format

```
 <rollback statement> ::=
     ROLLBACK [WORK] [KEEP <lock statement>]
```

Syntax Rules

1. The <lock statement> must not specify a <wait option>.

General Rules

1. The <rollback statement> aborts the current transaction. This means that any database modifications performed within the transaction are undone.
   The <rollback statement> implicitly opens a new transaction. Any locks set, either implicitly or explicitly, within the new transaction are assigned to this transaction.

2. If the <lock statement> is omitted, the locks assigned to the transaction are released.

3. If a <lock statement> is specified, the locks specified in it are maintained beyond the end of the transaction and then assigned to the implicitly opened new transaction - provided that the locks specified in the <lock statement> are assigned to the transaction being ended. Any locks assigned to the transaction being ended that are not specified in the <lock statement> are released.

4. All result tables generated within the current transaction are implicitly closed when the related transaction is ended using the <rollback statement>.

5. The <isolation spec> declared in the <connect statement> controls the setting of locks in the new transaction.

<subtrans statement>

Function

subdivides a transaction into subunits.

Format

```
 <subtrans statement> ::=
     SUBTRANS BEGIN
   | SUBTRANS END
   | SUBTRANS ROLLBACK
```

Syntax Rules

none

General Rules

1. SUBTRANS BEGIN opens a subtransaction; i.e., ADABAS records the present point in the transaction. This can be followed by any sequence of <sql statement>s. If this sequence does not contain an additional SUBTRANS BEGIN, then all database modifications performed since the SUBTRANS BEGIN can be undone using a SUBTRANS ROLLBACK.
   The sequence can, however, also contain additional SUBTRANS BEGIN statements which open additional subtransactions. This means several nested subtransactions may be open at the same time.

2. SUBTRANS END closes a subtransaction; i.e., ADABAS forgets the savepoint within the transaction defined in SUBTRANS BEGIN - provided that an open subtransaction exists. If more than one open subtransaction exists, the last opened subtransaction is closed; i.e., it is no longer considered to be an open subtransaction.

3. SUBTRANS ROLLBACK undoes all database modifications performed within a subtransaction and then closes the subtransaction. Any database modifications performed by any subtransactions within the subtransaction are undone, regardless of whether they were ended with SUBTRANS END or SUBTRANS ROLLBACK. All result tables generated within the subtransaction are closed.
   The condition here is that an open subtransaction exists. If more than one open subtransaction exists, the last opened subtransaction is rolled back. The subtransaction concerned is then no longer considered open.

4. The <subtrans statement> does not affect locks assigned to the transaction. In particular, SUBTRANS END and SUBTRANS ROLLBACK do not release any locks.

5. The <subtrans statement> is particularly useful in keeping the effects of subroutines or DB procedures atomic; i.e., it ensures that they either fulfil all their tasks or else have no effect. To achieve this, first of all, a SUBTRANS BEGIN is issued. If the subroutine succeeds in fulfilling its task, it is ended with a SUBTRANS END; in the event of an error, a SUBTRANS ROLLBACK is used to undo all modifications performed by the subroutine.

6. The <commit statement> and the <rollback statement> implicitly close any subtransactions still open.

<lock statement>

Function

assigns a lock to the current transaction.

Format

```
<lock statement> ::=
    LOCK [<wait option>] <lock spec> IN SHARE MODE
  | LOCK [<wait option>] <lock spec> IN EXCLUSIVE MODE
  | LOCK [<wait option>] <lock spec> IN SHARE MODE
                         <lock spec> IN EXCLUSIVE MODE
  | LOCK [<wait option>] <row lock spec> OPTIMISTIC

<wait option> ::=
    (WAIT)
  | (NOWAIT)

<lock spec> ::=
    <table lock spec>
  | <row lock spec>
  | <table lock spec> <row lock spec>

<table lock spec> ::=
    TABLE <table name>,...

<row lock spec> ::=
    <row spec>...

<row spec> ::=
    ROW <table name> KEY <key spec>,...
  | ROW <table name> CURRENT OF <result table name>
```

Syntax Rules

1. For tables defined without key columns, the implicit key column SYSKEY CHAR(8) BYTE can be used in a <key spec>.

2. If CURRENT OF <result table name> is specified, the result table <result table name> must have been specified with FOR UPDATE.

General Rules

1. The specified table <table name> can be a nontemporary base table, view table, snapshot table or a synonym. If <table name> identifies a view table, then locks are set on the base tables on which the view table is based. To set SHARE locks, the current user must have the SELECT privilege; to set EXCLUSIVE locks, the user needs the UPDATE, DELETE or INSERT privilege.

2. The specification of a <row spec> requires that the table identified by <table name>

has a key column; i.e., if <table name> identifies a view table, this must be updatable.

3. If the view table identified by <table name> is not updatable, then only a SHARE lock can be set for this view table. As a result of this SQL statement, all base tables underlying the <table name> are subsequently locked in SHARE mode.

4. If <table name> identifies a snapshot table, only a SHARE lock can be set for this table.

5. A <table lock spec> specifies a lock for the given table. A <row lock spec> specifies a lock for the table row denoted by the key values or a position in a result table.

6. SHARE defines a SHARE lock for the listed objects. If a SHARE lock is set, no concurrent transaction can modify the locked objects.

7. EXCLUSIVE defines an EXCLUSIVE lock for the listed objects. If an EXCLUSIVE lock is set, no concurrent transaction can modify the locked objects. Concurrent transactions can only read-access the locked objects in ISOLATION LEVEL 0. EXCLUSIVE locks for rows which have not yet been modified can be released using the <unlock statement> before the end of the transaction.

8. OPTIMISTIC defines an optimistic lock on rows. This lock only makes sense when it is used together with the ISOLATION LEVELs 0, 1, 10, and 15. An update operation of the current user on a row which has been locked by this user using an optimistic lock is only performed if this row has not been updated in the meantime by a concurrent transaction. If this row has been changed in the meantime by a concurrent transaction, the update operation of the current user is rejected. The optimistic lock is released in both cases. If the update operation was successful, an EXCLUSIVE lock is set for this row. If the update operation was not successful, it should be repeated after reading the row again with or without optimistic lock. In ISOLATION LEVEL 0, an explicit lock must be specified for the new read operation. In this way, it can be ensured that the update is done to the current state and that no modifications made in the meantime are lost. The request of an optimistic lock only collides with an EXCLUSIVE lock. Concurrent transactions do not collide with an optimistic lock.

9. If no lock has been assigned to a transaction for a data object, then a SHARE or EXCLUSIVE lock can be requested within any transaction and the lock is immediately assigned to the transaction.
   If a SHARE lock has been assigned to a transaction T for a data object, and if no lock has been assigned to any concurrent transaction for this data object, then the transaction T can request an EXCLUSIVE lock for this data object and the lock is immediately assigned to this transaction.
   If an EXCLUSIVE lock has been assigned to a transaction for a data object, then a SHARE lock can, but need not, be requested for this transaction.
   The matrix 'Transactions' at the beginning of this chapter shows the possible parallel locks.
   A lock collision exists in the cases which are marked with 'No'; i.e., after having requested a lock within a transaction, the user has to wait for the lock to be released until one of the above situations or one of the situations that are marked with 'Yes' in the matrix occurs.
10. Locks can be requested either implicitly or explicitly. Explicit lock requests are performed using the <lock statement>. Whether a lock is requested implicitly and how long it remains assigned to the transaction depends on the <isolation spec> in the <connect statement>.
    SHARE locks and EXCLUSIVE locks set to single table rows which have not yet been updated can be released within a transaction. EXCLUSIVE locks on updated table

rows or table locks cannot be released within a transaction.

11. The locks assigned to a transaction by a <lock statement> are normally released once this transaction is ended, provided that the <commit statement> or <rollback statement> ending the transaction does not contain a <lock statement>.

12. If the <wait option> (NOWAIT) is specified, ADABAS does not wait for a lock to be released by another transaction, but issues an error message if there is a lock collision. If there is no collision, the requested lock is set.

13. In the event of a lock collision, if either the <wait option> is omitted or (WAIT) is specified, the system waits for locks to be released, until the period specified by the installation parameter REQUEST TIMEOUT has elapsed.
    If ADABAS has to wait too long for locks to be released when setting explicit or implicit locks, it issues a return code to this effect. The user can then respond to this return code, e.g., by terminating the transaction. In these situations, ADABAS does not execute an implicit ROLLBACK WORK.
    Whenever ADABAS recognizes a deadlock caused by explicit or implicit locks, it ends the transaction with an implicit ROLLBACK WORK.

14. If reproducible results are needed for reading rows using a <select statement>, the read objects must be locked and the locks must be kept until reproduction. Reproducibility usually requires that the tables concerned are locked in SHARE mode, either explicitly using one or more <lock statement>s or implicitly by using the ISOLATION LEVEL 3. This ensures that no other user can modify the table. To ensure the reproducibility of the SQL statement SELECT DIRECT, it suffices to implicitly or explicitly lock the row to be read in SHARE mode.

15. The fewer objects are locked, the more transactions can operate simultaneously on the database without colliding with lock requests of other transactions. For this reason, unnecessary locks should be avoided and set locks should be released as soon as possible.

16. If a transaction explicitly or implicitly requests too many row locks (SHARE or EXCLUSIVE locks) on a table, ADABAS tries to obtain a table lock instead. If this causes collisions with other locks, ADABAS continues to request row locks. This means that table locks can be obtained without waiting for them. The limit beyond which ADABAS tries to transform row locks into table locks depends on the installation parameter MAXLOCKS.

<unlock statement>

Function

releases row locks.

Format

```
 <unlock statement> ::=
     UNLOCK <row lock spec> IN SHARE MODE
   | UNLOCK <row lock spec> IN EXCLUSIVE MODE
   | UNLOCK <row lock spec> IN SHARE MODE
            <row lock spec> IN EXCLUSIVE MODE
   | UNLOCK <row lock spec> OPTIMISTIC
```

Syntax Rules

none

General Rules

1. SHARE locks, optimistic locks, and EXCLUSIVE locks set for single table rows which have not yet been updated can be released within a transaction using the <unlock statement>.

2. EXCLUSIVE locks come into existence when rows are inserted, updated or deleted, or they are set, like optimistic locks, by including <lock option>s in a SELECT statement or by issuing <lock statement>s. If a row has been inserted, updated or deleted, its EXCLUSIVE lock cannot be released by the <unlock statement>.

3. The <unlock statement> does not fail even if the specified lock does not exist or cannot be released.

<release statement>

Function

ends the transaction and the ADABAS session of a user.

Format

```
 <release statement> ::=
     COMMIT   [WORK] RELEASE
   | ROLLBACK [WORK] RELEASE
```

Syntax Rules

none

General Rules

1. COMMIT WORK RELEASE concludes the current transaction without opening a new one. The session is ended for the user.

2. If ADABAS has to undo the current transaction implicitly, then COMMIT WORK RELEASE fails, and a new transaction will be opened. The session of the user is not ended in this case.

3. ROLLBACK WORK RELEASE aborts the current transaction without opening a new one. Any database modifications performed during the current transaction are undone. The session of the user is ended. ROLLBACK WORK RELEASE has the same effect as a <rollback statement> followed by COMMIT WORK RELEASE.

4. Ending a session using a <release statement> implicitly deletes all result tables, the data stored in temporary base tables and the metadata of these tables.

5. If the ADABAS accounting is enabled, information concerning the session is inserted in the table SYSACCOUNT of the SYSDBA at the SERVERDB where the session was opened.

System Tables

This chapter describes the system tables that are available in all SQLMODEs. These system tables belong to the user 'DOMAIN'. In all SQLMODEs other than ADABAS, the name of the user 'DOMAIN' must be placed in front of the name of the system table.

COLUMNS			Columns of all tables, views, snapshots, synonyms, and results accessible to the user
OWNER	CHAR	(18)	Owner name of the table, view, snapshot, synonym, result
TABLENAME	CHAR	(18)	Table, view, snapshot, synonym or result name
COLUMNNAME	CHAR	(18)	Column name
MODE	CHAR	(3)	Mode of the column (key / man / opt)
DATATYPE	CHAR	(10)	Data type of the column (boolean / char / date / fixed / float / long / time / timestamp)
CODETYPE	CHAR	(8)	Code type of the column (ascii / ebcdic / byte)
LEN	FIXED	(4)	Length or precision of the column
DEC	FIXED	(3)	Digits to the right of the decimal point in a FIXED-type column
COLUMNPRIVILEGES	CHAR	(8)	User's privileges for the column
DEFAULT	CHAR	(254)	Default value for the column
DOMAINNAME	CHAR	(18)	Domain name
POS	FIXED	(3)	Original position of the column in the table
KEYPOS	FIXED	(3)	Original position of the key in the table
CREATEDATE	DATE		Creation date of the column
CREATETIME	TIME		Creation time of the column
ALTERDATE	DATE		Alteration date of the column
ALTERTIME	TIME		Alteration time of the column
TABLETYPE	CHAR	(8)	Type of the table
COMMENT	LONG		Comment on columns of accessible tables, snapshots and views

COL_REFS_DOM			Relationship Column Refers to Domain
DEFOBJTYPE	CHAR	(6)	COLUMN
DEFOWNER	CHAR	(18)	Owner name of the table
DEFTABLENAME	CHAR	(18)	Table name
DEFCOLUMNNAME	CHAR	(18)	Column name
RELTYPE	CHAR	(6)	REFERS
REFOBJTYPE	CHAR	(6)	DOMAIN
REFOWNER	CHAR	(18)	Owner name of the domain
REFDOMAINNAME	CHAR	(18)	Domain name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

COL_USES_COL			Relationship Column Uses Column
DEFOBJTYPE	CHAR	(6)	COLUMN
DEFOWNER	CHAR	(18)	Owner name of the table
DEFTABLENAME	CHAR	(18)	Table name
DEFCOLUMNNAME	CHAR	(18)	Column name

RELTYPE	CHAR	(4)	USES
REFOBJTYPE	CHAR	(6)	COLUMN
REFOWNER	CHAR	(18)	Owner name of the table
REFTABLENAME	CHAR	(18)	Table name
REFCOLUMNNAME	CHAR	(18)	Column name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship
CONNECTEDUSERS			All connected users
USERNAME	CHAR	(18)	User name
TERMID	CHAR	(18)	Terminal identification
SESSION	FIXED	(10)	Session
CATALOG_CACHE_SIZE	FIXED	(10)	Catalog cache size
DBPROC_CACHE_SIZE	FIXED	(10)	DB procedure cache size
TEMP_CACHE_SIZE	FIXED	(10)	Temporary cache size
SERVERDB	CHAR	(18)	SERVERDB name
CONNECTPARAMETERS			Connect parameters for the current user
SQLMODE	CHAR	(8)	SQLMODE
ISOLEVEL	FIXED	(10)	ISOLATION LEVEL
TIMEOUT	FIXED	(10)	Value for the session timeout
CACHELIMIT	FIXED	(10)	CACHELIMIT value
TERMCHARSETNAME	CHAR	(18)	TERMCHAR SET name
CONSTRAINTS			<constraint definitions> on accessible tables
OWNER	CHAR	(18)	Owner name of the table
TABLENAME	CHAR	(18)	Name of the table with the <constraint definition>
CONSTRAINTNAME	CHAR	(18)	<constraint definition> name
DEFINITION	LONG		<constraint definition> text
DBFUNCPARAMS			Parameters of a DB function that is accessible to the user
OWNER	CHAR	(18)	Owner name of the DB function
DBFUNCNAME	CHAR	(18)	DB function name
PARAMETERNAME	CHAR	(18)	Parameter name
POS	FIXED	(3)	Original position of the parameter in the DB function
IN/OUT-TYPE	CHAR	(6)	Mode of the parameter (in / out)
DATATYPE	CHAR	(10)	Data type of the column (boolean / char / date / fixed / float / time / timestamp)
LEN	FIXED	(4)	Length or precision of the parameter
DEC	FIXED	(3)	Digits to the right of the decimal point in FIXED-type parameters
CREATEDATE	DATE		Creation date of the DB function
CREATETIME	TIME		Creation time of the DB function
DBFUNCTIONS			DB functions accessible to the user

OWNER	CHAR	(18)	Owner name of the DB function
DBFUNCNAME	CHAR	(18)	DB function name
CREATEDATE	DATE		Creation date of the DB function
CREATETIME	TIME		Creation time of the DB function
COMMENT	LONG		Comment on the DB function

DBF_CONT_PRM			Relationship DB Function Contains Parameter
DEFOBJTYPE	CHAR	(10)	DBFUNCTION
DEFOWNER	CHAR	(18)	Owner name of the DB function
DEFDBFUNCNAME	CHAR	(18)	DB function name
RELTYPE	CHAR	(8)	CONTAINS
REFOBJTYPE	CHAR	(11)	DBFUNCTIONPARAM
REFOWNER	CHAR	(18)	Owner name of the DB function
REFDBFUNCNAME	CHAR	(18)	DB function name
REFPARAMETERNAME	CHAR	(18)	Parameter name
POS	FIXED	(3)	Original position of the parameter in the DB function
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

DBF_REFS_MOD			Relationship DB Function Refers to Module
DEFOBJTYPE	CHAR	(10)	DBFUNCTION
DEFOWNER	CHAR	(18)	Owner name of the DB function
DEFDBFUNCNAME	CHAR	(18)	DB function name
RELTYPE	CHAR	(6)	REFERS
REFOBJTYPE	CHAR	(6)	MODULE
REFOWNER	CHAR	(18)	Owner name of the module
REFPROGRAMNAME	CHAR	(18)	Program name
REFMODULENAME	CHAR	(18)	Module name
REFPROGLANG	CHAR	(6)	Programming language of the module (c/cobol ...)
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

DBPROCEDURES			DB procedures accessible to the user
OWNER	CHAR	(18)	Owner name of the DB procedure
PROGRAMNAME	CHAR	(18)	Program name
DBPROCNAME	CHAR	(18)	DB procedure name
ALIASNAME	CHAR	(18)	Short name of the DB procedure
PARAMETER	FIXED	(3)	Number of parameters of the DB procedure
EXECUTABLE	CHAR	(3)	DB procedure is executable (yes/no)
GRANT	CHAR	(3)	User is authorized to grant the right to execute the DB procedure (yes/no)
CREATEDATE	DATE		Creation date of the DB procedure
CREATETIME	TIME		Creation time of the DB procedure
COMMENT	LONG		Comment on the DB procedure

DBPROCPARAMS			Parameters of a DB procedure that is accessible to the user

OWNER	CHAR	(18)	Owner name of the DB procedure
PROGRAMNAME	CHAR	(18)	Program name
DBPROCNAME	CHAR	(18)	DB procedure name
PARAMETERNAME	CHAR	(18)	Parameter name
POS	FIXED	(3)	Original position of the parameter in the DB procedure
IN/OUT-TYPE	CHAR	(6)	Mode of the parameter (in/out)
DATATYPE	CHAR	(10)	Data type of the parameter (boolean / char / date / fixed / float / time / timestamp)
LEN	FIXED	(4)	Length or precision of the parameter
DEC	FIXED	(3)	Digits to the right of the decimal point in a parameter
CREATEDATE	DATE		Creation date of the DB procedure
CREATETIME	TIME		Creation time of the DB procedure

DBP_CONT_PRM Relationship DB Procedure Contains Parameter

DEFOBJTYPE	CHAR	(11)	DBPROCEDURE
DEFOWNER	CHAR	(18)	Owner name of the DB procedure
DEFPROGRAMNAME	CHAR	(18)	Program name
DEFDBPROCNAME	CHAR	(18)	DB procedure name
RELTYPE	CHAR	(8)	CONTAINS
REFOBJTYPE	CHAR	(11)	DBPROCEDUREPARAM
REFOWNER	CHAR	(18)	Owner name of the DB procedure
REFPROGRAMNAME	CHAR	(18)	Program name
REFDBPROCNAME	CHAR	(18)	DB procedure name
REFPARAMETERNAME	CHAR	(18)	Parameter name
POS	FIXED	(3)	Original position of the parameter in the DB procedure
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

DBP_REFS_MOD Relationship DB Procedure Refers to Module

DEFOBJTYPE	CHAR	(11)	DBPROCEDURE
DEFOWNER	CHAR	(18)	Owner name of the DB procedure
DEFPROGRAMNAME	CHAR	(18)	Program name
DEFDBPROCNAME	CHAR	(18)	DB procedure name
RELTYPE	CHAR	(6)	REFERS
REFOBJTYPE	CHAR	(6)	MODULE
REFOWNER	CHAR	(18)	Owner name of the module
REFPROGRAMNAME	CHAR	(18)	Program name
REFMODULENAME	CHAR	(18)	Module name
REFPROGLANG	CHAR	(6)	Programming language of the module (c/cobol ...)
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

DOMAINCONSTRAINTS <constraint definition> for a domain

OWNER	CHAR	(18)	Owner name of the domain

DOMAINNAME	CHAR	(18)	Domain name
CONSTRAINTNAME	CHAR	(18)	<constraint definition> name
DEFINITION	LONG		<constraint definition> text

DOMAINS			All domains
OWNER	CHAR	(18)	Owner name of the domain
DOMAINNAME	CHAR	(18)	Domain name
DATATYPE	CHAR	(10)	Data type of the domain (boolean / char / date / fixed / float / long / time / timestamp)
CODETYPE	CHAR	(8)	Code type of the domain (ascii / ebcdic / byte)
LEN	FIXED	(4)	Length or precision of the domain
DEC	FIXED	(3)	Digits to the right of the decimal point in a FIXED-type domain
DEFAULT	CHAR	(254)	Default value for the domain
DEFINITION	LONG		Text of the domain definition
CREATEDATE	DATE		Creation date of the domain
CREATETIME	TIME		Creation time of the domain
COMMENT	LONG		Comment on the domain

EASYCOMMANDS			EASY commands of the user
OWNER	CHAR	(18)	Owner name of the EASY command
COMMANDNAME	CHAR	(18)	EASY command name
TABLEOWNER	CHAR	(18)	Owner name of the table
TABLENAME	CHAR	(18)	Table name
CREATEDATE	DATE		Creation date of the EASY command
CREATETIME	TIME		Creation time of the EASY command
ALTERDATE	DATE		Alteration date of the EASY command
ALTERTIME	TIME		Alteration time of the EASY command
COMMENT	LONG		Comment on the EASY command

EASYFORMS			EASY forms accessible to the user
OWNER	CHAR	(18)	Owner name of the EASY form
TABLENAME	CHAR	(18)	Table name
FORMNAME	CHAR	(18)	EASY form name
CREATEDATE	DATE		Creation date of the EASY form
CREATETIME	TIME		Creation time of the EASY form
ALTERDATE	DATE		Alteration date of the EASY form
ALTERTIME	TIME		Alteration time of the EASY form
COMMENT	LONG		Comment on the EASY form

ECM_USES_COL			Relationship EASY Command Uses Column
DEFOBJTYPE	CHAR	(18)	EASYCOMMAND
DEFOWNER	CHAR	(18)	Owner name of the EASY command
DEFCOMMANDNAME	CHAR	(18)	EASY command name
DEFTABLEOWNER	CHAR	(18)	Owner name of the table
DEFTABLENAME	CHAR	(18)	Table name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	COLUMN
REFOWNER	CHAR	(18)	Owner name of the table

REFTABLENAME	CHAR	(18)	Table name
REFCOLUMNNAME	CHAR	(18)	Column name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

