Contents

Manual Ordering

Manual Order Number: ESD611-032WOU

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 'Data Definition' describes the SQL statements for the definition of tables etc.
The chapter 'Authorization' explains the protective mechanisms against illegal access and illegal modifications to the data.
The chapter 'Data Manipulation' describes the SQL statements for the insertion, update, and deletion of data.
The chapter 'Data Retrieval' deals with the SQL statements for data access.
The chapter 'Transactions' deals with the mechanisms for the maintenance of the consistency as well as for the synchronization of the ADABAS server.
The chapter 'System Tables' describes the view tables that contain information about the database objects and their relationships to each other and to programs.
The chapter 'Restrictions' lists the restrictions which generally apply to data types, parameters, identifiers, etc.
The chapter 'Introduction' specifies the differences that exist between the syntax and semantics in ADABAS and the Definition of ORACLE7.
The chapter 'Error Messages' lists the error messages which can occur in addition to those specified 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, a number, a date value, or a value of a LONG column.

See also

Character String

LONG Column

Number

Date Value

Character String

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

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

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

LONG Column

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

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

Number

1. There are fixed point and floating point numbers.

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

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

4. All numbers can be compared to each other.

Date Value

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

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

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

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

Column

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

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

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

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

Index

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

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

Synonym

1. A synonym is another name for a table.

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

User and Usergroup

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

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

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

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

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

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

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

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

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

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

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

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

Privilege

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

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

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

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

Database

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

2. The catalog consists of metadata. The definitions of database objects such as base tables, view tables, synonyms, 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 chapter 'Session') of the user who wants to make these modifications was not started on the SERVERDB where the data to be modified is stored, the two SERVERDBs must be connected to each other within the network.

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

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

11. To prevent the two subnetworks from contradictory modifications to the replicated (metadata) data, ADABAS determines the subnetwork with the greater number (the so-called majority) of SERVERDBs within the whole network. The subnetwork containing the majority is then allowed to modify the (metadata) data. This procedure is called the majority concept. For two subnetworks of equal size, ADABAS decides the one that is to represent the majority. The minority subnetwork is not allowed to

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

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

Transaction

1. A transaction is a sequence of database operations which form a unit with regard to data backup and synchronization. Transactions are closed with COMMIT or ROLLBACK. If a transaction is closed with COMMIT, all modifications made to the database within the transaction are kept. If a transaction is aborted with ROLLBACK, all modifications made to the database within this transaction are cancelled.
   Modifications closed with COMMIT cannot be cancelled with ROLLBACK.
   Each data definition and authorizing statement is preceded and concluded with COMMIT.
   COMMIT and ROLLBACK implicitly open a new transaction.

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

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

Subtransaction

1. Within a transaction, subtransactions can be defined which let a series of database operations within a transaction appear as a unit with regard to modifications to the database.

2. SAVEPOINT defines a position, i.e., the starting point for a subtransaction within a transaction, and assigns a name to this position. Any modifications made in the meantime can be rolled back by using ROLLBACK TO SAVEPOINT with a specification of the name, without influencing database operations that were performed within the transaction before this subtransaction.

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

Session

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

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

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

Data Integrity

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

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

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

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

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

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

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

Snapshot Table

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

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

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

Backup and Recovery Concept

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

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

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

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

SQLMODE

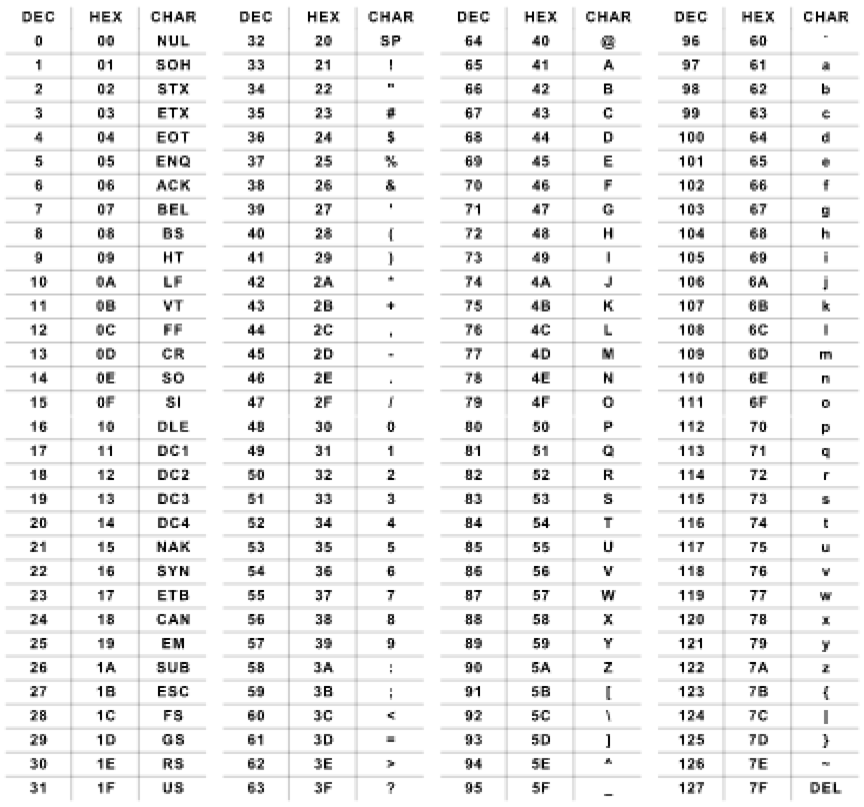
1. The database system ADABAS is able to perform correct ADABAS applications, as well as applications that are written according to the ANSI standard (ANSI X3.135-1992, Entry SQL), the definition of DB2 Version 3, or the definition of ORACLE7. ADABAS is able to check whether ADABAS applications conform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

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

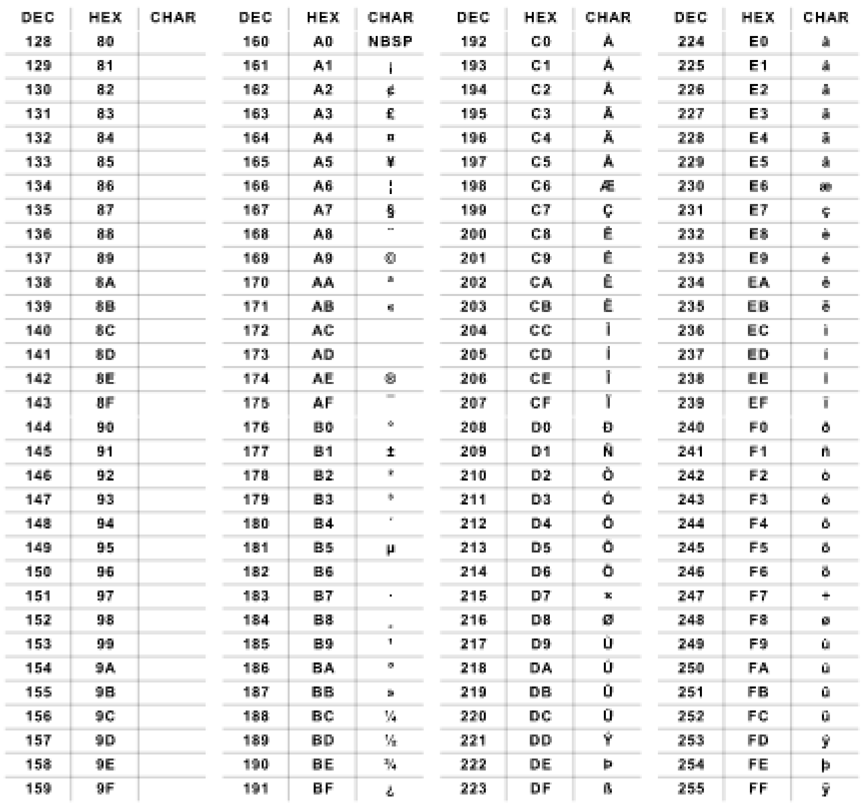
Code Tables

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

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



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



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

☐ possibly set by the operating system

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

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

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

<character>

Function

defines the elements of character strings and of key words.

Format

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

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

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

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

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

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



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

Syntax Rules

none

General Rules

none

<literal>

Function

specifies a non-NULL value.

Format

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

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

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

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

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

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

<sign> ::=
    +
  | -



<unsigned integer> ::=
    <digit>

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

<mantissa> ::=
    <fixed point literal>

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

Syntax Rules

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

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

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

General Rules

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

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

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

<token>

Function

specifies lexical units.

Format

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

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

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


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

  BAD           BEGINLOAD     BLOCKSIZE     BUFFER

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

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

  ENDLOAD       ENDPOS        EUR           EXPLAIN       EXPLICIT

  FIRSTPOS      FNULL         FORCE         FORMAT        FREAD
  FREEPAGE      FWRITE



  GATEWAY       GRANTED

  HEXTORAW      HOLD          HOURS

  IMPLICIT      INDEXNAME     INIT          INITRANS      INSTR
  INTERNAL      ISO

  JIS

  KEEP

  LABEL         LASTPOS       LAST_DAY      LOAD
  MAXTRANS      MDECLARE      MDELETE       MFETCH        MICROSECONDS
  MINSERT       MINUTES       MLOCK         MOD           MONITOR
  MONTHS      MONTHS_BETWEEN  MSELECT       MUPDATE

  NEW_TIME      NEXTVAL       NEXT_DAY      NOLOG         NORMAL
  NOSORT        NVL

  OFF           OPTIMISTIC    ORACLE        OUT           OVERWRITE

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

  QUICK

  RANGE         RAWTOHEX      RECONNECT     REFRESH       REPLICATION
  REST          RESTART       RESTORE       REUSE         RFETCH

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

  TABID         TABLEDEF      TEMP          TEMPLIMIT     TERMCHAR
  TIMEOUT       TO_CHAR       TO_DATE       TO_NUMBER     TRANSFILE
  TRIGGERDEF

  UNLOAD        UNLOCK        UNTIL         USA           USERID

  VERIFY        VERSION       VSIZE         VTRACE

  WAIT

  YEARS


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

  BEGIN         BINARY        BIT           BOOLEAN       BOTH
  BUFFERPOOL    BYTE

  CASCADE       CAST          CATALOG       CEIL          CEILING
  CHARACTER     CHR           CLOSE         COMMIT        CONCAT
  CONNECTED     CONSTRAINT    COS           COSH          COT
  COUNT         CURDATE       CURRENT_DATE  CURRENT_TIME  CURSOR
  CURTIME

  DATABASE      DATEDIFF      DAY           DAYNAME       DAYOFMONTH
  DAYOFWEEK     DAYOFYEAR     DBYTE         DEC           DECLARE
  DECODE        DEGREES       DESCRIBE      DIGITS        DIRECT
  DISCONNECT    DOMAIN        DOUBLE

  EBCDIC        EDITPROC      END           ENTRY         ENTRYDEF
  ESCAPE        EXCEPT        EXECUTE       EXP           EXPAND
  EXTRACT

  FALSE         FETCH         FIRST         FIXED         FLOOR
  FOREIGN
  GET           GRAPHIC       GREATEST

  HEX           HOUR

  IFNULL        IGNORE        INDICATOR     INITCAP       INNER
  INT           ISOLATION

  JOIN

  KEY

  LANGUAGE      LAST          LCASE         LEADING       LEAST
  LEFT          LENGTH        LFILL         LINK          LIST
  LN            LOCAL         LOCALSYSDBA   LOG           LOG10
  LOWER         LPAD          LTRIM

  MAKEDATE      MAKETIME      MAPCHAR       MAX           MICROSECOND
  MIN           MINUTE        MONTH         MONTHNAME

  NATURAL       NEXT          NO            NOROUND       NOW
  NUM           NUMERIC


  OBID          OBJECT        ONLY          OPEN          OPTIMIZE
  OUTER

  PACKED        PI            POWER         PRECISION     PREV
  PRIMARY       PROCEDURE

  RADIANS       READ          REAL          REFERENCED    REFERENCES
  REJECT        RELEASE       REPLACE       RESTRICT      RFILL
  RIGHT         ROLLBACK      ROUND         ROWNO         RPAD
  RTRIM

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

  TABLESPACE    TAN           TANH          TIME          TIMEDIFF
  TIMESTAMP     TIMEZONE      TOIDENTIFIER  TRAILING      TRANSACTION
  TRANSLATE     TRIM          TRUE          TRUNC         TRUNCATE

  UCASE         UNKNOWN       UPPER         USAGE         USERGROUP
  USING

  VALIDPROC     VALUE         VARGRAPHIC    VARIANCE      VARYING

  WEEKOFYEAR    WORK          WRITE

  YEAR

  ZONED

<reserved key word> ::=
  ADD           ALL           ALTER         AND           ANY
  AS            ASC           AUDIT

  BETWEEN       BY
  CHAR          CHECK         CLUSTER       COLUMN        COMMENT
  CONNECT       CREATE        CURRENT

  DATE          DECIMAL       DEFAULT       DELETE        DESC
  DISTINCT      DROP

  EXCLUSIVE     EXISTS

  FLOAT         FOR           FROM


  GRANT         GROUP

  HAVING

  IDENTIFIED    IN            INDEX         INSERT        INTEGER
  INTERSECT     INTO          IS

  LEVEL         LIKE          LOCK          LONG

  MINUS         MODE          MODIFY

  NOT           NOWAIT        NULL          NUMBER

  OF            ON            OPTION        OR            ORDER

  PCTFREE       PRIVILEGES    PUBLIC

  RAW           RENAME        RESOURCE      REVOKE        ROW
  ROWID         ROWNUM        ROWS

  SELECT        SET           SHARE         SMALLINT      SYNONYM
  SYSDATE

  TABLE         TO            TRIGGER

  UID           UNION         UNIQUE        UPDATE        USER

  VALUES        VARCHAR       VARCHAR2      VIEW

  WHENEVER      WHERE         WITH
```

```
<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>

<constraint name> ::=
    <identifier>

<index name> ::=
    <identifier>

<reference name> ::=
    <identifier>

<referential constraint name> ::=
    <identifier>

<result table name> ::=
    <identifier>

<sequence name> ::=
    <identifier>



<synonym name> ::=
    <identifier>

<termchar set name> ::=
    <identifier>

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

<parameter name> ::=
    :<identifier>

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

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

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

Syntax Rules

1. All names are truncated after the 18th character.

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

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

General Rules

1. A <user name> identifies a user.

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

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

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

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

8. 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.

