Contents

Manual Ordering

Introduction

Introduction

Concepts

Data Type
Parameter
Table
Column
User and Usergroup
Privilege
Database
Distributed Database
Transaction
Session
Data Integrity
Backup and Recovery Concept
SQLMODE
Code Tables

Common Elements

<character>
<literal>
<token>
Names
<column spec>
<parameter spec>
Specifying Values
<set function spec>
<expression>
<predicate>
<search condition>

SQL Statement

SQL Statement

Data Definition

<create schema statement>
<create view statement>

Authorization

<grant statement>

Data Manipulation

Data Manipulation
<insert statement>
<update statement>
<delete statement>

Data Retrieval

Data Retrieval
<query statement>
<open cursor statement>
<fetch statement>
<close statement>
<single select statement>

Transactions

Transactions
<connect statement>
<commit statement>
<rollback statement>
<release statement>

Restrictions

Restrictions

Differences

Differences

SQLSTATEs

SQLSTATEs

Syntax

Syntax

Manual Ordering

Manual Order Number: ESD611-033WOU

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 schema statement>' describes the SQL statements for the definition of tables etc.
The chapter '<grant 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 'Restrictions' lists the restrictions which generally apply to data types, parameters, identifiers, etc.
The chapter 'Differences' specifies the differences that exist between the syntax and semantics in ADABAS and the ANSI standard (ANSI X3.135-1992, Entry SQL).
The chapter 'SQLSTATEs' lists the messages and codes that can occur instead of those described in the ADABAS online help 'Messages and Codes'.
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 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 non-NULL value is a character string, or a number.

See also

Character String

Number

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 or EBCDIC). It defines the sort sequence to be used when comparing the values of this column.

3. All character strings can be compared to each other.

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.

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 etc.) that may have occurred during the assignment of values. For the transfer of 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 has a permanent memory representation and description.

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. Each table has a name that is unique within the whole database. To name result tables, names of existing tables can be used.

8. 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. 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.

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 view 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, 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, users, and usergroups.

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 the 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 WORK or ROLLBACK WORK. If a transaction is closed with COMMIT WORK, all modifications made to the database within the transaction are kept. If a transaction is aborted with ROLLBACK WORK, all modifications made to the database within this transaction are cancelled. Modifications closed with COMMIT WORK cannot be cancelled with ROLLBACK WORK.
   COMMIT WORK and ROLLBACK WORK 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.

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.

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 ANSI. 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	ã	99	63	Ã
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	ä	102	66	Ä
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	õ	235	EB	Õ
140	8C	ð	172	AC	Ð	204	CC	ò	236	EC	Ò
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	ô	239	EF	Ô
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>
  | <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> ::=
    # | @ | $

<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>,
    <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>...'

<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.

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 schema statement>, <column definition>').

2. A <string literal> of the type " 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> ::=
  ABS          ACOS         ADDDATE      ADDTIME      ALPHA
  ASCII        ASIN         ATAN         ATAN2        AUDIT

  BINARY       BOOLEAN      BUFFERPOOL   BYTE

  CEIL         CEILING      CHR          CLUSTER      COMMENT
  CONCAT       CONNECTED    COS          COSH         COT
  CURDATE      CURTIME

  DATABASE     DATEDIFF     DAYNAME      DAYOFMONTH   DAYOFWEEK
  DAYOFYEAR    DBYTE        DECODE       DEGREES      DIGITS
  DIRECT

  EBCDIC       EDITPROC     ENTRY        ENTRYDEF     EXCLUSIVE
  EXP          EXPAND

  FIXED        FLOOR

  GRAPHIC      GREATEST

  HEX

  IDENTIFIED   IFNULL       IGNORE       INDEX        INITCAP

  LCASE        LEAST        LENGTH       LFILL        LINK
  LIST         LN           LOCALSYSDBA  LOCK         LOG
```

```
  LOG10         LONG          LPAD          LTRIM

  MAKEDATE      MAKETIME      MAPCHAR       MICROSECOND   MINUS
  MODE          MODIFY        MONTHNAME

  NOROUND       NOW           NOWAIT        NUM           NUMBER

  OBID          OBJECT        OPTIMIZE

  PACKED        PCTFREE       PI            POWER         PREV

  RADIANS       RAW           REFERENCED    REJECT        RELEASE
  RENAME        REPLACE       RESOURCE      RFILL         ROUND
  ROW           ROWID         ROWNO         ROWNUM        RPAD
  RTRIM

  SELUPD        SHARE         SHOW          SIGN          SIN
  SINH          SOUNDEX       SQRT          STAMP         STATISTICS
  STDDEV        SUBDATE       SUBSTR        SUBTIME       SYNONYM
  SYSDATE       SYSDBA



  TABLESPACE    TAN           TANH          TIMEDIFF      TIMEZONE
  TOIDENTIFIER  TRIGGER       TRUNC         TRUNCATE

  UCASE         UID           USERGROUP

  VALIDPROC     VARCHAR2      VARGRAPHIC    VARIANCE

  WEEKOFYEAR

  ZONED


<reserved key word> ::=
  ACTION        ADD           ALL           ALTER         AND
  ANY           AS            ASC           AT            AVG

  BEGIN         BETWEEN       BIT           BOTH          BY

  CASCADE       CAST          CATALOG       CHAR          CHARACTER
  CHECK         CLOSE         COLUMN        COMMIT        CONNECT
  CONSTRAINT    COUNT         CREATE        CURRENT       CURRENT_DATE
  CURRENT_TIME  CURSOR

  DATE          DAY           DEC           DECIMAL       DECLARE
  DEFAULT       DELETE        DESC          DESCRIBE      DISCONNECT
  DISTINCT      DOMAIN        DOUBLE        DROP

  END           ESCAPE        EXCEPT        EXECUTE       EXISTS
  EXTRACT

  FALSE         FETCH         FIRST         FLOAT         FOR
  FOREIGN       FROM

  GET           GRANT         GROUP

  HAVING        HOUR

  IN            INDICATOR     INNER         INSERT        INT
  INTEGER       INTERSECT     INTO          IS            ISOLATION
```

```
  JOIN

  KEY

  LANGUAGE     LAST         LEADING      LEFT         LEVEL
  LIKE         LOCAL        LOWER

  MAX          MIN          MINUTE       MONTH


  NATURAL      NEXT         NO           NOT          NULL
  NUMERIC

  OF           ON           ONLY         OPEN         OPTION
  OR           ORDER        OUTER

  PRECISION    PRIMARY      PRIVILEGES   PROCEDURE    PUBLIC

  READ         REAL         REFERENCES   RESTRICT     REVOKE
  RIGHT        ROLLBACK     ROWS

  SCHEMA       SECOND       SELECT       SET          SMALLINT
  SOME         SUM

  TABLE        TIME         TIMESTAMP    TO           TRAILING
  TRANSACTION  TRANSLATE    TRIM         TRUE

  UNION        UNIQUE       UNKNOWN      UPDATE       UPPER
  USAGE        USER         USING

  VALUE        VALUES       VARCHAR      VARYING      VIEW

  WHENEVER     WHERE        WITH         WORK         WRITE

  YEAR


<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>

<alias name> ::=
    <identifier>

<column name> ::=
    <identifier>

<reference name> ::=
    <identifier>

<result table name> ::=
    <identifier>

<schema name> ::=
    <identifier>

<termchar set name> ::=
    <identifier>

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

<parameter name> ::=
    :<identifier>


<indicator name> ::=
    <parameter name>
```

Syntax Rules

1. Names must not be longer than 18 characters.

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.

2. A <usergroup name> identifies a usergroup.

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.

4. A new column name <alias name> defines the name of a column in a view table. It is defined in a <create view statement>.

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

6. 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.

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

8. A <schema name> identifies a schema generated by a <create schema statement>.

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

10. A <table name> identifies a table. An identifier is defined as <table name> by a <create table statement> or <create view 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.

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

12. 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.

<column spec>

Function

specifies a column in a table.

Format

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

Syntax Rules

none

General Rules

none

<parameter spec>

Function

specifies a parameter.

Format

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

Syntax Rules

1. The specification of INDICATOR is insignificant.

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 a value less than 0 indicates that the value represented by the parameter is the NULL value.

Specifying Values

Function

specifies a value.

Format

```
<value spec> ::=
    <literal>
  | <parameter spec>
  | NULL
  | USER
```

Syntax Rules

none

General Rules

1. The key word NULL denotes the NULL value.
2. The key word USER denotes the name of the current user.

<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 <column spec> )

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

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

<all set function name> ::=
    MAX
  | MIN
  | SUM
  | AVG
```

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 <column spec> to each row of a temporary result table or of a group and by eliminating all NULL values and duplicate values.
   If the set is empty and the <distinct function> is applied to the whole temporary result table, the result of AVG, MAX, MIN, and SUM 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.