ECM_USES_SNP			Relationship EASY Command Uses Snapshot
DEFOBJTYPE	CHAR	(18)	EASYCOMMAND
DEFOWNER	CHAR	(18)	Owner name of the EASY command
DEFCOMMANDNAME	CHAR	(18)	EASY command name
DEFTABLEOWNER	CHAR	(18)	Owner name of the table
DEFTABLENAME	CHAR	(18)	Table name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	Snapshot
REFOWNER	CHAR	(18)	Owner name of the snapshot table
REFSNAPSHOTNAME	CHAR	(18)	Snapshot table name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

ECM_USES_SYN			Relationship EASY Command Uses Synonym
DEFOBJTYPE	CHAR	(18)	EASYCOMMAND
DEFOWNER	CHAR	(18)	Owner name of the EASY command
DEFCOMMANDNAME	CHAR	(18)	EASY command name
DEFTABLEOWNER	CHAR	(18)	Owner name of the table
DEFTABLENAME	CHAR	(18)	Table name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	SYNONYM
REFOWNER	CHAR	(18)	Owner name of the synonym
REFSYNONYMNAME	CHAR	(18)	Synonym name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

ECM_USES_TAB			Relationship EASY Command Uses Table
DEFOBJTYPE	CHAR	(18)	EASYCOMMAND
DEFOWNER	CHAR	(18)	Owner name of the EASY command
DEFCOMMANDNAME	CHAR	(18)	EASY command name
DEFTABLEOWNER	CHAR	(18)	Owner name of the table
DEFTABLENAME	CHAR	(18)	Table name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	TABLE
REFOWNER	CHAR	(18)	Owner name of the table
REFTABLENAME	CHAR	(18)	Table name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

ECM_USES_VIE			Relationship EASY Command Uses View
DEFOBJTYPE	CHAR	(18)	EASYCOMMAND
DEFOWNER	CHAR	(18)	Owner name of the EASY command
DEFCOMMANDNAME	CHAR	(18)	EASY command name

DEFTABLEOWNER	CHAR	(18)	Owner name of the table
DEFTABLENAME	CHAR	(18)	Table name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	VIEW
REFOWNER	CHAR	(18)	Owner name of the view table
REFVIEWNAME	CHAR	(18)	View table name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

EFM_USES_COL			Relationship EASY Form Uses Column
DEFOBJTYPE	CHAR	(18)	EASYFORM
DEFOWNER	CHAR	(18)	Owner name of the table
DEFTABLENAME	CHAR	(18)	Table name
DEFFORMNAME	CHAR	(18)	EASY form name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	COLUMN
REFOWNER	CHAR	(18)	Owner name of the table
REFTABLENAME	CHAR	(18)	Table name
REFCOLUMNNAME	CHAR	(18)	Column name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

EFM_USES_SNP			Relationship EASY Form Uses Snapshot
DEFOBJTYPE	CHAR	(18)	EASYFORM
DEFOWNER	CHAR	(18)	Owner name of the table
DEFTABLENAME	CHAR	(18)	Table name
DEFFORMNAME	CHAR	(18)	EASY form name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	SNAPSHOT
REFOWNER	CHAR	(18)	Owner name of the snapshot table
REFSNAPSHOTNAME	CHAR	(18)	Snapshot table name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

EFM_USES_SYN			Relationship EASY Form Uses Synonym
DEFOBJTYPE	CHAR	(18)	EASYFORM
DEFOWNER	CHAR	(18)	Owner name of the table
DEFTABLENAME	CHAR	(18)	Table name
DEFFORMNAME	CHAR	(18)	Easy form name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	SYNONYM
REFOWNER	CHAR	(18)	Owner name of the synonym
REFSYNONYMNAME	CHAR	(18)	Synonym name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

EFM_USES_TAB			Relationship EASY Form Uses Table
DEFOBJTYPE	CHAR	(18)	EASYFORM
DEFOWNER	CHAR	(18)	Owner name of the table
DEFTABLENAME	CHAR	(18)	Table name

DEFFORMNAME	CHAR	(18)	EASY form name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	TABLE
REFOWNER	CHAR	(18)	Owner name of the table
REFTABLENAME	CHAR	(18)	Table name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

EFM_USES_VIE			Relationship EASY Form Uses View
DEFOBJTYPE	CHAR	(18)	EASYFORM
DEFOWNER	CHAR	(18)	Owner name of the table
DEFTABLENAME	CHAR	(18)	Table name
DEFFORMNAME	CHAR	(18)	Easy form name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	VIEW
REFOWNER	CHAR	(18)	Owner name of the view table
REFVIEWNAME	CHAR	(18)	View table name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

FKC_REFS_COL			Relationship Foreign Key Column Refers to Column (foreign key)
DEFOBJTYPE	CHAR	(6)	FOREIGNKEYCOLUMN
DEFOWNER	CHAR	(18)	Owner name of the table
DEFTABLENAME	CHAR	(18)	Table name
DEFCOLUMNNAME	CHAR	(18)	Column name
DEFFKEYNAME	CHAR	(18)	Name of the <referential constraint definition>
RELTYPE	CHAR	(6)	REFERS
REFOBJTYPE	CHAR	(6)	COLUMN
REFOWNER	CHAR	(18)	Owner name of the table
REFTABLENAME	CHAR	(18)	Table name
REFCOLUMNNAME	CHAR	(18)	Column name
RULE	CHAR	(18)	Delete rule
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship
FKEYCOMMENT	LONG		Comment on the <referential constraint definition>

FOK_REFS_TAB			Relationship Foreign Key Refers to Table
DEFOBJTYPE	CHAR	(10)	FOREIGNKEY
DEFOWNER	CHAR	(18)	Owner name of the table
DEFTABLENAME	CHAR	(18)	Table name
DEFFKEYNAME	CHAR	(18)	Name of the <referential constraint definition>
RELTYPE	CHAR	(6)	REFERS
REFOBJTYPE	CHAR	(5)	TABLE
REFOWNER	CHAR	(18)	Owner name of the table
REFTABLENAME	CHAR	(18)	Table name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

FOK_USES_COL			Relationship Foreign Key Uses Column
DEFOBJTYPE	CHAR	(10)	FOREIGNKEY
DEFOWNER	CHAR	(18)	Owner name of the table
DEFTABLENAME	CHAR	(18)	Table name
DEFFKEYNAME	CHAR	(18)	Name of the <referential constraint definition>
RELTYPE	CHAR	(4)	USES
REFOBJTYPE	CHAR	(6)	COLUMN
REFOWNER	CHAR	(18)	Owner name of the table
REFTABLENAME	CHAR	(18)	Table name
REFCOLUMNNAME	CHAR	(18)	Column name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

FOREIGNKEYS			<referential constraint definition>s accessible to the user
OWNER	CHAR	(18)	Owner name of the table
TABLENAME	CHAR	(18)	Table name
FKEYNAME	CHAR	(18)	Name of the <referential constraint definition>
RULE	CHAR	(18)	Delete rule
CREATEDATE	DATE		Creation date of the <referential constraint definition>
CREATETIME	TIME		Creation time of the <referential constraint definition>
COMMENT	LONG		Comment on the <referential constraint definition>

INDEXES			Indexes accessible to the user
OWNER	CHAR	(18)	Owner name of the index
TABLENAME	CHAR	(18)	Table name
INDEXNAME	CHAR	(18)	Index name
TYPE	CHAR	(6)	Type of the index (unique/null)
CREATEDATE	DATE		Creation date of the index
CREATETIME	TIME		Creation time of the index
COMMENT	LONG		Comment on the index

IND_USES_COL			Relationship Index Uses Column
DEFOBJTYPE	CHAR	(5)	INDEX
DEFOWNER	CHAR	(18)	Owner name of the index
DEFTABLENAME	CHAR	(18)	Table name
DEFINDEXNAME	CHAR	(18)	Index name
RELTYPE	CHAR	(4)	USES
REFOBJTYPE	CHAR	(6)	COLUMN
REFOWNER	CHAR	(18)	Owner name of the table
REFTABLENAME	CHAR	(18)	Table name
REFCOLUMNNAME	CHAR	(18)	Column name
TYPE	CHAR	(6)	Type of the index (unique/null)
POS	FIXED	(3)	Original position of the column in the index

SORT	CHAR	(4)	Sort order (asc/desc)
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship
INDEXCOMMENT	LONG		Comment on the index

LITERALS			Literals accessible to the user
OWNER	CHAR	(18)	Owner name of the literal
LITERALNAME	CHAR	(18)	Literal name
LANGUAGE	CHAR	(18)	Literal language
S_LABEL	CHAR	(8)	Small label
M_LABEL	CHAR	(12)	Medium label
L_LABEL	CHAR	(18)	Large label
XL_LABEL	CHAR	(80)	Extra large label
CREATEDATE	DATE		Creation date of the literal
CREATETIME	TIME		Creation time of the literal
ALTERDATE	DATE		Alteration date of the literal
ALTERTIME	TIME		Alteration time of the literal
COMMENT	LONG		Comment on the literal

MAPCHARSETS			All MAPCHAR SETs
MAPCHARSETNAME	CHAR	(18)	Name of the MAPCHAR SET
CODE	CHAR	(8)	Code type for which the MAPCHAR SET was defined (ascii/ebcdic)
INTERN	CHAR	(1)	The original form in hexadecimal format
MAP_CODE	CHAR	(2)	The target form in hexadecimal notation
MAP_CHARACTER	CHAR	(2)	The target form with printable characters

MODULES			Modules accessible to the user
OWNER	CHAR	(18)	Owner name of the module
PROGRAMNAME	CHAR	(18)	Program name
MODULENAME	CHAR	(18)	Module name
PROGLANG	CHAR	(18)	Programming language of the module (c/cobol ...)
CREATEDATE	DATE		Creation date of the module
CREATETIME	TIME		Creation time of the module
ALTERDATE	DATE		Alteration date of the module
ALTERTIME	TIME		Alteration time of the module
COMMENT	LONG		Comment on the module

MOD_CALL_DBP			Relationship Module Calls DB Procedure
DEFOBJTYPE	CHAR	(18)	MODULE
DEFOWNER	CHAR	(18)	Owner name of the module
DEFPROGRAMNAME	CHAR	(18)	Program name
DEFMODULENAME	CHAR	(18)	Module name
DEFPROGLANG	CHAR	(18)	Programming language of the module (c/cobol ...)
RELTYPE	CHAR	(18)	CALLS
REFOBJTYPE	CHAR	(18)	DBPROCEDURE
REFOWNER	CHAR	(18)	Owner name of the DB procedure

REFPROGRAMNAME	CHAR	(18)	Program name
REFDBPROCNAME	CHAR	(18)	DB procedure name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

MOD_CALL_MOD			Relationship Module Calls Module
DEFOBJTYPE	CHAR	(18)	MODULE
DEFOWNER	CHAR	(18)	Owner name of the module
DEFPROGRAMNAME	CHAR	(18)	Program name
DEFMODULENAME	CHAR	(18)	Module name
DEFPROGLANG	CHAR	(18)	Programming language of the module (c/cobol ...)
RELTYPE	CHAR	(18)	CALLS
REFOBJTYPE	CHAR	(18)	MODULE
REFOWNER	CHAR	(18)	Owner name of the module
REFPROGRAMNAME	CHAR	(18)	Program name
REFMODULENAME	CHAR	(18)	Module name
REFPROGLANG	CHAR	(18)	Programming language of the module (c/cobol ...)
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

MOD_USES_COL			Relationship Module Uses Column
DEFOBJTYPE	CHAR	(18)	MODULE
DEFOWNER	CHAR	(18)	Owner name of the module
DEFPROGRAMNAME	CHAR	(18)	Program name
DEFMODULENAME	CHAR	(18)	Module name
DEFPROGLANG	CHAR	(18)	Programming language of the module (c/cobol ...)
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	COLUMN
REFOWNER	CHAR	(18)	Owner name of the table
REFTABLENAME	CHAR	(18)	Table name
REFCOLUMNNAME	CHAR	(18)	Column name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

MOD_USES_DOM			Relationship Module Uses Domain
DEFOBJTYPE	CHAR	(18)	MODULE
DEFOWNER	CHAR	(18)	Owner name of the module
DEFPROGRAMNAME	CHAR	(18)	Program name
DEFMODULENAME	CHAR	(18)	Module name
DEFPROGLANG	CHAR	(18)	Programming language of the module (c/cobol ...)
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	DOMAIN
REFOWNER	CHAR	(18)	Owner name of the domain
REFDOMAINNAME	CHAR	(18)	Domain name

CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

MOD_USES_QCM			Relationship Module Uses QUERY Command
DEFOBJTYPE	CHAR	(18)	MODULE
DEFOWNER	CHAR	(18)	Owner name of the module
DEFPROGRAMNAME	CHAR	(18)	Program name
DEFMODULENAME	CHAR	(18)	Module name
DEFPROGLANG	CHAR	(18)	Programming language of the module (c/cobol ...)
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	QUERYCOMMAND
REFOWNER	CHAR	(18)	Owner name of the QUERY command
REFCOMMANDNAME	CHAR	(18)	QUERY command name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

MOD_USES_SNP			Relationship Module Uses Snapshot
DEFOBJTYPE	CHAR	(18)	MODULE
DEFOWNER	CHAR	(18)	Owner name of the module
DEFPROGRAMNAME	CHAR	(18)	Program name
DEFMODULENAME	CHAR	(18)	Module name
DEFPROGLANG	CHAR	(18)	Programming language of the module (c/cobol ...)
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	SNAPSHOT
REFOWNER	CHAR	(18)	Owner name of the snapshot table
REFSNAPSHOTNAME	CHAR	(18)	Snapshot table name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

MOD_USES_SYN			Relationship Module Uses Synonym
DEFOBJTYPE	CHAR	(18)	MODULE
DEFOWNER	CHAR	(18)	Owner name of the module
DEFPROGRAMNAME	CHAR	(18)	Program name
DEFMODULENAME	CHAR	(18)	Module name
DEFPROGLANG	CHAR	(18)	Programming language of the module (c/cobol ...)
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	SYNONYM
REFOWNER	CHAR	(18)	Owner name of the synonym
REFSYNONYMNAME	CHAR	(18)	Synonym name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

MOD_USES_TAB			Relationship Module Uses Table
DEFOBJTYPE	CHAR	(18)	MODULE
DEFOWNER	CHAR	(18)	Owner name of the module

DEFPROGRAMNAME	CHAR	(18)	Program name
DEFMODULENAME	CHAR	(18)	Module name
DEFPROGLANG	CHAR	(18)	Programming language of the module (c/cobol ...)
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	TABLE
REFOWNER	CHAR	(18)	Owner name of the table
REFTABLENAME	CHAR	(18)	Table name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

MOD_USES_VIE			Relationship Module Uses View
DEFOBJTYPE	CHAR	(18)	MODULE
DEFOWNER	CHAR	(18)	Owner name of the module
DEFPROGRAMNAME	CHAR	(18)	Program name
DEFMODULENAME	CHAR	(18)	Module name
DEFPROGLANG	CHAR	(18)	Programming language of the module (c/cobol ...)
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	VIEW
REFOWNER	CHAR	(18)	Owner name of the view table
REFVIEWNAME	CHAR	(18)	View table name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

PROGRAMS			Programs accessible to the user
OWNER	CHAR	(18)	Owner name of the program
PROGRAMNAME	CHAR	(18)	Program name
PROGLANG	CHAR	(18)	Programming language of the program (c/cobol ...)
CREATEDATE	DATE		Creation date of the program
CREATETIME	TIME		Creation time of the program
ALTERDATE	DATE		Alteration date of the program
ALTERTIME	TIME		Alteration time of the program
COMMENT	LONG		Comment on the program

PRO_CONT_MOD			Relationship Program Contains Module
DEFOBJTYPE	CHAR	(7)	PROGRAM
DEFOWNER	CHAR	(18)	Owner name of the program
DEFPROGRAMNAME	CHAR	(18)	Program name
DEFPROGLANG	CHAR	(18)	Programming language of the program (c/cobol ...)
RELTYPE	CHAR	(8)	CONTAINS
REFOBJTYPE	CHAR	(6)	MODULE
REFOWNER	CHAR	(18)	Owner name of the module
REFPROGRAMNAME	CHAR	(18)	Program name
REFMODULENAME	CHAR	(18)	Module name
REFPROGLANG	CHAR	(18)	Programming language of the module (c/cobol ...)
CREATEDATE	DATE		Creation date of the relationship

CREATETIME	TIME		Creation time of the relationship

QCM_USES_COL			Relationship QUERY Command Uses Column
DEFOBJTYPE	CHAR	(18)	QUERYCOMMAND
DEFOWNER	CHAR	(18)	Owner name of the QUERY command
DEFCOMMANDNAME	CHAR	(18)	QUERY command name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	COLUMN
REFOWNER	CHAR	(18)	Owner name of the table
REFTABLENAME	CHAR	(18)	Table name
REFCOLUMNNAME	CHAR	(18)	Column name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

QCM_USES_SNP			Relationship QUERY Command Uses Snapshot
DEFOBJTYPE	CHAR	(18)	QUERYCOMMAND
DEFOWNER	CHAR	(18)	Owner name of the QUERY command
DEFCOMMANDNAME	CHAR	(18)	QUERY command name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	SNAPSHOT
REFOWNER	CHAR	(18)	Owner name of the snapshot table
REFSNAPSHOTNAME	CHAR	(18)	Snapshot table name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

QCM_USES_SYN			Relationship QUERY Command Uses Synonym
DEFOBJTYPE	CHAR	(18)	QUERYCOMMAND
DEFOWNER	CHAR	(18)	Owner name of the QUERY command
DEFCOMMANDNAME	CHAR	(18)	QUERY command name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	SYNONYM
REFOWNER	CHAR	(18)	Owner name of the synonym
REFSYNONYMNAME	CHAR	(18)	Synonym name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

QCM_USES_TAB			Relationship QUERY Command Uses Table
DEFOBJTYPE	CHAR	(18)	QUERYCOMMAND
DEFOWNER	CHAR	(18)	Owner name of the QUERY command
DEFCOMMANDNAME	CHAR	(18)	QUERY command name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	TABLE
REFOWNER	CHAR	(18)	Owner name of the table
REFTABLENAME	CHAR	(18)	Table name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

QCM_USES_VIE			Relationship QUERY Command Uses View

DEFOBJTYPE	CHAR	(18)	QUERYCOMMAND
DEFOWNER	CHAR	(18)	Owner name of the QUERY command
DEFCOMMANDNAME	CHAR	(18)	QUERY command name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	VIEW
REFOWNER	CHAR	(18)	Owner name of the view table
REFVIEWNAME	CHAR	(18)	View table name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

QPCOMMANDS			QueryPlus commands accessible to the user
OWNER	CHAR	(18)	Owner name of the QueryPlus command
COMMANDNAME	CHAR	(150)	QueryPlus command name
CREATEDATE	DATE		Creation date of the QueryPlus command
CREATETIME	TIME		Creation time of the QueryPlus command
ALTERDATE	DATE		Alteration date of the QueryPlus command
ALTERTIME	TIME		Alteration time of the QueryPlus command
COMMENT	LONG		Comment on the QueryPlus command

QPC_USES_COL			Relationship QueryPlus Command Uses Column
DEFOBJTYPE	CHAR	(18)	QPCOMMAND
DEFOWNER	CHAR	(18)	Owner name of the QueryPlus command
DEFCOMMANDNAME	CHAR	(150)	QueryPlus command name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	COLUMN
REFOWNER	CHAR	(18)	Owner name of the table
REFTABLENAME	CHAR	(18)	Table name
REFCOLUMNNAME	CHAR	(18)	Column name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

QPC_USES_SNP			Relationship QueryPlus Command Uses Snapshot
DEFOBJTYPE	CHAR	(18)	QPCOMMAND
DEFOWNER	CHAR	(18)	Owner name of the QueryPlus command
DEFCOMMANDNAME	CHAR	(150)	QueryPlus command name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	SNAPSHOT
REFOWNER	CHAR	(18)	Owner name of the snapshot table
REFSNAPSHOTNAME	CHAR	(18)	Snapshot table name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

QPC_USES_SYN			Relationship QueryPlus Command Uses Synonym
DEFOBJTYPE	CHAR	(18)	QPCOMMAND
DEFOWNER	CHAR	(18)	Owner name of the QueryPlus command

DEFCOMMANDNAME	CHAR	(150)	QueryPlus command name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	SYNONYM
REFOWNER	CHAR	(18)	Owner name of the synonym
REFSYNONYMNAME	CHAR	(18)	Synonym name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

QPC_USES_TAB			Relationship QueryPlus Command Uses Table
DEFOBJTYPE	CHAR	(18)	QPCOMMAND
DEFOWNER	CHAR	(18)	Owner name of the QueryPlus command
DEFCOMMANDNAME	CHAR	(150)	QueryPlus command name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	TABLE
REFOWNER	CHAR	(18)	Owner name of the table
REFTABLENAME	CHAR	(18)	Table name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

QPC_USES_VIE			Relationship QueryPlus Command Uses View
DEFOBJTYPE	CHAR	(18)	QPCOMMAND
DEFOWNER	CHAR	(18)	Owner name of the QueryPlus command
DEFCOMMANDNAME	CHAR	(150)	QueryPlus command name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	VIEW
REFOWNER	CHAR	(18)	Owner name of the view table
REFVIEWNAME	CHAR	(18)	View table name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

QPEXCELLINKS			QueryPlus ExcelLinks accessible to the user
OWNER	CHAR	(18)	Owner name of the QueryPlus ExcelLink
EXCELLINKNAME	CHAR	(150)	QueryPlus ExcelLink name
CREATEDATE	DATE		Creation date of the QueryPlus ExcelLink
CREATETIME	TIME		Creation time of the QueryPlus ExcelLink
ALTERDATE	DATE		Alteration date of the QueryPlus ExcelLink
ALTERTIME	TIME		Alteration time of the QueryPlus ExcelLink
COMMENT	LONG		Comment on the QueryPlus ExcelLink

QPE_USES_QPC			Relationship QueryPlus ExcelLink Uses QueryPlus Command
DEFOBJTYPE	CHAR	(18)	QPEXCELLINK
DEFOWNER	CHAR	(18)	Owner name of the QueryPlus ExcelLink
DEFEXCELLINKNAME	CHAR	(150)	QueryPlus ExcelLink name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	QPCOMMAND
REFOWNER	CHAR	(18)	Owner name of the QueryPlus command
REFCOMMANDNAME	CHAR	(150)	QueryPlus command name
CREATEDATE	DATE		Creation date of the relationship

CREATETIME	TIME		Creation time of the relationship

QPE_USES_QPQ			Relationship QueryPlus ExcelLink Uses QueryPlus Query
DEFOBJTYPE	CHAR	(18)	QPEXCELLINK
DEFOWNER	CHAR	(18)	Owner name of the QueryPlus ExcelLink
DEFEXCELLINKNAME	CHAR	(150)	QueryPlus ExcelLink name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	QPQUERY
REFOWNER	CHAR	(18)	Owner name of the QueryPlus query
REFQUERYNAME	CHAR	(150)	QueryPlus query name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

QPQUERYS			QueryPlus queries accessible to the user
OWNER	CHAR	(18)	Owner name of the QueryPlus query
QUERYNAME	CHAR	(150)	QueryPlus query name
CREATEDATE	DATE		Creation date of the QueryPlus query
CREATETIME	TIME		Creation time of the QueryPlus query
ALTERDATE	DATE		Alteration date of the QueryPlus query
ALTERTIME	TIME		Alteration time of the QueryPlus query
COMMENT	LONG		Comment on the QueryPlus query