9. A <referential constraint name> identifies a referential integrity rule which is created by a <referential constraint definition> in the <create table statement> or in the <alter

table statement> defining delete or existence conditions between two tables.

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

11. A <sequence name> identifies a sequence which is generated by a <create sequence statement>.

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

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

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

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

16. 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>
  | <result table name>.<column name>
```

Syntax Rules

none

General Rules

none

<parameter spec>

Function

specifies a parameter.

Format

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

Syntax Rules

none

General Rules

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

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

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

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

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

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

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

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

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

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

Specifying Values

Function

specifies a value.

Format

```
<value spec> ::=
    <literal>
  | <parameter spec>
  | NULL
  | USER
  | [<owner>.]<sequence name>.NEXTVAL
  | [<owner>.]<sequence name>.CURRVAL
  | SYSDATE
  | UID

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

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.

3. [<owner>.]<sequence name>.NEXTVAL denotes the next value that will be generated for the specified <sequence name>. The rule according to which the next value is generated is specified in the <create sequence statement>. The interval between the values is also specified in the <create sequence statement>. Sequences can be used to generate unique values. These are not uninterrupted because values generated in a transaction that is rolled back cannot be reused.

4. [<owner>.]<sequence name>.CURRVAL denotes the last value that was generated for the specified sequence name using [<owner>.]<sequence name>.NEXTVAL.

5. SYSDATE denotes the current date and the current time.

6. UID denotes the user number of a user.

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

<function spec>

Function

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

Format

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

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

Syntax Rules

none

General Rules

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

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

See also

<arithmetic function>

<trigonometric function>

<string function>

<date and time function>

<special function>

<conversion function>

<arithmetic function>

Function

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

Format

```
<arithmetic function> ::=
    TRUNC   ( <expression>[, <expression>] )
  | ROUND   ( <expression>[, <expression>] )
  | CEIL    ( <expression> )
  | FLOOR   ( <expression> )
  | SIGN    ( <expression> )
  | ABS     ( <expression> )
  | POWER   ( <expression>, <expression> )
  | EXP     ( <expression> )
  | SQRT    ( <expression> )
  | LN      ( <expression> )
  | LOG     ( <expression>, <expression> )
  | MOD     ( <expression>, <expression> )
  | LENGTH  ( <expression> )
  | VSIZE   ( <expression> )
  | INSTR   ( <string spec>, <string spec>
              [,<expression>[, <expression>] ] )
  | ASCII   ( <expression> )
```

Syntax Rules

none

General Rules

1. TRUNC
   Let a and s be numbers.
   If s>0, then TRUNC(a,s) is the number a truncated s digits after the decimal point.
   If s=0, then TRUNC(a,s) is the integral part of a.
   If s<0, then TRUNC(a,s) is the number a truncated s digits before the decimal point.
   If s is not specified, then the value 0 is implicitly assumed for s.
   If s is not an integer value, then the integral part of s is used.
   If a is a floating point number, then the result is a floating point number. Otherwise, the result is a fixed point number. If a is the NULL value, then TRUNC(a,s) is the NULL value.
   For the description of the function TRUNC applied to DATE values or to character strings that conform to the date format, see chapter '<date and time function>>'.

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

If s is not an integer value, then the integral part of s is used.
If a is a floating point number, then the result is a floating point number. Otherwise, the result is a fixed point number. If a is the NULL value, then ROUND(a,s) is the NULL value.
For the description of the function ROUND applied to DATE values or to character strings that conform to the date format, see chapter '<date and time function>'.

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

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

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

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

   If a is the NULL value, then SIGN(a) is the NULL value.

6. ABS
   If a is a number, then ABS(a) is the absolute value of a. If a is the NULL value, then ABS(a) is the NULL value.

7. POWER
   Let a and b be numbers, then POWER(a,b) = ab. If b is not an integer value, then an error message is output. If a or b is the NULL value, then the result is the NULL value.

8. EXP
   Let a be a number, then EXP(a) = ea, where e = 2.71828183. If a is the NULL value, then the result is the NULL value.

9. SQRT
   Let a be a number > 0, then SQRT(a) is the square root of a. If a is a number = 0, then the result of SQRT(a) is 0. If a is a number < 0 or a is the NULL value, then the result is the NULL value.

10. LN
    Let a be a number, then LN(a) is the natural logarithm of a. If a is the NULL value, then the result is the NULL value.

11. LOG
    Let a be a number, then LOG(a,b) is the logarithm b to the base of a. If a or b is the NULL value, then the result is the NULL value.

12. MOD

If a and b are integer numbers and ABS(a)<1E18 and 0<b<1E18, then the following applies:
Let m be the integer remainder resulting from the integer division of a by b.

```
If m>=0, then MOD(a,b) = m
If m<0,  then MOD(a,b) = m+b
```

If b=0, then the result of MOD(a,b) is equal to a. If one of the specified conditions is not satisfied, an error message is output.

13. LENGTH
    LENGTH can be applied to any data type. LENGTH(a) indicates the length that would be needed to represent the characters of a. For numbers, the digits after the decimal point only consisting of '0' and the decimal point are not counted. If a is the NULL value, then LENGTH(a) is the NULL value.

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

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

16. ASCII
    ASCII can be applied to any data type.
    The result of ASCII(a) is a number that represents the decimal value of the first byte in the character representation of a. For numbers, this could be the minus sign. If a is the NULL value, then the result of ASCII(a) is the NULL value.

<trigonometric function>

Function

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

Format

```
<trigonometric function> ::=
    COS     ( <expression> )
  | SIN     ( <expression> )
  | TAN     ( <expression> )
  | COSH    ( <expression> )
  | SINH    ( <expression> )
  | TANH    ( <expression> )
```

Syntax Rules

none

General Rules

1. All <trigonometric function>s produce the NULL value as the result if the <expression> produces the NULL value.

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

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

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

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

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

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

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

<string function>

Function

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

Format

```
 <string function> ::=
     <string spec> || <string spec>
   | CONCAT    ( <string spec>, <string spec> )
   | SUBSTR    ( <string spec>, <expression>[, <expression>] )
   | LPAD      ( <string spec>, <unsigned integer>
                 [, <string literal> ] )
   | RPAD      ( <string spec>, <unsigned integer>
                 [, <string literal> ] )
   | LTRIM     ( <string spec>[, <string spec> ] )
   | RTRIM     ( <string spec>[, <string spec> ] )
   | UPPER     ( <string spec> )
   | LOWER     ( <string spec> )
   | INITCAP   ( <string spec> )
   | REPLACE   ( <string spec>, <string spec>[, <string spec> ] )
   | TRANSLATE ( <string spec>, <string spec>, <string spec> )
   | SOUNDEX   ( <string spec> )
```

Syntax Rules

none

General Rules

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

2. Concatenation, CONCAT
   The concatenation CONCAT(x,y) produces the same result as the concatenation x||y.

3. SUBSTR
   If x is a character string of length n, then SUBSTR(x,a,b) is that part of the character string x which begins at the ath character and has a length of b characters.
   SUBSTR(x,a) corresponds to SUBSTR(x,a,n-a+1) and produces all characters of the character string x from the ath character to the last character (nth).
   If b is specified as <unsigned integer>, then a value greater than (n-a+1) is also valid for b. In all the other cases, the value of b must not exceed the value (n-a+1). If b > (n-

a+1), then SUBSTR(x,a) is performed internally. As many blanks (code attribute ASCII or EBCDIC) or binary zeros (code attribute BYTE) are appended to the end of this result as are needed to give the result the length b.
If x, a or b is the NULL value, then SUBSTR(x,a,b) is the NULL value.

4. LPAD
   At the beginning of the character string defined as the first parameter, LPAD inserts the character defined as the third parameter as often as is needed to give the character string the length specified in the second parameter. If the first parameter is a character string with the code attribute ASCII or EBCDIC, then the third parameter must be a <string literal> consisting of a single character. If the first parameter is a character string with the code attribute BYTE, the third parameter must be a <hex literal> designating a single character, therefore consisting of two <hex digit>s. If the third parameter is not specified, then a blank (code attribute ASCII or EBCDIC) or a binary zero (code attribute BYTE) is inserted. If the first parameter is the NULL value, then LPAD produces the NULL value as the result. If the second or third parameter is the NULL value, then an error message is output.

5. RPAD
   At the end of the character string defined as the first parameter, RPAD inserts the character defined as the third parameter as often as is needed to give the character string the length specified in the second parameter. If the first parameter is a character string with the code attribute ASCII or EBCDIC, then the third parameter must be a <string literal> consisting of a single character. If the first parameter is a character string with the code attribute BYTE, then the third parameter must be a <hex literal> designating a single character, therefore consisting of two <hex digit>s. If the third parameter is not specified, then a blank (code attribute ASCII or EBCDIC) or a binary zero (code attribute BYTE) is inserted. If the first parameter is the NULL value, then RPAD produces the NULL value as the result. If the second or third parameter is the NULL value, then an error message is output.

6. LTRIM
   LTRIM removes all characters specified in the second parameter from the beginning of the character string specified as first parameter, so that the result of LTRIM begins with the first character that was not specified in the second parameter. If no second parameter is specified, then a blank is implicitly assumed. The length of the character string decreases accordingly. LTRIM applied to the NULL value produces the NULL value as the result.

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

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

9. INITCAP
   INITCAP changes the character string in such a way that the first character of a word is an uppercase character and the rest of the word consists of lowercase characters.

Words are separated by one or more characters which are neither letters nor digits. INITCAP applied to the NULL value produces the NULL value.

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

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

12. SOUNDEX
    SOUNDEX applies the soundex algorithm to the character string and produces a value of data type CHAR (4) as the result. SOUNDEX applied to the NULL value produces the NULL value as the result.

<date and time function>

Function

specifies functions which operate on the data type DATE.

Format

```
<date and time function> ::=
    ADD_MONTHS     ( <date and time expression>, <expression> )
  | MONTHS_BETWEEN ( <date and time expression>,
                     <date and time expression> )
  | LAST_DAY       ( <date and time expression> )
  | NEXT_DAY       ( <date and time expression>, <string spec> )
  | NEW_TIME       ( <date and time expression>,
                     <source timezone spec>,
                     <dest timezone spec> )
  | ROUND          ( <date and time expression>
                     [, <trunc and round format> ] )
  | TRUNC          ( <date and time expression>
                     [, <trunc and round format> ] )

<date and time expression> ::=
    <expression>

<source timezone spec> ::=
    <timezone spec>

<dest timezone spec> ::=
    <timezone spec>

<timezone spec> ::=
    'AST'
  | 'ADT'
  | 'BST'
  | 'BDT'
  | 'CST'
  | 'CDT'
  | 'EST'
  | 'EDT'
  | 'GMT'
  | 'HST'
  | 'HDT'



  | 'MST'
  | 'MDT'
  | 'NST'
  | 'PST'
  | 'PDT'
  | 'YST'
  | 'YDT'

<trunc and round format> ::=
    see <date and time function> in chapter 0
```

Syntax Rules

none

General Rules

1. The <date and time expression> must produce a date and time value as the result. This value must correspond to the default representation of the data type DATE and use correct language-specific terms.

2. ADD_MONTHS
   The <expression> in ADD_MONTHS must produce a numeric integer value.
   ADD_MONTHS(d,n) adds n months to the date d, where n may be negative.
   If d is the NULL value, then ADD_MONTHS(d,n) is the NULL value. If one of the specified conditions is not satisfied, then an error message is output.

3. MONTHS_BETWEEN
   MONTHS_BETWEEN(f,s) produces the number of months between the dates f and s. If f designates a date that precedes s, then the result is negative. The result is a floating point number with 10 significant digits. The fractional digits of this number designate the part of a month consisting of 31 days. If f or s is the NULL value, then the result of MONTHS_BETWEEN(f,s) is the NULL value.

4. LAST_DAY
   LAST_DAY(d) produces the date of the last day of the month specified in d. If d is the NULL value, then the result of LAST_DAY(d) is the NULL value.

5. NEXT_DAY
   NEXT_DAY(d,c) produces the date of the next day of week with the name c which follows the date d. If the date d designates the day of week with the specified name, then the result of NEXT_DAY(d,c) is the date one week after the specified date d. If d is the NULL value, then NEXT_DAY(d,c) is the NULL value. If c does not designate a day of week known in the specified language, then an error message is output.

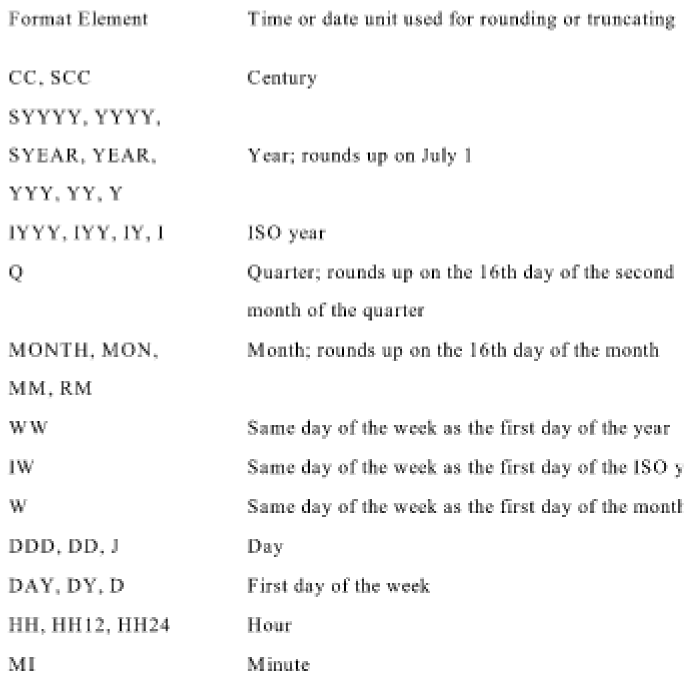
6. NEW_TIME
   NEW_TIME(d,stz,dtz) produces the date and time specification for the time zone dtz if the specification in d is valid for the time zone stz. The date specification may be changed for the result value. Valid time zones are:

'AST','ADT'	Atlantic Standard or Daylight Time
'BST','BDT'	Bering Standard or Daylight Time
'CST','CDT'	Central Standard or Daylight Time
'EST','EDT'	Eastern Standard or Daylight Time
'GMT'	Greenwich Mean Time
'HST','HDT'	Alaska-Hawaii Standard or Daylight Time
'MST','MDT'	Mountain Standard or Daylight Time
'NST'	Newfoundland Standard Time
'PST','PDT'	Pacific Standard or Daylight Time
'YST','YDT'	Yukon Standard or Daylight Time

7. ROUND
   ROUND(d,trf) produces the indicated date and time specification d as the result, rounded or truncated to the time or date unit, e.g., hours, that is specified in the <trunc and round format>. If no second parameter is specified, then 'DD' is used as the default format. ROUND applied to the NULL value produces the NULL value as the

result.