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.
   If the set is empty and the <all function> is applied to the whole temporary result table, the result of AVG, MAX, MIN, and SUM is the NULL value.
   If there is no group to which the <all function> could be applied, the result table is empty.
   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 <column spec>) is the number of values of the argument in the <distinct 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. 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>

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

<sign> ::=
    +

  | -

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

Syntax Rules

none

General Rules

1. The arithmetical operators * and / can only be applied to numeric data types.

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

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

4. 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').

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

6. If no parentheses are used, the operators have the following precedence: <sign> has a higher precedence than the multiplicative operators * and / 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>
  | <comparison predicate>
  | <exists predicate>
  | <in predicate>
  | <join predicate>
  | <like predicate>
  | <null predicate>
  | <quantified 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. All alphanumeric columns can be compared to each other. Numbers can be compared to each other.

See also

<between predicate>

<comparison predicate>

<exists predicate>

<in predicate>

<join predicate>

<like predicate>

<null predicate>

<quantified 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.

<comparison predicate>

Function

specifies a comparison between two values.

Format

```
<comparison predicate> ::=
    <expression> <comp op> <expression>
  | <expression> <comp op> <subquery>

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

Syntax Rules

1. The <subquery> must produce a single-column result table which contains no more than one row.

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) 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 or EBCDIC) chosen for this column.

5. If x or y are NULL values, or if the result of the <subquery> is empty, then (x <comp op> y) is unknown.

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

<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 is contained in a given set of values.

Format

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

Syntax Rule

1. The <subquery> must produce a single-column result table.

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 <value spec>s. S is a set of values. The value x and the values in S must be comparable with each other.

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

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

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

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

<join predicate>

Function

specifies a join.

Format

```
<join predicate> ::=
    <expression> <comp op> <expression>
```

Syntax Rules

none

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. 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>

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

Syntax Rules

1. The hexadecimal values X'1E' and X'1F' cannot be specified as <string literal>, but only as a parameter value.

General Rules

1. The <expression> of the <like expression> must produce an alphanumeric value.

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

3. A <match set> is a character.
   Thereby '_' and X'1E' stand for any character, <match char> for itself.

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 escape character can only be placed before a <match char> which is identical to the escape character, before a <match string>, or before a <match set> that is no <match char>.
   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. (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> <subquery>

<quantifier> ::=
    ALL
  | <some>

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

Syntax Rules

1. The <subquery> must produce a single-column result table.

General Rules

1. Let x be the result of the <expression> and S the result of the <subquery>. 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.

<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 schema statement>
  | <create table statement>
  | <create view statement>
  | <grant statement>

  | <insert statement>
  | <update statement>
  | <delete statement>

  | <query statement>
  | <open cursor statement>
  | <fetch statement>
  | <close statement>
  | <single select statement>

  | <connect statement>
  | <commit statement>
  | <rollback statement>
  | <release statement>
```

Syntax Rules

none

General Rules

1. The SQL statements of the 1st block are described in the chapter '<create schema statement>'.
2. The SQL statements of the 2nd block are described in the chapter '<grant 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. All SQL statements can be embedded in programming languages. For a detailed description, refer to the online help on the precompilers.

<create schema statement>

Function

creates a schema.

Format

```
<create schema statement> ::=
    CREATE SCHEMA AUTHORIZATION <schema name>
```

Syntax Rules

none

General Rules

1. The execution of the <create schema statement> has no semantic significance.

See also

<create table statement>
<column definition>
<constraint definition>
<referential constraint definition>
<key definition>
<unique definition>

<create table statement>

Function

creates a base table.

Format

```
<create table statement> ::=
    CREATE TABLE <table name> (<table description element>,...)

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

Syntax Rules

1. 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>. Otherwise, <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. 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. The current user must have DBA or RESOURCE status.
5. The current user becomes the owner of the created table. The user obtains the INSERT, UPDATE, DELETE, SELECT, and REFERENCES privilege for this table.

<column definition>

Function

defines a table column.

Format

```
<column definition> ::=
    <column name> <data type> <column attributes>

<data type> ::=
    CHAR[ACTER] [(<unsigned integer>)]
  | DEC[IMAL]   (<unsigned integer> [,<unsigned integer>])
  | NUMERIC (<unsigned integer> [,<unsigned integer>])
  | SMALLINT
  | INT[EGER]
  | FLOAT [(<unsigned integer>)]
  | REAL
  | DOUBLE PRECISION

<column attributes> ::=
    [<default spec>]
    [NOT NULL [<unique spec>] ]
    [REFERENCES <referenced table> [(<referenced column>)]]
    [<constraint definition>]

<unique spec> ::=
    PRIMARY KEY
  | UNIQUE

<referenced table> ::=
    <table name>

<referenced column> ::=
    <column name>

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



<default value> ::=
    <literal>
  | NULL
  | USER
```

Syntax Rules

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

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) defines 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. 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.

4. DEC[IMAL](p,s) and NUMERIC(p,s) define 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. SMALLINT is equivalent to NUMERIC(5,0) CHECK (<column name> BETWEEN -32768 AND 32767). INT[EGER] is equivalent to NUMERIC(10,0) CHECK (<column name> BETWEEN -2147483648 AND 2147473647).

5. 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. If p is omitted, FLOAT(16) is assumed. REAL is equivalent to FLOAT(16). DOUBLE PRECISION is equivalent to FLOAT(18).

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

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

8. 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.

9. 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.

10. If PRIMARY KEY is specified, then the column makes up the key of the table. ADABAS ensures that the key values of a table are unique.

11. If UNIQUE is specified, ADABAS ensures that different rows of the table do not have the same value in the column <column name>.

12. If a table is defined without a key column, ADABAS implicitly generates the key column SYSKEY CHAR(8) BYTE. This column has neither the code attribute ASCII nor EBCDIC and can therefore not be compared with other columns. 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.

13. 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 can only be specified for columns of the data type CHAR[ACTER](n) where n >= 18.

14. 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>)].

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

<constraint definition>

Function

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

Format

```
<constraint definition> ::=
    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. The NULL value implicitly satisfies all <search condition>s, unless it is explicitly excluded.

3. 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.

4. 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.

5. 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 (<referencing column>,...)
    REFERENCES <referenced table> [(<referenced column>,...)]