QPQ_USES_COL			Relationship QueryPlus Query Uses Column
DEFOBJTYPE	CHAR	(18)	QPQUERY
DEFOWNER	CHAR	(18)	Owner name of the QueryPlus query
DEFQUERYNAME	CHAR	(150)	QueryPlus query name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	COLUMN
REFOWNER	CHAR	(18)	Owner name of the table
REFTABLENAME	CHAR	(18)	Table name
REFCOLUMNNAME	CHAR	(18)	Column name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

QPQ_USES_SNP			Relationship QueryPlus Query Uses Snapshot
DEFOBJTYPE	CHAR	(18)	QPQUERY
DEFOWNER	CHAR	(18)	Owner name of the QueryPlus query
DEFQUERYNAME	CHAR	(150)	QueryPlus query name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	SNAPSHOT
REFOWNER	CHAR	(18)	Owner name of the snapshot table
REFSNAPSHOTNAME	CHAR	(18)	Snapshot table name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

QPQ_USES_SYN			Relationship QueryPlus Query Uses Synonym

DEFOBJTYPE	CHAR	(18)	QPQUERY
DEFOWNER	CHAR	(18)	Owner name of the QueryPlus query
DEFQUERYNAME	CHAR	(150)	QueryPlus query name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	SYNONYM
REFOWNER	CHAR	(18)	Owner name of the synonym
REFSYNONYMNAME	CHAR	(18)	Synonym name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

QPQ_USES_TAB			Relationship QueryPlus Query Uses Table
DEFOBJTYPE	CHAR	(18)	QPQUERY
DEFOWNER	CHAR	(18)	Owner name of the QueryPlus query
DEFQUERYNAME	CHAR	(150)	QueryPlus query name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	TABLE
REFOWNER	CHAR	(18)	Owner name of the table
REFTABLENAME	CHAR	(18)	Table name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

QPQ_USES_VIE			Relationship QueryPlus Query Uses View
DEFOBJTYPE	CHAR	(18)	QPQUERY
DEFOWNER	CHAR	(18)	Owner name of the QueryPlus query
DEFQUERYNAME	CHAR	(150)	QueryPlus query name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	VIEW
REFOWNER	CHAR	(18)	Owner name of the view table
REFVIEWNAME	CHAR	(18)	View table name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

QPWORDLINKS			QueryPlus WordLinks accessible to the user
OWNER	CHAR	(18)	Owner name of the QueryPlus WordLink
WORDLINKNAME	CHAR	(150)	QueryPlus WordLink name
CREATEDATE	DATE		Creation date of the QueryPlus WordLink
CREATETIME	TIME		Creation time of the QueryPlus WordLink
ALTERDATE	DATE		Alteration date of the QueryPlus WordLink
ALTERTIME	TIME		Alteration time of the QueryPlus WordLink
COMMENT	LONG		Comment on the QueryPlus WordLink

QPW_USES_QPC			Relationship QueryPlus WordLink Uses QueryPlus Command
DEFOBJTYPE	CHAR	(18)	QPWORDLINK
DEFOWNER	CHAR	(18)	Owner name of the QueryPlus WordLink
DEFWORDLINKNAME	CHAR	(150)	QueryPlus WordLink name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	QPCOMMAND
REFOWNER	CHAR	(18)	Owner name of the QueryPlus command

REFCOMMANDNAME	CHAR	(150)	QueryPlus command name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

QPW_USES_QPQ			Relationship QueryPlus WordLink Uses QueryPlus Query
DEFOBJTYPE	CHAR	(18)	QPWORDLINK
DEFOWNER	CHAR	(18)	Owner name of the QueryPlus WordLink
DEFWORDLINKNAME	CHAR	(150)	QueryPlus WordLink name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	QPQUERY
REFOWNER	CHAR	(18)	Owner name of the QueryPlus query
REFQUERYNAME	CHAR	(150)	QueryPlus query name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

QUERYCOMMANDS			QUERYcommands accessible to the user
OWNER	CHAR	(18)	Owner name of the QUERY command
COMMANDNAME	CHAR	(18)	QUERY command name
CREATEDATE	DATE		Creation date of the QUERY command
CREATETIME	TIME		Creation time of the QUERY command
ALTERDATE	DATE		Alteration date of the QUERY command
ALTERTIME	TIME		Alteration time of the QUERY command
COMMENT	LONG		Comment on the QUERY command

SEQUENCES			Sequences accessible to the user
OWNER	CHAR	(18)	Owner name of the sequence
SEQUENCE_NAME	CHAR	(18)	Sequence name
MIN_VALUE	FIXED	(10)	Minimum value of the sequence
MAX_VALUE	FIXED	(10)	Maximum value of the sequence
INCREMENT_BY	FIXED	(10)	Value by which the sequence is incremented
CYCLE_FLAG	CHAR	(1)	Does the sequence wrap around on reaching the limit?
ORDER_FLAG	CHAR	(1)	Are sequence numbers generated in order?
CACHE_SIZE	FIXED	(10)	Number of sequence values loaded into the cache
LAST_NUMBER	FIXED	(10)	Last sequence number written to disk
CREATEDATE	DATE		Creation date of the sequence
CREATETIME	TIME		Creation time of the sequence
COMMENT	LONG		Comment on the sequence

SERVERDBS			All SERVERDBs
NO	FIXED	(4)	SERVERDB number
STATE	CHAR	(8)	SERVERDB state
MAJORITY	CHAR	(8)	SERVERDB belongs to the majority (yes/no)
SERVERDB	CHAR	(18)	SERVERDB name
SERVERNODE	CHAR	(64)	SERVERNODE in the network

SNAPSHOTDEFS — Definition of a snapshot table accessible to the user

OWNER	CHAR	(18)	Owner name of the snapshot table
SNAPSHOTNAME	CHAR	(18)	Snapshot table name
FAST_REFRESHABLE	CHAR	(3)	Snapshot table can be refreshed fast (yes/no)
MASTER_OWNER	CHAR	(18)	Owner name of the base table on which the snapshot table was built
MASTER_TABLENAME	CHAR	(18)	Table name of the base table on which the snapshot table was built
LEN	FIXED	(4)	Length of the snapshot table definition
DEFINITION	LONG		Text of the snapshot table definition

SNAPSHOTS — Snapshot tables accessible to the user

OWNER	CHAR	(18)	Owner name of the snapshot table
SNAPSHOTNAME	CHAR	(18)	Snapshot table name
PRIVILEGES	CHAR	(30)	User's privileges for the snapshot table
TYPE	CHAR	(8)	Type of the table
CREATEDATE	DATE		Creation date of the snapshot table
CREATETIME	TIME		Creation time of the snapshot table
UPDSTATDATE	DATE		Date of the last <update statistics> performed on the snapshot table
UPDSTATTIME	TIME		Time of the last <update statistics> performed on the snapshot table
ALTERDATE	DATE		Alteration date of the snapshot table
ALTERTIME	TIME		Alteration time of the snapshot table
REPLICATION	CHAR	(3)	Snapshot table is replicated (yes/no/null)
SERVERDB	CHAR	(18)	SERVERDB name
SERVERNODE	CHAR	(64)	SERVERNODE in the network
COMMENT	LONG		Comment on the snapshot table

SNP_CONT_COL — Relationship Snapshot Contains Column

DEFOBJTYPE	CHAR	(5)	SNAPSHOT
DEFOWNER	CHAR	(18)	Owner name of the snapshot table
DEFSNAPSHOTNAME	CHAR	(18)	Snapshot table name
RELTYPE	CHAR	(8)	CONTAINS
REFOBJTYPE	CHAR	(6)	COLUMN
REFOWNER	CHAR	(18)	Owner name of the snapshot table
REFTABLENAME	CHAR	(18)	Table name
REFCOLUMNNAME	CHAR	(18)	Column name
POS	FIXED	(3)	Original position of the column in the snapshot table
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

SNP_USES_SYN — Relationship Snapshot Uses Synonym

DEFOBJTYPE	CHAR	(8)	SNAPSHOT
DEFOWNER	CHAR	(18)	Owner name of the snapshot table
DEFSNAPSHOTNAME	CHAR	(18)	Snapshot table name
RELTYPE	CHAR	(4)	USES

REFOBJTYPE	CHAR	(5)	SYNONYM
REFOWNER	CHAR	(18)	Owner name of the synonym
REFSYNONYMNAME	CHAR	(18)	Synonym name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

SNP_USES_TAB			Relationship Snapshot Uses Table
DEFOBJTYPE	CHAR	(8)	SNAPSHOT
DEFOWNER	CHAR	(18)	Owner name of the snapshot table
DEFSNAPSHOTNAME	CHAR	(18)	Snapshot table name
RELTYPE	CHAR	(4)	USES
REFOBJTYPE	CHAR	(5)	TABLE
REFOWNER	CHAR	(18)	Owner name of the table
REFTABLENAME	CHAR	(18)	Table name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

SNP_USES_VIE			Relationship Snapshot Uses View
DEFOBJTYPE	CHAR	(8)	SNAPSHOT
DEFOWNER	CHAR	(18)	Owner name of the snapshot table
DEFSNAPSHOTNAME	CHAR	(18)	Snapshot table name
RELTYPE	CHAR	(4)	USES
REFOBJTYPE	CHAR	(5)	VIEW
REFOWNER	CHAR	(18)	Owner name of the view table
REFVIEWNAME	CHAR	(18)	View table name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

SYNONYMS			Synonyms accessible to the user
OWNER	CHAR	(18)	Owner name of the synonym
SYNONYMNAME	CHAR	(18)	Synonym name
TABLEOWNER	CHAR	(18)	Owner name of the table
TABLENAME	CHAR	(18)	Table name
CREATEDATE	DATE		Creation date of the synonym
CREATETIME	TIME		Creation time of the synonym
COMMENT	LONG		Comment on the synonym

SYN_REFS_TAB			Relationship Synonym Refers to Table
DEFOBJTYPE	CHAR	(7)	SYNONYM
DEFOWNER	CHAR	(18)	Owner name of the synonym
DEFSYNONYMNAME	CHAR	(18)	Synonym name
RELTYPE	CHAR	(6)	REFERS
REFOBJTYPE	CHAR	(5)	TABLE
REFOWNER	CHAR	(18)	Owner name of the table
REFTABLENAME	CHAR	(18)	Table name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

TABLES			Tables, views, snapshots, synonyms, results accessible to the user
OWNER	CHAR	(18)	Owner name of the table, view, snapshot, synonym, result
TABLENAME	CHAR	(18)	Table, view, snapshot, synonym, result name
PRIVILEGES	CHAR	(30)	User's privileges for the table, view, snapshot, synonym, result
TYPE	CHAR	(8)	Table type (table / view / synonym / snapshot / result)
CREATEDATE	DATE		Creation date of the table, view, snapshot, synonym, result
CREATETIME	TIME		Creation time of the table, view, snapshot, synonym, result
UPDSTATDATE	DATE		Date of the last <update statistics> performed on the table
UPDSTATTIME	TIME		Time of the last <update statistics> performed on the table
ALTERDATE	DATE		Alteration date of the table
ALTERTIME	TIME		Alteration time of the table
REPLICATION	CHAR	(3)	Table is replicated (yes/no/null)
SERVERDB	CHAR	(18)	SERVERDB name
SERVERNODE	CHAR	(64)	SERVERNODE in the network
SNAPSHOT_LOG	CHAR	(3)	Table has a snapshot log (yes/no)
COMMENT	LONG		Comment on the table, view, snapshot, synonym

TAB_CONT_COL			Relationship Table Contains Column
DEFOBJTYPE	CHAR	(5)	TABLE
DEFOWNER	CHAR	(18)	Owner name of the table
DEFTABLENAME	CHAR	(18)	Table name
RELTYPE	CHAR	(8)	CONTAINS
REFOBJTYPE	CHAR	(6)	COLUMN
REFOWNER	CHAR	(18)	Owner name of the table
REFTABLENAME	CHAR	(18)	Table name
REFCOLUMNNAME	CHAR	(18)	Column name
POS	FIXED	(3)	Original position of the column in the table
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

TAB_CONT_TRG			Relationship Table Contains Trigger
DEFOBJTYPE	CHAR	(5)	TABLE
DEFOWNER	CHAR	(18)	Owner name of the table
DEFTABLENAME	CHAR	(18)	Table name
RELTYPE	CHAR	(8)	CONTAINS
REFOBJTYPE	CHAR	(7)	TRIGGER
REFOWNER	CHAR	(18)	Owner name of the table
REFTABLENAME	CHAR	(18)	Table name
REFTRIGGERNAME	CHAR	(18)	Trigger name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

TAB_USES_CON Relationship Table Uses Constraint

DEFOBJTYPE	CHAR	(5)	TABLE
DEFOWNER	CHAR	(18)	Owner name of the table
DEFTABLENAME	CHAR	(18)	Table name
RELTYPE	CHAR	(4)	USES
REFOBJTYPE	CHAR	(10)	CONSTRAINT
REFOWNER	CHAR	(18)	Owner name of the table
REFTABLENAME	CHAR	(18)	Table name
REFCONSTRAINTNAME	CHAR	(18)	<constraint definition> name

TERMCHARSETS All TERMCHAR SETs

TERMCHARSETNAME	CHAR	(18)	Name of the TERMCHAR SET
CODE	CHAR	(8)	Code type for which the TERMCHAR SET was defined (ascii/ebcdic)
STATE	CHAR	(8)	TERMCHAR SET is activated (enabled/disabled)
INTERN	CHAR	(1)	The original form in hexadecimal format
EXTERN	CHAR	(1)	The terminal-specific variant in hexadecimal format
COMMENT	CHAR	(8)	Comment on the TERMCHAR SET

TRG_CONT_PRM Relationship Trigger Contains Parameter

DEFOBJTYPE	CHAR	(7)	TRIGGER
DEFOWNER	CHAR	(18)	Owner name of the table
DEFTABLENAME	CHAR	(18)	Table name
DEFTRIGGERNAME	CHAR	(18)	Trigger name
RELTYPE	CHAR	(8)	CONTAINS
REFOBJTYPE	CHAR	(12)	TRIGGERPARAM
REFOWNER	CHAR	(18)	Owner name of the table
REFTABLENAME	CHAR	(18)	Table name
REFTRIGGERNAME	CHAR	(18)	Trigger name
REFPARAMETERNAME	CHAR	(18)	Parameter name
POS	FIXED	(3)	Original position of the parameter in the trigger
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

TRG_REFS_MOD Relationship Trigger Refers to Module

DEFOBJTYPE	CHAR	(7)	TRIGGER
DEFOWNER	CHAR	(18)	Owner name of the table
DEFTABLENAME	CHAR	(18)	Table name
DEFTRIGGERNAME	CHAR	(18)	Trigger name
RELTYPE	CHAR	(6)	REFERS
REFOBJTYPE	CHAR	(6)	MODULE
REFOWNER	CHAR	(18)	Owner name of the module
REFPROGRAMNAME	CHAR	(18)	Program name
REFMODULENAME	CHAR	(18)	Module name
REFPROGLANG	CHAR	(6)	Programming language of the module (c/cobol

			...)
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

TRIGGERPARAMS			Parameters of a trigger that is accessible to the user
OWNER	CHAR	(18)	Owner name of the table
TABLENAME	CHAR	(18)	Table name
TRIGGERNAME	CHAR	(18)	Trigger name
PARAMETERNAME	CHAR	(18)	Parameter name
POS	FIXED	(3)	Original position of the parameter in the trigger
NEW/OLD-TYPE	CHAR	(3)	Version of the parameter (new/old)
DATATYPE	CHAR	(10)	Data type of the column (boolean / char / date / fixed / float / time / timestamp)
LEN	FIXED	(4)	Length or precision of the column
DEC	FIXED	(3)	Digits to the right of the decimal point in FIXED-type parameters
CREATEDATE	DATE		Creation date of the trigger
CREATETIME	TIME		Creation time of the trigger

TRIGGERS			Triggers accessible to the user
OWNER	CHAR	(18)	Owner name of the table
TABLENAME	CHAR	(18)	Table name
TRIGGERNAME	CHAR	(18)	Trigger name
INSERT	CHAR	(3)	Type of the trigger
UPDATE	CHAR	(3)	Type of the trigger
DELETE	CHAR	(3)	Type of the trigger
CREATEDATE	DATE		Creation date of the trigger
CREATETIME	TIME		Creation time of the trigger
DEFINITION	LONG		Text of the trigger definition
COMMENT	LONG		Comment on the trigger

USERS			All users
OWNER	CHAR	(18)	Owner name of the user
GROUPNAME	CHAR	(18)	Group name
USERNAME	CHAR	(18)	User name
USERMODE	CHAR	(8)	Class of the user (sysdba / dba / resource / standard)
CONNECTMODE	CHAR	(8)	Connect mode (multiple/single)
PERMLIMIT	FIXED	(10)	PERMLIMIT value
TEMPLIMIT	FIXED	(10)	TEMPLIMIT value
MAXTIMEOUT	FIXED	(10)	TIMEOUT value
COSTWARNING	FIXED	(10)	COSTWARNING value
COSTLIMIT	FIXED	(10)	COSTLIMIT value
CACHELIMIT	FIXED	(10)	CACHELIMIT value
CREATEDATE	DATE		Creation date of the user
CREATETIME	TIME		Creation time of the user
ALTERDATE	DATE		Alteration date of the user

ALTERTIME	TIME		Alteration time of the user
PWCREADATE	DATE		Creation date of the password
PWCREATIME	TIME		Creation time of the password
SERVERDB	CHAR	(18)	SERVERDB name
SERVERNODE	CHAR	(64)	SERVERNODE in the network
COMMENT	LONG		Comment on the user
USR_OWNS_DBF			Relationship User Owns DB Function
DEFOBJTYPE	CHAR	(4)	USER
DEFOWNER	CHAR	(18)	Owner name of the user
DEFGROUPNAME	CHAR	(18)	Group name
DEFUSERNAME	CHAR	(18)	User name
RELTYPE	CHAR	(4)	OWNS
REFOBJTYPE	CHAR	(10)	DBFUNCTION
REFOWNER	CHAR	(18)	Owner name of the DB function
REFDBFUNCNAME	CHAR	(18)	DB function name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship
USR_OWNS_DOM			Relationship User Owns Domain
DEFOBJTYPE	CHAR	(4)	USER
DEFOWNER	CHAR	(18)	Owner name of the user
DEFGROUPNAME	CHAR	(18)	Group name
DEFUSERNAME	CHAR	(18)	User name
RELTYPE	CHAR	(4)	OWNS
REFOBJTYPE	CHAR	(6)	DOMAIN
REFOWNER	CHAR	(18)	Owner name of the domain
REFDOMAINNAME	CHAR	(18)	Domain name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship
USR_OWNS_USR			Relationship User Owns User
DEFOBJTYPE	CHAR	(4)	USER
DEFOWNER	CHAR	(18)	Owner name of the user
DEFGROUPNAME	CHAR	(18)	Group name
DEFUSERNAME	CHAR	(18)	User name
RELTYPE	CHAR	(4)	OWNS
REFOBJTYPE	CHAR	(4)	USER
REFOWNER	CHAR	(18)	Owner name of the user
REFGROUPNAME	CHAR	(18)	Group name
REFUSERNAME	CHAR	(18)	User name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship
USR_USES_COL			Relationship User Uses Column
DEFOBJTYPE	CHAR	(4)	USER
DEFOWNER	CHAR	(18)	Owner name of the user
DEFGROUPNAME	CHAR	(18)	Group name

DEFUSERNAME	CHAR	(18)	User name
RELTYPE	CHAR	(4)	USES
REFOBJTYPE	CHAR	(6)	COLUMN
REFOWNER	CHAR	(18)	Owner name of the table
REFTABLENAME	CHAR	(18)	Table, view or snapshot name
REFCOLUMNNAME	CHAR	(18)	Column name
PRIVILEGES	CHAR	(30)	User's privileges for the column
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

USR_USES_DBP			Relationship User Uses DB Procedure
DEFOBJTYPE	CHAR	(4)	USER
DEFOWNER	CHAR	(18)	Owner name of the user
DEFGROUPNAME	CHAR	(18)	Group name
DEFUSERNAME	CHAR	(18)	User name
RELTYPE	CHAR	(4)	USES
REFOBJTYPE	CHAR	(11)	DBPROCEDURE
REFOWNER	CHAR	(18)	Owner name of the DB procedure
REFPROGRAMNAME	CHAR	(18)	Program name
REFDBPROCNAME	CHAR	(18)	DB procedure name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

USR_USES_PRO			Relationship User Uses Program
DEFOBJTYPE	CHAR	(18)	USER
DEFOWNER	CHAR	(18)	Owner name of the user
DEFGROUPNAME	CHAR	(18)	Group name
DEFUSERNAME	CHAR	(18)	User name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	PROGRAM
REFOWNER	CHAR	(18)	Owner name of the program
REFPROGRAMNAME	CHAR	(18)	Program name
REFPROGLANG	CHAR	(18)	Programming language of the program (c/cobol ...)
PRIVILEGES	CHAR	(30)	User's privileges for the program
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

USR_USES_QCM			Relationship User Uses QUERY Command
DEFOBJTYPE	CHAR	(18)	USER
DEFOWNER	CHAR	(18)	Owner name of the user
DEFGROUPNAME	CHAR	(18)	Group name
DEFUSERNAME	CHAR	(18)	User name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	QUERYCOMMAND
REFOWNER	CHAR	(18)	Owner name of the query command
REFCOMMANDNAME	CHAR	(18)	Query command name
PRIVILEGES	CHAR	(30)	User's privileges for the QUERY command
CREATEDATE	DATE		Creation date of the relationship

CREATETIME	TIME		Creation time of the relationship

USR_USES_QPC			Relationship User Uses QueryPlus Command
DEFOBJTYPE	CHAR	(18)	USER
DEFOWNER	CHAR	(18)	Owner name of the user
DEFGROUPNAME	CHAR	(18)	Group name
DEFUSERNAME	CHAR	(18)	User name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	QPCOMMAND
REFOWNER	CHAR	(18)	Owner name of the QueryPlus command
REFCOMMANDNAME	CHAR	(150)	QueryPlus command name
PRIVILEGES	CHAR	(30)	User's privileges for the QueryPlus command
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

USR_USES_QPE			Relationship User Uses QueryPlus ExcelLink
DEFOBJTYPE	CHAR	(18)	USER
DEFOWNER	CHAR	(18)	Owner name of the user
DEFGROUPNAME	CHAR	(18)	Group name
DEFUSERNAME	CHAR	(18)	User name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	QPEXCELLINK
REFOWNER	CHAR	(18)	Owner name of the QueryPlus ExcelLink
REFEXCELLINKNAME	CHAR	(150)	QueryPlus ExcelLink name
PRIVILEGES	CHAR	(30)	User's privileges for the QueryPlus ExcelLink
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

USR_USES_QPQ			Relationship User Uses QueryPlus Query
DEFOBJTYPE	CHAR	(18)	USER
DEFOWNER	CHAR	(18)	Owner name of the user
DEFGROUPNAME	CHAR	(18)	Group name
DEFUSERNAME	CHAR	(18)	User name
RELTYPE	CHAR	(18)	USES
REFOBJTYPE	CHAR	(18)	QPQUERY
REFOWNER	CHAR	(18)	Owner name of the QueryPlus query
REFQUERYNAME	CHAR	(150)	QueryPlus query name
PRIVILEGES	CHAR	(30)	User's privileges for the QueryPlus query
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