8. TRUNC
TRUNC(d,trf) produces the indicated date and time specification d as the result, rounded or truncated to the time or date unit, e.g., hours, that is specified in the <trunc and round format>. If no second parameter is specified, then 'DD' is used as the default format. TRUNC applied to the NULL value produces the NULL value as the result.



Format Element	Time or date unit used for rounding or truncating
CC, SCC	Century
SYYYY, YYYY, SYEAR, YEAR, YYY, YY, Y	Year; rounds up on July 1
IYYY, IYY, IY, I	ISO year
Q	Quarter; rounds up on the 16th day of the second month of the quarter
MONTH, MON, MM, RM	Month; rounds up on the 16th day of the month
WW	Same day of the week as the first day of the year
IW	Same day of the week as the first day of the ISO y
W	Same day of the week as the first day of the mont
DDD, DD, J	Day
DAY, DY, D	First day of the week
HH, HH12, HH24	Hour
MI	Minute

...Table 1

<special function>

Function

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

Format

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

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

<search expression> ::=
    <expression>

<result expression> ::=
    <expression>

<check expression> ::=
    <expression>

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

Syntax Rules

none

General Rules

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

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

NULL value.

3. DECODE
The data types of the <check expression> and of the <search expression>s must be comparable. The data types of the <result expression>s and the optional <default expression> must be comparable. The data types of the <search expression>s and of the <result expression>s need not be comparable.
DECODE compares the result of the <check expression> with one <search expression> result after the other. If conformity is established, the result of DECODE is the result of the <result expression> which is included in the <search and result spec> containing the matching <search expression>. If the result of the <check expression> and the result of a <search expression> is the NULL value, then conformity is established.
If no conformity can be established, DECODE produces the result of the <default expression>. If no <default expression> is specified, then the result of DECODE is the NULL value.

<conversion function>

Function

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

Format

```
<conversion function> ::=
    TO_NUMBER ( <string spec>[, <number format> ] )
  | CHR       ( <expression> )
  | RAWTOHEX  ( <expression> )
  | HEXTORAW  ( <expression> )
  | TO_CHAR   ( <expression>[, <date or number format> ] )
  | TO_DATE   ( <expression>[, <date format> ] )

<date or number format> ::=
    <number format>
  | <date format>

<number format> ::=
    see <conversion function> in chapter 0

<date format> ::=
    see <conversion function> in chapter 0
```

Syntax Rules

none

General Rules

1. TO_NUMBER
   TO_NUMBER can be applied to character strings with the code attribute ASCII or EBCDIC. TO_NUMBER transforms the character string specified in the first parameter into the corresponding numeric format. The specified character string to be transformed must have the format specified in the second parameter. TO_NUMBER applied to the NULL value produces the NULL value.

2. CHR
   Let a be a number. CHR(a) produces a character as the result that corresponds to MOD(TRUNC(a,0), 256) in the current code type.
   If a is the NULL value, then CHR(a) is the NULL value.

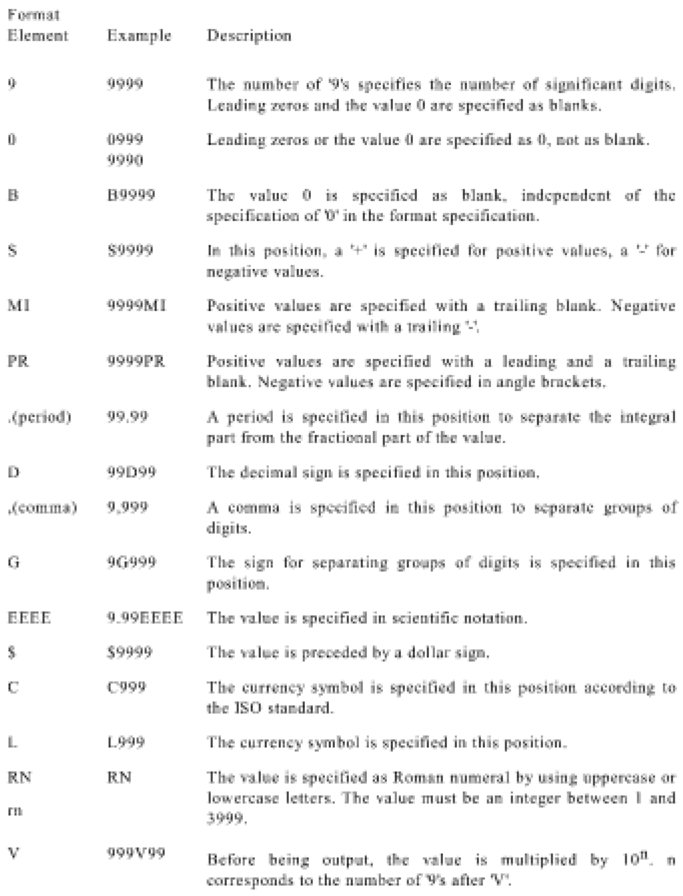
3. RAWTOHEX
   RAWTOHEX produces the hexadecimal representation of the argument. RAWTOHEX can be applied to alphanumeric and numeric values with the restriction that these character strings can only contain up to 127 characters. RAWTOHEX applied to the NULL value produces the NULL value as the result.

4. HEXTORAW

HEXTORAW transforms a character string with the code attribute ASCII or EBCDIC which contains a hex representation into a character string with the code attribute BYTE. HEXTORAW applied to the NULL value produces the NULL value as the result.

5. TO_CHAR
TO_CHAR transforms the specified first parameter into a character string in the format specified in the optional second parameter. TO_CHAR can only be applied to numeric values or date values. The formats represented in <conversion function> or <conversion function> are valid, depending on the data type of the first parameter. The names of the days and weeks as well as numbers written out as words are indicated in the currently defined language. If only one parameter is specified, then a format is used for numeric values which is suitable for outputting the number of significant digits and the number of digits after the decimal point. For date values, this is the default format 'DD-MM-YY'. If the value of the first numeric parameter is too large or, in the case of negative values, too small to be transformed into the specified format, then the result of TO_CHAR is a character string filled with '#'.
In addition to the format elements listed in the tables 2 and 3, the format element 'FM' can be included repeatedly in the format specification if this should be necessary. If no 'FM' is specified, then trailing blanks and leading zeros or blanks are inserted into the result of TO_CHAR to give it a fixed length. This padding can be suppressed by specifying the format element 'FM'. Another 'FM' activates the padding, a third 'FM' disables it again, etc.
TO_CHAR applied to the NULL value produces the NULL value as the result.

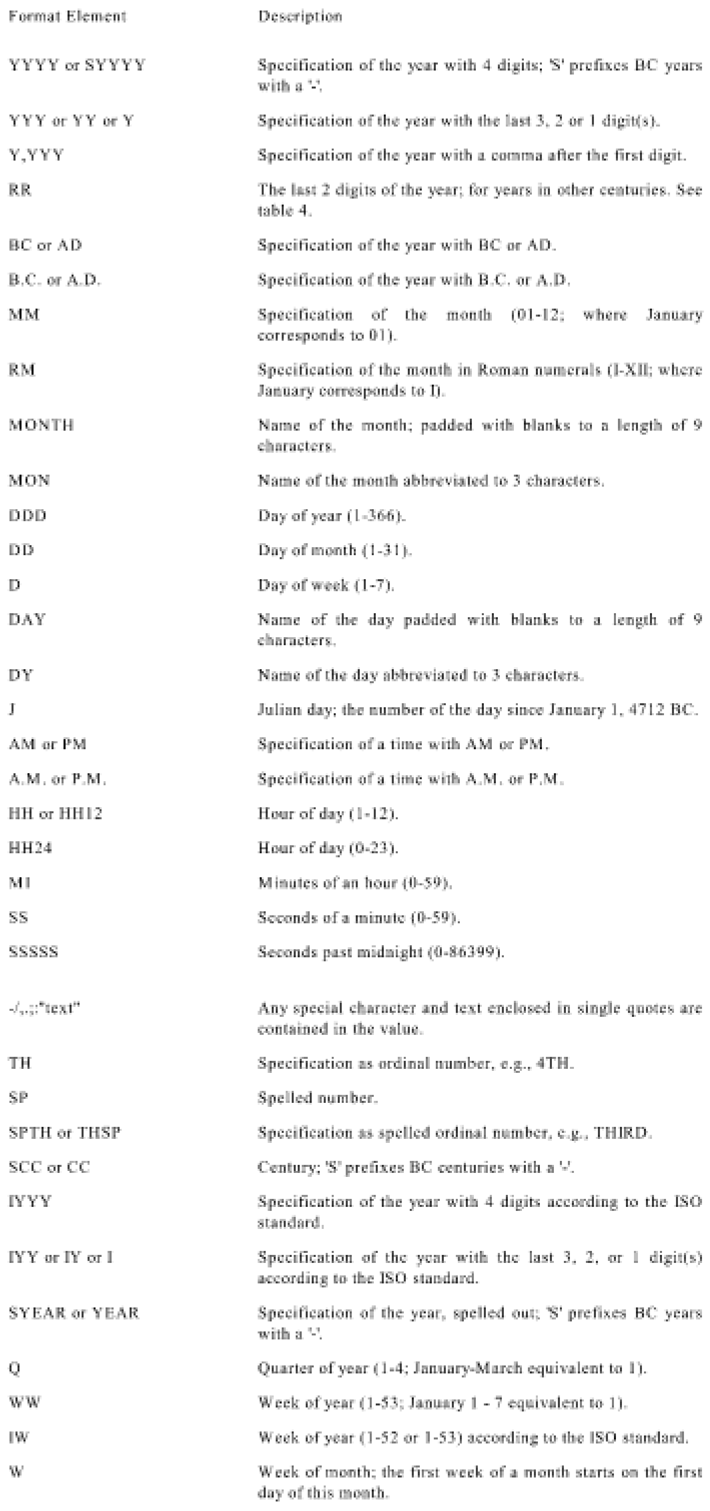
6. TO_DATE
TO_DATE can be applied to character strings with the code attribute ASCII or EBCDIC or to numbers which are implicitly converted into character strings. TO_DATE transforms the character string specified in the first parameter into a date specification. The character string to be transformed must have the format specified in the optional second parameter. Only the format elements specified above the blank line in <conversion function> are valid as <date format>s. The names of days or months must be specified in the current language. If only one parameter is specified, then the default format 'DD-MM-YY' is used. If J for the Julian day is specified as <date format>, then the first parameter must designate a number.
Each format element may be included only once in the format specification. Some format elements exclude each other, as there are: 'DDD' and 'J'; 'YYYY', 'YYY', 'YY', 'Y', 'IYYY', 'IYY', 'IY' and 'I'; 'HH', 'HH12' and 'HH24'; 'A.M.', 'P.M.', 'AM' and 'PM'; 'A.'D.', 'B.C.', 'AD' and 'BC'; 'MM', 'MON' and 'MONTH'; 'D', 'DY' and 'DAY'; 'HH24' along with 'A.M.', 'P.M.', 'AM' and 'PM'. It must be ensured that the data of two format elements do not contradict each other. For example, if the year and the Julian day were specified, the day must lie within the corresponding year; if the seconds of the day and the hours of the day were specified, the specified second must belong to the corresponding hour.
In addition to the format elements listed in <conversion function>, the format element 'FX' can be included in the format specification. Without an 'FX' specification, the specified value must not be exactly like the format specification. This means that the specified text does not have to be identical with the text contained in the format specification, but only have its length. Numbers need not have the specified number of digits. If the format element 'FX' is used, the format specification must be observed exactly. To avoid having to specify numbers with leading zeros, the format element 'FX' can be used along with the format element 'FM' (see function TO_CHAR) in the form 'FXFM'.
TO_DATE applied to the NULL value produces the NULL value as the result.



Format Element	Example	Description
9	9999	The number of '9's specifies the number of significant digits. Leading zeros and the value 0 are specified as blanks.
0	0999 9990	Leading zeros or the value 0 are specified as 0, not as blank.
B	B9999	The value 0 is specified as blank, independent of the specification of '0' in the format specification.
S	S9999	In this position, a '+' is specified for positive values, a '-' for negative values.
MI	9999MI	Positive values are specified with a trailing blank. Negative values are specified with a trailing '-'.
PR	9999PR	Positive values are specified with a leading and a trailing blank. Negative values are specified in angle brackets.
.(period)	99.99	A period is specified in this position to separate the integral part from the fractional part of the value.
D	99D99	The decimal sign is specified in this position.
,(comma)	9,999	A comma is specified in this position to separate groups of digits.
G	9G999	The sign for separating groups of digits is specified in this position.
EEEE	9.99EEEE	The value is specified in scientific notation.
$	$9999	The value is preceded by a dollar sign.
C	C999	The currency symbol is specified in this position according to the ISO standard.
L	L999	The currency symbol is specified in this position.
RN rn	RN	The value is specified as Roman numeral by using uppercase or lowercase letters. The value must be an integer between 1 and 3999.
V	999V99	Before being output, the value is multiplied by 10^n. n corresponds to the number of '9's after 'V'.

Table 2

In a <number format>, the format elements MI and PR can only be specified as the last format element, and the format element S only as the first or last format element. If no MI, PR or S format element is specified in a <number format>, a positive number is preceded by a blank, and a negative number by a minus sign.