<referencing column> ::=
    <column name>
```

Syntax Rules

none

General Rules

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

2. The referencing table and the <referenced table> must be base tables.

3. If the <referenced table> does not exist at execution time of the <create table statement>, then it must be generated within the current transaction. Otherwise, the transaction fails.

4. The current user must have the REFERENCES privilege for the <referenced table>.

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

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

7. 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.

8. 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>.

9. 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.

10. 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>.

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

12. The deletion of a row from the <referenced table> fails whenever at least one matching row exists.

13. 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.

<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>; 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. The sum of the lengths of the key columns must not exceed 255 characters.

4. Key columns are NOT NULL columns.

5. 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. The <unique definition> is part of a <create table statement>. Each <column name> must identify a column of the table defined by the <create table statement> and may occur only once.

2. Each column identified by <column name> must be a NOT NULL column.

3. The <unique definition> ensures that no two distinct rows of the table contain the same value in all columns identified by <column name>.

<create view statement>

Function

creates a view table.

Format

```
<create view statement> ::=
    CREATE 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 number of <alias name>s must be equal to the number of columns in the result table generated by the <query expression>.

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

3. The user must have the SELECT privilege for all tables 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 the view table when he is authorized to grant the SELECT privilege to other users for all tables that occur in the view definition. The user has the INSERT, UPDATE, or DELETE privilege when he has the corresponding privileges for the table 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 the table on which the view table is based.

4. 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.

5. The view table is always identical to the table that would be obtained as the result of the <query expression>.

6. 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 UNION.

   c) The <search condition> of the <query expression> in the <create view statement> contains a <subquery>.

7. A view table is called updatable if it is based on just one base table, if it is not a complex view table, and if it is not based on a complex view table.

8. 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 the table 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 the table of the <from clause> as <select column>.

9. 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>.

10. 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 the table of the <from clause> of the <create view statement>.

11. 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.

<grant statement>

Function

grants privileges for tables and single columns.

Format

```
<grant statement> ::=
    GRANT <table privileges> ON <table name>
    TO <grantee>,... [WITH GRANT OPTION]

<table privileges> ::=
    ALL PRIVILEGES
  | <privilege>,...

<privilege> ::=
    INSERT
  | UPDATE [(<column name>,...)]
  | SELECT
  | DELETE
  | REFERENCES [(<column name>,...)]

<grantee> ::=
    PUBLIC
  | <user name>
  | <usergroup name>
```

Syntax Rules

none

General Rules

1. The <table privileges> define a set of privileges for the table identified by <table name>.
   The user must have the authorization to grant privileges for the specified table. For base tables, the owner of the table has this authorization.
   For view 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 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. 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 table. 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 table. 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 table. The current user must have the authorization to grant the SELECT privilege.

5. The DELETE privilege allows the user identified by <grantee> to delete rows from the specified table. The current user must have the authorization to grant the DELETE privilege.

6. 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.

7. All privileges which the user is authorized to grant for the table using ALL PRIVILEGES are granted to the users identified by the sequence of <grantee>s.

8. <grantee> must not be identical with the <user name> of the current user and the name of the table owner.

9. <grantee> must not denote a member of a usergroup.

10. If PUBLIC is specified, the listed privileges are granted to all users, both to current ones and to any created later.

11. 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.

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>

<insert columns and values> ::=
    [(<column name>,...)] VALUES (<value spec>,...)
  | [(<column name>,...)] <query expression>
```

Syntax Rules

1. A column specified in the optional sequence of <column name>s is a target column. Target columns can be specified in any order.

2. If no sequence of <column name>s is specified, this has the same effect as the specification of a sequence of <column name>s which contains 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 values must equal the number of target columns. The ith <value spec> 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.

2. If <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.
   A column must not occur more than once in a sequence of <column name>s.

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, on which

the view table is based. In this case, the target columns of <table name> correspond to the columns of the base table, on which the view table is based. In the following paragraphs, the term target column refers to the corresponding column of the base table.

6. If there is no <query expression> in the <insert statement>, exactly one row is inserted into the table <table name>. 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 there is already a row with the key specified for the row to be inserted, the <insert statement> fails.

8. 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.

9. If there are <constraint definition>s for the base table 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.

10. If the base table 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.

11. Let C be a target column and v a non-NULL value to be stored in C.

12. If C is a numeric column, v must be a number within the permitted range of values of C.

13. 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.

14. 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.

15. An <insert statement> sets the third entry of SQLERRD in the SQLCA (see the Precompiler online help) to the number of inserted rows.

16. 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 <table name>
           <update columns and values>
           [WHERE <search condition>]
  | UPDATE <table name>
           <update columns and values>
           WHERE CURRENT OF <result table name>

<update columns and values> ::=
    SET <set update clause>,...

<set update clause> ::=
    <column name> = <expression>
```

Syntax Rules

1. Columns whose values are to be updated are called target columns.

2. The <expression> in a <set update clause> must not contain a <set function spec>.

General Rules

1. <table name> must identify an existing base table or view table.

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 table on which the view table is based. In this case, the target columns of <table name> correspond to columns of the base table, on which the view table is based. In the following paragraphs, the term target column always refers to the corresponding column in the base table.

5. Values of key columns defined by a user for a <create table statement> can be updated. The implicit key column SYSKEY, if created, cannot be updated.

6. <update columns and values> identifies one or more target columns and new values

for these columns. The optional <search condition> or, in case of CURRENT OF, the cursor position within the result table <result table name> determines the rows of the specified table to be updated

7. If neither a <search condition> nor CURRENT OF <result table name> is specified, all rows of the specified table are updated.

8. If 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>.

9. 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>.

10. 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 is updatable, see '0'.

11. 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.

12. If no row is found for which the conditions defined by the optional clauses are satisfied, the SQLSTATE 02000 - ROW NOT FOUND - is set.

13. If there are <constraint definition>s for the base table 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.

14. 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.

15. For each row in which the value of a <referenced column> of a <referential constraint definition> is to be updated with 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.

16. Let C be a target column and v a non-NULL value for the modification of C.

17. If C is a numeric column, then v must be a number within the permitted range of values for C.

18. 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.

19. 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.

20. 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>
           [WHERE <search condition>]
  | DELETE FROM <table name>
           WHERE CURRENT OF <result table name>
```

Syntax Rules

none

General Rules

1. <table name> must identify an existing base table or view table.

2. The current user must have the DELETE 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 DELETE privilege because the view table is not updatable.

3. If <table name> identifies a view table, rows are deleted from the base table, on which the view table is based.

4. 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 <search condition> nor CURRENT OF <result table name> is specified, all rows of the specified table are deleted.

6. If 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.

7. 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>.

8. 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 is updatable; i.e., that it was created without DISTINCT and without <set function spec>. Afterwards, the cursor is positioned behind the result table row.

9. 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.

10. If no row is found which satisfies the conditions defined by the optional clauses, the SQLSTATE 02000 - ROW NOT FOUND - is set.

11. For each row deleted in the course of the <delete statement> which comes from a <referenced table> of at least one <referential constraint definition>, ADABAS checks whether there is a matching row in the corresponding foreign key table. If there is at least one matching row, the <delete statement> fails.

12. 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>.

13. If errors occur in the course of the <delete statement>, the statement fails, leaving the table unmodified.

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>

<declare cursor statement> ::=
    DECLARE <result table name> CURSOR FOR <select statement>

<select statement> ::=
    <query expression>
    [<order clause>]
```

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 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>.

3. A result table or, more precisely, the underlying base table, is updatable if the <query statement> satisfies the following conditions:

   a) The <query expression> may only consist of one <query spec>.

   b) One base table or one updatable view table may only be specified in the <from clause> of the <query spec>.

   c) DISTINCT, GROUP BY or HAVING must not be specified.

   d) <expression>s must not contain a <set function spec>.

   e) The <select statement> must not contain an <order clause>.

See also

<query expression>

<query spec>
<table expression>
<order clause>

<query expression>

Function

specifies an unordered result table.

Format

```
<query expression> ::=
    <query primary>
  | <query expression> UNION [ALL] <query primary>

<query primary> ::=
    <query spec>
  | (<query expression>)
```

Syntax Rules

none

General Rules

1. 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>.

2. 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 have the same data type, the same length, and the same scale.