USR_USES_QPW			Relationship User Uses QueryPlus WordLink
DEFOBJTYPE	CHAR	(18)	USER
DEFOWNER	CHAR	(18)	Owner name of the user
DEFGROUPNAME	CHAR	(18)	Group name
DEFUSERNAME	CHAR	(18)	User name
RELTYPE	CHAR	(18)	USES

REFOBJTYPE	CHAR	(18)	QPWORDLINK
REFOWNER	CHAR	(18)	Owner name of the QueryPlus WordLink
REFWORDLINKNAME	CHAR	(150)	QueryPlus WordLink name
PRIVILEGES	CHAR	(30)	User's privileges for the QueryPlus WordLink
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

USR_USES_TAB			Relationship User Uses table
DEFOBJTYPE	CHAR	(4)	USER
DEFOWNER	CHAR	(18)	Owner name of the user
DEFGROUPNAME	CHAR	(18)	Group name
DEFUSERNAME	CHAR	(18)	User name
RELTYPE	CHAR	(4)	USES
REFOBJTYPE	CHAR	(5)	TABLE
REFOWNER	CHAR	(18)	Owner name of the table
REFTABLENAME	CHAR	(18)	Table name
PRIVILEGES	CHAR	(30)	User's privileges for the table
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

VERSIONS			Version
KERNEL	CHAR	(40)	Version of the ADABAS server
RUNTIMEENVIRONMENT	CHAR	(40)	Version of the runtime environment

VIEWDEFS			Definition of a view accessible to the user
OWNER	CHAR	(18)	Owner name of the view table
VIEWNAME	CHAR	(18)	View table name
LEN	FIXED	(4)	Length of the view table definition
DEFINITION	LONG		Text of the view table definition

VIEWS			View tables accessible to the user
OWNER	CHAR	(18)	Owner name of the view table
VIEWNAME	CHAR	(18)	View table name
PRIVILEGES	CHAR	(30)	User's privileges for the view table
TYPE	CHAR	(8)	Type of the table
CREATEDATE	DATE		Creation date of the view table
CREATETIME	TIME		Creation time of the view table
UPDSTATDATE	DATE		Date of the last <update statistics> performed on the view table
UPDSTATTIME	TIME		Time of the last <update statistics> performed on the view table
ALTERDATE	DATE		Alteration date of the view
ALTERTIME	TIME		Alteration time of the view
REPLICATION	CHAR	(3)	Table is replicated (yes/no/null)
SERVERDB	CHAR	(18)	SERVERDB name
SERVERNODE	CHAR	(64)	SERVERNODE in the network
COMMENT	LONG		Comment on the view table

VIE_CONT_COL			Relationship View Contains Column
DEFOBJTYPE	CHAR	(5)	VIEW
DEFOWNER	CHAR	(18)	Owner name of the view table
DEFVIEWNAME	CHAR	(18)	View table name
RELTYPE	CHAR	(8)	CONTAINS
REFOBJTYPE	CHAR	(6)	COLUMN
REFOWNER	CHAR	(18)	Owner name of the view table
REFTABLENAME	CHAR	(18)	Table name
REFCOLUMNNAME	CHAR	(18)	Column name
POS	FIXED	(3)	Original position of the column in the view table
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

VIE_USES_SNP			Relationship View Uses Snapshot
DEFOBJTYPE	CHAR	(4)	VIEW
DEFOWNER	CHAR	(18)	Owner name of the view table
DEFVIEWNAME	CHAR	(18)	View table name
RELTYPE	CHAR	(4)	USES
REFOBJTYPE	CHAR	(5)	SNAPSHOT
REFOWNER	CHAR	(18)	Owner name of the snapshot table
REFSNAPSHOTNAME	CHAR	(18)	Snapshot table name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

VIE_USES_SYN			Relationship View Uses Synonym
DEFOBJTYPE	CHAR	(4)	VIEW
DEFOWNER	CHAR	(18)	Owner name of the view table
DEFVIEWNAME	CHAR	(18)	View table name
RELTYPE	CHAR	(4)	USES
REFOBJTYPE	CHAR	(5)	SYNONYM
REFOWNER	CHAR	(18)	Owner name of the synonym
REFSYNONYMNAME	CHAR	(18)	Synonym name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

VIE_USES_TAB			Relationship View Uses Table
DEFOBJTYPE	CHAR	(4)	VIEW
DEFOWNER	CHAR	(18)	Owner name of the view table
DEFVIEWNAME	CHAR	(18)	View table name
RELTYPE	CHAR	(4)	USES
REFOBJTYPE	CHAR	(5)	TABLE
REFOWNER	CHAR	(18)	Owner name of the view table
REFTABLENAME	CHAR	(18)	Table name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

VIE_USES_VIE			Relationship View Uses View

DEFOBJTYPE	CHAR	(4)	VIEW
DEFOWNER	CHAR	(18)	Owner name of the view table
DEFVIEWNAME	CHAR	(18)	View table name
RELTYPE	CHAR	(4)	USES
REFOBJTYPE	CHAR	(5)	VIEW
REFOWNER	CHAR	(18)	Owner name of the view table
REFVIEWNAME	CHAR	(18)	View table name
CREATEDATE	DATE		Creation date of the relationship
CREATETIME	TIME		Creation time of the relationship

Statistics

The units in which ADABAS addresses hard disks is 4 KB. In this chapter, the term 'page' is used for such a unit.

<update statistics statement>

Function

defines the storage requirements of tables and indexes as well as the value distribution of indexes and columns, and stores this information in the catalog.

Format

```
 <update statistics statement> ::=
     UPDATE STAT[ISTICS] COLUMN <table name>.<column name>
   | UPDATE STAT[ISTICS] COLUMN (<column name>,...)
                                FOR <table name>
   | UPDATE STAT[ISTICS] [<owner>.]<table name>
   | UPDATE STAT[ISTICS] [<owner>.][<identifier>]*
```

Syntax Rules

none

General Rules

1. If a <table name> is specified, the table must be a non-temporary base table or a snapshot table, and the user must have a privilege for it.

2. If a <column name> is specified, this column must exist in the table <table name>.

3. Specifying <identifier>* has the same effect as issuing the <update statistics statement> for all base tables for which the current user has a privilege, and whose <table name> begins with <identifier>.

4. The SYSDBA can use UPDATE STATISTICS * to execute the <update statistics statement> for all base tables whose HOME SERVERDB is the HOME SERVERDB of the SYSDBA - regardless of whether the SYSDBA has privileges for these tables.

5. The <update statistics statement> implicitly performs a <commit statement> for each base table; i.e., the transaction within which the <update statistics statement> has been executed is closed.

6. The execution of the <update statistics statement> has the effect that information about the table, such as the number of rows, the number of used pages, the sizes of indexes, the value distribution within columns or indexes, etc., is stored in the catalog. These values are used by the ADABAS optimizer to optimize SQL statements.

7. When a <create index statement> is executed, the above-mentioned information is stored in the catalog for the index as well as for the base table for which this index is being defined. No information is stored for other indexes defined on this base table.

8. The statistical values stored in the catalog can be retrieved by selecting the system table OPTIMIZERSTATISTICS. Each row of the table describes statistical values of indexes, columns or the size of a table:

OPTIMIZERSTATISTICES

OWNER	CHAR	(18)	owner of the table for which statistical information is available
TABLENAME	CHAR	(18)	name of table for which statistical information is available
COLUMNNAME	CHAR	(18)	name of a column for which statistical information is available
INDEXNAME	CHAR	(18)	name of an index for which statistical information is available
DISTINCTVALUES	FIXED	(10)	number of different values if the current row describes a column or an index; otherwise, the number of rows in a table
PAGECOUNT	FIXED	(10)	number of pages used by an index if the current row describes an index; number of pages in a base table if the current row describes a table; otherwise; NULL
AVGLISTLENGTH	FIXED	(10)	average number of keys in an index list if the current row describes an index; otherwise, NULL

Statistical System Tables

During the installation of ADABAS, a set of system tables is created on each SERVERDB. These system tables can be used to select information about the configuration, structures and sizes of database objects.
On each SERVERD, these tables are owned by the SYSDBA created during the configuration. The specification of the <owner> is not required for the access to the tables.

DBPARAMETERS			parameter of a SERVERDB
DESCRIPTION	CHAR	(18)	description of how to interpret the column VALUE
VALUE	CHAR	(64)	value

This table contains the parameters defined for the SERVERDB by using the ADABAS component CONTROL. The column DESCRIPTION contains the following values:

SERVERDB
VALUE contains the logical SERVERDB name

SYSDEVSPACE
VALUE contains the logical name of the first system DEVSPACE of ADABAS

MIRR_SYSDEVSPACE
VALUE contains the logical name of the mirror DEVSPACE of the system DEVSPACE if mirrored DEVSPACEs are defined

TRANSACTION_LOG
VALUE contains the logical name of the transaction log DEVSPACE

ARCHIVE_LOG
VALUE contains the logical name of the first archive log DEVSPACE of ADABAS

MIRR_ARCHIVE_LOG
VALUE contains the logical name of the mirror DEVSPACE of the archive log if mirrored DEVSPACEs are defined

CONTROLUSERID
VALUE contains the name of the CONTROL user

MAXDEVSPACES
VALUE contains the maximum number of DEVSPACEs

MAXDATADEVSPACES
VALUE contains the maximum number of data DEVSPACEs

MAXSERVERDB
VALUE contains the maximum number of SERVERDBs in the distributed database

MAXBACKUPDEVS
VALUE contains the maximum number of backup devices

SERVERTASKS
VALUE contains the maximum number of servers for the handling of remote tasks

MAXUSERTASKS
VALUE contains the maximum number of users who can simultaneously establish sessions with the SERVERDB

MAXDATAPAGES
VALUE contains the maximum number of data pages of the SERVERDB

MAXCPU
VALUE contains the number of CPUs available to ADABAS

DATA_CACHE_PAGES
VALUE contains the size of the data cache in pages

PROC_DATA_PAGES
VALUE contains the size of the storage area in pages available for variables in DB procedures, DB functions and triggers

PROC_CODE_PAGES
VALUE contains the size of the storage area in pages available for the code of DB procedures and triggers

TEMP_CACHE_PAGES
VALUE contains the size of the storage area in pages available for temporary pages in the session-specific caches

CATALOG_CACHE_PAGS
VALUE contains the size of the storage area in pages available for catalog information in the session-specific caches

CONV_CACHE_PAGES
VALUE contains the size of the converter cache in pages

MAXLOCKS
VALUE contains the maximum number of locks and lock requests

RUNDIRECTORY
VALUE contains the path name of the directory where diagnose information will be stored

OPMSG1
VALUE contains the logical name of the device for the output of priority 1 messages

OPMSG2
VALUE contains the logical name of the device for the output of priority 2 messages

CONFIGURATION			configuration parameters of the SERVERDB
DESCRIPTION	CHAR	(40)	description of how to interpret the value in the column CHAR_VALUE or NUMERIC_VALUE
CHAR_VALUE	CHAR	(40)	alphanumeric value
NUMERIC_VALUE	FIXED	(10)	numeric value

The column DESCRIPTION contains the following values:

DEFAULT CODE
In this row, the column CHAR_VALUE contains the code (ASCII or EBCDIC) used to store columns of the data type CHAR

DATE TIME FORMAT
In this row, the column CHAR_VALUE contains the date and time formats (EUR, INTERNAL, ISO, JIS, USA) used to represent columns of the data type DATE, TIME or TIMESTAMP

SESSION TIMEOUT
The column NUMERIC_VALUE contains the timeout value for the maximum time of inactivity in seconds

LOCK TIMEOUT
The column NUMERIC_VALUE contains the timeout value for inactive locks in seconds

REQUEST TIMEOUT
The column NUMERIC_VALUE contains the timeout value for lock requests in seconds

LOG MODE
The column CHAR_VALUE describes the log mode (DEMO, SINGLE, NORMAL, DUAL)

LOG SEGMENT SIZE
The column NUMERIC_VALUE contains the size of a log segment in pages

NO OF ARCHIVE LOGS
The column NUMERIC_VALUE contains the number of archive log DEVSPACEs

NO OF DATA DEVSPACES
The column NUMERIC_VALUE contains the number of data DEVSPACEs

MIRRORED DEVSPACES
The column CHAR_VALUE contains information about mirrored DEVSPACEs (YES, NO)

SYS DEVSPACE SIZE
The column NUMERIC_VALUE contains the size of the system DEVSPACE in pages

SYS DEVSPACE NAME
The column CHAR_VALUE contains the logical name of the system DEVSPACE

TRANSACTION LOG SIZE
The column NUMERIC_VALUE contains the size of the transaction log in pages

TRANSACTION LOG NAME
The column CHAR_VALUE contains the name of the transaction log

DATA DEVSPACE * SIZE
The column NUMERIC_VALUE contains the size of the data DEVSPACE in pages

DATA DEVSPACE * NAME
The column CHAR_VALUE contains the name of a data DEVSPACE

DATADEVSPACES			usage of data DEVSPACEs
DEVSPACENAME	CHAR	(40)	logical name of the data DEVSPACE
DEVSPACESIZE	FIXED	(10)	size of the DEVSPACE in pages
MAXDATAPAGENO	FIXED	(10)	largest created page number
USEDPERMPAGES	FIXED	(10)	number of DEVSPACE pages used for permanent objects
PCTUSEDPERM	FIXED	(10)	percentage of the pages used for permanent objects
USEDTMPPAGES	FIXED	(10)	number of DEVSPACE pages used for temporary objects
PCTUSEDTMP	FIXED	(10)	percentage of the pages used for temporary objects
UNUSEDPAGES	FIXED	(10)	number of unused pages
PCTUNUSED	FIXED	(10)	percentage of unused pages

INDEXSTATISTICS			information about structure and size of indexes
OWNER	CHAR	(18)	owner of a table
TABLENAME	CHAR	(18)	table name
INDEXNAME	CHAR	(18)	index name (NULL for unnamed indexes)
COLUMNNAME	CHAR	(18)	name of an indexed column
DESCRIPTION	CHAR	(40)	description of how to interpret the following columns
CHAR_VALUE	CHAR	(12)	alphanumeric value
NUMERIC_VALUE	FIXED	(10)	numeric value

The column DESCRIPTION contains the following values:

ROOT PNO
NUMERIC_VALUE contains the page number of the B* tree root

FILETYPE
CHAR_VALUE contains the type of the B* tree

USED PAGES
NUMERIC_VALUE contains the number of pages used by the index

INDEX PAGES
NUMERIC_VALUE contains the number of B* tree index pages used by the index

LEAF PAGES
NUMERIC_VALUE contains the number of leaf pages used by the index

INDEX LEVELS
NUMERIC_VALUE contains the number of B* tree index levels

SPACE USED IN ALL   PAGES(%)
NUMERIC_VALUE contains the percentage of the index pages used

SPACE USED IN ROOT PAGE(%)
NUMERIC_VALUE contains the percentage of the B* tree root page used

SPACE USED IN INDEX PAGES(%)
NUMERIC_VALUE contains the percentage of the B* tree index pages used

SPACE USED IN INDEX PAGES(%) MIN
NUMERIC_VALUE contains the minimum percentage of the B* tree index pages used

SPACE USED IN INDEX PAGES(%) MAX
NUMERIC_VALUE contains the maximum percentage of the B* tree index pages used

SPACE USED IN LEAF PAGES(%)
NUMERIC_VALUE contains the percentage of the B* tree leaf pages used

SPACE USED IN LEAF PAGES(%) MIN
NUMERIC_VALUE contains the minimum percentage of the B* tree leaf pages used

SPACE USED IN LEAF PAGES(%) MAX
NUMERIC_VALUE contains the maximum percentage of the B* tree leaf pages used

SECONDARY KEYS (INDEX LISTS)
NUMERIC_VALUE contains the number of different values in the indexed columns

AVG SECONDARY KEY LENGTH
NUMERIC_VALUE contains the average length of the index values

MIN SECONDARY KEY LENGTH
NUMERIC_VALUE contains the minimum length of the index values

MAX SECONDARY KEY LENGTH
NUMERIC_VALUE contains the maximum length of the index values

AVG SEPARATOR LENGTH
NUMERIC_VALUE contains the average length of a B* tree separator

MIN SEPARATOR LENGTH
NUMERIC_VALUE contains the minimum length of the separator

MAX SEPARATOR LENGTH
NUMERIC_VALUE contains the maximum length of the separator

PRIMARY KEYS
NUMERIC_VALUE contains the number of tables identified by OWNER and

TABLENAME

AVG PRIMARY KEYS PER LIST
NUMERIC_VALUE contains the average number of keys per index list

MIN PRIMARY KEYS PER LIST
NUMERIC_VALUE contains the minimum number of keys per index list

MAX PRIMARY KEYS PER LIST
NUMERIC_VALUE contains the maximum number of keys per index list

VALUES WITH SELECTIVITY <= 1%
NUMERIC_VALUE contains the number of index lists with a selectivity <= 1%

VALUES WITH SELECTIVITY <= 5%
NUMERIC_VALUE contains the number of index lists with a selectivity between 1% and 5%

VALUES WITH SELECTIVITY <= 10%
NUMERIC_VALUE contains the number of index lists with a selectivity between 5% and 10%

VALUES WITH SELECTIVITY <= 25%
NUMERIC_VALUE contains the number of index lists with a selectivity between 10% and 25%.

VALUES WITH SELECTIVITY > 25%
NUMERIC_VALUE contains the number of index lists with a selectivity > 25%

LOCKSTATISTICS			information about the lock list contents
SESSION	FIXED	(10)	user session identification
TRANSACTION	FIXED	(10)	transaction identification
SERVERDBNO	FIXED	(5)	SERVERDB identification
PROCESS	FIXED	(10)	user process identification
USERNAME	CHAR	(18)	user name
TERMID	CHAR	(18)	terminal identification
REMOTEUSER	CHAR	(3)	'YES' for lock entries of remote SERVERDBs; otherwise, NO
PENDINGLOCK	CHAR	(3)	'YES' for 'pending' locks; otherwise; 'NO'
LOCKMODE	CHAR	(14)	lock mode
LOCKREQUESTMODE	CHAR	(14)	lock request mode
OWNER	CHAR	(18)	table owner
TABLENAME	CHAR	(18)	table name
ROWIDLENGTH	FIXED	(3)	length of the key of the locked row
ROWID	CHAR	(120)	prefix of the key of the locked row
ROWIDHEX	CHAR	(40)	prefix of the key of the row in hexadecimal representation

LOCKLISTSTATISTICS			information about the lock list usage
DESCRIPTION	CHAR	(40)	description of how to interpret the contents of the column VALUE
VALUE	CHAR	(12)	value

The column DESCRIPTION contains the following values:

ENTRIES

VALUE contains the number of entries available in the lock list

USED ENTRIES
VALUE contains the number of entries for locks and lock requests

USED ENTRIES(%)
VALUE contains the percentage of used entries available in the lock list

AVG USED ENTRIES
VALUE contains the average number of entries for locks and lock requests

AVG USED ENTRIES(%)
VALUE contains the average percentage of used entries for locks and lock requests

MAX USED ENTRIES
VALUE contains the maximum number of entries for locks and lock requests

MAX USED ENTRIES(%)
VALUE contains the maximum percentage of used entries for locks and lock requests

LOCK ESCALATION
VALUE contains the number of lock escalations

TRANSACTIONS HOLDING LOCKS
VALUE contains the number of transactions with assigned locks

TRANSACTIONS REQUESTING LOCKS
VALUE contains the number of transactions requesting locks

CHECKPOINT WANTED
If the column VALUE contains the value 'TRUE', the lock list is closed, i.e., no EXCLUSIVE lock can be assigned to a transaction without EXCLUSIVE lock because a checkpoint was requested

SHUTDOWN WANTED
If the column VALUE contains the value 'TRUE', the lock list is closed because a shutdown was requested

SERVERDBSTATISTICS			information about the use of the SERVERDB
SERVERDBSIZE	FIXED	(10)	SERVERDB size in pages
MAXDATAPAGENO	FIXED	(10)	largest page number of the SERVERDB
USEDPERMPAGES	FIXED	(10)	number of SERVERDB pages used for non-temporary objects
PCTUSEDPERM	FIXED	(10)	percentage of pages used for non-temporary objects
USEDTMPPAGES	FIXED	(10)	number of SERVERDB pages used for temporary objects
PCTUSEDTMP	FIXED	(10)	percentage of pages used for temporary objects
UNUSEDPAGES	FIXED	(10)	number of unused pages
PCTUNUSED	FIXED	(10)	percentage of unused pages
UPDATEDPERMPAGES	FIXED	(10)	number of modified pages for permanent objects
LOGSIZE	FIXED	(10)	log size in pages
USEDLOGPAGES	FIXED	(10)	number of log pages used
PCTUSEDLOGPAGES	FIXED	(10)	percentage of log pages used
RESERVEDLOGPAGES	FIXED	(10)	reserved log pages
LOGSEGMENTSIZE	FIXED	(10)	log segment size in pages
COMPLETESEGMENTS	FIXED	(10)	number of completed log segments
SAVEPOINTS	FIXED	(10)	number of savepoints written
CHECKPOINTS	FIXED	(10)	number of checkpoints written

PAGESPERSAVEPOINT	FIXED	(10)	average savepoint distance in log pages
PAGESPERCHECKPOINT	FIXED	(10)	average checkpoint distance in log pages

TABLESTATISTICS			information about structure and size of base tables
OWNER	CHAR	(18)	table owner
TABLENAME	CHAR	(18)	table name
DESCRIPTION	CHAR	(40)	description of how to interpret the following columns
CHAR_VALUE	CHAR	(12)	alphanumeric value
NUMERIC_VALUE	FIXED	(10)	numeric value

The column DESCRIPTION contains the following values:

`ROOT PNO`
NUMERIC_VALUE contains the page number of the B* tree root

`FILETYPE`
CHAR_VALUE contains the B* tree type

`USED PAGES`
NUMERIC_VALUE contains the number of pages used by the table

`INDEX PAGES`
NUMERIC_VALUE contains the number of pages used by the table in the B* tree index

`LEAF PAGES`
NUMERIC_VALUE contains the number of leaf pages used by the table

`INDEX LEVELS`
NUMERIC_VALUE contains the number of B* tree index levels

`SPACE USED IN ALL   PAGES(%)`
NUMERIC_VALUE contains the percentage of index pages used

`SPACE USED IN ROOT  PAGE(%)`
NUMERIC_VALUE contains the percentage of the B* tree root page used

`SPACE USED IN INDEX PAGES(%)`
NUMERIC_VALUE contains the percentage of the B* tree index pages used

`SPACE USED IN INDEX PAGES(%) MIN`
NUMERIC_VALUE contains the minimum percentage of the B* tree index pages used

`SPACE USED IN INDEX PAGES(%) MAX`
NUMERIC_VALUE contains the maximum percentage of the B* tree index pages used

`SPACE USED IN LEAF  PAGES(%)`
NUMERIC_VALUE contains the percentage of the B* tree leaf pages used

`SPACE USED IN LEAF  PAGES(%) MIN`
NUMERIC_VALUE contains the minimum percentage of the B* tree leaf pages used

`SPACE USED IN LEAF  PAGES(%) MAX`
NUMERIC_VALUE contains the maximum percentage of the B* tree leaf pages used

`ROWS`
NUMERIC_VALUE contains the number of table rows

`AVG ROWS PER PAGE`
NUMERIC_VALUE contains the average number of rows per page

`MIN ROWS PER PAGE`
NUMERIC_VALUE contains the minimum number of rows per page

`MAX ROWS PER PAGE`
NUMERIC_VALUE contains the maximum number of rows per page

`AVG ROW LENGTH`
NUMERIC_VALUE contains the average length of rows

`MIN ROW LENGTH`
NUMERIC_VALUE contains the minimum length of rows

`MAX ROW LENGTH`
NUMERIC_VALUE contains the maximum length of rows

`AVG KEY LENGTH`
NUMERIC_VALUE contains the average length of keys

`MIN KEY LENGTH`
NUMERIC_VALUE contains the minimum length of keys

`MAX KEY LENGTH`
NUMERIC_VALUE contains the maximum length of keys

`AVG SEPARATOR LENGTH`
NUMERIC_VALUE contains the average length of the separator

`MIN SEPARATOR LENGTH`
NUMERIC_VALUE contains the minimum length of the separator

`MAX SEPARATOR LENGTH`
NUMERIC_VALUE contains the maximum length of the separator

`DEFINED LONG COLUMNS`
NUMERIC_VALUE contains the number of defined columns of the data type LONG

`AVG LONG   COLUMN LENGTH`
NUMERIC_VALUE contains the average length of LONG columns

`MIN LONG   COLUMN LENGTH`
NUMERIC_VALUE contains the minimum length of LONG columns

`MAX LONG   COLUMN LENGTH`
NUMERIC_VALUE contains the maximum length of LONG columns

`LONG COLUMN PAGES`
NUMERIC_VALUE contains the number of pages of all LONG columns of the table

`AVG PAGES PER LONG COLUMN`
NUMERIC_VALUE contains the average number of pages of the table per LONG column

`MIN PAGES PER LONG COLUMN`
NUMERIC_VALUE contains the smallest LONG column of the table in pages

`MAX PAGES PER LONG COLUMN`
NUMERIC_VALUE contains the largest LONG column of the table in pages

TRANSACTIONS			information about active transactions of a SERVERDB
SESSION	FIXED	(10)	user session identification
TRANSACTION	FIXED	(10)	transaction identification
SERVERDBNO	FIXED	(5)	SERVERDB identification
PROCESS	FIXED	(10)	user process identification
USERNAME	CHAR	(18)	user name

CONNECTDATE	DATE		
CONNECTTIME	TIME		session begin
TERMID	CHAR	(18)	terminal identification
REMOTEUSER	CHAR	(3)	'YES' for lock entries of remote SERVERDBs; otherwise, 'NO'
PENDINGLOCK	CHAR	(3)	'YES' for 'pending'locks; otherwise, 'NO'
LOCKMODE	CHAR	(14)	lock mode
LOCKREQUESTMODE	CHAR	(14)	lock request mode

USERSTATISTICS			information about the resources used by users
USERNAME	CHAR	(18)	user name
USERMODE	CHAR	(8)	user class
PERMLIMIT	FIXED	(10)	maximum number of pages that can be used for permanent objects
PERMLCOUNT	FIXED	(10)	number of pages currently used for permanent objects
TEMPLIMIT	FIXED	(10)	maximum number of pages that can be used for temporary objects
TEMPCOUNT	FIXED	(10)	number of pages currently used for temporary objects

ADABAS Monitor

See

<monitor statement>

<monitor statement>

Function

enables or disables the database monitoring.