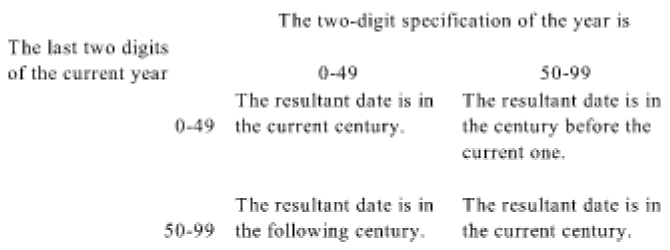
Format Element	Description
YYYY or SYYYY	Specification of the year with 4 digits; 'S' prefixes BC years with a '-'.
YYY or YY or Y	Specification of the year with the last 3, 2 or 1 digit(s).
Y,YYY	Specification of the year with a comma after the first digit.
RR	The last 2 digits of the year; for years in other centuries. See table 4.
BC or AD	Specification of the year with BC or AD.
B.C. or A.D.	Specification of the year with B.C. or A.D.
MM	Specification of the month (01-12; where January corresponds to 01).
RM	Specification of the month in Roman numerals (I-XII; where January corresponds to I).
MONTH	Name of the month; padded with blanks to a length of 9 characters.
MON	Name of the month abbreviated to 3 characters.
DDD	Day of year (1-366).
DD	Day of month (1-31).
D	Day of week (1-7).
DAY	Name of the day padded with blanks to a length of 9 characters.
DY	Name of the day abbreviated to 3 characters.
J	Julian day; the number of the day since January 1, 4712 BC.
AM or PM	Specification of a time with AM or PM.
A.M. or P.M.	Specification of a time with A.M. or P.M.
HH or HH12	Hour of day (1-12).
HH24	Hour of day (0-23).
MI	Minutes of an hour (0-59).
SS	Seconds of a minute (0-59).
SSSSS	Seconds past midnight (0-86399).
-/,.;:"text"	Any special character and text enclosed in single quotes are contained in the value.
TH	Specification as ordinal number, e.g., 4TH.
SP	Spelled number.
SPTH or THSP	Specification as spelled ordinal number, e.g., THIRD.
SCC or CC	Century; 'S' prefixes BC centuries with a '-'.
IYYY	Specification of the year with 4 digits according to the ISO standard.
IYY or IY or I	Specification of the year with the last 3, 2, or 1 digit(s) according to the ISO standard.
SYEAR or YEAR	Specification of the year, spelled out; 'S' prefixes BC years with a '-'.
Q	Quarter of year (1-4; January-March equivalent to 1).
WW	Week of year (1-53; January 1 - 7 equivalent to 1).
IW	Week of year (1-52 or 1-53) according to the ISO standard.
W	Week of month; the first week of a month starts on the first day of this month.

Table 3

The names of the months or days, of Roman numerals, or of numbers written out as words are written in uppercase or lowercase characters as specified in the format elements. For example, 'DAY' produces 'MONDAY' as the result, whereas 'Day' yields 'Monday' and 'day' yields 'monday' as the result.

The format elements TH, SP, SPTH and THSP are only valid after format elements for the specification of numbers.

Table 4 shows the way in which the format element 'RR' takes effect. This format element can be used to store data from the previous or following century in the database, although only the last two digits of the century have been specified.



	The two-digit specification of the year is	
The last two digits of the current year	0-49	50-99
0-49	The resultant date is in the current century.	The resultant date is in the century before the current one.
50-99	The resultant date is in the following century.	The resultant date is in the current century.

Table 4

<set function spec>

Function

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

Format

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

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

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

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

Syntax Rules

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

General Rules

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

2. The argument of a <distinct function> is a set of values. This set is generated by applying the <expression> to each row of a temporary result table or of a group and by eliminating all NULL values and duplicate values.
   If the set is empty and the <distinct function> is applied to the whole temporary result table, the result of AVG, MAX, MIN, STDDEV, SUM, and VARIANCE is the NULL value, and the result of COUNT is 0.
   If there is no group to which the <distinct function> could be applied, the result table is empty.

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, STDDEV, SUM, and VARIANCE is the NULL value, and the result of COUNT is 0.
If there is no group to which the <all function> could be applied, the result table is empty.
The result of an <all function> is independent of whether the key word ALL is specified or not.

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

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

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

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 NUMBER(*).

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

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

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

<expression>

Function

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

Format

```
<expression> ::=
    <arithmetic expression>
  | <datetime expression>

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

Syntax Rules

none

General Rules

none

See also

<arithmetic expression>
<datetime expression>

<arithmetic expression>

Function

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

Format

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

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

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

<sign> ::=
    +

  | -

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

Syntax Rules

none

General Rules

1. The arithmetical operators * and /, as well as + and - can only be applied to numeric data types. In addition, the operators + and - can also be applied to <datetime expression>s (see chapter 0).

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

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

4. If both operators are <datetime expression>s, then then result is a floating point number indicating the number of days between the two <datetime expression>s.

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

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

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

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

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

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

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

7. 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.

<datetime expression>

Function

specifies a value which is formed by applying arithmetical operators to values of the data type DATE.

Format

```
<datetime expression> ::=
    <datetime primary>
  | <datetime expression> + <factor>
  | <datetime expression> - <factor>
  | <factor> + <datetime expression>

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

Syntax Rules

none

General Rules

1. Only the functions which produce a result of the data type DATE are valid within a <datetime primary>.
2. The result of <factor> must be a numeric value.
3. The result of <factor> indicates the number of days which are to be added to or subtracted from the date specified by the <datetime expression>.
   During the calculation, overflows (e.g., for 'DEC-31-95' + 5.6) and underflows are carried over for each part.
   The result must lie between 01-01-0001 and 12-31-9999.
4. The result of a <datetime expression> is the NULL value if any <datetime primary> or a <factor> has the NULL value.

<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>
  | <rownum predicate>
```

Syntax Rules

none

General Rules

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

2. Columns with the same code attribute can be compared to each other. Columns with the different code attributes ASCII and EBCDIC can be compared to each other. Columns of the code attributes ASCII and EBCDIC can be compared to date values. In the <between predicate>, <comparison predicate>, <in predicate>, <join predicate>, and <like predicate>, character strings are converted into numbers and numbers into character strings, if required.

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

See also

<between predicate>

<comparison predicate>

<exists predicate>

<in predicate>

<join predicate>

<like predicate>
<null predicate>
<quantified predicate>
<rownum 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 or between lists of values.

Format

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

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

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

Syntax Rules

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

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

General Rules

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

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

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

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

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

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

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

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

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

<exists predicate>

Function

checks whether a result table contains at least one row.

Format

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

Syntax Rules

none

General Rules

1. The truth value of an <exists predicate> is either true or false.

2. Let T be the result table produced by <subquery>. (EXISTS T) is true if and only if T contains at least one row.

<in predicate>

Function

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

Format

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

Syntax Rule

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

General Rules

1. Let x be the result of the <expression> and S be either the result of the <subquery> or the values of the sequence of <expression>s. S is a set of values. The value x and the values in S must be comparable with each other.
2. If an <expression list> is specified to the left of the operator IN, then let x be the value list consisting of the result of the <expression>s x1, x2, ..., xn of this value list. Let S be either the result of the <subquery> that consists of a set of value lists s or a sequence of value lists s. A value list s consists of the results of the <expression>s s1, s2, ..., sn. A value xm must be comparable with all values sm.
3. x=s is true if xm=sm is valid for all m=1, ..., n. x=s is false if there is at least one m for which xm=sm is false. x=s is unknown if there is no m for which xm=sm is false and if there is at least one m for which xm=sm is unknown.
4. If x=s is true for at least one value or value list s of S, then (x IN S) is true.
5. If x=s is not true for any value or any value list s of S and x=s is unknown for at least one value or value list s of S, then (x IN S) is unknown.
6. If S is empty or if x=s is false for every value or value list s of S, then (x IN S) is false.
7. (x NOT IN S) has the same result as NOT(x IN S).

<join predicate>

Function

specifies a join.

Format

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

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

Syntax Rules

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

General Rules

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

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

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

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

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

   The term of underlying base tables is explained in detail in the chapter '<from clause>'.

5. Usually, rows are only transferred to the result table if they have a counterpart corresponding to the <comp op> in the other table specified in the <join predicate>.
   If it must be ensured that every row of a table is contained in the result table at least once, the <outer join indicator> must be specified on the side of <comp op> where the other table is specified.
   If it is not possible to find at least one counterpart for a table row in the other table, this row is used to build a row for the result table. The NULL value is then used for the output columns which are usually formed from the other table's columns.

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

<like predicate>

Function

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

Format

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

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

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

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

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

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

Syntax Rules

none

General Rules

1. The <expression> of the <like expression> must produce an alphanumeric value, or a date 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 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> (<expression>,...)
  | <expression> <comp op> <quantifier> <subquery>
  | <expression list> <equal or not>
    <quantifier> (<expression list>,...)
  | <expression list> <equal or not> <quantifier> <subquery>

<quantifier> ::=
    ALL
  | <some>

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

Syntax Rules

1. The <subquery> must produce a result table which contains as many columns as <expression>s are specified to the left of the operator.
2. Each <expression list> specified to the right of <equal or not> must contain as many <expression>s as are specified in the <expression list> to the left of <equal or not>.

General Rules

1. Let x be the result of the <expression> and S the result of the <subquery> or sequence of <expression>s. S is a set of values. The value x and the values in S must be comparable with each other.
2. If S is empty or (x <comp op> s) is true for every value s of S, then (x <comp op> ALL S) is true.
3. If (x <comp op> s) is not false for any value s of S and (x <comp op> s) is unknown for at least one value s of S, then (x <comp op> ALL S) is unknown.
4. If (x <comp op> s) is false for at least one value s of S, then (x <comp op> ALL S) is false.
5. If (x <comp op> s) is true for at least one value s of S, then (x <comp op> <some> S) is true.
6. If (x <comp op> s) is not true for any value s of S and (x <comp op> s) is unknown for at least one value s of S, then (x <comp op> <some> S) is unknown.

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

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

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

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

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

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

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

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

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

<rownum predicate>

Function

limits the number of rows of a result table.

Format

```
<rownum predicate> ::=
    ROWNUM <  <rownum spec>
  | ROWNUM <= <rownum spec>

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

Syntax Rules

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

General Rules

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

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

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

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

<search condition>

Function

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

Format

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

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

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

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

Syntax Rules

none

General Rules

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

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

3. The following rules apply to NOT:

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

4. The following rules apply to AND:

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

5. The following rules apply to OR:

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

SQL Statement

Function

specifies any SQL statement.

Format

```
<sql statement> ::=
    <create table statement>
  | <drop table statement>
  | <alter table statement>
  | <create synonym statement>
  | <drop synonym statement>
  | <create snapshot statement>
  | <drop snapshot statement>
  | <create snapshot log statement>
  | <drop snapshot log statement>
  | <create view statement>
  | <drop view statement>
  | <create index statement>
  | <drop index statement>
  | <create sequence statement>
  | <drop sequence statement>
  | <oracle ddl statement>
  | <comment statement>

  | <create user statement>
  | <drop user statement>
  | <grant statement>
  | <revoke statement>

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

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



  | <connect statement>
  | <commit statement>
  | <rollback statement>
  | <rollback to statement>
  | <savepoint statement>
  | <lock statement>
  | <release statement>
```

Syntax Rules

none

General Rules

1. The SQL statements of the 1st block are described in the chapter 'Data Definition'.

2. The SQL statements of the 2nd block are described in the chapter 'Authorization'.

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.

Data Definition

Every data definition statement is preceded and concluded by an implicit <commit statement>.

<create table statement>

Function

creates a base table.

Format

```
<create table statement> ::=
    CREATE TABLE <table name> [(<table description element>,...)]
          [<oracle option>...] [AS <query expression>]

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

<oracle option> ::=
    PCTFREE <unsigned integer>
  | PCTUSED <unsigned integer>
  | INITTRANS <unsigned integer>
  | MAXTRANS <unsigned integer>
  | TABLESPACE <identifier>
  | STORAGE <storage clause>



<storage clause> ::=
    ([INITIAL <unsigned integer>] [NEXT <unsigned integer>]
    [MINEXTENTS <unsigned integer>]
    [MAXEXTENTS <unsigned integer>]
    [PCTINCREASE <unsigned integer>])
```

Syntax Rules

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

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

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

General Rules

1. Omitting the <owner> in the <table name> has the same effect as specifying the current user as <owner>. 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. If a <query expression> is specified, a base table is created with the same structure as the result table defined by the <query expression>. If <column definition>s are specified, then each <column definition> may only consist of a <column name>, and the number of <column definition>s must equal the number of columns in the result table generated by the <query expression>. The <data type> of the ith column of the generated base table corresponds to that of the ith column in the result table generated by the <query expression>. The result table must not contain LONG columns. If the <create table statement> contains no <column definition>s, the column names are taken from the result table as well.
   The rows of the result table are implicitly inserted into the generated base table.
   The same restrictions apply for the <query expression> here as for the <query expression> of an <insert statement>.

6. The current user becomes the owner of the created table. The user obtains every privilege (INSERT, UPDATE, DELETE, SELECT, ALTER, INDEX, and REFERENCES privilege) for this table.

7. The <oracle options> and the <storage clause> contained therein are meaningless for ADABAS.

See also

<column definition>

<constraint definition>

<referential constraint definition>

<key definition>

<unique definition>

<column definition>

Function

defines a table column.

Format

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

<data type> ::=
    CHAR    [(<unsigned integer>)]
  | VARCHAR [(<unsigned integer>)]
  | LONG [RAW]
  | NUMBER  [(<unsigned integer> [,<unsigned integer>])]
  | NUMBER  [(* [,<unsigned integer>])]
  | DATE
  | RAW     [(<unsigned integer>)]

<column attributes> ::=
    [<default spec>]
    [NULL        [CONSTRAINT <constraint name>] ]
    [NOT NULL    [CONSTRAINT <constraint name>] ]
    [UNIQUE      [CONSTRAINT <constraint name>] ]
    [PRIMARY KEY [CONSTRAINT <constraint name>] ]
    [REFERENCES <referenced table> [(<referenced column>)] ]
    [<constraint definition>]

<default spec> ::=
    DEFAULT <expression>

<referenced table> ::=
    <table name>

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

Syntax Rules

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

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

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

General Rules

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

2. CHAR (n) and VARCHAR (n) define an alphanumeric column with the length attribute n. The length attribute must be greater than 0 and less than or equal to 2000. 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 (n) is specified, the value n determines whether ADABAS stores the values of this column in fixed length or in variable length. If the values are to be stored in variable length regardless of n, VARCHAR must be specified. Otherwise, specifying VARCHAR has the same effect as CHAR.