3. The names of the result table columns are formed from the names of the <select column>s of the first <query spec>.

4. Let T_1 be the left operand of UNION, Let T_2 be the right operand. Let R be the result of the operation on T_1 and T_2.

5. If UNION is specified, R contains all rows of T_1 and T_2.

6. DISTINCT is implicitly assumed for the <query expression>s belonging to T_1 and T_2 if ALL is not specified. All duplicate rows are removed from R.
   A row is a duplicate of another row if both rows have identical values in each column. NULL values are assumed to be identical.

<query spec>

Function

specifies an unordered result table.

Format

```
<query spec> ::=
    SELECT [<distinct spec>] <select column>,...
    <table expression>

<distinct spec> ::=
    DISTINCT
  | ALL

<select column> ::=
    <table columns>
  | <derived column>

<table columns> ::=
    *

<derived column> ::=
    <expression> [ [AS] <result column name>]

<result column name> ::=
    <identifier>
```

Syntax Rules

1. 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.

General Rules

1. 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>.

2. 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.

3. 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.
   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>.

4. The specification of <table columns> in a <select column> is an abbreviation of the specification of the result table columns.

5. If a <select column> of the format '*' is specified, then 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.
The implicitly generated column SYSKEY is not passed.

6. 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> [AS] <result column name>', 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 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.

7. 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>.

8. 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

none

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>

<subquery>

Correlated Subquery

<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>]
```

Syntax Rules

none

General Rules

1. Each <table spec> specifies a table identifier.

2. If a <table spec> specifies no <reference name>, the <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. 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.

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, 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 base or view tables.

8. 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.

9. <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.

<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> unless the <where clause> was specified in a <subquery> that belongs to a <having clause> of a higher-level <query spec>, and a grouping column of this higher-level <query spec> was specified as correlated column in the <set function spec>. For an explanation of the terms 'higher level' and 'correlated column' see the chapter 'Correlated Subquery'.

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> 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.

4. 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 <column spec>,...
```

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, and COUNT 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.

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, and COUNT.

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 column name 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.

Format

```
<subquery> ::=
    (<query expression>)
```

Syntax Rules

1. A <subquery> used in a <comparison predicate>, <in predicate> or <quantified predicate> 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>.

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>]
  | <column spec> [<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 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>.

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.

<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 must be deleted before their names can be used for other result tables.

2. All result tables are implicitly closed at the end of the transaction using the <commit statement> or <rollblack 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. 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.

5. 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.

6. If the result table is empty, the SQLSTATE 02000 - ROW NOT FOUND - is set.

7. 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 <result table name> INTO <parameter spec>,...
```

Syntax Rules

none

General Rules

1. Let C be the position in the result table. The SQLSTATE 02000 - 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.

2. If 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.

3. If 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.

4. 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.

5. 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.

6. 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.

7. 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 <fetch statement>s. Depending on the ISOLATION LEVEL selected, this can also be the reason for locking problems occurring with a FETCH, e.g., SQLSTATE 40001 - LOCK REQUEST TIMEOUT.

<close statement>

Function

closes a result table.

Format

```
<close statement> ::=
    CLOSE <result table name>
```

Syntax Rules

none

General Rules

1. The result table with the specified name is closed. Its name can be used to denote another result table.

2. Existing result tables must be closed before their names can be used for other result tables.

3. All result tables are implicitly closed at the end of the transaction using the <commit statement> or <rollback statement>.

4. 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>]
```

Syntax Rules

none

General Rules

1. For an empty result table, the SQLSTATE 02000 - ROW NOT FOUND - is set.

2. 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.

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 implicitly set by ADABAS in the course of processing an <sql statement>. These locks are assigned to the transaction that contains the <sql statement>. ADABAS distinguishes between SHARE locks and EXCLUSIVE locks which either refer to rows or tables. 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).
The locks assigned to a transaction are released at the end of the transaction, making the respective database objects accessible again to other transactions.

The following table gives a schematic overview on the possible parallel locks. EXCL means EXCLUSIVE.

	Let a transaction have a(n)					
	EXCL	SHARE	EXCL	SHARE	EXCL	SHARE
Can another transaction	lock on a table		lock on a row		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 and 3.

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 ANSI.

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. The <isolation spec> specified in the <connect statement> is applied to each newly opened transaction.

5. Locks can be requested implicitly. In this case, the type of lock depends on the <isolation spec> in the <connect statement>. How long an implicit lock is maintained also depends on the <isolation spec>. For modifications of metadata of the catalog, ADABAS ensures, however, that no inconsistencies occur through concurrent transactions, regardless of the selected <isolation spec>. It is therefore possible that collisions with other locks occur even for the ISOLATION LEVEL 0.

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 means that a SHARE lock is assigned to the transaction for each read row R_1 in a table. When the next row R_2 in the same table is read, the lock on R_1 is released and a SHARE lock is assigned to the transaction for the row R_2.
   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. ISOLATION LEVEL 2 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 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.

9. ISOLATION LEVEL 3 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.

10. If the <isolation spec> is omitted, ISOLATION LEVEL 3 is assumed.

11. 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.

12. 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.

13. 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.

14. 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.

15. 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>.

16. 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.

17. 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, 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.

18. 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.

19. 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
```

Syntax Rules

none

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. The locks assigned to the transaction are released.

3. All result tables are implicitly closed when a transaction is ended using the <commit statement>.

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
```

Syntax Rules

none

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. The locks assigned to the transaction are released.

3. All result tables are implicitly closed when the related transaction is ended using a <rollback statement>.

4. The <isolation spec> declared in the <connect statement> controls the setting of locks in the new transaction.

<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.

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.

Restrictions

Maximum values :

Number of tables		unlimited
Length of an identifier	18	characters
Internal length of a table row	4047	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 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

Differences

1. Between the ANSI standard and ADABAS, there are differences with regard to the implicit addition of the <owner> if this specification has been omitted in the <table name>.

2. In addition to the ANSI standard, in ADABAS, X'1F' and X'1E' are accepted in the <like expression> of a <like predicate> as equivalents of '%' and '_'.

3. In contrast to the ANSI standard, the <create schema statement> has no semantic significance in ADABAS.

SQLSTATEs

00000 SUCCESS

Explanation:
The SQL statement was successfully executed.

User Action:
No user action is required.

01003 NULL VALUE IN SET FUNCTION ELIMINATED

Explanation:
The SQL statement was successfully executed. At least one NULL value was eliminated from a <set function spec>.

User Action:
No user action is required.

01004 VARIABLE MAY BE TRUNCATED

Explanation:
The SQL statement was successfully executed. The value in the parameter or in the database column was too long to be stored completely in the corresponding database column or parameter. It was truncated.

User Action:
No user action is required. If this is not desired, modify either the format of the database column or the format of the parameter.

02000 ROW NOT FOUND

Explanation:
There is no (further) table row which meets the qualification.

User Action:
No user action is required.

07008 TOO MANY PARAMETERS FOR DESCRIPTOR

Explanation:
Too many parameters are specified for the descriptor variable.

User Action:
Reduce the number of parameters in the SQL statement. The maximum number is 300.

08001 SERVERDB NOT ACCESSIBLE

Explanation:
The attempt was made to start a session using an ADABAS component. This is not

possible, because
1. the SERVERDB name was incorrectly specified or
2. the database server was not started.

User Action:
1. Check the specified SERVERDB name.
2. Start the database server.

08002 SESSION ALREADY CONNECTED

Explanation:
A database session has already been opened with this number.

User Action:
Either close the database session beforehand or remove the CONNECT from the application program.

08003

USER MUST BE CONNECTED

Explanation:
An ADABAS session can only be opened with a <connect statement>.

User Action:
Specify a <connect statement>.

CONNECT FAILED, CHECK SERVERDB

Explanation:
An error was detected while processing the CONNECT statement.