Format

```
 <monitor statement> ::=
     MONITOR ON
   | MONITOR OFF
```

Syntax Rules

none

General Rules

1. If MONITOR ON is specified, counters registering internal ADABAS events are kept, to be used for tuning measures. All counters are initialized with 0.

2. MONITOR OFF disables the counters for the internal ADABAS events. The counters are not reset.

3. The counters for the internal events kept by ADABAS can be retrieved by selecting system tables. The system tables are created by the SYSDBA during the installation. They produce results for users with DBA status. For non-authorized users, the error message 100 ROW NOT FOUND is output. The specification of the <owner> is not required for the access to the tables. The tables have the following structure:

```
DESCRIPTIONCHAR(40)
VALUE      CHAR(12)
```

   Each row contains a counter value which is described by the value contained in the column DESCRIPTION.
   The following monitor system tables are provided:

   MONITOR_CACHES

   contains information about the operations performed on the different ADABAS caches. The column DESRIPTION contains the following values:

   DATA CACHE ACCESSES
   number of accesses to the ADABAS data cache

   DATA CACHE ACCESSES SUCCESSFUL
   number of successful accesses to the data cache

   DATA CACHE ACCESSES UNSUCCESSFUL
   number of unsuccessful accesses to the data cache

DATA CACHE HIT RATE (%)
percentage of successful accesses to the data cache

FILE DIRECTORY CACHE ACCESSES
number of accesses to the ADABAS file cache

FILE DIRECTORY CACHE ACCESSES SUCCESSFUL
number of successful accesses to the file cache

FILE DIRECTORY CACHE ACCESSES UNSUCCESSFUL
number of unsuccessful accesses to the file cache

FILE DIRECTORY CACHE HIT RATE (%)
percentage of successful accesses to the file cache

FBM CACHE ACCESSES
number of accesses to the Free Block Management cache

FBM CACHE ACCESSES SUCCESSFUL
number of successful accesses to the Free Block Management cache

FBM CACHE ACCESSES UNSUCCESSFUL
number of unsuccessful accesses to the Free Block Management cache

FBM CACHE HIT RATE (%)
percentage of successful accesses to the Free Block Management cache

CONVERTER CACHE ACCESSES
number of accesses to the converter cache

CONVERTER CACHE ACCESSES SUCCESSFUL
number of successful accesses to the converter cache

CONVERTER CACHE ACCESSES UNSUCCESSFUL
number of unsuccessful accesses to the converter cache

CONVERTER CACHE HIT RATE (%)
percentage of successful accesses to the converter cache

USM CACHE ACCESSES
number of accesses to the User Storage Management cache

USM CACHE ACCESSES SUCCESSFUL
number of successful accesses to the User Storage Management cache

USM CACHE ACCESSES UNSUCCESSFUL
number of unsuccessful accesses to the User Storage Management cache

USM CACHE HIT RATE (%)
percentage of successful accesses to the User Storage Management cache

LOG CACHE ACCESSES
number of accesses to the log cache

LOG CACHE ACCESSES SUCCESSFUL
number of successful accesses to the log cache

LOG CACHE ACCESSES UNSUCCESSFUL
number of unsuccessful accesses to the log cache

LOG CACHE HIT RATE (%)
percentage of successful accesses to the log cache

CATALOG CACHE ACCESSES
number of accesses to the session-specific catalog cache

CATALOG CACHE ACCESSES SUCCESSFUL
number of successful accesses to the session-specific catalog cache

`CATALOG CACHE ACCESSES UNSUCCESSFUL`
number of unsuccessful accesses to the session-specific catalog cache

`CATALOG CACHE HIT RATE (%)`
percentage of successful accesses to the session-specific catalog cache

`TEMP CACHE ACCESSES`
number of accesses to the session-specific cache for temporary pages

`TEMP CACHE ACCESSES SUCCESSFUL`
number of successful accesses to the session-specific cache for temporary pages

`TEMP CACHE ACCESSES UNSUCCESSFUL`
number of unsuccessful accesses to the session-specific cache for temporary pages

`TEMP CACHE HIT RATE (%)`
percentage of successful accesses to the session-specific cache for temporary pages

MONITOR_LOAD

contains information about the executed SQL statements and access methods.
The column DESCRIPTION contains the following values:

`SQL COMMANDS`
number of executed SQL statements

`PREPARES`
number of parsed SQL statements

`EXECUTES`
number of executions of previously parsed SQL statements

`COMMITS`
number of executed <commit statement>s

`ROLLBACKS`
number of executed <rollback statement>s

`LOCKS AND UNLOCKS`
number of executed <lock statement>s and <unlock statement>s

`SUBTRANS BEGINS`
number of SQL statements for the opening of a subtransaction

`SUBTRANS ENDS`
number of SQL statements for the conclusion of a subtransaction

`SUBTRANS ROLLBACKS`
number of SQL statements for the rollback of a subtransaction

`CREATES`
number of executed SQL statements for the creation of database objects

`ALTERS`
number of executed SQL statements for the alteration of database objects

`DROPS`
number of executed SQL statements for the dropping of database objects

`SELECTS AND FETCHES`
number of executed SQL statements for data access

`SELECTS AND FETCHES, ROWS READ`
number of rows considered for the access of data

SELECTS AND FETCHES, ROWS QUAL
number of rows considered for the access of data satisfying conditions

INSERTS
number of executed SQL statement for the insertion of rows

INSERTS, ROWS INSERTED
number of rows inserted

UPDATES
number of executed SQL statements for the update of rows

UPDATES, ROWS READ
number of rows considered for the update of data

UPDATES, ROWS UPDATED
number of rows updated

DELETES
number of executed SQL statements for the deletion of rows

DELETES, ROWS READ
number of rows considered for the deletion of data

DELETES, ROWS DELETED
number of rows deleted

SHOWS
number of SQL statements for the reading of metadata of the catalog

DBPROC CALLS
number of DB procedure calls

TRIGGER CALLS
number of trigger calls

PRIMARY KEY ACCESSES
number of search operations with direct access using the key

PRIMARY KEY ACCESSES, ROWS READ
number of rows read by direct access using the key

PRIMARY KEY ACCESSES, ROWS QUAL
number of rows read by direct access using the key, satisfying conditions

PRIMARY KEY RANGE ACCESSES
number of search operations with accesses within a range of keys

PRIMARY KEY RANGE ACCESSES, ROWS READ
number of rows read within a range of keys

PRIMARY KEY RANGE ACCESSES, ROWS QUAL
number of rows read within a range of keys, satisfying conditions

INDEX ACCESSES
number of search operations with accesses to an index

INDEX ACCESSES, ROWS READ
number of rows directly accessed using an index

INDEX ACCESSES, ROWS QUAL
number of rows indirectly accessed using an index, satisfying conditions

INDEX RANGE ACCESSES
number of search operations using an index range

INDEX RANGE ACCESSES, ROWS READ
number of rows indirectly accessed using an index range

INDEX RANGE ACCESSES, ROWS QUAL
number of rows indirectly accessed using an index range, satisfying conditions

ISOLATED INDEX ACCESSES
number of search operations completely or partially satisfied by an index without accessing the corresponding row

ISOLATED INDEX ACCESSES, ROWS READ
number of keys accessed within the search operations denoted in ISOLATED INDEX ACCESSES

ISOLATED INDEX ACCESSES, ROWS QUAL
number of keys accessed within the search operations denoted in ISOLATED INDEX ACCESSES, satisfying conditions

ISOLATED INDEX RANGE ACCESSES
number of search operations using a part of an index with values within a range without accessing the rows of the base table

ISOLATED INDEX RANGE ACCESSES, ROWS READ
number of primary/secondary keys accessed within the search operations denoted by ISOLATED INDEX RANGE ACCESSES

ISOLATED INDEX RANGE ACCESSES, ROWS QUAL
number of primary/secondary keys accessed within the search operations denoted by ISOLATED INDEX RANGE ACCESSES, satisfying conditions

TABLE SCANS
number of search operations through the whole base table

TABLE SCANS, ROWS READ
number of rows accessed within search operations through the whole base table

TABLE SCANS, ROWS QUAL
number of rows accessed within search operations through the whole base table, satisfying conditions

ISOLATED INDEX SCANS
number of search operations for which a complete index was accessed without accessing rows of the base table

ISOLATED INDEX SCANS, ROWS READ
number of index rows accessed within the search operations described under ISOLATED INDEX SCANS

ISOLATED INDEX SCANS, ROWS QUAL
number of index rows accessed within the search operations described under ISOLATED INDEX SCANS, satisfying conditions

MEMORY SORTS / SORT&MERGE
number of sorting operations in the main memory to build temporary indexes

MEMORY SORTS / SORT&MERGE, ROWS READ
number of rows read to build temporary indexes

SORTS BY INSERTION
number of sorting operations by inserts

SORTS BY INSERTION, ROWS INSERTED
number of rows inserted during the sorting operation

MONITOR_LOCK

contains information about operations performed by the ADABAS lock manager. The column DESCRIPTION contains the following values:

`LOCK LIST AVG USED ENTRIES`
average number of entries in the lock list

`LOCK LIST MAX USED ENTRIES`
maximum number of entries in the lock list

`LOCK LIST COLLISIONS`
number of lock collisions

`LOCK LIST ESCALATIONS`
number of lock escalations

`LOCK LIST INSERTED ROW   ENTRIES`
number of inserted row locks

`LOCK LIST INSERTED TABLE ENTRIES`
number of inserted table locks

MONITOR_LOG

contains information about operations executed by the ADABAS logging. The column DESCRIPTION contains the following values:

`LOG PAGE PHYSICAL READS`
number of physically read log pages

`LOG PAGE PHYSICAL WRITES`
number of physically written log pages

`LOG QUEUE PAGES`
size of the log queue in pages

`LOG QUEUE MAX USED PAGES`
maximum number of used log queue pages

`LOG QUEUE INSERTS`
number of insert operations in the log queue

`LOG QUEUE OVERFLOWS`
number of log queue overflows

`LOG QUEUE GROUP COMMITS`
number of group commits

`LOG QUEUE WAITS FOR LOG PAGE WRITE`
number of waiting times for log write operations

`LOG QUEUE MAX WAITS PER LOG PAGE`
maximum number of waiting times per log page

`LOG QUEUE AVG WAITS PER LOG PAGE`
average number of waiting times per log page

MONITOR_PAGES

contains information about accesses to pages. The column DESCRIPTION has the following values:

`VIRTUAL  READS`
: number of virtual read operations

`VIRTUAL  WRITES`
: number of virtual write operations

`PHYSICAL READS`
: number of physical read operations

`PHYSICAL WRITES`
: number of physical write operations

`CATALOG        VIRTUAL  READ`
: number of virtual catalog read operations

`CATALOG        VIRTUAL  WRITES`
: number of virtual catalog write operations

`CATALOG        PHYSICAL READS`
: number of physical catalog read operations

`CATALOG        PHYSICAL WRITES`
: number of physical catalog write operations

`FBM PAGE       PHYSICAL READS`
: number of physically read free storage space management pages

`FBM PAGE       PHYSICAL WRITES`
: number of physically written free storage space management pages

`CONVERTER PAGE PHYSICAL READS`
: number of physically read converter pages

`CONVERTER PAGE PHYSICAL WRITES`
: number of physically written converter pages

`USM PAGE       PHYSICAL READS`
: number of physically read User Space Management pages

`USM PAGE       PHYSICAL WRITES`
: number of physically written User Space Management pages

`PERM PAGE      VIRTUAL  READS`
: number of virtually read permanent pages

`PERM PAGE      VIRTUAL  WRITES`
: number of virtually written permanent pages

`PERM PAGE      PHYSICAL READS`
: number of physically read permanent pages

`PERM PAGE      PHYSICAL WRITES`
: number of physically written permanent pages

`TEMP PAGE      VIRTUAL  READS`
: number of virtually read temporary pages

`TEMP PAGE      VIRTUAL  WRITES`
: number of virtually written temporary pages

`TEMP PAGE      PHYSICAL READS`
: number of physically read temporary pages

`TEMP PAGE      PHYSICAL WRITES`
: number of physically written temporary pages

`LEAF PAGE      VIRTUAL  READS`
: number of virtually read leaf pages

`LEAF PAGE      VIRTUAL  WRITES`

number of virtually written leaf pages

LEAF PAGE PHYSICAL READS
number of physically read leaf pages

LEAF PAGE PHYSICAL WRITES
number of physically written leaf pages

LEVEL1 PAGE VIRTUAL READS
number of virtually read index pages on level 1

LEVEL1 PAGE VIRTUAL WRITES
number of virtually written index pages on level 1

LEVEL1 PAGE PHYSICAL READS
number of physically read index pages on level 1

LEVEL1 PAGE PHYSICAL WRITES
number of physically written index pages on level 1

LEVEL2 PAGE VIRTUAL READS
number of virtually read index pages on level 2

LEVEL2 PAGE VIRTUAL WRITES
number of virtually written index pages on level 2

LEVEL2 PAGE PHYSICAL READS
number of physically read index pages on level 2

LEVEL2 PAGE PHYSICAL WRITES
number of physically written index pages on level 2

LEVEL3 PAGE VIRTUAL READS
number of virtually read index pages on level 3

LEVEL3 PAGE VIRTUAL WRITES
number of virtually written index pages on level 3

LEVEL3 PAGE PHYSICAL READS
number of physically read index pages on level 3

LEVEL3 PAGE PHYSICAL WRITES
number of physically written index pages on level 3

MONITOR_ROW

contains information about operations on row level. The column DESCRIPTION contains the following values:

BD ADD RECORD PERM
number of rows inserted in permanent tables

BD ADD RECORD TEMP
number of rows inserted in temporary tables

BD REPL RECORD PERM
number of rows updated in permanent tables

BD REPL RECORD TEMP
number of rows updated in temporary tables

BD DEL RECORD PERM
number of rows deleted from permanent tables

BD DEL RECORD TEMP

number of rows deleted from temporary tables

`BD GET  RECORD PERM`
number of rows selected from permanent tables specifying the key

`BD GET  RECORD TEMP`
number of rows selected from temporary tables specifying the key

`BD NEXT RECORD PERM`
number of rows selected from permanent tables specifying the predecessor key

`BD NEXT RECORD TEMP`
number of rows selected from temporary tables specifying the predecessor key

`BD PREV RECORD PERM`
number of rows selected from permanent tables specifying the successor key

`BD PREV RECORD TEMP`
number of rows selected from temporary tables specifying the successor key

`BD SELECT DIRECT RECORD`
number of rows selected specifying the key

`BD SELECT NEXT   RECORD`
number of rows selected specifying the predecessor key

`BD SELECT PREV   RECORD`
number of rows selected specifying the successor key

`BD ADD TO   INDEX LIST PERM`
number of insert operations in permanent indexes

`BD ADD TO   INDEX LIST TEMP`
number of insert operations in temporary indexes

`BD DEL FROM INDEX LIST PERM`
number of delete operations from permanent indexes

`BD DEL FROM INDEX LIST TEMP`
number of delete operations from temporary indexes

`BD GET      INDEX LIST PERM`
number of accesses to permanent indexes

`BD GET      INDEX LIST TEMP`
number of accesses to temporary indexes

MONITOR_SERVERDB

contains information about the ADABAS sender and receiver processes. The column DESCRIPTION contains the following values:

`DISTRIBUTION MESSAGES RECEIVED`
number of orders received from remote SERVERDBs

`DISTRIBUTION MESSAGES SENT`
number of orders sent to remote SERVERDBs

`DISTRIBUTION MESSAGES DELAYED`
number of orders received from remote SERVERDBs which could not be handled immediately

`DISTRIBUTION SERVER JOBS`
number of server jobs

`DISTRIBUTION MESSAGE DESCR CACHE OVERFLW`

number of overflows of the message description cache

DISTRIBUTION MESSAGE CACHE OVERFLOWS
number of overflows of the message cache

MONITOR_TRANS

contains information about transactions. The column DESCRIPTION contains the following values:

SQL COMMANDS
number of SQL statements

WRITE TRANSACTIONS
number of transactions with modifying operations

KB CALLS
number of KB orders

MONITOR_VTRACE

contains information about the vtrace output. The column DESCRIPTION contains the following values:

VTRACE I/O OPERATIONS
number of vtrace output operations

VTRACE I/O OPERATIONS LOCKED
number of delayed vtrace output operations

MONITOR

This table is the combination of all monitor tables described so far.

Restrictions

Maximum values :

Number of tables		unlimited
Length of an identifier	18	characters
Internal length of a table row	4047	characters
Length of a LONG column	2147483647	characters
Columns per table (with KEY)	255	columns
Columns per table (without KEY)	254	columns
Number of key columns	127	columns
Precision of numeric values	18	digits
Sum of internal lengths of all key columns	255	characters
Sum of internal lengths of all columns belonging to an index	255	characters
Length of sort columns in SELECT	250	characters
Number of result columns	254	columns
Number of join tables in SELECT	16	tables
Number of join conditions in a WHERE clause of a SELECT	64	
Number of named indexes per table	256	
Number of correlated columns in an SQL statement	64	
Number of correlated tables in an SQL statement	16	
Number of SERVERDBs in a distributed database	2048	SERVERDBs
Number of DEVSPACEs	64	DEVSPACEs
Length of an SQL statement	8240	characters
Number of parameters in an SQL statement	300	parameters

Compatibility with Former Versions

1. The specification of the SQLMODE SQL-DB in the <connect statement> is still possible.

2. A <range spec> in the following format can be specified instead of a <constraint definition> in the <create table statement>:

```
 <range spec> ::=
     RANGE [NOT] BETWEEN <literal> AND <literal>
     RANGE [NOT] IN (<value spec>,...)
```

   If a <range spec> is specified for an optional column, the <constraint definition> defined by it implicitly contains the NULL value. If this effect is not desired, NOT NULL must be specified in addition to the <range spec>. If a <default spec> was specified in addition, the <default value> must satisfy the <range spec>.

3. Instead of the <isolation spec>, the specifications LOCK EXPLICIT, LOCK NORMAL, and LOCK IMPLICIT are allowed.

   - LOCK EXPLICIT corresponds to ISOLATION LEVEL 0.
   - LOCK NORMAL corresponds to ISOLATION LEVEL 15.
   - LOCK IMPLICIT corresponds to ISOLATION LEVEL 2 with the restriction that no table SHARE locks are set during the execution of an <sql statement>.

4. The <sql statement>s CREATE LINK and DROP LINK are still available. In contrast to former versions, the <referential constraint name> (link name) must be unique together with the name of the referencing table , no longer with the name of the <referenced table>.

<create link statement>

Function

defines existence conditions between the rows of two tables.

Format

```
 <create link statement> ::=
     CREATE LINK <referential constraint name>
     FOREIGN KEY <referencing table>
     (<referencing column>,...)
     <references spec>
     [<delete rule>]
```

Syntax Rules

none

General Rules

1. Executing the <create link statement> has the same effect as defining a corresponding <referential constraint definition> in the <create table statement> or an <alter table statement> of the referencing table.

2. The same rules which are valid for a <referential constraint definition> apply to the <create link statement>.

3. The <referential constraint name> must be different from all existing <referential constraint name>s of the referencing table.

4. Each row R of the referencing table must satisfy one of the following conditions:

   i) R is the matching row of the <referential constraint definition>.
   ii) R contains the NULL value in one of the columns of the <referencing column>s.
   iii) The <delete rule> defines ON DELETE SET DEFAULT and R contains the default value in all columns of the <referencing column>s.

<drop link statement>

Function

drops a <referential constraint definition> between two tables.

Format

```
<drop link statement> ::=
    DROP LINK <referential constraint name>
    REFERENCES <referenced table>
```

Syntax Rules

none

General Rules

1. The user must be the owner of one of the two tables linked by the <referential constraint definition>, and the user must have the REFERENCES privilege on the corresponding table.

2. The meta data of the specified <referential constraint definition> is dropped from the catalog.

3. As <referential constraint definition>s are required for the updatability of join

view tables, dropping a <referential constraint definition> can have the effect that a view table based on the <referenced table> and the referencing table can no longer be updated.

4. The <sql statement>s for catalog and statistical information are still available. This chapter contains a list of the <query statement>s that, issued on the system tables, should be used to replace the <sql statement>s for catalog and statistical information.