4. LONG defines an alphanumeric column of any length which can be used in the <insert statement>, in the <update columns and values> of the <update statement>, as <select column>, and in the <null predicate>. The specification of RAW has the effect that the characters in the column are not interpreted as ASCII or EBCDIC characters but as characters with the code attribute BYTE.

5. NUMBER defines a column where numeric values (fixed point values, floating point values) are stored. NUMBER (p,s) defines a numeric column with the precision p and the scale s. NUMBER without a specification of p and NUMBER(*) both define a column where floating point values are stored.

6. DATE defines an alphanumeric column where date and time values are stored.
   A value of the data type DATE is output in the format 'DD-MON-YY'. Thereby,
   'DD'   stands for a two-digit identifier of a day (01-31),
   'MON' stands for a three-letter identifier of a month,
   'YY'   stands for a two-digit identifier of a year.
   The three-letter identifier of the month is language-specific and can assume the value JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV or DEC.
   On input, the date specification can comprise up to 26 characters, where the identifiers of the day and year can have one or two digits each. Any number of characters which are neither letters nor digits can be specified instead of the '-'. The identifier of the month can be specified in uppercase or lowercase characters or in any combination of them. The time value contained in the data type DATE is set to the midnight value.
   The function SYSDATE can be used to retrieve the current date. The time value contained in the data type is set to the current time. The <conversion function> TO_CHAR must be used to output date and time values stored in a column of the data type DATE.

7. RAW (n) defines a character string with the length n, whose characters are not interpreted as ASCII or EBCDIC characters but as characters with the code attribute BYTE. If the length attribute is omitted, n=1 is assumed.

8. The NULL value can be inserted into columns which are not part of the key and for which neither a <default spec> nor NOT NULL was defined. The NULL specification has therefore no significance.

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

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

11. 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 value in the <default spec> is stored in the column. If there is no <default spec>, the NULL value is stored in the column.

12. 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.

13. 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.

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

15. If a table is defined without a key column, ADABAS implicitly generates the key column SYSKEY RAW(8). 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.

16. In terms of the <default spec> syntax, ADABAS already allows for an <expression>. Currently, however, ADABAS can only support <expression>s that consist of one <value spec>. Otherwise, the <default spec> is ignored.

17. 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>)].

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

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

```
CHARACTER(n)          is mapped to  CHAR(n)
INT[EGER]             is mapped to  NUMBER(18)
SMALLINT              is mapped to  NUMBER(18)
DEC[IMAL] [(*)]       is mapped to  NUMBER(18)
DEC[IMAL](*,s)        is mapped to  NUMBER(18,s)
DEC[IMAL](p)          is mapped to  NUMBER(p)
DEC[IMAL](p,s)        is mapped to  NUMBER(p,s)
NUMERIC(p,s)          is mapped to  NUMBER(p,s)
DOUBLE PRECISION      is mapped to  NUMBER(*)
REAL                  is mapped to  NUMBER(*)
FLOAT [(*)]           is mapped to  NUMBER(*)
FLOAT(p)              is mapped to  NUMBER(*)
LONG VARCHAR          is mapped to  LONG
```

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

```
CHAR(n), RAW (n)
          n <=  30    : n + 1
     21.   < n <= 254    : n + 1  for key columns,
                        n + 2  otherwise
    22.    < n           : n + 3
VARCHAR(n)
     23.   < n <= 254    : n + 1  for key columns,
                        n + 2  otherwise
    24.    < n           : n + 3
LONG                   :  9
NUMBER (p,s)           : (p+1) DIV 2 + 2
NUMBER [(* [,s])]      : 11
```

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

<constraint definition>

Function

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

Format

```
 <constraint definition> ::=
     [CONSTRAINT <constraint name>] CHECK (<search condition>)
```

Syntax Rules

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

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

General Rules

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

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

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

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

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

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

<referential constraint definition>

Function

defines existence conditions between the rows of two tables.

Format

```
<referential constraint definition> ::=
    [CONSTRAINT <constraint name>]
    FOREIGN KEY (<referencing column>,...)
    REFERENCES <referenced table> [(<referenced column>,...)]
    [ON DELETE CASCADE]

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

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

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

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

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

8. If the <referenced column>s do not identify the key of the <referenced table>, then the <referenced table> must have a <unique definition> whose <column name>s match the <referenced column>s.

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

must match the data type and length of the corresponding <referenced column>.

10. 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.

11. 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>.

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

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

14. 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> ::=
     [CONSTRAINT <constraint name>] PRIMARY KEY (<column name>,...)
     [USING INDEX <oracle option>
```

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

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

5. Key columns are NOT NULL columns.

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

<unique definition>

Function

defines the uniqueness of column value combinations.

Format

```
 <unique definition> ::=
     [CONSTRAINT <constraint name>] UNIQUE (<column name>,...)
     [USING INDEX <oracle option>]
```

Syntax Rules

none

General Rules

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

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

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

<drop table statement>

Function

drops a base table.

Format

```
 <drop table statement> ::=
     DROP TABLE <table name> [CASCADE CONSTRAINTS]
```

Syntax Rules

none

General Rules

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

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

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

4. If the table to be dropped is <referenced table> of a <referential constraint definition> and CASCADE CONSTRAINTS is not specified, then the <drop table statement> fails.

<alter table statement>

Function

alters properties of a table.

Format

```
 <alter table statement> ::=
     ALTER TABLE <table name> <add definition>

   | ALTER TABLE <table name> <modify definition>
```

Syntax Rules

none

General Rules

1. The <table name> must be the name of an existing base table.
2. The current user must have the ALTER privilege for the table identified by <table name>.

See also

<add definition>

<modify definition>

<add definition>

Function

defines additional properties for a table.

Format

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

Syntax Rules

none

General Rules

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

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

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

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

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

<modify definition>

Function

alters the properties of a column.

Format

```
 <modify definition> ::=
     MODIFY (<column name> <data type>)
```

Syntax Rules

none

General Rules

1. The data type of a key column or foreign key column cannot be altered.
2. A specified <data type> replaces the existing <data type>. The new data type must be compatible with the former data type, or, more precisely:
   a) [VAR]CHAR(n) can be changed to [VAR]CHAR(m) with m>= n.

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

   c) RAW(n) can be changed to RAW(m) with m>=n.
3. In some cases, the <modify definition> has the effect that a new table column is defined implicitly. This column is not visible to the user. If the addition of the new column could have the effect that the maximum number of columns would be exceeded, the <alter table statement> fails.
4. The expansion of a column of the base table can have the effect that the maximum length of a row is exceeded. In this case, the <alter table statement> fails.
5. The expansion of a column of the base table can have the effect that the column of a view table defined on this base table becomes too long. In this case, the <alter table statement> fails.
6. Changing the data type of a column can have the effect that indexes defined across the column are implicity recreated. Expanding a column can have the effect that an index consisting of several columns becomes too wide. In this case, the <alter table statement> fails.

<create synonym statement>

Function

defines a synonym for a table name.

Format

```
 <create synonym statement> ::=
     CREATE SYNONYM [<owner>.]<synonym name> FOR <table name>
```

Syntax Rules

none

General Rules

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

<drop synonym statement>

Function

drops a synonym for a table name.

Format

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

Syntax Rules

none

General Rules

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

<create snapshot statement>

Function

creates a snapshot table.

Format

```
<create snapshot statement> ::=
    CREATE SNAPSHOT <table name>
    [<oracle snapshot options>] [<refresh spec>]
    AS <query expression>

<oracle snapshot options> ::=
      <oracle option>
    | CLUSTER <identifier> (<column name>,...)

<refresh spec> ::=
    REFRESH [<refresh kind>]
    [START WITH <date and time expression>]
    [NEXT <date and time expression>]

<refresh kind> ::=
    FAST
  | COMPLETE
  | FORCE
```

Syntax Rules

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

General Rules

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

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

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

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

5. The <query expression> must not make reference to a snapshot table.

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

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

8. ADABAS distinguishes between simple and complex snapshot tables. Simple snapshot tables have the following properties:

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

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

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

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

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

9. To tally the contents of the snapshot table with the contents of the result table defined by the <query expression>, the <refresh statement> can be used in SQLMODE ADABAS. ADABAS distinguishes between two methods of executing the <refresh statement>:

   a) If the snapshot table is a simple snapshot table and the base table on which the snapshot table is based has a snapshot log, then this snapshot log can be used to determine the differences between the contents of the snapshot table and the result table of the <query expression>. Only these differences are transferred to update the snapshot table. In many cases, this is more convenient than to transfer the complete result table into the snapshot table.

   b) All rows of the snapshot table are deleted. Then all rows of the result table defined by the <query expression> are inserted.

10. The <oracle snapshot options> are meaningless for ADABAS.

<drop snapshot statement>

Function

drops a snapshot table.

Format

```
 <drop snapshot statement> ::=
     DROP SNAPSHOT <table name>
```

Syntax Rules

none

General Rules

1. <table name> must identify a snapshot table.
2. The current user must be the owner of the snapshot table.
3. The metadata and all rows of the snapshot table are dropped.
4. All indexes, synonyms and view tables defined on the snapshot table are dropped.
5. If <table name> identifies a simple snapshot table and the underlying base table has a snapshot log, then any information of the snapshot log is dropped that is only relevant for refresh operations on the snapshot table to be dropped. If the snapshot table to be dropped is the only simple snapshot table based on the base table, then the corresponding snapshot log is not written until the next simple snapshot table is created on this base table.

<create snapshot log statement>

Function

creates a snapshot log.

Format

```
 <create snapshot log statement> ::=
     CREATE SNAPSHOT LOG ON <table name> [<oracle option>]
```

Syntax Rules

none

General Rules

1. <table name> must identify a non-temporary base table.

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

3. The <create snapshot log statement> creates a snapshot log for the base table identified by <table name>. In a snapshot log, ADABAS stores information about the modified rows of the table. This information can be used later with a <refresh statement> to update a snapshot table without having to execute the complete <query expression>, because only the modifications made since the last execution of the <refresh statement> are performed. In many cases, this is convenient because the data transfer between the SERVERDBs is reduced considerably.

4. ADABAS only writes the snapshot log if there is at least one simple snapshot table based on the table <table name>. Otherwise, the snapshot log is created but not filled when rows of the table are modified.

5. The <oracle option>s are meaningless in ADABAS.

<drop snapshot log statement>

Function

drops a snapshot log.

Format

```
 <drop snapshot log statement> ::=
     DROP SNAPSHOT LOG ON <table name>
```

Syntax Rules

none

General Rules

1. The base table identified by <table name> must have a snapshot log.
2. The current user must be the owner of the base table.
3. The snapshot log and the information contained in it are dropped. If rows of the base table are modified, these modifications are no longer recorded in the snapshot log.
4. After dropping the snapshot log, the <query expression> must be executed completely to update snapshot tables that are based on the base table <table name>.

<create view statement>

Function

creates a view table.

Format

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

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. If the specification of REPLACE is omitted, the <table name> must not be identical to the name of an existing table.

3. If REPLACE is specified, then <table name> may be identical to the name of an existing view table. In this case, the definition of the existing view table is replaced by the new definition. ADABAS then attempts to adapt privileges granted for the existing view table to the new view definition; usually, the privileges for the view table are kept in this way. Privileges are only removed implicitly if conflicts occur that cannot be resolved by ADABAS. Should there be large differences between the two view definitions, then the <create view statement> can fail in the following case:

   a) The <create view statement> of a view table based on the existing view table cannot be executed free of errors on the new view definition.

4. 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.

5. The <alias name>s define the column names of the view table. If no <alias name>s are specified, then the column names of the result table generated by the <query expression> are applied to the view table. The column names of the view table must be unique. Otherwise, <alias name>s must be specified for the result table generated by the <query expression>. The column descriptions for the view table are taken from the corresponding columns in the <query expression>. The <from clause> of the <query expression> may contain one or more tables.

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

7. A view table is a complex view table if one of the following conditions is satisfied:

   a) The definition of the view table contains DISTINCT or GROUP BY.

   b) The <create view statement> contains MINUS, INTERSECT, or UNION.

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

   d) The <create view statement> contains an outer join, that is, an <outer join indicator> in a <join predicate> of the <search condition>.

8. A view table is called updatable if it is 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.

9. The owner of the view table has the INSERT privilege; i.e., the user may specify a view table in the <insert statement> as the table into which insertion is to be made if the following conditions are satisfied:

   a) The view table is updatable.

   b) The owner of the view table has the INSERT privilege for 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>.

10. The owner of the view table has the UPDATE privilege for a column of the view table; i.e., the user may specify a column in the <update statement> as column to be updated if the following conditions are satisfied:

   a) The view table is updatable.

   b) The owner of the view table has the UPDATE privilege for the <table columns> or the <column name> defining the column.

   c) The column is defined by a specification of <table columns> or by a <column name>, but not by an <expression> with more than one <column name>.

11. The owner of the view table has the DELETE privilege for the view table; i.e., the user may specify a view table in the <delete statement> as the table from which a column or row is to be deleted if the following conditions are satisfied:

    a) The view table is updatable.

    b) The owner of the view table has the DELETE privilege for the table of the <from clause> of the <create view statement>.

12. If the <create view statement> contains the WITH CHECK OPTION, then the owner of the view table must have the INSERT, UPDATE, or DELETE privilege for the view table.
    The specification of WITH CHECK OPTION has the effect that the <insert statement> or <update statement> issued on the view table does not create any rows which subsequently could not be selected via the view table; i.e., the <search condition> of the view table must be true for any resulting rows.
    The CHECK OPTION is inherited; i.e., if a view table V was defined WITH CHECK OPTION and V occurs in the <from clause> of an updatable view table V1, then only those rows can be inserted or altered using V1 which can be selected using V.

<drop view statement>

Function

drops a view table.

Format

```
 <drop view statement> ::=
     DROP VIEW <table name>
```

Syntax Rules

none

General Rules

1. The table name must denote an existing view table.

2. The user must be the owner of the specified view table.

3. The metadata of the view table and all dependent synonyms, view tables and privileges are dropped. The tables on which the view table was created remain unaffected. All snapshot tables derived from the view table to be dropped remain unaffected. ADABAS marks them in such a way that the <query expression> defining the snapshot tables must be performed again when the <refresh statement> is executed the next time. This means that the <refresh statement> fails if the dropped table has not been recreated in the meantime.

<create index statement>

Function

creates an index for a base table or a snapshot table.