User Action:
1. Check the name of the database and correct it, if necessary.
2. A check must be made as to whether the database is running. If need be, the DBA must perform a RESTART.
3. Data communication with the database must be re-established.

SERVERDB SYSTEM NOT YET AVAILABLE

Explanation:
The database exists, but it is in the startup or shutdown phase, with the result that a connection cannot be established at the moment.

User Action:
Repeat the CONNECT at a later point in time.

08004

USER ALREADY CONNECTED TO THIS USER TASK

Explanation:
A user already connected to the database entered another <connect statement>.

User Action:
If the user wants to continue working under another user name, he must specify first the <release statement> and then a <connect statement>.

USER ALREADY CONNECTED

Explanation:
A user attempts to connect to ADABAS under a user name which was defined with EXCLUSIVE in a <create user statement>, <create usergroup statement>, <alter user statement>, or <alter usergroup statement>. Another user has connected to ADABAS using this name.

User Action:
The user must wait until the other user has disconnected from ADABAS using a <release statement>.
To enable a user to have several simultaneous connects, the owner of the user or usergroup must specify NOT EXCLUSIVE for this user using the <alter user statement> or <alter usergroup statement>.

CONNECT STATEMENT SYNTAX WRONG

Explanation:
There is an error in the CONNECT statement syntax.

User Action:
For the exact description of the syntax, refer to the chapter <connect statement>.
Correct the statement accordingly.

IMPLICIT CONNECT: MISSING USER OR SERVERDB

Explanation:
During the execution of an implicit CONNECT, the user entries or the database name could not be found.

User Action:
Open an XUSER file, or enter the CONNECT parameters using runtime options.

MISSING USERNAME FOR CONNECT

Explanation:
No user/password combination could be found for an implicit CONNECT enabled by the CHECK option.

User Action:
Write the CONNECT statement as the first parameter into the program, or specify the option USER, or create an XUSER file for an implicit CONNECT.

MISSING USERNAME OR PASSWORD FOR CONNECT

Explanation:
A username or password specification is missing in a CONNECT statement, or there is no XUSER file for an implicit CONNECT.

User Action:
Specify username and password for the CONNECT statement using options, or open an XUSER file.

SERVERDB MUST BE RESTARTED

Explanation:
It is not possible to work on the SERVERDB because it was shut down.

User Action:
The SERVERDB must be started with the Operating / Restart / Warm menu function in the ADABAS component CONTROL.

UNKNOWN USER NAME/PASSWORD COMBINATION

Explanation:
The specified combination of user name and password is unknown. ADABAS can only be accessed by a combination that is known to the database.

User Action:
Change the user or password specification in the SQL statement.

0A000 SYSTEM ERROR: NOT YET IMPLEMENTED

Explanation:
This SQL statement is not yet implemented. It will be available in future versions.

User Action:
A user action is not possible.

21000 MORE THAN ONE RESULT ROW NOT ALLOWED

Explanation:
1. This error may occur when a <single select statement> is executed and more than one row complies with the <search condition>.
2. The error can also occur when a <subquery> specified in a <comparison predicate> or a <set update clause> of an <update statement> is executed and more than one row complies with the <search condition>.

User Action:
1. The <single select statement> can be replaced with a <query statement> and a sequence of <fetch statement>s, or it can be ensured that at most one row complies with the conditions by expanding the <search condition>.
2. The <comparison predicate> can be replaced with a <quantified predicate>. The attempt can be made to change the <subquery> in such a way that the <subquery> contains at most one row as the result by specifying DISTINCT or by expanding the <search condition>.

22001

INPUT VARIABLE HAS BEEN TRUNCATED

Explanation:
The contents of a character string is longer than the database is capable of storing, or a floating point number was truncated.

User Action:
Set the input variable to a string which corresponds to the length of the database variable or adapt the input variable to the format used in the database.

CONSTANT MUST BE COMPATIBLE WITH COLUMN TYPE AND LENGTH

Explanation:
The specified constant does not match the data type for this column.

User Action:
Use a <query statement> issued on the system table DOMAIN.COLUMNS to find out the definition of the affected column. Specify a constant with the correct data type.

ASSIGNMENT IMPOSSIBLE, CHAR VALUE TOO LONG

Explanation:
In an <insert statement> with <query expression> or in an <update statement>, the attempt was made to assign a character string to a column of data type CHAR. This character string was too long. The error message is returned for the first occurring value with an exceeding length, not while analyzing the maximum column lengths.

User Action:
Use SELECT ... WHERE LENGTH (<column name>) > <unsigned integer> in SQLMODE ADABAS to find out the rows containing a value with an exceeding length.
The length of the corresponding column can be increased in SQLMODE ADABAS by an <alter table statement>.

22002 MISSING INDICATOR VARIABLE, OUTPUT PARAMETER WITH NULL VALUE

Explanation:
The indicator variable required for returning the NULL value to the SQL variable is missing.

User Action:
Specify an indicator variable with the parameter.

22003

INVALID EXPONENT

Explanation:

There is one of two possible causes.

1. A numeric final or temporary result is greater or less than the values which can be represented by a floating point number.
2. A numeric value is greater or less than permitted by the data type of a specified column.

User Action:

1. If a numeric temporary result is too large or too small, the attempt can be made to prevent an overflow or underflow by rearranging the arithmetic operations.
2. Use a <query statement> issued on the system table DOMAIN.COLUMNS to find out the data type of the column. The values must be corrected accordingly.

NUMERIC INPUT PARAMETER OVERFLOW

Explanation:
The numeric input value is too large for the database.

User Action:
Check the size of the input value and of the range of values valid for the database and modify them, if necessary.

NUMERIC OUTPUT PARAMETER OVERFLOW

Explanation:
An ADABAS database value is too large for the SQL variable or the parameter.

User Action:
Enlarge the value range for the SQL variable or parameter.

22005 INCOMPATIBLE DATA TYPES

Explanation:
The data type of the SQL variable or parameter is not compatible with the ADABAS data type.

User Action:
Change the data type of the SQL variable or parameter to match the ADABAS data type for the table column.
If this message is returned during precompilation and the data types do be compatible, precompile with NOCHECK option (see the chapter 'General Rules' of the Precompiler online help).

22007

INVALID DATE FORMAT

Explanation:
The specified value is not a valid date value.
User Action:

Correct the date value.

INVALID TIME FORMAT

Explanation:
The specified value is not a valid time value.

User Action:
Correct the time value.

INVALID TIMESTAMP FORMAT

Explanation:
The specified value is not a valid timestamp value.

User Action:
Correct the timestamp value.

22012 INVALID NUMERIC EXPRESSION

Explanation:
It was intended to perform a division by 0.

User Action:
Check whether this error can be prevented by using appropriate <predicate>s.

22019 INVALID ESCAPE VALUE

Explanation:
The input host variable is longer than 1 byte.

User Action:
Use a correct host variable.

INVALID ESCAPE VALUE

Explanation:
Exactly one character is valid for an escape value.

User Action:
Reduce the escape value to one character.

22023 INVALID NUMERIC INPUT PARAMETER VALUE

Explanation:
The input value cannot be converted.

User Action:
Specify the value in a valid notation (see SQL variable types in the chapter 'General SQL Variable Conventions' of the Precompiler online help).

22024 UNTERMINATED C STRING

Explanation:
The zero byte delimiter is missing.

User Action:
Insert a zero byte.

22025 INVALID ESCAPE SEQUENCE

Explanation:
The escape character may only be placed before a <match char> which is identical to the escape character, before a <match string>, or before a <match set> which is not a <match char>.

User Action:
Remove the exceeding escape character. Afterwards, the SQL statement can be reissued.

23000

INTEGRITY VIOLATION

Explanation:
Insertions or updates would violate integrity constraints specified in the base or view table definition.