   Note that the names of tables, domains, users, etc., must be enclosed in single quotation marks. Names specified as <simple identifier>s must be specified in uppercase characters. Names specified as <special identifier>s are entered without enclosing <double quotes> in the desired combination of upper- and lowercases. If <double quotes> belong to the <special identifier>, they are not doubled on input.

   In the following list, a distinction is made between examples of catalog information determining a set of objects (list) and examples determining the structure or definition of just one object (structure or definition).

   The structure of the statistical information result tables frequently consisted of a row that contained a DESCRIPTION and the value belonging to this description. For some of these informative functions, system tables are provided now that contain the complete information in one row in appropriately named columns. In the following list, the attempt was made to specify a <query statement> that does not modify the structure of the result tables. As information coming from one row must be split into several rows, the <query statement> is quite complicated. If it is not necessary to keep the structure of the result tables used so far, the simplified formats of the <query statement>s should be used.

COLUMN

```
List
-------------------------------------------------------------------
SHOW COLUMN                 SELECT *
<owner>.<table name>.       FROM DOMAIN.COLUMNS
           <column name>    WHERE owner = <owner>
                            AND tablename = <table name>
                            AND columnname = <column name>
-------------------------------------------------------------------
```

CONNECT PARAM

```
List
-------------------------------------------------------------------
SHOW CONNECT PARAM          SELECT *
                            FROM DOMAIN.CONNECTPARAMETERS
-------------------------------------------------------------------
```

CONSTRAINT

```
List
-------------------------------------------------------------------
SHOW CONSTRAINT             SELECT *
```

```
                          FROM DOMAIN.CONSTRAINTS

SHOW CONSTRAINT           SELECT *
<owner>.<table name>      FROM DOMAIN.CONSTRAINTS
                          WHERE owner LIKE <owner>
                          AND tablename LIKE <table name>
-------------------------------------------------------------------

Definition
-------------------------------------------------------------------
SHOW CHECK                SELECT definition
<owner>.<table name>.     FROM DOMAIN.CONSTRAINTS
       <constraint name>  WHERE owner LIKE <owner>
                          AND tablename LIKE <table name>
                          AND constraintname LIKE <constraint name>
-------------------------------------------------------------------
```

DBPROCEDURE

```
List
-------------------------------------------------------------------
SHOW DBPROCEDURE          SELECT *
<owner>.<program name>.   FROM DOMAIN.DBPROCEDURES
        <procedure name>  WHERE owner LIKE <owner>
                          AND programname LIKE <program name>
                          AND dbprocname LIKE <procedure name>
-------------------------------------------------------------------

Parameter
-------------------------------------------------------------------
SHOW PARAM DBPROC         SELECT *
<owner>.<program name>.   FROM DOMAIN.DBPROCPARAMS
        <procedure name>  WHERE owner = <owner>
                          AND programname = <program name>
                          AND dbprocname = <procedure name>
-------------------------------------------------------------------
```

DOMAIN

```
List
-------------------------------------------------------------------
SHOW DOMAIN               SELECT *
                          FROM DOMAIN.DOMAINS

SHOW DOMAIN               SELECT *
<domain name>             FROM DOMAIN.DOMAINS
                          WHERE domainname LIKE <domain name>
-------------------------------------------------------------------

Definition
-------------------------------------------------------------------
SHOW DOMAINDEF            SELECT definition
<domain name>             FROM DOMAIN.DOMAINS
                          WHERE domainname = <domain name>
-------------------------------------------------------------------

Domain Constraint
-------------------------------------------------------------------
SHOW CHECK                SELECT definition
<domain name>             FROM DOMAIN.DOMAINCONSTRAINTS
```

```
                          WHERE domainname = <domain name>
-------------------------------------------------------------------
```

FOREIGN KEY

```
List
-------------------------------------------------------------------
SHOW FOREIGN KEY          SELECT defowner owner,
                                 deftablename tablename,
                                 defcolumnname columnname,
                                 defrefname refname,
                                 refowner,
                                 reftablename,
                                 refcolumnname,
                                 rule,
                                 createdate "DATE",
                                 createtime "TIME",
                                 comment
                          FROM DOMAIN.FKC_REFS_COL

SHOW FOREIGN KEY          SELECT defowner owner,
<owner>.<table name>             deftablename tablename,
                                 defcolumnname columnname,
                                 defrefname refname,
                                 refowner,
                                 reftablename,
                                 refcolumnname,
                                 rule,
                                 createdate "DATE",
                                 createtime "TIME",
                                 comment
                          FROM DOMAIN.FKC_REFS_COL
                          WHERE defowner = <owner>
                          AND deftablename LIKE <table name>
-------------------------------------------------------------------
```

INDEX

```
List
-------------------------------------------------------------------
SHOW INDEX                SELECT defowner owner,
                                 deftablename tablename,
                                 defindexname indexname,
                                 type,
                                 refcolumnname columnname,
                                 pos,
                                 sort,
                                 createdate "DATE",
                                 createtime "TIME",
                                 comment
                          FROM DOMAIN.IND_USES_COL
                          ORDER BY owner,
                                   tablename,
                                   indexname,
                                   pos

SHOW INDEX                SELECT defowner owner,
<owner>.<table name>             deftablename tablename,
                                 defindexname indexname,
                                 type,
                                 refcolumnname columnname,
                                 pos,
                                 sort,
                                 createdate "DATE",
                                 createtime "TIME",
                                 comment
                          FROM DOMAIN.IND_USES_COL
                          WHERE defowner = <owner>
                          AND deftablename LIKE <table name>
                          ORDER BY owner,
                                   tablename,
                                   indexname,
                                   pos
--------------------------------------------------------------------
```

MAPCHARSET

```
List
--------------------------------------------------------------------
SHOW MAPCHARSET               SELECT *
                              FROM DOMAIN.MAPCHARSETS

SHOW MAPCHARSET               SELECT *
<mapcharset name>             FROM DOMAIN.MAPCHARSETS
                              WHERE mapcharsetname LIKE
                                                   <mapcharset name>
--------------------------------------------------------------------
```

PRIMARY KEY

```
List
--------------------------------------------------------------------
SHOW PRIMARY KEY OF           SELECT *
<owner>.<table name>          FROM DOMAIN.COLUMNS
                              WHERE owner = <owner>
                              AND tablename = <table name>
                              AND keypos IS NOT NULL
                              ORDER BY keypos
--------------------------------------------------------------------
```

PRIVILEGES

```
List
--------------------------------------------------------------------
SHOW PRIV GRANTED             SELECT refowner owner,
TO <user name>                       reftablename tablename,
ON <owner>.<table name>              refcolumnname columnname,
                                     privileges,
                                     defusername grantor
                              FROM DOMAIN.USR_USES_COL
                              WHERE defusername LIKE <user name>
                              AND refowner LIKE <owner>
                              AND reftablename LIKE <table name>

SHOW PRIV ON                  SELECT refowner owner,
<owner>.<table name>                 reftablename tablename,
                              refcolumnname columnname,
                              privileges,
                              defusername grantor
                          FROM DOMAIN.USR_USES_COL
                          WHERE defusername = USERGROUP
                          AND refowner LIKE <owner>
                          AND reftablename LIKE <table name>
-------------------------------------------------------------------
```

SERVERDB

```
List
-------------------------------------------------------------------
SHOW SERVERDB             SELECT *
                          FROM DOMAIN.SERVERDBS

SHOW SERVERDB             SELECT *
<serverdb name>           FROM DOMAIN.SERVERDBS
                          WHERE serverdb LIKE <serverdb name>
-------------------------------------------------------------------
```

SYNONYM

```
List
-------------------------------------------------------------------
SHOW SYNONYM              SELECT defsynonymname synonymname,
                                 refowner owner,
                                 reftablename tablename
                          FROM DOMAIN.SYN_REFS_TAB

SHOW SYNONYM              SELECT defsynonymname synonymname,
<synonym name>                   refowner owner,
                                 reftablename tablename
                          FROM DOMAIN.SYN_REFS_TAB
                          WHERE defsynonymname LIKE <synonym name>
-------------------------------------------------------------------
```

SYSDBA

```
List
-------------------------------------------------------------------
SHOW SYSDBA               SELECT SYSDBA
                          FROM LOCALSYSDBA.DUAL

SHOW SYSDBA               SELECT SYSDBA (<user name>)
OF <user name>            FROM LOCALSYSDBA.DUAL
-------------------------------------------------------------------
```

TABLE

```
List
-------------------------------------------------------------------
SHOW TABLE                SELECT *
                          FROM DOMAIN.TABLES
                          ORDER BY owner,tablename
```

```
SHOW TABLE                SELECT *
<owner>.<table name>      FROM DOMAIN.TABLES
                          WHERE owner LIKE <owner>
                          AND tablename LIKE <table name>
-------------------------------------------------------------------

Structure
-------------------------------------------------------------------
SHOW TABLEDEF             SELECT *
<owner>.<table name>      FROM DOMAIN.COLUMNS
                          WHERE owner = <owner>
                          AND tablename = <table name>
                          ORDER BY pos
-------------------------------------------------------------------
```

TERMCHARSET

```
List
-------------------------------------------------------------------
SHOW TERMCHARSET          SELECT *
                          FROM DOMAIN.TERMCHARSETS

SHOW TERMCHARSET          SELECT *
<termcharset name>        FROM DOMAIN.TERMCHARSETS
                          WHERE termcharsetname LIKE
                                                <termcharset name>
-------------------------------------------------------------------
```

TRIGGER

```
List
-------------------------------------------------------------------
SHOW TRIGGER              SELECT *
                          FROM DOMAIN.TRIGGERS

SHOW TRIGGER              SELECT *
<owner>.<table name>.     FROM DOMAIN.TRIGGERS
          <trigger name>  WHERE owner LIKE <owner>
                          AND tablename LIKE <table name>
                          AND triggername LIKE <trigger name>

SHOW TRIGGER              SELECT *
<trigger name>            FROM DOMAIN.TRIGGERS
OF <owner>.<table name>   WHERE owner LIKE <owner>
                          AND tablename LIKE <table name>
                          AND triggername LIKE <trigger name>
-------------------------------------------------------------------

Definition
-------------------------------------------------------------------
SHOW TRIGGERDEF           SELECT definition
<trigger name>            FROM DOMAIN.TRIGGERS
OF <owner>.<table name>   WHERE owner = <owner>
                          AND tablename = <table name>
                          AND triggername = <trigger name>
-------------------------------------------------------------------

Parameter
-------------------------------------------------------------------
SHOW PARAM TRIGGER        SELECT *
```

```
<trigger name>             FROM DOMAIN.TRIGGERPARAMS
OF <owner>.<table name>    WHERE owner = <owner>
                           AND tablename = <table name>
                           AND triggername = <trigger name>
-------------------------------------------------------------------
```

USER

List

```
-------------------------------------------------------------------
SHOW USER                  SELECT *
                           FROM DOMAIN.USERS

SHOW USER <user name>      SELECT *
                           FROM DOMAIN.USERS
                           WHERE username LIKE <user name>
                           OR groupname LIKE <user name>

SHOW USER CURRENT          SELECT *
                           FROM DOMAIN.USERS
                           WHERE ((username = ' '
                           AND groupname = USERGROUP)
                           OR username = USERGROUP)
-------------------------------------------------------------------
```

USER CONNECTED

List

```
-------------------------------------------------------------------
SHOW USER CONNECTED        SELECT *
                           FROM DOMAIN.CONNECTEDUSERS
-------------------------------------------------------------------
```

VERSION

List

```
-------------------------------------------------------------------
SHOW VERSION               SELECT *
                           FROM DOMAIN.VERSIONS
-------------------------------------------------------------------
```

VIEW

List

```
-------------------------------------------------------------------
SHOW TABLE                 SELECT *
                           FROM DOMAIN.VIEWS
                           ORDER BY owner,tablename

SHOW TABLE                 SELECT *
<owner>.<table name>       FROM DOMAIN.VIEWS
                           WHERE owner LIKE <owner>
                           AND tablename LIKE <table name>
-------------------------------------------------------------------
```

Structure

```
-------------------------------------------------------------------
SHOW TABLEDEF             SELECT *
<owner>.<table name>      FROM DOMAIN.COLUMNS
                          WHERE owner = <owner>
                          AND tablename = <table name>
                          ORDER BY pos
-------------------------------------------------------------------
```

Definition

```
-------------------------------------------------------------------
SHOW VIEW                 SELECT definition
<owner>.<table name>      FROM DOMAIN.VIEWDEFS
                          WHERE owner = <owner>
                          AND tablename = <table name>
-------------------------------------------------------------------
```

OPTIMIZE STATISTICS

List

```
-------------------------------------------------------------------
SHOW OPTIMIZE STATISTICS  SELECT columnname,indexname,
<owner>.<table name>      distinctvalues,pagecount,avglistlength
                          FROM FROM SYSDBA.OPTIMIZERSTATISTICS
                          WHERE owner = <owner>
                          AND tablename LIKE <table name>
-------------------------------------------------------------------
```

STATISTICS CONFIGURATION

List

```
-------------------------------------------------------------------
SHOW STATISTICS CONFIG    SELECT SUBSTR(DESCRIPTION,1,40),
                                 DECODE(CHAR_VALUE,NULL,
                                 LFILL(CHR(NUMERIC_VALUE),' ',12),
                                 SUBSTR(CHAR_VALUE,1,40))
                          FROM SYSDBA.CONFIGURATION
-------------------------------------------------------------------
```

STATISTICS DEVSPACE

List

```
-------------------------------------------------------------------
SHOW STATISTICS DEVSPACE  SELECT SUBSTR('PAGES',1,40),
<devspace name>                  FIXED(DEVSPACESIZE,12)
                          FROM SYSDBA.DATADEVSPACES
                          WHERE devspacename LIKE <devspace name>
                          UNION ALL
                          SELECT 'LAST DATA PAGE NO',
                                 MAXDATAPAGENO
                          FROM SYSDBA.DATADEVSPACES
                          WHERE devspacename LIKE <devspace name>
                          UNION ALL
                          SELECT 'USED PERM PAGES',
                                 USEDPERMPAGES
                          FROM SYSDBA.DATADEVSPACES
                          WHERE devspacename LIKE <devspace name>
                          UNION ALL
                          SELECT 'USED PERM PAGES (%)',
                                 PCTUSEDPERM
                          FROM SYSDBA.DATADEVSPACES
                          WHERE devspacename LIKE <devspace name>
                          UNION ALL
                          SELECT 'USED TEMP PAGES',
                                 USEDTMPPAGES
                          FROM SYSDBA.DATADEVSPACES
                          WHERE devspacename LIKE <devspace name>
                          UNION ALL
                          SELECT 'USED TEMP PAGES (%)',
                                 PCTUSEDTMP
                          FROM SYSDBA.DATADEVSPACES
                          WHERE devspacename LIKE <devspace name>
                          UNION ALL

                          SELECT 'UNUSED    PAGES',
                                 UNUSEDPAGES
                          FROM SYSDBA.DATADEVSPACES
                          WHERE devspacename LIKE <devspace name>
                          UNION ALL
                          SELECT 'UNUSED    PAGES (%)',
                                 PCTUNUSED
                          FROM SYSDBA.DATADEVSPACES
                          WHERE devspacename LIKE <devspace name>

SHOW STATISTICS DEVSPACE  SELECT DEVSPACESIZE,MAXDATAPAGENO,
<devspace name>                  USEDPERMPAGES,PCTUSEDPERM,
                                 USEDTMPPAGES,PCTUSEDTMP,
                                 UNUSEDPAGES,PCTUNUSED
                          FROM SYSDBA.DATADEVSPACES
                          WHERE devspacename LIKE <devspace name>
--------------------------------------------------------------------
```

```
 <devspace name> ::=
     <string literal>
```

STATISTICS INDEX

List

```
--------------------------------------------------------------------
SHOW STATISTICS INDEX     SELECT SUBSTR(DESCRIPTION,1,40),
<owner>.<table name>.            DECODE(CHAR_VALUE,NULL,
           <column name>         LFILL(CHR(NUMERIC_VALUE),' ',12),
                                 SUBSTR(CHAR_VALUE,1,40))
                          FROM SYSDBA.INDEXSTATISTICS
                          WHERE owner = <owner>
                          AND tablename LIKE <table name>
                          AND columnname LIKE <column name>

SHOW STATISTICS INDEX     SELECT SUBSTR(DESCRIPTION,1,40),
<index name> OF                  DECODE(CHAR_VALUE,NULL,
<owner>.<table name>             LFILL(CHR(NUMERIC_VALUE),' ',12),
                                 SUBSTR(CHAR_VALUE,1,40))
                          FROM SYSDBA.INDEXSTATISTICS
                          WHERE owner = <owner>
                          AND tablename LIKE <table name>
                          AND indexname LIKE <index name>
--------------------------------------------------------------------
```

STATISTICS LOCK

```
List
-------------------------------------------------------------------
SHOW STATISTICS LOCK      SELECT
                          OWNER,TABLENAME,ROWIDLENGTH,ROWIDHEX,
                          DECODE(LOCKMODE,NULL,LOCKREQUESTMODE,
                          LOCKMODE) LOCKMODE,PENDINGLOCK,
                          SERVERDBNO,SESSION,TRANSACTION,
                          DECODE(REMOTEUSER,'YES','<remote>',
                          USERNAME) USERNAME,TERMID,PROCESS
                          FROM SYSDBA.LOCKSTATISTICS

SHOW STATISTICS           SELECT *
LOCK CONFIG               FROM SYSDBA.LOCKLISTSTATISTICS

SHOW STATISTICS           SELECT DISTINCT OWNER,TABLENAME,
LOCK TABLE                DECODE(LOCKMODE,NULL,LOCKREQUESTMODE,
<owner>.<table name>      LOCKMODE) LOCKMODE,PENDINGLOCK,
                          SERVERDBNO,SESSION,TRANSACTION,
                          DECODE(REMOTEUSER,'YES','<remote>',
                          USERNAME) USERNAME,TERMID,PROCESS
                          FROM SYSDBA.LOCKSTATISTICS
                          WHERE owner LIKE <owner>
                          AND tablename LIKE <table name>

SHOW STATISTICS           SELECT SERVERDBNO,SESSION,TRANSACTION,
LOCK USER                 DECODE(REMOTEUSER,'YES','<remote>',
                          USERNAME) USERNAME,TERMID,PROCESS,
                          DECODE(LOCKMODE,NULL,LOCKREQUESTMODE,
                          LOCKMODE) LOCKMODE,PENDINGLOCK
                          FROM SYSDBA.TRANSACTIONS
-------------------------------------------------------------------
```

STATISTICS LOG

```
List
-------------------------------------------------------------------
SHOW STATISTICS LOG       SELECT SUBSTR(DESCRIPTION,1,40),
                          SUBSTR(CHAR_VALUE,1,12)
                          FROM SYSDBA.CONFIGURATION
                          WHERE DESCRIPTION = 'LOG MODE'
                          UNION ALL
                          SELECT 'LOG PAGES',
                          LFILL(CHR(LOGSIZE),' ',12)
                          FROM SYSDBA.SERVERDBSTATISTICS
                          UNION ALL

                          SELECT 'USED     LOG PAGES',
                          LFILL(CHR(USEDLOGPAGES),' ',12)
                          FROM SYSDBA.SERVERDBSTATISTICS
                          UNION ALL
                          SELECT 'USED     LOG PAGES (%)',
                          LFILL(CHR(PCTUSEDLOGPAGES),' ',12)
                          FROM SYSDBA.SERVERDBSTATISTICS
                          UNION ALL
                          SELECT 'UNUSED   LOG PAGES',
                          LFILL(CHR(UNUSEDLOGPAGES),' ',12)
                          FROM SYSDBA.SERVERDBSTATISTICS
                          UNION ALL
                          SELECT 'UNUSED   LOG PAGES (%)',
                          LFILL(CHR(PCTUNUSEDLOGPAGES),' ',12)
                          FROM SYSDBA.SERVERDBSTATISTICS
                          UNION ALL
                          SELECT 'RESERVED LOG PAGES',
                          LFILL(CHR(RESERVEDLOGPAGES),' ',12)
                          FROM SYSDBA.SERVERDBSTATISTICS
                          UNION ALL
                          SELECT 'LOG SEGMENT SIZE',
                          LFILL(CHR(LOGSEGMENTSIZE),' ',12)
                          FROM SYSDBA.SERVERDBSTATISTICS
                          UNION ALL
                          SELECT 'LOG SEGMENTS COMPLETED',
                          LFILL(CHR(COMPLETESEGMENTS),' ',12)
                          FROM SYSDBA.SERVERDBSTATISTICS
                          UNION ALL
                          SELECT 'SAVEPOINTS',
                          LFILL(CHR(SAVEPOINTS),' ',12)
                          FROM SYSDBA.SERVERDBSTATISTICS
                          UNION ALL
                          SELECT 'CHECKPOINTS',
                          LFILL(CHR(CHECKPOINTS),' ',12)
                          FROM SYSDBA.SERVERDBSTATISTICS
                          UNION ALL
                          SELECT 'LOG PAGES PER SAVEPOINT',
                          LFILL(CHR(PAGESPERSAVEPOINT),' ',12)
                          FROM SYSDBA.SERVERDBSTATISTICS
                          UNION ALL
                          SELECT 'LOG PAGES PER CHECKPOINT',
                          LFILL(CHR(PAGESPERCHECKPOINT),' ',12)
                          FROM SYSDBA.SERVERDBSTATISTICS


SHOW STATISTICS LOG       SELECT
                          CHAR_VALUE,LOGSIZE,USEDLOGPAGES,
                          PCTUSEDLOGPAGES,UNUSEDLOGPAGES,
                          PCTUNUSEDLOGPAGES,RESERVEDLOGPAGES,
                          LOGSEGMENTSIZE,COMPLETESEGMENTS,
                          SAVEPOINTS,CHECKPOINTS,
                          PAGESPERSAVEPOINT,PAGESPERCHECKPOINT
                          FROM SYSDBA.SERVERDBSTATISTICS,
                               SYSDBA.CONFIGURATION
                          WHERE DESCRIPTION = 'LOG MODE'
-------------------------------------------------------------------
```

STATISTICS MAPCHAR SET

```
List
-------------------------------------------------------------------
SHOW STATISTICS           SELECT INTERN,"MAP CODE","MAP CHARACTER"
MAPCHAR SET               FROM DOMAIN.MAPCHARSETS
<mapcharset name>         WHERE mapcharsetname LIKE
                                               <mapcharset name>
-------------------------------------------------------------------
```