Format

```
<create index statement> ::=
    CREATE [UNIQUE] INDEX <index spec>
    [<oracle option>]

<index spec> ::=
    <index name> ON <table name> (<index clause>,...)

<index clause> ::=
    <column name> [<order spec>]

<order spec> ::=
    ASC
  | DESC
```

Syntax Rules

1. The <index spec> must not contain more than 16 <column name>s.

General Rules

1. The table identified by <table name> must be an existing base table or snapshot table.

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

3. Up to 256 indexes may be created per table.

4. If an index was created on exactly one column, then it is not possible to create another one-column index on this column.

5. If the <index name> is the only difference between the index defined by the <create index statement> and an existing index for the table, then the <create index statement> fails.

6. The sum of the internal lengths of the columns to be indexed must not exceed 255 characters.

7. The current user must be the owner of the table identified by <table name> or have the INDEX privilege for the table.

8. The index is created across the specified table columns. The secondary key consists of the specified columns of the table, in the specified order. The specification of ASC or DESC has the effect that the index values are stored in ascending or descending order. If the specification of ASC or DESC is omitted, ASC is implicitly assumed.

9. The index and the metadata describing the index are stored on the HOME SERVERDB of the table identified by <table name>.

10. If UNIQUE is specified, ADABAS ensures that no two rows of the specified table have the same values in the indexed columns. NULL values in one-column indexes are considered to be non-identical.

11. Indexes facilitate the access via non-key columns. But the maintenance of indexes means additional overhead in connection with <insert statement>s, <update statement>s and <delete statement>s. ASC or DESC can be specified to support the processing in a specific sort sequence that corresponds to the index definition.

12. The <oracle options> are meaningless for ADABAS.

<drop index statement>

Function

drops an index and its description.

Format

```
<drop index statement> ::=
    DROP INDEX <index name> [ON <table name>]
```

Syntax Rules

none

General Rules

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

2. The specified index must exist.

3. If the <index name> clearly denotes an index, the specification 'ON <table name>' can be omitted.

4. The current user must be the owner of the table identified by <table name> or have the INDEX privilege for the table <table name>.

5. The metadata of the specified index is deleted from the catalog. The storage space occupied by the index is released.

<create sequence statement>

Function

defines a generator for integer numbers.

Format

```
<create sequence statement> ::=
    CREATE SEQUENCE [<owner>.]<sequence name>
    [INCREMENT BY <integer>]
    [START WITH <integer>]
    [<maxvalue spec>]
    [<minvalue spec]
    [<cycle spec]
    [<cache spec>]
    [<order sequence spec>]

<maxvalue spec> ::=
      MAXVALUE <integer>
    | NOMAXVALUE

<minvalue spec> ::=
      MINVALUE <integer>
    | NOMINVALUE

<cycle spec> ::=
      CYCLE
    | NOCYCLE

<cache spec> ::=
      CACHE
    | NOCACHE

<order sequence spec> ::=
      ORDER
    | NOORDER
```

Syntax Rules

1. INCREMENT BY <integer>, START WITH <integer>, <maxvalue spec>, <minvalue spec>, <cycle spec>, <cache spec>, and <order sequence spec> can be specified in any order.

General Rules

1. The current user must have RESOURCE or DBA status.

2. <owner> must identify the current user.

3. The current user becomes the owner of the sequence.

4. The <create sequence statement> generates a database object that produces integer values. This object is generated on the HOME SERVERDB of the current user. In the following, it is called a sequence.

5. Up to eight sequences can be generated on each SERVERDB.

6. The integer numbers generated by the sequence can be used to assign key values.

7. INCREMENT BY defines the difference between the next sequence value and the last value assigned. A negative value for INCREMENT BY generates a descending sequence. If no value is specified for INCREMENT BY, then the value 1 is used.

8. START WITH defines the first sequence value. If no START WITH value is specified, then MAXVALUE is used for descending sequences and MINVALUE for ascending sequences.

9. MINVALUE is the smallest value generated by the sequence.

10. MAXVALUE is the largest value generated by the sequence.

11. If CYCLE is specified, MINVALUE is produced for ascending sequences after assigning MAXVALUE; and MAXVALUE is produced for descending sequences after assigning MINVALUE.

12. If NOCYCLE is specified, a request for a sequence value fails if the end of the sequence has been reached; i.e., if MAXVALUE has been assigned for an ascending sequence and MINVALUE has been assigned for a descending sequence.

13. The specification of CACHE or NOCACHE is meaningless for ADABAS.

<drop sequence statement>

Function

drops a generator for integer numbers.

Format

```
 <drop sequence statement> ::=
     DROP SEQUENCE [<owner>.]<sequence name>
```

Syntax Rules

none

General Rules

1. <owner> must be identical to the name of the current user.
2. <sequence name> must identify a sequence of the current user.

<oracle ddl statement>

Function

creates database objects which are meaningless for ADABAS.

Format

```
 <oracle ddl statement> ::=
     <create tablespace statement>
   | <create rollback segment statement>
   | <create public rollback segment statement>
```

Syntax Rules

none

General Rules

1. ADABAS accepts the <oracle ddl statement>s, but these have no effect. This facilitates the transfer of existing ORACLE applications to ADABAS.

<comment statement>

Function

creates a comment for a table or column.

Format

```
<comment statement> ::=
    COMMENT ON TABLE <table name> IS <string literal>
  | COMMENT ON COLUMN
          <table name>.<column name> IS <string literal>
  | COMMENT ON <table name> ( <column comment>,... )

<column comment> ::=
    <column name> IS <string literal>
```

Syntax Rules

1. The <string literal> may have up to 254 characters.

General Rules

1. The comment is stored in the catalog.

Authorization

Every authorizing statement is preceded and concluded by an implicit <commit statement>.

<create user statement>

Function

defines a user.

Format

```
 <create user statement> ::=
     CREATE USER <user name> IDENTIFIED BY <password>
     [<oracle user option> ...]

 <oracle user option> ::=
     DEFAULT TABLESPACE <identifier>
   | TEMPORARY TABLESPACE <identifier>
   | QUOTA <unsigned integer [ K | M ] ON <identifier>
   | QUOTA UNLIMITED
   | PROFILE <identifier>
```

Syntax Rules

none

General Rules

1. The <create user statement> defines a user. The existence and the properties of the user are recorded in the catalog in the form of metadata.
2. The current user must have DBA status. The user is the owner of the generated user.
3. The <user name> to be defined must not denote an existing user.
4. The <password> must be specified when an ADABAS session is opened. It ensures that only authorized users obtain access to ADABAS.
5. The user obtains the RESOURCE user status. This gives the specified user the right to define private data and to grant other users privileges for these objects.
6. The <oracle user option>s are meaningless for ADABAS.

<drop user statement>

Function

drops the definition of a user.

Format

```
 <drop user statement> ::=
     DROP USER <user name> CASCADE
```

Syntax Rules

none

General Rules

1. The current user must have owner authorization over the user to be dropped.

2. At the time when the <drop user statement> is executed, the user identified by <user name> must not be connected to any SERVERDB of the database.

3. The SERVERDB where the <drop user statement> is to be executed must belong to the majority.

4. If the user to be dropped does not belong to a usergroup and is the owner of synonyms or tables, and CASCADE is not specified, the <drop user statement> fails. If CASCADE is specified, all synonyms and tables of the user to be dropped, as well as indexes, privileges, view tables, etc. based on these objects are dropped.

5. If a user with DBA status is dropped, any users generated by him remain untouched. The SYSDBA of the HOME SERVERDB of the dropped DBA becomes the new owner of this user.

6. The metadata of the user to be dropped is dropped from the catalog.

<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
  | INDEX
  | ALTER
  | 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 and snapshot tables, it may happen that not even the owner is authorized to grant all privileges. Which privileges a user may grant for a view table or snapshot table is determined by ADABAS upon generation of the table. The result depends on the type of the table, as well as on the user's privileges for the tables selected in the view table or snapshot table. The owner of a table can retrieve the privileges he is allowed to grant by selecting the system table DOMAIN.PRIVILEGES.

2. The INSERT privilege allows the user identified by <grantee> to insert rows into the specified 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 INDEX privilege allows the user identified by <grantee> to execute the <create index statement> and the <drop index statement> for the specified table. The INDEX privilege can only be granted for base tables and snapshot tables, and the current user must have the authorization to grant the INDEX privilege.

7. The ALTER privilege allows the user identified by <grantee> to alter the table definition of the specified table. The ALTER privilege can only be granted for base tables, and the current user must have the authorization to grant the ALTER privilege.

8. 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.

9. 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.

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

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

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

13. 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.

<revoke statement>

Function

revokes privileges.

Format

```
 <revoke statement> ::=
     REVOKE <table privileges> ON <table name> FROM <grantee>,...
```

Syntax Rules

none

General Rules

1. The owner of a table can revoke the privileges granted for this table from any user. By specifying ALL, the owner of the table revokes all privileges granted for the table from the user.

2. If a user is not the owner of the table, he may only revoke the privileges he has granted. If a user who is not the owner of the table specifies ALL, he revokes all privileges he has granted for this table from the user identified by <grantee>.

3. The <revoke statement> can cascade; i.e., revoking a privilege from one user can have the effect that this privilege is revoked from other users who may have received this privilege from the user specified in the <revoke statement>. More precisely:
   Let U1, U2, and U3 be users. U1 grants U2 the privilege set P WITH GRANT OPTION, and U2 grants U3 the privilege set P', P' <= P. If U1 revokes the privilege set P'',
   P'' <= P from the user U2, then the privilege set (P' * P'') is implicitly revoked from U3.

4. If the SELECT privilege is revoked from the owner of a view table for a table occurring in the <table expression> of the view definition, the view table is dropped, along with all view tables, privileges, and synonyms based on this view table.

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 (<expression>,...)
   | [(<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 <expression>s must equal the number of target columns. The ith <expression> is assigned the ith column name.

4. The number of <select column>s specified in the <query expression> must equal the number of target columns.

General Rules

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

2. If <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 value of the <default spec>.

   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 value of the <default spec>.

   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 tables into which rows are to be inserted using the <insert statement> are 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. If v is the result of a <query expression>, fractional digits are rounded, if necessary.

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. If C is an alphanumeric column with the code attribute BYTE, then v must be a hexadecimal character string with a length not exceeding the length attribute of C. Trailing binary zeros are disregarded in determining the length of v.
    If the length of v is shorter than the length attribute of C, then v is lengthened by the corresponding number of binary zeros.

15. If C is a column defined with the data type DATE, then v must be a value that corresponds to the format of date values.

16. If a <literal> or a <parameter spec> is used as <expression>, a character string is converted into a number or a number is converted into a character string whenever possible and required by the data type of the target column.

17. 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.

18. The <insert statement> can only be used to assign a value to columns of the data type LONG if it contains a parameter or NULL specification. The assignment of values to LONG columns is therefore only possible with some ADABAS tools. For details, refer to the corresponding manuals.

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

20. 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> [<reference name>]
           <update columns and values>
           [WHERE <search condition>]
  | UPDATE <table name> [<reference 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>
  | <column name> = <subquery>
  | <column name>,... = (<expression>,...)
```

Syntax Rules

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

2. The number of the specified <extended expression>s must equal the number of target columns. The ith <extended expression> is assigned to the ith target column.

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

4. The <subquery> must produce a single-column result table with up to one row.

General Rules

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

2. All target columns must identify columns of the table <table name>, and each target column may only be listed once.

3. The current user must have the UPDATE privilege for each target column in <table name>.
   If <table name> identifies a view table, it may happen that not even the owner of the view table is able to update column values because the view table is not updatable.

4. If <table name> identifies a view table, column values are only updated in rows which belong to the base 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 '<query statement>'.

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 message 100 - 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. The <subquery> must produce a result table containing up to one row.

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

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

values for C. If v is the result of an <expression> which is not made up of a single <numeric literal>, then fractional digits are rounded whenever necessary.

19. If C is an alphanumeric column with the code attribute ASCII or EBCDIC, then v must be a character string with a length 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.

20. If C is an alphanumeric column with the code attribute BYTE, then v must be a hexadecimal character string with a length that does not exceed the length attribute of C. Trailing binary zeros are disregarded in determining the length of v.
If the length of v is shorter than the length attribute of C, then v is lengthened by the corresponding number of binary zeros.

21. If C is a column defined with the data type DATE, then v must be a value that corresponds to the format of date values.

22. If a <literal> or a <parameter spec> is used as <extended expression>, then a character string is converted into a number or a number is converted into a character string whenever possible and required by the data type of the target column.

23. The <update statement> can only be used to assign a new value to columns of the data type LONG if it contains a parameter or NULL specification. The assignment of values to LONG columns is therefore only possible with some ADABAS tools. For details, refer to the corresponding manuals.

24. 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.

25. Should errors occur in the process of updating a row, the <update statement> fails, leaving the table unmodified.

<delete statement>

Function

deletes rows from a table.

Format

```
<delete statement> ::=
    DELETE FROM <table name> [<reference name>]
           [WHERE <search condition>]
  | DELETE FROM <table name> [<reference name>]
           WHERE CURRENT OF <result table name>
```

Syntax Rules

none

General Rules

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

2. The current user must have the DELETE privilege for the table identified by <table name>.
   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 message 100 - 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.

<truncate statement>

Function

deletes all rows from a table.

Format

```
 <truncate statement> ::=
     TRUNCATE <table name> [<storage reuse spec>]

 <storage reuse spec> ::=
     DROP STORAGE
   | REUSE STORAGE
```

Syntax Rules

none

General Rules

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

2. The current user must be the owner of the table identified by <table name>.

3. All rows of the specified table are deleted.

4. The difference between a <delete statement> without <search condition> and without KEY or CURRENT OF <result table name> specification and a <truncate statement> is the following: the deletion of rows executed using a <truncate statement> cannot be cancelled using a <rollback statement>. Triggers created for the table will not be activated.

5. If the specified table is the <referenced table> of a <referential constraint definition>, the <truncate statement> fails.

6. The <storage reuse spec> is meaningless for ADABAS.

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>
  | <select statement>

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

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

<order and update clause> ::=
    <order clause> [<update clause>]
  | <update clause> [<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 or GROUP BY must not be specified.

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

4. An <update clause> can only be specified for updatable result tables. For updatable result tables, a position within a particular result table always corresponds to a position in the underlying table and thus, ultimately, to a position in a base table.

See also

<query expression>

<query spec>

<table expression>

<subquery>

<order clause>

<update clause>

<query expression>

Function

specifies an unordered result table.

Format

```
<query expression> ::=
    <query primary>
  | <query expression> UNION     <query primary>
  | <query expression> INTERSECT <query primary>
  | <query expression> MINUS     <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 be comparable.
   Numeric columns can be compared to each other. If all ith <select column>s are numeric columns, the ith column of the result table is a numeric column.
   Alphanumeric columns with the code attribute BYTE can be compared to each other.
   Alphanumeric columns with the code attribute ASCII or EBCDIC can be compared to each other and to date values.
   If all ith <select column>s are date values, the ith column of the result table is a date value.
   In all the other cases, the ith column of the result table is an alphanumeric column.
   Comparable columns with differing code attributes are converted.
   If columns are comparable but have different lengths, the corresponding column of the result table has the maximum length of the underlying columns.