User Action:
The error message specifies the column which would violate the integrity constraints.
Correct the input value for the corresponding column.

DUPLICATE KEY

Explanation:
There is already a table row with the key to be inserted.

User Action:
Check whether the existing table row contains the desired values. If this is not the case, check whether values in the existing table row can be replaced with the desired values. If a new table row must be inserted, change the value of the key to be inserted in order to prevent key collisions.

REFERENTIAL INTEGRITY VIOLATED

Explanation:
There is one of three possible causes.

1. An <insert statement> or <update statement> issued on a table that is the referencing table of a <referential constraint definition> produces a row that is not a matching row of the <referential constraint definition>.
2. When deleting rows from a <referenced table> of a <referential constraint definition> with <action> RESTRICT in the <delete rule>, a matching row exists.
3. When executing a <referential constraing definition>, the <referenced table>

or referencing table contains rows which conflict with the <referential constraint definition>.

User Action:
1. Display the definition of the <referential constraint definition> using a <query statement> issued on the system table DOMAIN.COL_REFS_COL. Correct the <insert statement> or <update statement> according to this definition.
2. Use an appropriate <query statement> to determine which row of the referencing table prevents the desired <referenced table> rows from being deleted.
3. Use an appropriate <query statement> to determine which row of the <referenced table> or referencing table conflicts with the <referential constraint definition> to be created. Modify or delete the row concerned, or correct the <referential constraint definition> to be created.

DUPLICATE KEY IN INDEX

Explanation:
There is already a table row with the specified secondary key. UNIQUE was specified for the secondary key.

User Action:
Correct the value of the secondary key to be inserted in the SQL statement in order to avoid a key value collision.
The error message specifies the column or multiple-column index already containing the specified values.

24000

DUPLICATE RESULT TABLE NAME

Explanation:
A result table generated by DECLARE CURSOR must be closed using a <close statement>, before the result table name can be used to open a new result table within the transaction.

User Action:
Insert a <close statement> into the ADABAS application.

SQL STATEMENT NOT ALLOWED WITHOUT PREVIOUS FETCH

Explanation:
The attempt was made to issue an SQL statement with CURRENT OF <result table name>, without having previously issued a successful <fetch statement> on the specified result table.

ADABAS Action
Repeat the SQL statement, once you have issued a successful <fetch statement> for the result table.

UNKNOWN RESULT TABLE

Explanation:
There is no result table (any more) with the specified name.

User Action:
Use a <query statement> issued on the system table DOMAIN.TABLES to find out the names of the existing result tables. Correct the name of the result table or check why the result table with the specified name was deleted.
A <commit statement> and <rollback statement> implicitly close all result tables.

26000 UNKNOWN STATEMENT NAME

Explanation:
The statement name is unknown.

User Action:
Issue the PREPARE statement or correct it.

40001

LOCK REQUEST TIMEOUT

Explanation:
The lock request or an implicit lock conflicts with the locks of another user. The maximum waiting time for granting the lock has elapsed (installation parameter REQUEST TIMEOUT).

User Action:
In some cases, the error message contains a more detailed description of the error. The lock request can be reissued. To avoid possible deadlock situations, it is advisable to roll back the transaction by using a <rollback statement>.

WORK ROLLED BACK

Explanation:
Your transaction was implicitly cancelled and rolled back by an implicit <rollback statement>, because

1. you failed to carry out any ADABAS operations within a certain period of time (installation parameter LOCK TIMEOUT), but held locks which other users were waiting for, or because
2. the SERVERDB was in a deadlock situation. A deadlock situation is a situation in which two or more users hold locks and request further locks that are held by the respective other users. In the simplest case of two users, one user holds one lock at least and requests another lock. But this lock is held by another user who, on the other hand, waits for the lock held by the first user. This situation can only be resolved if one of the users releases the lock already obtained.

User Action:
In some cases, the error message contains a more detailed description of the error. In both cases, the lock requests must be checked and modified, if necessary. The last transaction must be repeated.
It may also be necessary to check and modify the value of the installation

parameter LOCK TIMEOUT.

40003 SESSION INACTIVITY TIMEOUT (WORK ROLLED BACK)

Explanation:
Your transaction was implicitly cancelled and rolled back by an implicit <rollback statement>. The ADABAS session was implicitly terminated, since you failed to carry out any ADABAS operations within a certain period of time (installation parameter SESSION TIMEOUT or TIMEOUT value specified with the <connect statement>).

User Action:
Repeat the <connect statement> and specify a larger TIMEOUT value, if necessary. It may also be necessary to check and modify the value of the installation parameter SESSION TIMEOUT.

42000

MISSING IDENTIFIER

Explanation:
An <identifier> is missing.

User Action:
The error position indicates the location of the missing <identifier>. Insert an <identifier> into the SQL statement.

IDENTIFIER TOO LONG

Explanation:
The specified identifier is longer than 18 characters.

User Action:
Specify an identifier that does not exceed 18 characters.

MISSING INTEGER

Explanation:
An integer is missing.

User Action:
Insert an integer into the SQL statement.

MISSING CONSTANT

Explanation:
A constant is missing in the SQL statement.

User Action:
Insert a constant into the SQL statement.

PARAMETER SPEC NOT ALLOWED IN THIS CONTEXT

Explanation:
1. The attempt was made to specify a parameter in a <select column>.
2. The error message can also be returned if a <comparison predicate> of the format '<parameter spec> <comp op> <parameter spec>' occurs within the <search condition>.
3. The error message can also occur if a <comparison predicate>, <in predicate>, or <quantified predicate> specifies a comparison between a parameter and a <subquery>.

User Action:
1. Replace the parameter with a constant.
2. It is useful to check such a condition within the ADABAS application, not in ADABAS. If this is not possible, replace one of the two parameters with a constant, a column name, or an <expression> which does not only contain parameters.
3. Replace the parameter with a constant, so that the data type of the parameter becomes unique.

`RESERVED IDENTIFIER NOT ALLOWED`

Explanation:
The specified name is a reserved keyword and must not be used to identify database objects.

User Action:
Correct the SQL statement using another <identifier>.

`INVALID KEYWORD OR MISSING DELIMITER`

Explanation:
The SQL statement contains an incorrect keyword or a keyword that is unknown, or a keyword or delimiter is missing.

User Action:
Correct the SQL statement according to the syntax description.

`COLUMN MUST BE GROUP COLUMN`

Explanation:
A column which is not a group column was specified in a <select column> or <having clause>. Columns which are not group columns may only occur in arguments of the functions COUNT, SUM, AVG, MAX, or MIN.

User Action:
Insert the specified column into the <group clause> as further group column or remove it from the <select column> or <having clause>.

`UNKNOWN COLUMN NAME`

Explanation:
There is no column with the specified name in any of the specified tables.

User Action:
Use <query statement>s issued on the system table DOMAIN.COLUMNS to find out the names of the columns existing in the tables. Correct the column name.

UNKNOWN TABLE NAME

Explanation:
A table with the specified name is not known to you. This table may not exist; or this table exists but you have no privileges for it.

User Action:
Use a <query statement> issued on the system table DOMAIN.TABLES to find out the names of the tables for which you have privileges. Then correct the table name. It may be sufficient to place the missing <owner> in front of it. Otherwise, create a table with the desired name or check why you have no privileges for the existing table.

UNION COLUMNS MUST BE COMPATIBLE

Explanation:
In a <query expression> with at least a UNION specification, all sequences of <select column>s must designate the same number of <select column>s. The data types amd lengths of the corresponding columns must be identical. It is also necessary that only <column spec>s or '*' may be specified in the sequences of <select column>s of the <query spec>s connected by UNION. The specification of <literal>s is not allowed.

User Action:
This request cannot be made.

INVALID SQL STATEMENT

Explanation:
The SQL statement either contains a typing error within the first two keywords, or is unknown, or is not permitted in this ADABAS version.