STATISTICS SERVERDB

```
List
-------------------------------------------------------------------
SHOW STATISTICS SERVERDB  SELECT SUBSTR('PAGES',1,40),
                                 FIXED(SERVERDBSIZE,12)
                          FROM SYSDBA.SERVERDBSTATISTICS
                          UNION ALL
                          SELECT 'MAX DATA  PAGE NO',
                                 MAXDATAPAGENO
                          FROM SYSDBA.SERVERDBSTATISTICS
                          UNION ALL
                          SELECT 'USED PERM PAGES',
                                 USEDPERMPAGES
                          FROM SYSDBA.SERVERDBSTATISTICS
                          UNION ALL
                          SELECT 'USED PERM PAGES (%)',
                                 PCTUSEDPERM
                          FROM SYSDBA.SERVERDBSTATISTICS
                          UNION ALL
                          SELECT 'USED TEMP PAGES',
                                 USEDTMPPAGES
                          FROM SYSDBA.SERVERDBSTATISTICS
                          UNION ALL
                          SELECT 'USED TEMP PAGES (%)',
                                 PCTUSEDTMP
                          FROM SYSDBA.SERVERDBSTATISTICS
                          UNION ALL
                          SELECT 'UNUSED    PAGES',
                                 UNUSEDPAGES
                          FROM SYSDBA.SERVERDBSTATISTICS
                          UNION ALL
                          SELECT 'UNUSED    PAGES (%)',
                                 PCTUNUSED
                          FROM SYSDBA.SERVERDBSTATISTICS
                          UNION ALL
                          SELECT 'UPDATED PERM PAGES',
                                 UPDATEDPERMPAGES
                          FROM SYSDBA.SERVERDBSTATISTICS

SHOW STATISTICS SERVERDB  SELECT SERVERDBSIZE,MAXDATAPAGENO,
                                 USEDPERMPAGES,PCTUSEDPERM,
                                 USEDTMPPAGES,PCTUSEDTMP,
                                 UNUSEDPAGES,PCTUNUSED,
                                 UPDATEDPERMPAGES
                          FROM SYSDBA.SERVERDBSTATISTICS
-------------------------------------------------------------------
```

STATISTICS TABLE

List

```
-------------------------------------------------------------------
SHOW STATISTICS TABLE     SELECT SUBSTR(DESCRIPTION,1,40),
<owner>.<table name>             DECODE(CHAR_VALUE,NULL,
                                 LFILL(CHR(NUMERIC_VALUE),' ',12),
                                 SUBSTR(CHAR_VALUE,1,40))
                          FROM SYSDBA.TABLESTATISTICS
                          WHERE owner = <owner>
                          AND tablename LIKE <table name>
-------------------------------------------------------------------
```

STATISTICS TERMCHAR SET

List

```
-------------------------------------------------------------------
SHOW STATISTICS           SELECT *
TERMCHAR SET              FROM DOMAIN.TERMCHARSETS
```

```
SHOW STATISTICS           SELECT *
TERMCHAR SET              FROM DOMAIN.TERMCHARSETS
<termcharset name>        WHERE termcharsetname LIKE
                                               <termcharset name>
-------------------------------------------------------------------
```

STATISTICS USER

```
List
-------------------------------------------------------------------
SHOW STATISTICS USER      SELECT *
<user name>               FROM SYSDBA.USERSTATISTICS
                          WHERE username LIKE <user name>
-------------------------------------------------------------------
```

MONITOR

```
List
-------------------------------------------------------------------
SHOW MONITOR ALL          SELECT *
                          FROM SYSDBA.MONITOR

SHOW MONITOR CACHES       SELECT *
                          FROM SYSDBA.MONITOR_CACHES

SHOW MONITOR DBPROC       SELECT *
                          FROM SYSDBA.MONITOR_DBPROC

SHOW MONITOR LOAD         SELECT *
                          FROM SYSDBA.MONITOR_LOAD

SHOW MONITOR LOCK         SELECT *
                          FROM SYSDBA.MONITOR_LOCK

SHOW MONITOR LOG          SELECT *
                          FROM SYSDBA.MONITOR_LOG

SHOW MONITOR PAGES        SELECT *
                          FROM SYSDBA.MONITOR_PAGES

SHOW MONITOR ROW          SELECT *
                          FROM SYSDBA.MONITOR_ROW

SHOW MONITOR SERVERDB     SELECT *
                          FROM SYSDBA.MONITOR_SERVERDB

SHOW MONITOR TRANSACTION  SELECT *
                          FROM SYSDBA.MONITOR_TRANS

SHOW MONITOR VTRACE       SELECT *
                          FROM SYSDBA.MONITOR_VTRACE
-------------------------------------------------------------------
```