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 T1 be the left operand of UNION, MINUS or INTERSECT. Let T2 be the right operand. Let R be the result of the operation on T1 and T2.
   A row is a duplicate of another row if both rows have identical values in each column. NULL values are assumed to be identical.

5. If UNION is specified, R contains all rows of T1 and T2.

6. If MINUS is specified, then R contains all rows of T1 which have no duplicate rows in T2.

7. If INTERSECT is specified, then R contains all rows of T1 which have a duplicate row in T2. One row of T2 can only be a duplicate row of just one row of T1. More than one row of T1 cannot have the same duplicate row in T2.

8. DISTINCT is implicitly assumed for the <query expression>s belonging to T1 and T2. All duplicate rows are removed from R.

9. If parentheses are missing, then UNION, MINUS, and INTERSECT will be evaluated from left to right.

<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>
  | <rownum column>

<table columns> ::=
    *
  | <table name>.*
  | <reference name>.*

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

<rownum column> ::=
    ROWNUM [<result column name>]

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

Syntax Rules

1. The specification of a column of the data type LONG in a <select column> is only valid in the uppermost sequence of <select column>s in a <query statement> or <single select statement> if the <distinct spec> DISTINCT has not been used there.
   For restrictions to these options refer to the Precompiler online help, as well as to the manuals of the other components.

2. If a <select column> contains a <set function spec>, the sequence of <select column>s to which the <select column> belongs must not contain any <table columns>, and every column name occurring in an <expression> must denote a grouping column, or the <expression> must consist of grouping columns.

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 <table name>.* or <reference name>.* is an abbreviation of the specification of all columns of the underlying table. The first column name of the result table is taken from the first column name of the underlying table, the second column name of the result table corresponds to the second column name of the underlying table, etc. The order of the column names of the underlying table corresponds to the order determined when the underlying table is defined.
   The implicitly generated column SYSKEY is not passed.

7. The specification of a <derived column> in a <select column> defines a column of the result table. If a column of the result table has the form '<expression> <result column name>', then this result column gets the name <result column name>. If no <result column name> is specified and the <expression> is a <column spec> which denotes a column of the temporary result table, then the column of the result table gets the column name of the temporary result table. If no <result column name> is specified and the <expression> is no <column spec>, then the column gets the name 'EXPRESSION_', where '_' denotes a number with up to three digits, starting with 'EXPRESSION1', 'EXPRESSION2', etc.

8. If a <rownum column> is specified, a column of data type NUMBER(10) is generated having the name ROWNUM. It contains the values 1, 2, 3,... which represent a numbering of the result table rows. If the <rownum column> was specified in the form 'ROWNUM <result column name>', then the result column is given the name <result column name>.
   A <rownum column> must not be ordered by using ORDER BY.

9. Each column of a result table has exactly the same data type, the same length, the same precision, and the same scale as the <derived column> or the column underlying the <table columns>.
   This does not apply to the data type DATE. This is represented with a length of 26 characters.

10. Every column name specified in a <select column> must uniquely identify a column of one of the tables underlying the <query spec>. If need be, the column name must be qualified by the table identifier.

<table expression>

Function

specifies a simple or a grouped result table.

Format

```
 <table expression> ::=
     <from clause>
     [<where clause>]
     [<group clause>]
     [<having clause>]
```

Syntax Rules

1. The order of the <group clause> and <having clause> can be inverted.

General Rules

1. A <table expression> produces a temporary result table. If there are no optional clauses, this temporary result table is the result of the <from clause>. Otherwise, each specified clause is applied to the result of the previous clause and the table is the result of the last specified clause. The temporary result table contains all columns of all tables listed in the <from clause>.

See also

<from clause>

<where clause>

<group clause>

<having clause>

<from clause>

Function

specifies a table that is made up of one or more tables.

Format

```
 <from clause> ::=
     FROM <table spec>,...

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

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, a snapshot 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, view, or snapshot 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>.

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 'Correlated Subquery'), 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 <expression>,...
```

Syntax Rules

none

General Rules

1. Each column name specified in the <group clause> must uniquely denote a column of the tables underlying the <query spec>. If necessary, the column name must be qualified with the table identifier.

2. The <group clause> allows the functions SUM, AVG, MIN, MAX, COUNT, STDDEV, and VARIANCE to be applied not only to entire result tables but also to groups of rows within a result table. A group is defined by the grouping columns specified in GROUP BY. All rows of a group have the same values in the grouping columns. Rows containing the NULL value in a grouping column are combined to form a group.

3. GROUP BY generates one row for each group in the result table. Therefore, the <select column>s in the <query spec> may only contain those grouping columns and operations on grouping columns, as well as those <expression>s that use the functions SUM, AVG, MIN, MAX, COUNT, STDDEV, and VARIANCE.
4. If there is no row that satisfies the conditions indicated in the <where clause> and a <group clause> was specified, then the result table is empty.

<having clause>

Function

specifies the characteristics of a group.

Format

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

Syntax Rules

none

General Rules

1. Each <expression> that is not specified in the argument of a <set function spec> but occurs in the <search condition> must denote a grouping column.

2. The <search condition> is applied to each group of the result table. The result of the <having clause> is a table that only contains those groups for which the <search condition> is satisfied.

<subquery>

Function

specifies a result table that can be used in certain predicates and for the update of column values.

Format

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

Syntax Rules

1. A <subquery> used in a <set update clause> of an <update statement> must only form a single-column result table.

General Rules

1. The result of a <subquery> is a result table.
2. Subqueries can be used in certain predicates such as the <comparison predicate>, <exists predicate>, <in predicate>, and <quantified predicate>.
3. Subqueries can only be used in the <set update clause> of the <update statement>.

See also

Correlated Subquery

Correlated Subquery

Certain predicates can contain subqueries. These subqueries, in turn, can contain other subqueries, etc. A <subquery> containing subqueries is at a higher level than the subqueries included.
Within the <search condition> of a <subquery>, column names may occur that belong to tables contained in the <from clause> of higher-level subqueries. A <subquery> of this kind is called a correlated subquery. Tables that are used in subqueries in such a way are called correlated tables. No more than 16 correlated tables are allowed within an SQL statement. Columns that are used in subqueries in such a way are called correlated columns. Their number in an SQL statement is limited to 64.
If the qualifying table name or reference name does not clearly identify a table of a higher level, the table at the lowest level is taken from these non-unique tables.
If the column name is not qualified by the table name or reference name, the tables at higher levels are scanned for it. The column name must be unique in all tables of the <from clause> to which the table found belongs.
If a correlated subquery is used, the values of one or more columns of a temporary result row at a higher level are included in the <search condition> of a <subquery> at a lower level, whereby the result of the subquery is used for the definite qualification of the higher-level temporary result row.

Example:

We look at a table HOTEL which contains the column names NAME, CITY, HNO, and a table ROOM which contains the column names HNO and PRICE. For every city, the names of all hotels are searched which have prices less than the average price of the city concerned.

```
SELECT name, city
FROM   hotel X, room
WHERE  X.hno = room.hno
  AND  room.price < ( SELECT AVG(room.price)
                 FROM   hotel, room
                 WHERE  hotel.hno  = room.hno
                   AND  hotel.city = X.city )
```

<order clause>

Function

specifies a sorting sequence for a result table.

Format

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

<sort spec> ::=
    <unsigned integer> [<sort option>]
  | <expression> [<sort option>]

<sort option> ::=
    ASC
  | DESC
```

Syntax Rules

1. The maximum number of <sort spec>s that form the sort criterion is 16.

2. If the <query expression> consists of more than one <query spec>, the specification of a <sort spec> is only allowed in the form <unsigned integer> [<sort option>].

General Rules

1. If a <query spec> is specified with DISTINCT, the total of the internal lengths of all sorting columns must not exceed 246 characters; otherwise, 250 characters.

2. Column names in the <sort spec>s must be columns of the tables specified in the <from clause>.

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.

<update clause>

Function

specifies that a result table is to become updatable.

Format

```
 <update clause> ::=
     FOR UPDATE [OF <column name>,...] [NOWAIT]
```

Syntax Rules

none

General Rules

1. The specified column names must denote columns in the table underlying the <query spec>. They need not occur in a <select column>.

2. The <query statement> containing the <update clause> must generate an updatable result table.

3. All columns of the underlying base table are updatable if the user has the corresponding privileges, regardless of whether they were specified as <column name> or not.

   If a column x is contained

   - in an index and
   - in the <search condition> of the <query statement> and
   - in a <set update clause> of the <update statement> in the form 'x = <expression>', where <expression> contains the column x,

   then it is strongly recommended to specify the column x as <column name> in the <update clause>.

   If at least one of these conditions is not satisfied, the column should not be specified.

4. Using the <update clause> has the effect that EXCLUSIVE locks are set for the selected rows.

5. If (NOWAIT) is specified, ADABAS does not wait for the release of a data object locked by another user, but it returns an error message in the case that a collision occurs. If no collision exists, the desired lock is set. If (NOWAIT) is not specified and a collision occurs, then the release of the locked data object is waited for (but only as long as is specified by the installation parameter REQUEST TIMEOUT).

<open cursor statement>

Function

generates the result table previously defined with the specified name.

Format

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

Syntax Rules

none

General Rules

1. Existing result tables are implicitly deleted by the generation of a result table having the same name.

2. 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.

3. If the name of a result table is identical to that of a base table, view table, snapshot table or a synonym, these tables cannot be accessed during the existence of the result table.

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 return code 100 - ROW NOT FOUND - is set.

7. If the result table is not empty, the value 0 is returned in the third entry of SQLERRD in the SQLCA (see the Precompiler online help).

<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 return code 100 - ROW NOT FOUND - is output and no values are assigned to the parameters if any of the following conditions is satisfied:

   a) The result table is empty.

   b) C is positioned on or after the last result table row.

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; or p must be an alphanumeric parameter and v, after having been converted, must be representable in p. If v is a character string, p must

be an alphanumeric parameter or a numeric parameter in which v can be represented after having been converted.

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., error message -51 - LOCK REQUEST TIMEOUT.

8. If a result table that was physically created contains LONG columns and if the ISOLATION LEVELs 0 and 1 are used, then it is not sure that the contents of the LONG columns are consistent with the other columns. If the result table was not physically created, consistency is not ensured in ISOLATION LEVEL 0. For this reason, it is recommended to ensure consistency by using a <lock statement> or the ISOLATION LEVELs 2 or 3.

9. For a successful <fetch statement>, the third entry of SQLERRD in the SQLCA (see the Precompiler online help) is set to the number of rows which have been read so far from the result table <result table name> using a <fetch statement>.

<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. Result tables are implicitly closed when a result table with the same name is generated.

3. 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>]
     [<update clause>]
```

Syntax Rules

1. The order of the <group clause> and <having clause> can also be inverted.

General Rules

1. The specification of a column of the data type LONG in a <select column> is only valid in the uppermost sequence of <select column>s in a <single select statement> if the <distinct spec> DISTINCT was not used there.
   For restrictions to these options refer to the Precompiler online help as well as to the manuals of the other components.

2. For an empty result table, the return code 100 - ROW NOT FOUND - is set.

3. If the result table contains at least 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.

Each data definition and authorizing statement represents a transaction of its own, because it is preceded and concluded by a <commit statement>.

The <commit statement> and the <rollback statement> both implicitly open a new transaction.

Since ADABAS permits concurrent transactions on the same database objects, locks on rows, tables and the catalog are necessary to isolate individual transactions. Locks are either implicitly set by ADABAS in the course of processing an <sql statement> or explicitly set using the <lock statement>. These locks are assigned to the transaction that contains the <sql statement> or <lock statement>. ADABAS distinguishes between SHARE locks and EXCLUSIVE locks which either refer to rows or tables. 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 <savepoint statement> can be used to subdivide a transaction into additional atomic units, whose modifications can be undone using ROLLBACK TO <savepoint name>.

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 ORACLE.

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 or explicitly. Locks are requested explicitly using the <lock statement>. Whether a lock must be requested implicitly or explicitly depends on the <isolation spec> in the <connect statement>. How long an implicit SHARE lock is maintained also depends on the <isolation spec>. Implicitly set EXCLUSIVE locks cannot be released within a transaction. Explicit lock requests are always possible, regardless of the <isolation spec>.

6. ISOLATION LEVEL 0 means that rows can be read without requesting SHARE locks; i.e., no SHARE locks are implicitly requested. For this reason, there is no guarantee that a given row will still be in the same state when it is read again within the same transaction as when it was accessed earlier, since it may have been modified in the meantime by a concurrent transaction.
   Furthermore, there is no guarantee that the state of a read row has already been recorded in the database using COMMIT WORK.
   When rows are inserted, updated or deleted, implicit EXCLUSIVE locks are assigned to the transaction for the rows concerned. These cannot be released until the end of the transaction.

7. ISOLATION LEVEL 1 means that a SHARE lock is assigned to the transaction for each read row R1 in a table. When the next row R2 in the same table is read, the lock on R1 is released and a SHARE lock is assigned to the transaction for the row R2.
   For data retrieval by using a <query statement>, ADABAS makes sure that, at the time each row is read, no EXCLUSIVE lock has been assigned to other transactions for the given row. It is, however, impossible to predict whether a <query statement> causes a SHARE lock for a row of the specified table or not and for which row this may occur.
   When rows are inserted, updated or deleted, implicit EXCLUSIVE locks are assigned to the transaction for the rows concerned. These cannot be released until the end of the transaction.

8. 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 1 is assumed.