User Action:
Correct the typing errors, or specify another SQL statement.

INVALID UNSIGNED INTEGER

Explanation:
No valid number was specified.

User Action:
Correct the SQL statement.

INVALID DATATYPE

Explanation:
The specified data type is unknown.

User Action:

The valid data types are described in the chapter '<column definition>' in this online help. Use one of the data types specified there.

INVALID TABLE NAME

Explanation:
The specified table name does not comply with the syntax for <identifier>s.

User Action:
Correct the table name specified in the SQL statement.

INVALID END OF SQL STATEMENT

Explanation:
According to the syntax, the specified SQL statement is not allowed.

User Action:
The error position shows the location where the specified SQL statement deviates from the permitted syntax.
Correct the SQL statement accordingly.

VARIABLE IN VIEW DEFINITION NOT ALLOWED IN VIEW

Explanation:
Parameters must not be specified in the <create view statement>.

User Action:
Use constants instead of variables.

MISSING VALUE SPECIFICATION

Explanation:
A value is missing, or the specified value is not allowed.

User Action:
Correct the value specified in the SQL statement, or insert a value into the SQL statement.

MISSING NUMERIC CONSTANT

Explanation:
A number is missing.

User Action:
The error position indicates the location of the missing number. Insert a number into the SQL statement.

NUMERIC CONSTANT TOO LONG

Explanation:
A number was entered which
1. contains more than 18 digits or

2. does not comply with the definition of the range of values.

User Action:
The number which was incorrectly entered may be found out from the position specification in the error message.
1. The number must be reduced to 18 significant digits and be specified as <floating point literal>, if necessary.
2. The definition of the range of values must be checked and the specification of the number must be corrected accordingly.

MISSING STRING CONSTANT

Explanation:
A string constant is missing in the issued SQL statement.

User Action:
Insert a <string literal> into the SQL statement.

TOO FEW COLUMNS

Explanation:
For a <referential constraint definition>, less <referencing column>s than <referenced column>s were specified. The number of columns specified for the referencing table must correspond to the number of the <referenced column>s or the <referenced table> specified implicitly or explicitly.

User Action:
Use a <query statement> issued on the system table DOMAIN.COLUMNS to determine the definition of the key columns of the <referenced table>. Use a <query statement> issued on the system table DOMAIN.IND_USES_COL to find out the indexes of the <referenced table>. The specification of the referencing table columns must be adapted accordingly.

TOO FEW VALUES

Explanation:
In case of an <insert statement> or <update statement>, the number of specified values is less than the number of column names (possibly implicitly specified).

User Action:
Adapt the number of specified values in the SQL statement to the number of specified column names.
Use a <query statement> issued on the system table DOMAIN.COLUMNS for an <insert statement> without column name specification to determine the definition of the used table.

TOO MANY VARIABLES

Explanation:
A maximum of 300 variables may be specified per SQL statement.

User Action:

Decrease the number of variables in the SQL statement. Some variables must be replaced with constant values. If this is not possible, split the SQL statement into several SQL statements.

TOO MANY VALUES

Explanation:
In case of an <insert statement> or <update statement>, the number of specified values exceeds the number of column names (possibly implicitly specified).

User Action:
Adapt the number of specified values in the SQL statement to the number of specified column names.
Use a <query statement> issued on the system table DOMAIN.COLUMNS for an <insert statement> without column name specification to find out the definition of the used table.

MISSING PRIVILEGE

Explanation:
You are not authorized to execute the SQL statement.

User Action:
Use a <show privileges statement> to find out the privileges you have received for the specified table. It is not possible to execute the desired SQL statement.

44000 VIEW VIOLATION

Explanation:
An <insert statement> or <update statement> was issued for a view table. At least one of the rows specified in the SQL statement does not satisfy the <search condition>s of all underlying view tables defined 'WITH CHECK OPTION'.

User Action:
Display the definition of the view table using a <query statement> issued on the system table DOMAIN.VIEWDEFS. Correct the <insert statement> or <update statement> according to this definition.

Ixxxx

Explanation:
SQLSTATEs starting with 'I' are SQLSTATEs which are not predefined by the standard. For the Explanation and User Actions, see the online help Messages and Codes. To find out the pertinent description, replace the 'I' of the SQLSTATE with a '-'. Leading '0's are omitted.

User Action:
See the online help Messages and Codes.

Sxxxx SYSTEM ERROR

Explanation:
These error messages should not occur during normal operation on a consistent

database.
With some errors, an implicit SHUTDOWN is issued.

User Action:
As a rule, the database should be shut down and the ADABAS Support be informed. It is possible to create a trace of the last database activities. Write this trace to magnetic tape and send it to the ADABAS Support for error tracing and correction.

Syntax

```
<alias name> ::=
     <identifier>

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

<all set function name> ::=
     MAX
   | MIN
   | SUM
   | AVG

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

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

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

<boolean term> ::=
     <boolean factor>
   | <boolean term> AND <boolean factor>
<character> ::=
     <digit>
   | <letter>
   | <extended letter>
   | <language specific character>
   | <special character>

<close statement> ::=
     CLOSE <result table name>

<column attributes> ::=
     [<default spec>]
     [NOT NULL [<unique spec>] ]
     [REFERENCES <table name> [(<column name>)]]
     [<constraint definition>]

<column definition> ::=
     <column name> <data type> <column attributes>


<column name> ::=
     <identifier>

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

<commit statement> ::=
     COMMIT WORK

<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>

<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>)

<create schema statement> ::=
     CREATE SCHEMA AUTHORIZATION <schema name>

<create table statement> ::=
     CREATE TABLE <table name> (<table description element>...)

<create view statement> ::=
     CREATE VIEW <table name> [(<alias name>,...)]
     AS <query expression>
     [WITH CHECK OPTION]

<data type> ::=
     CHAR[ACTER] [(<unsigned integer>)]
   | DEC[IMAL]   (<unsigned integer> [,<unsigned integer>])
   | NUMERIC (<unsigned integer> [,<unsigned integer>])
   | SMALLINT
   | INT[EGER]
   | FLOAT [(<unsigned integer>)]
   | REAL
   | DOUBLE PRECISION

<declare cursor statement> ::=
     DECLARE <result table name> CURSOR FOR <select statement>

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

<default value> ::=
     <literal>
   | NULL
   | USER

<delete statement> ::=
     DELETE FROM <table name>
            [WHERE <search condition>]
   | DELETE FROM <table 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> [ [AS] <result column name>]
```

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

<distinct function> ::=
     <set function name> ( DISTINCT <column spec> )

<distinct spec> ::=
     DISTINCT
   | ALL

<double quotes> ::=
     "

<exists predicate> ::=
     EXISTS <subquery>

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


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

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

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

<fetch statement> ::=
     FETCH <result table name> INTO <parameter spec>,...

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

<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>,...

<grant statement> ::=
     GRANT <table privileges> ON <table name>
     TO <grantee>,... [WITH GRANT OPTION]

<grantee> ::=
     PUBLIC
   | <user name>
   | <usergroup name>

<group clause> ::=
     GROUP BY <column spec>,...
```

```
<having clause> ::=
     HAVING <search condition>

<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 (<value spec>,...)

<indicator name> ::=
     <parameter name>

<insert columns and values> ::=
     [(<column name>,...)] VALUES (<value spec>,...)
   | [(<column name>,...)] <query expression>

<insert statement> ::=
     INSERT INTO <table name> <insert columns and values>

<isolation spec> ::=
     ISOLATION LEVEL <unsigned integer>

<join predicate> ::=
     <expression> <comp op> <expression>

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

<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>]

<literal> ::=
```

```
     <string literal>
   | <numeric literal>

<mantissa> ::=
     <fixed point literal>

<match char> ::=
     Every character except
     %, X'1F', <underscore>, X'1E'.