Syntax

```
<add definition> ::=
     ADD  <column definition>,...
   | ADD (<column definition>,...)
   | ADD <constraint definition>
   | ADD <key definition>
   | ADD REPLICATION

<alias name> ::=
     <identifier>

<all function> ::=
     <set function name> ( [ALL] <expression> )

<alter data type> ::=
     <data type>
   | <domain name>

<alter definition> ::=
     COLUMN <column name> <alter data type>
   | COLUMN <column name> NOT     NULL
   | COLUMN <column name> DEFAULT NULL
   | COLUMN <column name> ADD   <default spec>
   | COLUMN <column name> ALTER <default spec>
   | COLUMN <column name> DROP DEFAULT
   | ALTER CONSTRAINT <constraint name> CHECK <search condition>
   | ALTER <key definition>

<alter password statement> ::=
     ALTER PASSWORD <old password> TO <new password>
   | ALTER PASSWORD <user name> <new password>

<alter table statement> ::=
     ALTER TABLE <table name> <add definition>
   | ALTER TABLE <table name> <drop definition>
   | ALTER TABLE <table name> <alter definition>
   | ALTER TABLE <table name> <referential constraint definition>
   | ALTER TABLE <table name> DROP FOREIGN KEY
     <referential constraint name>


<alter user statement> ::=
     ALTER USER <user name> [<user mode>]
            [PERMLIMIT   <altered value>]
            [TEMPLIMIT   <altered value>]
            [TIMEOUT     <altered value>]
            [COSTWARNING <altered value>]
            [COSTLIMIT   <altered value>]
            [CACHELIMIT  <altered value>]
            [[NOT] EXCLUSIVE]

<alter usergroup statement> ::=
     ALTER USERGROUP <usergroup name> [<usergroup mode>]
            [PERMLIMIT   <altered value>]
            [TEMPLIMIT   <altered value>]
            [TIMEOUT     <altered value>]
            [COSTWARNING <altered value>]
            [COSTLIMIT   <altered value>]
            [CACHELIMIT  <altered value>]
            [[NOT] EXCLUSIVE]
```

```
<altered value> ::=
     <unsigned integer>
   | NULL

<arithmetic function> ::=
     TRUNC   ( <expression>[, <expression>] )
   | ROUND   ( <expression>[, <expression>] )
   | NOROUND ( <expression> )
   | FIXED   ( <expression>[, <unsigned integer>
               [, <unsigned integer>] ] )
   | CEIL    ( <expression> )
   | FLOOR   ( <expression> )
   | SIGN    ( <expression> )
   | ABS     ( <expression> )
   | POWER   ( <expression>, <expression> )
   | EXP     ( <expression> )
   | SQRT    ( <expression> )
   | LN      ( <expression> )
   | LOG     ( <expression>, <expression> )
   | PI
   | LENGTH  ( <expression> )
   | INDEX   ( <string spec>, <string spec>
               [,<expression>[, <expression>] ] )

<between predicate> ::=
     <expression> [NOT] BETWEEN <expression> AND <expression>

<bool predicate> ::=
     <column spec> [ IS [NOT] <bool spec> ]

<bool spec> ::=
     TRUE
   | FALSE

<boolean factor> ::=
     [NOT] <boolean primary>

<boolean primary> ::=
     <predicate>
   | (<search condition>)

<boolean term> ::=
     <boolean factor>
   | <boolean term> AND <boolean factor>

<cascade option> ::=
      CASCADE
    | RESTRICT

<character> ::=
     <digit>
   | <letter>
   | <extended letter>
   | <hex digit>
   | <language specific character>
   | <special character>

<check expression> ::=
     <expression>

<clear snapshot log statement> ::=
     CLEAR SNAPSHOT LOG ON <table name>
```

```
<close statement> ::=
     CLOSE [<result table name>]

<code spec> ::=
     ASCII
   | EBCDIC
   | BYTE

<column attributes> ::=
     [<key or not null spec>]
     [<default spec>]
     [<constraint definition>]
     [REFERENCES <table name> [(column name)]]
     [UNIQUE]


<column definition> ::=
     <column name> <data type> <column attributes>
   | <column name> <domain name> [<key or not null spec>]

<column name> ::=
     <identifier>

<column spec> ::=
     <column name>
   | <table name>.<column name>
   | <reference name>.<column name>
   | <result table name>.<column name>

<comment> ::=
     <string literal>
   | <parameter name>

<comment on statement> ::=
     COMMENT ON <object spec> IS <comment>

<commit statement> ::=
     COMMIT [WORK] [KEEP <lock statement>]

<comp op> ::=
     < | > | <> | != | = | <= | >=
   | ¬= | ¬< | ¬> for a computer with the code type EBCDIC
   | ~= | ~< | ~> for a computer with the code type ASCII

<comparison predicate> ::=
     <expression> <comp op> <expression>
   | <expression> <comp op> <subquery>
   | <expression list> <equal or not> (<expression list>)
   | <expression list> <equal or not> <subquery>

<complement sign> ::=
     ^
   | ~
   | ¬

<connect statement> ::=
     CONNECT <user spec>
     IDENTIFIED BY <password spec>
     [SQLMODE <sqlmode spec>]
     [<isolation spec>]
     [TIMEOUT <unsigned integer>]
     [CACHELIMIT <unsigned integer>]
     [TERMCHAR SET <termchar set name>]
```

```
<constraint definition> ::=
     CHECK      <search condition>
   | CONSTRAINT <search condition>
   | CONSTRAINT <constraint name> CHECK <search condition>

<constraint name> ::=
     <identifier>

<conversion function> ::=
     NUM       ( <expression> )
   | CHR       ( <expression>[, <unsigned  integer> ] )
   | HEX       ( <expression> )
   | CHAR      ( <expression>[, <datetimeformat> ] )

<create domain statement> ::=
     CREATE DOMAIN <domain name> <data type>
     [<default spec>] [<constraint definition>]

<create index statement> ::=
     CREATE [UNIQUE] INDEX <index spec>

<create snapshot log statement> ::=
     CREATE SNAPSHOT LOG ON <table name>

<create snapshot statement> ::=
     CREATE SNAPSHOT <table name> [(<alias name>,...)]
     AS <query expression>

<create synonym statement> ::=
     CREATE SYNONYM [<owner>.]<synonym name> FOR <table name>

<create table statement> ::=
     CREATE TABLE <table name> [(<table description element>,...)]
           [<table option>]
           [AS <query expression> [<duplicates clause>] ]
   | CREATE TABLE <table name> LIKE <source table>
             [<table option>]

<create user statement> ::=
     CREATE USER <user name> PASSWORD <password>
            [<user mode>]
            [PERMLIMIT   <unsigned integer>]
            [TEMPLIMIT   <unsigned integer>]
            [TIMEOUT     <unsigned integer>]
            [COSTWARNING <unsigned integer>]
            [COSTLIMIT   <unsigned integer>]
            [CACHELIMIT  <unsigned integer>]
            [[NOT] EXCLUSIVE]
            [AT <serverdb name>]

   | CREATE USER <like user> PASSWORD <password>
            LIKE <source user>
   | CREATE USER <user name> PASSWORD <password>
            USERGROUP <usergroup name>

<create usergroup statement> ::=
     CREATE USERGROUP <usergroup name>
            [<usergroup mode>]
            [PERMLIMIT   <unsigned integer>]
            [TEMPLIMIT   <unsigned integer>]
            [TIMEOUT     <unsigned integer>]
```

```
              [COSTWARNING <unsigned integer>]
              [COSTLIMIT   <unsigned integer>]
              [CACHELIMIT  <unsigned integer>]
              [[NOT] EXCLUSIVE]
              [AT <serverdb name>]

<create view statement> ::=
     CREATE [OR REPLACE] VIEW <table name> [(<alias name>,...)]
     AS <query expression>
     [WITH CHECK OPTION]

<data type> ::=
     CHAR[ACTER] (<unsigned integer>) [<code spec>]
   | VARCHAR (<unsigned integer>) [<code spec>]
   | LONG [VARCHAR] [<code spec>]
   | BOOLEAN
   | FIXED (<unsigned integer> [,<unsigned integer>])
   | FLOAT (<unsigned integer>)
   | DATE
   | TIME
   | TIMESTAMP

<date function> ::=
     ADDDATE    ( <date or timestamp expression>, <expression> )
   | SUBDATE    ( <date or timestamp expression>, <expression> )
   | DATEDIFF   ( <date or timestamp expression>,
                  <date or timestamp expression> )
   | DAYOFWEEK  ( <date or timestamp expression> )
   | WEEKOFYEAR ( <date or timestamp expression> )
   | DAYOFMONTH ( <date or timestamp expression> )
   | DAYOFYEAR  ( <date or timestamp expression> )
   | MAKEDATE   ( <expression>, <expression> )
   | DAYNAME    ( <date or timestamp expression> )
   | MONTHNAME  ( <date or timestamp expression> )

<date or timestamp expression> ::=
     <expression>


<datetimeformat> ::=
     EUR
   | INTERNAL
   | ISO
   | JIS
   | USA

<db procedure> ::=
     [<owner>.]<program name>.<procedure name>

<declare cursor statement> ::=
     DECLARE <result table name> CURSOR FOR <select statement>

<default expression> ::=
     <expression>

<default predicate> ::=
     <column spec> <comp op> DEFAULT

<default spec> ::=
     DEFAULT <default value>

<default value> ::=
     <literal>
   | NULL
   | USER
   | USERGROUP
   | DATE
   | TIME
   | TIMESTAMP
   | STAMP
   | TRUE
   | FALSE

<delete rule> ::=
     ON DELETE CASCADE
   | ON DELETE RESTRICT
   | ON DELETE SET DEFAULT
   | ON DELETE SET NULL

<delete statement> ::=
     DELETE [FROM] <table name> [<reference name>]
            [KEY <key spec>,...]
            [WHERE <search condition>]
   | DELETE [FROM] <table name> [<reference name>]
            WHERE CURRENT OF <result table name>


<delimiter token> ::=
     ( | ) | , | . | + | - | * | /
   | < | > | <> | != | = | <= | >=
   | ¬= | ¬< | ¬> for a computer with the code type EBCDIC
   | ~= | ~< | ~> for a computer with the code type ASCII

<derived column> ::=
     <expression> [<result column name>]
   | <result column name> = <expression>

<digit> ::=
     0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<dir or position> ::=
     <dir spec>
   | <position>
   | SAME

<dir spec> ::=
     FIRST
   | LAST
   | NEXT
   | PREV

<dir1 spec> ::=
     FIRST
   | LAST

<dir2 spec> ::=
     NEXT
   | PREV

<distinct function> ::=
     <set function name> ( DISTINCT <expression> )

<distinct spec> ::=
     DISTINCT
   | ALL
```

```
<domain name> ::=
     [<owner>.]<identifier>

<double quotes> ::=
     "


<drop definition> ::=
     DROP  <column name>,...  [<cascade option>]
   | DROP (<column name>,...) [<cascade option>]
   | DROP CONSTRAINT <constraint name>
   | DROP PRIMARY KEY
   | DROP REPLICATION

<drop domain statement> ::=
     DROP DOMAIN <domain name>

<drop index statement> ::=
     DROP INDEX <index name> [ON <table name>]
    | DROP INDEX <table name>.<column name>

<drop snapshot statement> ::=
     DROP SNAPSHOT <table name>

<drop snapshot log statement> ::=
     DROP SNAPSHOT LOG ON <table name>

<drop synonym statement> ::=
     DROP SYNONYM [<owner>.]<synonym name>

<drop table statement> ::=
     DROP TABLE <table name> [<cascade option>]

<drop user statement> ::=
     DROP USER <user name> [<cascade option>]

<drop usergroup statement> ::=
     DROP USERGROUP <usergroup name> [<cascade option>]

<drop view statement> ::=
     DROP VIEW <table name> [<cascade option>]

<duplicates clause> ::=
     REJECT DUPLICATES
   | IGNORE DUPLICATES
   | UPDATE DUPLICATES

<equal or not> ::=
     =
   | <>
   | ¬= for a computer with the code type EBCDIC
   | ~= for a computer with the code type ASCII

<exists predicate> ::=
     EXISTS <subquery>


<exists table statement> ::=
     EXISTS TABLE <table name>

<explain statement> ::=
     EXPLAIN [(<result table name>)] <query statement>
   | EXPLAIN [(<result table name>)] <single select statement>
```

```
<exponent> ::=
     [<sign>] [ [<digit>] <digit>] <digit>

<expression> ::=
     <term>
   | <expression> + <term>
   | <expression> - <term>

<expression list> ::=
     (<expression>,...)

<extended expression> ::=
     <expression>
   | DEFAULT
   | STAMP

<extended letter> ::=
     # | @ | $

<extended value spec> ::=
     <value spec>
   | DEFAULT
   | STAMP

<extraction function> ::=
     YEAR        ( <date or timestamp expression> )
   | MONTH       ( <date or timestamp expression> )
   | DAY         ( <date or timestamp expression> )
   | HOUR        ( <time or timestamp expression> )
   | MINUTE      ( <time or timestamp expression> )
   | SECOND      ( <time or timestamp expression> )
   | MICROSECOND ( <expression> )
   | TIMESTAMP   ( <expression>[, <expression> ] )
   | DATE        ( <expression> )
   | TIME        ( <expression> )

<factor> ::=
     [<sign>] <primary>

<fetch statement> ::=
     FETCH [<dir or position>] [<result table name>]
     INTO <parameter spec>,...

<first character> ::=
     <letter>
   | <extended letter>
   | <language specific character>

<first password character> ::=
     <letter>
   | <extended letter>
   | <language specific character>
   | <digit>

<fixed point literal> ::=
     [<sign>] <unsigned integer>[.<unsigned integer>]
   | [<sign>] <unsigned integer>.
   | [<sign>] .<unsigned integer>

<floating point literal> ::=
     <mantissa>E<exponent>
   | <mantissa>e<exponent>
```

```
<from clause> ::=
     FROM <table spec>,...

<function spec> ::=
     <arithmetic function>
   | <trigonometric function>
   | <string function>
   | <date function>
   | <time function>
   | <extraction function>
   | <special function>
   | <conversion function>
   | <userdefined function>

<grant statement> ::=
     GRANT <priv spec>,... TO <grantee>,... [WITH GRANT OPTION]
   | GRANT EXECUTE ON <db procedure> TO <grantee>,...

<grant user statement> ::=
     GRANT USER <granted users>
     [FROM <user name>] TO <user name>

<grant usergroup statement> ::=
     GRANT USERGROUP <granted usergroups>
     [FROM <user name>] TO <user name>

<granted users> ::=
     <user name>,...
   | *


<granted usergroups> ::=
     <usergroup name>,...
   | *

<grantee> ::=
     PUBLIC
   | <user name>
   | <usergroup name>

<group clause> ::=
     GROUP BY <expression>,...

<having clause> ::=
     HAVING <search condition>

<hex digit> ::=
     0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
   | A | B | C | D | E | F
   | a | b | c | d | e | f

<hex digit seq> ::=
     <hex digit> <hex digit>
   | <hex digit seq> <hex digit> <hex digit>

<hex literal> ::=
     x''
   | X''
   | x'<hex digit seq>'
   | X'<hex digit seq>'

<hours> ::=
     <expression>

<identifier> ::=
     <simple identifier>
   | <double quotes><special identifier><double quotes>

<identifier tail character> ::=
     <letter>
   | <extended letter>
   | <language specific character>
   | <digit>
   | <underscore>

<in predicate> ::=
     <expression> [NOT] IN <subquery>
   | <expression> [NOT] IN (<expression>,...)
   | <expression list> [NOT] IN <subquery>
   | <expression list> [NOT] IN (<expression list>,...)

<index clause> ::=
     <column name> [<order spec>]

<index name> ::=
     <identifier>

<index name spec> ::=
     INDEX <column name>
   | INDEXNAME <index name>

<index pos spec> ::=
     INDEX <column name> = <value spec>
   | INDEXNAME <index name> VALUES (<value spec>,...)

<index spec> ::=
     <unnamed index spec>
   | <named   index spec>

<indicator name> ::=
     <parameter name>

<insert columns and values> ::=
     [(<column name>,...)] VALUES (<extended expression>,...)
   | [(<column name>,...)] <query expression>
   | SET <set insert clause>,...

<insert statement> ::=
     INSERT [INTO] <table name> <insert columns and values>
     [<duplicates clause>]

<isolation spec> ::=
     ISOLATION LEVEL <unsigned integer>

<join predicate> ::=
     <expression> [<outer join indicator>]
             <comp op>
     <expression> [<outer join indicator>]

<key definition> ::=
     PRIMARY KEY (<column name>,...)

<key or not null spec> ::=
     [PRIMARY] KEY
   | NOT NULL [WITH DEFAULT]
```

```
<key spec> ::=
     <column name> = <value spec>


<key word> ::=
     <not restricted key word>
   | <restricted key word>
   | <reserved key word>

<language specific character> ::=
     Every letter that occurs in a North, Central or South
     European language, but is not contained in <letter>
     (e.g. the German umlauts, French grave accent, etc.).

<letter> ::=
     A | B | C | D | E | F | G | H | I | J | K | L | M
   | N | O | P | Q | R | S | T | U | V | W | X | Y | Z
   | a | b | c | d | e | f | g | h | i | j | k | l | m
   | n | o | p | q | r | s | t | u | v | w | x | y | z

<like expression> ::=
     <expression>
   | '<pattern element>...'

<like predicate> ::=
     <expression> [NOT] LIKE <like expression>
                  [ESCAPE <expression>]

<like user> ::=
     <user name>

<literal> ::=
     <string literal>
   | <numeric literal>

<lock option> ::=
     WITH LOCK <with lock info>

<lock spec> ::=
     <table lock spec>
   | <row lock spec>
   | <table lock spec> <row lock spec>

<lock statement> ::=
     LOCK [<wait option>] <lock spec> IN SHARE MODE
   | LOCK [<wait option>] <lock spec> IN EXCLUSIVE MODE
   | LOCK [<wait option>] <lock spec> IN SHARE MODE
                          <lock spec> IN EXCLUSIVE MODE
   | LOCK [<wait option>] <row lock spec> OPTIMISTIC

<mantissa> ::=
     <fixed point literal>


<mapchar set name> ::=
     <identifier>

<match char> ::=
     Every character except
     %, *, X'1F', <underscore>, ?, X'1E', (.

<match class> ::=
     <match range>
   | <match element>

<match element> ::=
     Every character except ).

<match range> ::=
     <match element>-<match element>

<match set> ::=
     <underscore>
   | ?
   | X'1E'
   | <match char>
   | ([<complement sign>]<match class>...)

<match string> ::=
     %
   | *
   | X'1F'

<minutes> ::=
     <expression>

<monitor statement> ::=
     MONITOR ON
   | MONITOR OFF

<named index spec> ::=
     <index name> ON <table name> ( <index clause>,... )

<named query expression> ::=
     <named query term>
   | <named query expression> UNION  [ALL] <query term>
   | <named query expression> EXCEPT [ALL] <query term>

<named query primary> ::=
     <named query spec>
   | (<named query expression>)


<named query spec> ::=
     SELECT [<distinct spec>]
     <result table name> (<select column>,...)
     <table expression>

<named query term> ::=
     <named query primary>
   | <named query term> INTERSECT [ALL] <query primary>

<named select statement> ::=
     <named query expression>
     [<order clause>]
     [<update clause>]
     [<lock option>]
     [FOR REUSE]

<new password> ::=
     <password>

<new synonym name> ::=
     <synonym name>
```

```
<new table name> ::=
     <identifier>

<next stamp statement> ::=
     NEXT STAMP [FOR <tablename>] [INTO] <parameter name>

<not restricted key word> ::=
   ACCOUNTING   ACTIVATE     ADABAS       ADD_MONTHS   AFTER
   ANALYZE      ANSI

   BAD          BEGINLOAD    BLOCKSIZE    BUFFER

   CACHELIMIT   CACHES       CANCEL       CLEAR        COLD
   COMPLETE     CONFIG       CONSOLE      CONSTRAINTS  COPY
   COSTLIMIT    COSTWARNING  CURRVAL

   DATA         DAYS         DB2          DBA          DBFUNCTION
   DBPROC       DBPROCEDURE  DEGREE       DESTPOS      DEVICE
   DEVSPACE     DIAGNOSE     DISABLE      DIV          DOMAINDEF
   DSETPASS     DUPLICATES   DYNAMIC

   ENDLOAD      ENDPOS       EUR          EXPLAIN      EXPLICIT

   FIRSTPOS     FNULL        FORCE        FORMAT       FREAD
   FREEPAGE     FWRITE

   GATEWAY      GRANTED

   HEXTORAW     HOLD         HOURS

   IMPLICIT     INDEXNAME    INIT         INITRANS     INSTR
   INTERNAL     ISO

   JIS

   KEEP

   LABEL        LASTPOS      LAST_DAY     LOAD

   MAXTRANS     MDECLARE     MDELETE      MFETCH       MICROSECONDS
   MINSERT      MINUTES      MLOCK        MOD          MONITOR
   MONTHS       MONTHS_BETWEEN MSELECT      MUPDATE

   NEW_TIME     NEXTVAL      NEXT_DAY     NOLOG        NORMAL
   NOSORT       NVL

   OFF          OPTIMISTIC   ORACLE       OUT          OVERWRITE

   PAGES        PARAM        PARSE        PARSEID      PARTICIPANTS
   PASSWORD     PATTERN      PCTUSED      PERMLIMIT    POS
   PRIV         PROC         PSM

   QUICK

   RANGE        RAWTOHEX     RECONNECT    REFRESH      REPLICATION
   REST         RESTART      RESTORE      REUSE        RFETCH

   SAME         SAPR3        SAVE         SAVEPOINT    SEARCH
   SECONDS      SEGMENT      SELECTIVITY  SEQUENCE     SERVERDB
   SHUTDOWN     SNAPSHOT     SOUNDS       SOURCEPOS    SQLID
   SQLMODE      STANDARD     STARTPOS     STAT         STATE
   STORAGE      STORE        SUBPAGES     SUBTRANS
   TABID        TABLEDEF     TEMP         TEMPLIMIT    TERMCHAR
   TIMEOUT      TO_CHAR      TO_DATE      TO_NUMBER    TRANSFILE
   TRIGGERDEF

   UNLOAD       UNLOCK       UNTIL        USA          USERID

   VERIFY       VERSION      VSIZE        VTRACE

   WAIT

   YEARS

<null predicate> ::=
     <expression> IS [NOT] NULL

<numeric literal> ::=
     <fixed point literal>
   | <floating point literal>

<object spec> ::=
     COLUMN  <table name>.<column name>
   | DBPROC  <db procedure>
   | DOMAIN  <domain name>
   | INDEX   <index name> ON <table name>
   | INDEX   <table name>.<column name>
   | TABLE   <table name>
   | TRIGGER <trigger name> ON <table name>
   | USER    <user name>
   | VIEW    <table name>
   | <parameter name>

<old password> ::=
     <password>

<old synonym name> ::=
     <synonym name>

<old table name> ::=
     <table name>

<open cursor statement> ::=
     OPEN <result table name>

<order clause> ::=
     ORDER BY <sort spec>,...

<order spec> ::=
     ASC
   | DESC

<outer join indicator> ::=
     (+)

<owner> ::=
     <user name>
   | <usergroup name>
   | TEMP

<parameter name> ::=
     :<identifier>

<parameter spec> ::=
    <parameter name> [<indicator name>]
```

```
<password> ::=
     <identifier>
   | <first password character> [<identifier tail character>...]

<password spec> ::=
     <parameter name>

<pattern element> ::=
     <match string>
   | <match set>

<pos1 spec> ::=
     <index name spec>
   | <index pos spec> [KEY <key spec>,...]
   | KEY <key spec>,...

<pos2 spec> ::=
     [<index pos spec>] KEY <key spec>,...

<position> ::=
     POS (<unsigned integer>)
   | POS (<parameter spec>)

<predicate> ::=
     <between predicate>
   | <bool predicate>
   | <comparison predicate>
   | <default predicate>
   | <exists predicate>
   | <in predicate>
   | <join predicate>
   | <like predicate>
   | <null predicate>
   | <quantified predicate>
   | <rowno predicate>
   | <sounds predicate>

<prefix> ::=
     <identifier>

<primary> ::=
     <value spec>
   | <column spec>
   | <function spec>
   | <set function spec>
   | (<expression>)

<priv spec> ::=
     <table privileges> ON [TABLE] <table name>,...

<privilege> ::=
     INSERT
   | UPDATE [(<column name>,...)]
   | SELECT [(<column name>,...)]
   | SELUPD [(<column name>,...)]
   | DELETE
   | INDEX
   | ALTER
   | REFERENCES [(<column name>,...)]

<procedure name> ::=
     <identifier>

<program name> ::=
     <identifier>

<quantified predicate> ::=
     <expression> <comp op> <quantifier> (<expression>,...)
   | <expression> <comp op> <quantifier> <subquery>
   | <expression list> <equal or not>
     <quantifier> (<expression list>,...)
   | <expression list> <equal or not> <quantifier> <subquery>

<quantifier> ::=
     ALL
   | <some>

<query expression> ::=
     <query term>
   | <query expression> UNION  [ALL] <query term>
   | <query expression> EXCEPT [ALL] <query term>

<query primary> ::=
     <query spec>
   | (<query expression>)

<query spec> ::=
     SELECT [<distinct spec>] <select column>,...
     <table expression>

<query statement> ::=
     <declare cursor statement>
   | <named select statement>
   | <select statement>

<query term> ::=
     <query primary>
   | <query term> INTERSECT [ALL] <query primary>


<reference name> ::=
     <identifier>

<referenced column> ::=
     <column name>

<referenced table> ::=
     <table name>

<referencing column> ::=
     <column name>

<referential constraint definition> ::=
     FOREIGN KEY  [<referential constraint name>]
     (<referencing column>,...)
     REFERENCES <referenced table> [(<referenced column>,...)]
     [<delete rule>]

<referential constraint name> ::=
     <identifier>

<refresh statement> ::=
     REFRESH SNAPSHOT <table name> [COMPLETE]
```

```
<regular token> ::=
     <literal>
   | <key word>
   | <identifier>
   | <parameter name>

<release statement> ::=
     COMMIT   [WORK] RELEASE
   | ROLLBACK [WORK] RELEASE

<rename column statement> ::=
     RENAME COLUMN <table name>.<column name> TO <column name>

<rename synonym statement> ::=
     RENAME SYNONYM <old synonym name> TO <new synonym name>

<rename table statement> ::=
     RENAME TABLE <old table name> TO <new table name>

<rename view statement> ::=
     RENAME VIEW <old table name> TO <new table name>

<reserved key word> ::=
   ABS          ACOS         ADDDATE      ADDTIME      ALL
   ALPHA        ALTER        ANY          ASCII        ASIN
   ATAN         ATAN2        AVG

   BINARY       BIT          BOOLEAN      BYTE

   CEIL         CEILING      CHAR         CHARACTER    CHECK
   CHR          COLUMN       CONNECTED    CONSTRAINT   COS
   COSH         COT          COUNT        CURDATE      CURRENT
   CURTIME

   DATABASE     DATE         DATEDIFF     DAY          DAYNAME
   DAYOFMONTH   DAYOFWEEK    DAYOFYEAR    DBYTE        DEC
   DECIMAL      DECODE       DEFAULT      DEGREES      DELETE
   DIGITS       DIRECT       DISTINCT     DOUBLE

   EBCDIC       ENTRY        ENTRYDEF     EXCEPT       EXISTS
   EXP          EXPAND

   FIRST        FIXED        FLOAT        FLOOR        FOR
   FROM

   GRAPHIC      GREATEST     GROUP

   HAVING       HEX          HOUR

   IFNULL       IGNORE       INDEX        INITCAP      INSERT
   INT          INTEGER      INTERSECT    INTO

   KEY

   LAST         LCASE        LEAST        LEFT         LENGTH
   LFILL        LINK         LIST         LN           LOCALSYSDBA
   LOG          LOG10        LONG         LOWER        LPAD
   LTRIM

   MAKEDATE     MAKETIME     MAPCHAR      MAX          MICROSECOND
   MIN          MINUTE       MONTH        MONTHNAME

   NEXT         NOROUND      NOT          NOW          NULL
   NUM          NUMERIC

   OBJECT       OF           ORDER

   PACKED       PI           POWER        PREV         PRIMARY

   RADIANS      REAL         REFERENCED   REJECT       REPLACE
   RFILL        RIGHT        ROUND        ROWID        ROWNO
   RPAD         RTRIM

   SECOND       SELECT       SELUPD       SET          SHOW
   SIGN         SIN          SINH         SMALLINT     SOME
   SOUNDEX      SQRT         STAMP        STATISTICS   STDDEV
   SUBDATE      SUBSTR       SUBTIME      SUM          SYSDBA

   TABLE        TAN          TANH         TIME         TIMEDIFF
   TIMESTAMP    TIMEZONE     TO           TOIDENTIFIER TRANSLATE
   TRIM         TRUNC        TRUNCATE

   UCASE        UNION        UPDATE       UPPER        USER
   USERGROUP

   VALUE        VALUES       VARCHAR      VARGRAPHIC   VARIANCE

   WEEKOFYEAR   WHERE        WITH

   YEAR

   ZONED

<restricted key word> ::=
   ACTION       ADD          AND          AS           ASC
   AT           AUDIT

   BEGIN        BETWEEN      BOTH         BUFFERPOOL   BY

   CASCADE      CAST         CATALOG      CLOSE        CLUSTER
   COMMENT      COMMIT       CONCAT       CONNECT      CREATE
   CURRENT_DATE CURRENT_TIME CURSOR

   DECLARE      DESC         DESCRIBE     DISCONNECT   DOMAIN
   DROP

   EDITPROC     END          ESCAPE       EXCLUSIVE    EXECUTE
   EXTRACT

   FALSE        FETCH        FOREIGN

   GET          GRANT

   IDENTIFIED   IN           INDICATOR    INNER        IS
   ISOLATION

   JOIN

   LANGUAGE     LEADING      LEVEL        LIKE         LOCAL
   LOCK

   MINUS        MODE         MODIFY

   NATURAL      NO           NOWAIT       NUMBER

   OBID         ON           ONLY         OPEN         OPTIMIZE
   OPTION       OR           OUTER

   PCTFREE      PRECISION    PRIVILEGES   PROCEDURE    PUBLIC

   RAW          READ         REFERENCES   RELEASE      RENAME
   RESOURCE     RESTRICT     REVOKE       ROLLBACK     ROW
   ROWNUM       ROWS

   SCHEMA       SHARE        SYNONYM      SYSDATE

   TABLESPACE   TRAILING     TRANSACTION  TRIGGER      TRUE

   UID          UNIQUE       UNKNOWN      USAGE        USING

   VALIDPROC    VARCHAR2     VARYING      VIEW

   WHENEVER     WORK         WRITE

<result column name> ::=
     <identifier>

<result expression> ::=
     <expression>

<result table name> ::=
     <identifier>

<revoke statement> ::=
     REVOKE <priv spec>,... FROM <grantee>,... [<cascade option>]
   | REVOKE EXECUTE ON <db procedure> FROM <grantee>,...

<rollback statement> ::=
     ROLLBACK [WORK] [KEEP <lock statement>]

<row lock spec> ::=
     <row spec>...

<row spec> ::=
     ROW <table name> KEY <key spec>,...
   | ROW <table name> CURRENT OF <result table name>

<rowno column> ::=
     ROWNO [<result column name>]
   | <result column name> = ROWNO

<rowno predicate> ::=
     ROWNO <  <rowno spec>
   | ROWNO <= <rowno spec>

<rowno spec> ::=
     <unsigned integer>
   | <parameter spec>

<search and result spec> ::=
     <search expression>, <result expression>

<search condition> ::=
     <boolean term>
   | <search condition> OR <boolean term>

<search expression> ::=
     <expression>
```

```
<seconds> ::=
     <expression>

<select column> ::=
     <table columns>
   | <derived column>
   | <rowno column>
   | <stamp column>

<select direct statement: positioned> ::=
     SELECT DIRECT <select column>,...
     INTO <parameter spec>,...
     FROM <table name>
     WHERE CURRENT OF <result table name>
     [<lock option>]

<select direct statement: searched> ::=
     SELECT DIRECT <select column>,...
     INTO <parameter spec>,...
     FROM <table name>
     KEY <key spec>,...
     [<where clause>]
     [<lock option>]

<select ordered format1: positioned> ::=
     SELECT <dir1 spec> <select column>,...
            INTO <parameter spec>,...
            FROM <table name>
            [<index name spec>]
            WHERE CURRENT OF <result table name>
            [<lock option>]
   | SELECT <dir1 spec> <select column>,...
            INTO <parameter spec>,...
            FROM <table name>
            [<index pos spec>]
            WHERE CURRENT OF <result table name>
            [<lock option>]


<select ordered format1: searched> ::=
     SELECT <dir1 spec> <select column>,...
     INTO <parameter spec>,...
     FROM <table name>
     [<pos1 spec>]
     [<where clause>]
     [<lock option>]

<select ordered format2: positioned> ::=
     SELECT <dir2 spec> <select column>,...
     INTO <parameter spec>,...
     FROM <table name>
     [<index pos spec>]
     WHERE CURRENT OF <result table name>
     [<lock option>]

<select ordered format2: searched> ::=
     SELECT <dir2 spec> <select column>,...
     INTO <parameter spec>,...
     FROM <table name>
     <pos2 spec>
     [<where clause>]
     [<lock option>]
```

```
<select ordered statement: positioned> ::=
     <select ordered format1: positioned>
   | <select ordered format2: positioned>

<select ordered statement: searched> ::=
     <select ordered format1: searched>
   | <select ordered format2: searched>

<select statement> ::=
     <query expression>
     [<order clause>]
     [<update clause>]
     [<lock option>]
     [FOR REUSE]

<serverdb name> ::=
     <string literal>

<servernode name> ::=
     <string literal>


<set function name> ::=
     COUNT
   | MAX
   | MIN
   | SUM
   | AVG
   | STDDEV
   | VARIANCE

<set function spec> ::=
     COUNT (*)
   | <distinct function>
   | <all function>

<set insert clause> ::=
     <column name> = <extended value spec>

<set update clause> ::=
     <column name> = <extended expression>
   | <column name> = <subquery>

<sign> ::=
     +
   | -

<simple identifier> ::=
     <first character> [<identifier tail character>...]

<single select statement> ::=
     SELECT [<distinct spec>] <select column>,...
     INTO <parameter spec>,...
     FROM <table spec>,...
     [<where clause>]
     [<having clause>]
     [<lock option>]

<some> ::=
     SOME
   | ANY

<sort option> ::=
     ASC
   | DESC

<sort spec> ::=
     <unsigned integer> [<sort option>]
   | <expression> [<sort option>]

<sounds predicate> ::=
   <expression> [NOT] SOUNDS [LIKE] <expression>

<source table> ::=
     <table name>

<source user> ::=
     <user name>

<special character> ::=
     Every character except <digit>, <letter>, <extended letter>,
     <hex digit>, <language specific character> and the character
     for the line end in a file.

<special function> ::=
     VALUE    ( <expression>, <expression>,... )
   | GREATEST ( <expression>, <expression>,... )
   | LEAST    ( <expression>, <expression>,... )
   | DECODE   ( <check expression>,
                <search and result spec>,...
                [, <default expression> ] )

<special identifier> ::=
     <special identifier character>...

<special identifier character> ::=
     Any character.

<sql statement> ::=
     <create table statement>
   | <drop table statement>
   | <alter table statement>
   | <rename table statement>
   | <rename column statement>
   | <exists table statement>
   | <create domain statement>
   | <drop domain statement>
   | <create synonym statement>
   | <drop synonym statement>
   | <rename synonym statement>
   | <create snapshot statement>
   | <drop snapshot statement>
   | <create snapshot log statement>
   | <drop snapshot log statement>
   | <create view statement>
   | <drop view statement>
   | <rename view statement>
   | <create index statement>
   | <drop index statement>
   | <comment on statement>


   | <create user statement>
   | <create usergroup statement>
   | <drop user statement>
   | <drop usergroup statement>
   | <alter user statement>
   | <alter usergroup statement>
   | <grant user statement>
   | <grant usergroup statement>
   | <alter password statement>
   | <grant statement>
   | <revoke statement>

   | <insert statement>
   | <update statement>
   | <delete statement>
   | <refresh statement>
   | <clear snapshot log statement>
   | <next stamp statement>

   | <query statement>
   | <open cursor statement>
   | <fetch statement>
   | <close statement>
   | <single select statement>
   | <select direct statement: searched>
   | <select direct statement: positioned>
   | <select ordered statement: searched>
   | <select ordered statement: positioned>
   | <explain statement>

   | <connect statement>
   | <commit statement>
   | <rollback statement>
   | <subtrans statement>
   | <lock statement>
   | <unlock statement>
   | <release statement>

   | <update statistics statement>
   | <monitor statement>

<sqlmode spec> ::=
     ADABAS
   | ANSI
   | DB2
   | ORACLE

<stamp column> ::=
     STAMP [<result column name>]
   | <result column name> = STAMP

<string function> ::=
     <string spec> || <string spec>
   | <string spec> & <string spec>
   | SUBSTR    ( <string spec>, <expression>[, <expression>] )
   | LFILL     ( <string spec>, <string literal>
                 [,<unsigned integer> ] )
   | RFILL     ( <string spec>, <string literal>
                 [,<unsigned integer> ] )
   | LPAD      ( <string spec>, <expression>, <string literal>
                 [,<unsigned integer> ] )
   | RPAD      ( <string spec>, <expression>, <string literal>
                 [,<unsigned integer> ] )
   | TRIM      ( <string spec>[, <string spec> ] )
   | LTRIM     ( <string spec>[, <string spec> ] )
   | RTRIM     ( <string spec>[, <string spec> ] )
   | EXPAND    ( <string spec>, <unsigned integer> )
   | UPPER     ( <string spec> )
   | LOWER     ( <string spec> )
   | INITCAP   ( <string spec> )
   | REPLACE   ( <string spec>, <string spec>[, <string spec> ] )
   | TRANSLATE ( <string spec>, <string spec>, <string spec> )
   | MAPCHAR   ( <string spec>[, <unsigned integer> ]
                 [, <mapchar set name> ] )
   | ALPHA     ( <string spec>[, <unsigned integer> ] )
   | ASCII     ( <string spec> )
   | EBCDIC    ( <string spec> )
   | SOUNDEX   ( <string spec> )

<string literal> ::=
     ''
   | '<character>'...
   | <hex literal>

 <string spec> ::=
     <expression>

<subquery> ::=
     (<query expression>)

<subtrans statement> ::=
     SUBTRANS BEGIN
   | SUBTRANS END
   | SUBTRANS ROLLBACK

<synonym name> ::=
     <identifier>


<table columns> ::=
     *
   | <table name>.*
   | <reference name>.*

<table description element> ::=
     <column definition>
   | <constraint definition>
   | <key definition>
   | <referential constraint definition>
   | <unique definition>

<table expression> ::=
     <from clause>
     [<where clause>]
     [<group clause>]
     [<having clause>]

<table lock spec> ::=
     TABLE <table name>,...

<table name> ::=
     [<owner>.]<identifier>

<table option> ::=
     WITH REPLICATION
   | IGNORE ROLLBACK

<table privileges> ::=
     ALL [PRIV[ILEGES]]
   | <privilege>,...
```

```
<table spec> ::=
     <table name> [<reference name>]
   | <result table name> [<reference name>]
   | (<query expression>) [<reference name>]

<term> ::=
     <factor>
   | <term> * <factor>
   | <term> / <factor>
   | <term> DIV <factor>
   | <term> MOD <factor>

<termchar set name> ::=
     <identifier>

<time expression> ::=
     <expression>

<time function> ::=
     ADDTIME  ( <time or timestamp expression>, <time expression> )
   | SUBTIME  ( <time or timestamp expression>, <time expression> )
   | TIMEDIFF ( <time or timestamp expression>,
                <time or timestamp expression> )
   | MAKETIME ( <hours>, <minutes>, <seconds> )

<time or timestamp expression> ::=
     <expression>

<token> ::=
     <regular token>
   | <delimiter token>

<trigger name> ::=
     <identifier>

<trigonometric function> ::=
     COS     ( <expression> )
   | SIN     ( <expression> )
   | TAN     ( <expression> )
   | COT     ( <expression> )
   | COSH    ( <expression> )
   | SINH    ( <expression> )
   | TANH    ( <expression> )
   | ACOS    ( <expression> )
   | ASIN    ( <expression> )
   | ATAN    ( <expression> )
   | ATAN2   ( <expression>, <expression> )
   | RADIANS ( <expression> )
   | DEGREES ( <expression> )

<underscore> ::=
     _

<unique definition> ::=
     UNIQUE (<column name>,...)

<unlock statement> ::=
     UNLOCK <row lock spec> IN SHARE MODE
   | UNLOCK <row lock spec> IN EXCLUSIVE MODE
   | UNLOCK <row lock spec> IN SHARE MODE
            <row lock spec> IN EXCLUSIVE MODE
   | UNLOCK <row lock spec> OPTIMISTIC
```

```
<unnamed index spec> ::=
     <table name>.<column name> [<order spec>]

<unsigned integer> ::=
     <digit>...

<update clause> ::=
     FOR UPDATE [OF <column name>,...]

<update columns and values> ::=
     SET <set update clause>,...
   | (<column name>,...) VALUES (<extended value spec>,...)

<update statement> ::=
     UPDATE [OF] <table name> [<reference name>]
            <update columns and values>
            [KEY <key spec>,...]
            [WHERE <search condition>]
   | UPDATE [OF] <table name> [<reference name>]
            <update columns and values>
            WHERE CURRENT OF <result table name>

<update statistics statement> ::=
     UPDATE STAT[ISTICS] COLUMN <table name>.<column name>
   | UPDATE STAT[ISTICS] COLUMN (<column name>,...)
                                FOR <table name>
   | UPDATE STAT[ISTICS] [<owner>.]<table name>
   | UPDATE STAT[ISTICS] [<owner>.][<identifier>]*

<user mode> ::=
     DBA
   | RESOURCE
   | STANDARD

<user name> ::=
     <identifier>

<user spec> ::=
     <parameter name>
   | <user name>

<userdefined function> ::=
     Each DB function defined by any user.
<usergroup mode> ::=
     RESOURCE
   | STANDARD

<usergroup name> ::=
     <identifier>

<value spec> ::=
     <literal>
   | <parameter spec>
   | NULL
   | USER
   | USERGROUP
   | LOCALSYSDBA
   | SYSDBA [(<user name>)]
   | SYSDBA [(<usergroup name>)]
   | DATE
   | TIME
   | TIMESTAMP
   | TIMEZONE
   | TRUE
   | FALSE

<wait option> ::=
     (WAIT)
   | (NOWAIT)

<where clause> ::=
     WHERE <search condition>

<with lock info> ::=
     [(NOWAIT)] [EXCLUSIVE] [ISOLATION LEVEL <unsigned integer>]
   | [(NOWAIT)]  OPTIMISTIC [ISOLATION LEVEL <unsigned integer>]
```