<match set> ::=
     <underscore>
   | X'1E'
   | <match char>

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

<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>


<open cursor statement> ::=
     OPEN <result table name>

<order clause> ::=
     ORDER BY <sort spec>,...

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

<parameter name> ::=
     :<identifier>

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

<password spec> ::=
     <parameter name>

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

<predicate> ::=
     <between predicate>
   | <comparison predicate>
   | <exists predicate>
   | <in predicate>
   | <join predicate>
   | <like predicate>
```

```
   | <null predicate>
   | <quantified predicate>

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

<privilege> ::=
     INSERT
   | UPDATE [(<column name>,...)]
   | SELECT
   | DELETE
   | REFERENCES [(<column name>,...)]

<quantified predicate> ::=
     <expression> <comp op> <quantifier> <subquery>


<quantifier> ::=
     ALL
   | <some>

<query expression> ::=
     <query primary>
   | <query expression> UNION [ALL] <query primary>

<query primary> ::=
     <query spec>
   | (<query expression>)

<query spec> ::=
     SELECT [<distinct spec>] <select column>,...
     <table expression>

<query statement> ::=
     <declare cursor statement>

<reference name> ::=
     <identifier>

<referenced column> ::=
     <column name>

<referenced table> ::=
     <table name>

<referencing column> ::=
     <column name>

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

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

<release statement> ::=
     COMMIT   WORK RELEASE
```

```
   | ROLLBACK WORK RELEASE

<reserved key word> ::=
   ACTION       ADD          ALL          ALTER        AND
   ANY          AS           ASC          AT           AVG

   BEGIN        BETWEEN      BIT          BOTH         BY

   CASCADE      CAST         CATALOG      CHAR         CHARACTER
   CHECK        CLOSE        COLUMN       COMMIT       CONNECT
   CONSTRAINT   COUNT        CREATE       CURRENT      CURRENT_DATE
   CURRENT_TIME CURSOR

   DATE         DAY          DEC          DECIMAL      DECLARE
   DEFAULT      DELETE       DESC         DESCRIBE     DISCONNECT
   DISTINCT     DOMAIN       DOUBLE       DROP

   END          ESCAPE       EXCEPT       EXECUTE      EXISTS
   EXTRACT

   FALSE        FETCH        FIRST        FLOAT        FOR
   FOREIGN      FROM

   GET          GRANT        GROUP

   HAVING       HOUR

   IN           INDICATOR    INNER        INSERT       INT
   INTEGER      INTERSECT    INTO         IS           ISOLATION

   JOIN

   KEY

   LANGUAGE     LAST         LEADING      LEFT         LEVEL
   LIKE         LOCAL        LOWER

   MAX          MIN          MINUTE       MONTH

   NATURAL      NEXT         NO           NOT          NULL
   NUMERIC

   OF           ON           ONLY         OPEN         OPTION
   OR           ORDER        OUTER

   PRECISION    PRIMARY      PRIVILEGES   PROCEDURE    PUBLIC

   READ         REAL         REFERENCES   RESTRICT     REVOKE
   RIGHT        ROLLBACK     ROWS

   SCHEMA       SECOND       SELECT       SET          SMALLINT
   SOME         SUM

   TABLE        TIME         TIMESTAMP    TO           TRAILING
   TRANSACTION  TRANSLATE    TRIM         TRUE

   UNION        UNIQUE       UNKNOWN      UPDATE       UPPER
   USAGE        USER         USING

   VALUE        VALUES       VARCHAR      VARYING      VIEW

   WHENEVER     WHERE        WITH         WORK         WRITE
```

```
   YEAR

<restricted key word> ::=
   ABS          ACOS         ADDDATE      ADDTIME      ALPHA
   ASCII        ASIN         ATAN         ATAN2        AUDIT

   BINARY       BOOLEAN      BUFFERPOOL   BYTE

   CEIL         CEILING      CHR          CLUSTER      COMMENT
   CONCAT       CONNECTED    COS          COSH         COT
   CURDATE      CURTIME

   DATABASE     DATEDIFF     DAYNAME      DAYOFMONTH   DAYOFWEEK
   DAYOFYEAR    DBYTE        DECODE       DEGREES      DIGITS
   DIRECT

   EBCDIC       EDITPROC     ENTRY        ENTRYDEF     EXCLUSIVE
   EXP          EXPAND

   FIXED        FLOOR

   GRAPHIC      GREATEST

   HEX

   IDENTIFIED   IFNULL       IGNORE       INDEX        INITCAP

   LCASE        LEAST        LENGTH       LFILL        LINK
   LIST         LN           LOCALSYSDBA  LOCK         LOG
   LOG10        LONG         LPAD         LTRIM

   MAKEDATE     MAKETIME     MAPCHAR      MICROSECOND  MINUS
   MODE         MODIFY       MONTHNAME

   NOROUND      NOW          NOWAIT       NUM          NUMBER

   OBID         OBJECT       OPTIMIZE

   PACKED       PCTFREE      PI           POWER        PREV

   RADIANS      RAW          REFERENCED   REJECT       RELEASE
   RENAME       REPLACE      RESOURCE     RFILL        ROUND
   ROW          ROWID        ROWNO        ROWNUM       RPAD
   RTRIM


   SELUPD       SHARE        SHOW         SIGN         SIN
   SINH         SOUNDEX      SQRT         STAMP        STATISTICS
   STDDEV       SUBDATE      SUBSTR       SUBTIME      SYNONYM
   SYSDATE      SYSDBA

   TABLESPACE   TAN          TANH         TIMEDIFF     TIMEZONE
   TOIDENTIFIER TRIGGER      TRUNC        TRUNCATE

   UCASE        UID          USERGROUP

   VALIDPROC    VARCHAR2     VARGRAPHIC   VARIANCE

   WEEKOFYEAR

   ZONED

<result column name> ::=
```

```
     <identifier>

<result table name> ::=
     <identifier>

<rollback statement> ::=
     ROLLBACK WORK

<schema name> ::=
     <identifier>

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

<select column> ::=
     <table columns>
   | <derived column>

<select statement> ::=
     <query expression>
     [<order clause>]

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


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

<set update clause> ::=
     <column name> = <expression>

<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>]
   | <column spec> [<sort option>]

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

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

<special identifier character> ::=
     Any character.


<sql statement> ::=
     <create schema statement>
   | <create table statement>
   | <create view statement>

   | <grant statement>

   | <insert statement>
   | <update statement>
   | <delete statement>

   | <query statement>
   | <open cursor statement>
   | <fetch statement>
   | <close statement>
   | <single select statement>

   | <connect statement>
   | <commit statement>
   | <rollback statement>
   | <release statement>

<sqlmode spec> ::=
     ADABAS
   | ANSI
   | DB2
   | ORACLE

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

<subquery> ::=
     (<query expression>)

<table columns> ::=
     *

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

<table privileges> ::=
     ALL PRIVILEGES
   | <privilege>,...

<table spec> ::=
     <table name> [<reference name>]

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

<termchar set name> ::=
     <identifier>

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

<underscore> ::=
     _

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

<unique spec> ::=
     PRIMARY KEY
   | UNIQUE

<unsigned integer> ::=
     <digit>...

<update columns and values> ::=
     SET <set update clause>,...

<update statement> ::=
     UPDATE <table name>
            <update columns and values>
            [WHERE <search condition>]
   | UPDATE <table name>
            <update columns and values>
            WHERE CURRENT OF <result table name>

<user name> ::=
     <identifier>


<user spec> ::=
     <parameter name>
   | <user name>

<usergroup name> ::=
     <identifier>

<value spec> ::=
     <literal>
   | <parameter spec>
```

```
   | NULL
   | USER

<where clause> ::=
     WHERE <search condition>
```