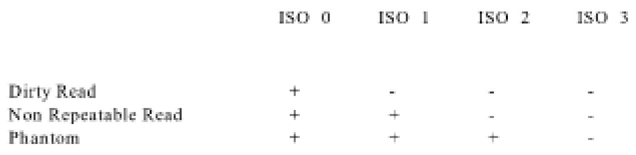
11. Which <isolation spec> is selected affects both the degree of concurrency and the guaranteed consistency. A high degree of concurrency is characterized by a state in which a maximum number of concurrent transactions can process a database without long waiting periods for locks to be released. As 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 ADABAS 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. 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. The <isolation spec> declared in the <connect statement> controls the setting of locks in the new transaction.

<rollback to statement>

Function

undoes the modifications of a transaction back to a defined savepoint.

Format

```
<rollback to statement> ::=
    ROLLBACK TO [SAVEPOINT] <savepoint name>

<savepoint name> ::=
    <identifier>
```

Syntax Rules

none

General Rules

1. <savepoint name> must identify an active savepoint within the current transaction defined by the <savepoint statement>.

2. All database modifications are undone which were made within the current transaction since the execution of the <savepoint statement> identified by the <savepoint name>.

3. All savepoints that were created by <savepoint statement>s executed after the identified <savepoint statement> are deactivated. Savepoints created by <savepoint statement>s executed prior to the execution of the identified <savepoint statement> are not affected.

4. The identified savepoint remains active; i.e., it is possible to repeat the <rollback to statement> with the same <savepoint name> several times during the same transaction.

5. The <rollback to statement> has no effect on the locks assigned to the transaction. In particular, no locks are released for the <rollback to statement>.

<savepoint statement>

Function

defines a savepoint within a transaction.

Format

```
 <savepoint statement> ::=
     SAVEPOINT <savepoint name>

 <savepoint name> ::=
     <identifier>
```

Syntax Rules

none

General Rules

1. The <savepoint statement> opens a subtransaction; i.e., ADABAS records the present point in the transaction and assigns the <savepoint name> to it. The savepoint is marked as active. This can be followed by any sequence of <sql statement>s. A <rollback to statement> can then be used to undo all the database modifications performed after the <savepoint statement>.
   The sequence of <sql statement>s can contain additional <savepoint statement>s.

2. The <savepoint name>s within a transaction must differ from each other. If a <savepoint name> is assigned twice during a transaction, the first point with the <savepoint name> defined within the transaction using the first <savepoint statement> becomes inactive.
3. The <savepoint statement>, as the <rollback to statement>, does not affect any locks assigned to the transaction. In particular, no locks are released for the two SQL statements.

4. The <savepoint statement> is particularly useful in keeping the effects of subroutines atomic; i.e., it ensures that they either fulfil all their tasks or else have no effect. To achieve this, first of all, a <savepoint statement> is issued. In the event that the subroutine fails to successfully fulfil its task, all modifications executed by the subroutine are undone using a <rollback to statement>.

5. The <commit statement> and the <rollback statement> implicitly close any subtransactions still open.

<lock statement>

Function

assigns a lock to the current transaction.

Format

```
 <lock statement> ::=
     LOCK TABLE <table name>,... IN <lock spec> MODE [NOWAIT]

  <lock spec> ::=
     SHARE
   | ROW SHARE
   | SHARE UPDATE
   | EXCLUSIVE
   | ROW EXCLUSIVE
   | SHARE ROW EXCLUSIVE
```

Syntax Rules

none

General Rules

1. The specified table <table name> can be a base table, view table, snapshot table or a synonym. If <table name> identifies a view table, then locks are set on the base tables on which the view table is based. To set SHARE locks, the current user must have the SELECT privilege; to set EXCLUSIVE locks, the user needs the UPDATE, DELETE or INSERT privilege.

2. If the view table identified by <table name> is not updatable, then only a SHARE lock can be set for this view table. As a result of this SQL statement, all base tables underlying the <table name> are subsequently locked in SHARE mode.

3. If <table name> identifies a snapshot table, only a SHARE lock can be set for this table.

4. SHARE, ROW SHARE and SHARE UPDATE define a SHARE lock for the listed tables. If a SHARE lock has been set, no concurrent transaction can modify the locked table.

5. EXCLUSIVE, ROW EXCLUSIVE and SHARE ROW EXCLUSIVE define EXCLUSIVE locks for the listed tables. If an EXCLUSIVE lock is set, no concurrent transaction can modify the locked tables. Concurrent transactions can only read-access locked tables in ISOLATION LEVEL 0.

6. If no lock has been assigned to a transaction for a data object, then a SHARE or EXCLUSIVE lock can be requested within any transaction and the lock is immediately

assigned to the transaction.
If a SHARE lock has been assigned to a transaction T for a data object, and if no lock has been assigned to any concurrent transaction for this data object, then the transaction T can request an EXCLUSIVE lock for this data object and the lock is immediately assigned to this transaction.
If an EXCLUSIVE lock has been assigned to a transaction for a data object, then a SHARE lock can, but need not, be requested for this transaction.
The matrix 'Transactions'at the beginning of this chapter shows the possible parallel locks.
A lock collision exists in the cases which are marked with 'No'; i.e., after having requested a lock within a transaction, the user has to wait for the lock to be released until one of the above situations or one of the situations that are marked with 'Yes' in the matrix occurs.

7. Locks can be requested either implicitly or explicitly. Explicit lock requests are performed using the <lock statement>. Whether a lock is requested implicitly and how long it remains assigned to the transaction depends on the <isolation spec> in the <connect statement>.

8. The locks assigned to a transaction by a <lock statement> are released at the end of the transaction.

9. If NOWAIT is specified, ADABAS does not wait for a lock to be released by another transaction, but issues an error message if there is a lock collision. If there is no collision, the requested lock is set.

10. In the event of a lock collision, if NOWAIT is omitted, the system waits for locks to be released, until the period specified by the installation parameter REQUEST TIMEOUT has elapsed.
If ADABAS has to wait too long for locks to be released when setting explicit or implicit locks, it issues an error message to this effect. The user can then respond to this error message, e.g., by terminating the transaction. In these situations, ADABAS does not execute an implicit ROLLBACK WORK.
Whenever ADABAS recognizes a deadlock caused by explicit or implicit locks, it ends the transaction with an implicit ROLLBACK WORK.

11. If reproducible results are needed for reading rows using a <select statement>, the read objects must be locked and the locks must be kept until reproduction. Reproducibility usually requires that the tables concerned are locked in SHARE mode, either explicitly using one or more <lock statement>s or implicitly by using the ISOLATION LEVEL 3. This ensures that no other user can modify the table.

12. The fewer objects are locked, the more transactions can operate simultaneously on the database without colliding with lock requests of other transactions. For this reason, unnecessary locks should be avoided and set locks should be released as soon as possible.

13. If a transaction implicitly requests too many row locks (SHARE or EXCLUSIVE locks) on a table, ADABAS tries to obtain a table lock instead. If this causes collisions with other locks, ADABAS continues to request row locks. This means that table locks can be obtained without waiting for them. The limit beyond which ADABAS tries to transform row locks into table locks depends on the installation parameter MAXLOCKS.

<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.

System Tables

This chapter describes the system tables that are equivalent to those of ORACLE7 and are only known in the SQLMODE ORACLE. These system tables belong to the user 'SYS' the name of whom can, but need not, be placed in front of the name of the system table. All system tables the names of which start with DBA_ are only visible to users with DBA status; all the other tables are visible to all users. System tables the names of which start with USER_ contain information about objects related to the current user (e.g., that are owned by the current user). System tables the names of which start with ALL_ contain information about objects known to the user; i.e., objects that are owned by him or for which he has privileges.

System tables denoted by an '*' after the table name are known to ADABAS, but do not contain any entries because the corresponding information or objects do not exist in ADABAS. An '*' at the beginning of a column description means that this column always contains the NULL value, because this information is not available in the format required for ADABAS. The table and column descriptions are those given in ORACLE7. Therefore, they also refer to objects that are not available in ADABAS, e.g, PACKAGE.

If you want to retrieve information about certain tables, users, etc., ensure that their names are enclosed in single quotation marks. Names specified as <simple identifier>s must be specified in uppercase characters. Names specified as <special identifier>s are specified in the desired combination of upper- and lowercase characters without the enclosing <double quotes>. If <double quotes> belong to the <special identifier>, they are not entered twice.

ALL_CATALOG		All tables, views, synonyms, sequences accessible to the user
OWNER	VARCHAR2 (30)	Owner of the object
TABLE_NAME	VARCHAR2 (30)	Name of the object
TABLE_TYPE	VARCHAR2 (11)	Type of the object

ALL_COL_COMMENTS		Comments on columns of accessible tables and views
OWNER	VARCHAR2 (30)	Owner of the object
TABLE_NAME	VARCHAR2 (30)	Name of the object
COLUMN_NAME	VARCHAR2 (30)	Name of the column
COMMENTS	VARCHAR2 (254)	Comment on the column

ALL_COL_PRIVS		Grants on columns for which the user is the grantor, grantee, owner, or an enabled role or PUBLIC is the grantee
GRANTOR	VARCHAR2 (30)	Name of the user who performed the grant
GRANTEE	VARCHAR2 (30)	Name of the user to whom access was granted
TABLE_SCHEMA	VARCHAR2 (30)	Schema of the object
TABLE_NAME	VARCHAR2 (30)	Name of the object
COLUMN_NAME	VARCHAR2 (30)	Name of the column
PRIVILEGE	VARCHAR2 (30)	Column Privilege
GRANTABLE	VARCHAR2 (3)	Privilege is grantable

ALL_COL_PRIVS_MADE		Grants on columns for which the user is owner or grantor
GRANTEE	VARCHAR2 (30)	Name of the user to whom access was granted
OWNER	VARCHAR2 (30)	Username of the owner of the object
TABLE_NAME	VARCHAR2 (30)	Name of the object
COLUMN_NAME	VARCHAR2 (30)	Name of the column
GRANTOR	VARCHAR2 (30)	Name of the user who performed the grant
PRIVILEGE	VARCHAR2 (30)	Column Privilege
GRANTABLE	VARCHAR2 (3)	Privilege is grantable

ALL_COL_PRIVS_RECD		Grants on columns for which the user, PUBLIC or enabled role is the grantee
GRANTEE	VARCHAR2 (30)	Name of the user to whom access was granted
OWNER	VARCHAR2 (30)	Username of the owner of the object
TABLE_NAME	VARCHAR2 (30)	Name of the object
COLUMN_NAME	VARCHAR2 (30)	Name of the column
GRANTOR	VARCHAR2 (30)	Name of the user who performed the grant
PRIVILEGE	VARCHAR2 (30)	Column privilege
GRANTABLE	VARCHAR2 (3)	Privilege is grantable

ALL_CONSTRAINTS		Constraint definitions on accessible tables
OWNER	VARCHAR2 (30)	Owner of the table
CONSTRAINT_NAME	VARCHAR2 (30)	Name associated with constraint definition
CONSTRAINT_TYPE	VARCHAR2 (2)	Type of constraint definition
TABLE_NAME	VARCHAR2 (30)	Name associated with table with constraint definition
SEARCH_CONDITION	VARCHAR2 (254)	Text of search condition for table check
R_OWNER	VARCHAR2 (30)	Owner of table used in referential constraint
R_CONSTRAINT_NAME	VARCHAR2 (30)	Name of unique constraint definition for referenced table
DELETE_RULE	VARCHAR2 (18)	The delete rule for a referential constraint
STATUS	VARCHAR2 (8)	enforcement status of constraint - ENABLED or DISABLED

ALL_CONS_COLUMNS		Information about accessible columns in constraint definitions
OWNER	VARCHAR2 (30)	Owner of the constraint definition
CONSTRAINT_NAME	VARCHAR2 (30)	Name associated with the constraint definition
TABLE_NAME	VARCHAR2 (30)	Name associated with table with constraint definition
COLUMN_NAME	VARCHAR2 (30)	Name associated with column specified in the constraint definition
POSITION	NUMBER	Original position of column in definition

ALL_DB_LINKS	*	Database links accessible to the user

The structure of the system table ALL_db_links and the meaning of its columns correspond to those of the system table DBA_db_links. In contrast to DBA_db_links, ALL_db_links does not contain the column PASSWORD.

ALL_DEF_AUDIT_OPTS		Auditing options for newly created objects
ALT	VARCHAR2 (3)	Auditing ALTER WHENEVER SUCCESSFUL / UNSUCCESSFUL
AUD	VARCHAR2 (3)	Auditing AUDIT WHENEVER SUCCESSFUL / UNSUCCESSFUL
COM	VARCHAR2 (3)	Auditing COMMENT WHENEVER SUCCESSFUL / UNSUCCESSFUL
DEL	VARCHAR2 (3)	Auditing DELETE WHENEVER SUCCESSFUL / UNSUCCESSFUL
GRA	VARCHAR2 (3)	Auditing GRANT WHENEVER SUCCESSFUL / UNSUCCESSFUL
IND	VARCHAR2 (3)	Auditing INDEX WHENEVER SUCCESSFUL / UNSUCCESSFUL
INS	VARCHAR2 (3)	Auditing INSERT WHENEVER SUCCESSFUL / UNSUCCESSFUL
LOC	VARCHAR2 (3)	Auditing LOCK WHENEVER SUCCESSFUL / UNSUCCESSFUL
REN	VARCHAR2 (3)	Auditing RENAME WHENEVER SUCCESSFUL / UNSUCCESSFUL
SEL	VARCHAR2 (3)	Auditing SELECT WHENEVER SUCCESSFUL / UNSUCCESSFUL
UPD	VARCHAR2 (3)	Auditing UPDATE WHENEVER SUCCESSFUL / UNSUCCESSFUL
REF	VARCHAR2 (3)	Auditing REFERENCES WHENEVER SUCCESSFUL / UNSUCCESSFUL (not used)
EXE	VARCHAR2 (3)	Auditing EXECUTE WHENEVER SUCCESSFUL / UNSUCCESSFUL

ALL_DEPENDENCIES		Dependencies to and from objects accessible to the user
OWNER	VARCHAR2 (30)	Owner of the object
NAME	VARCHAR2 (30)	Name of the object
TYPE	VARCHAR2 (12)	Type of the object
REFERENCED_OWNER	VARCHAR2 (30)	Owner of referenced object (remote owner if remote object)
REFERENCED_NAME	VARCHAR2 (30)	Name of referenced object
REFERENCED_TYPE	VARCHAR2 (12)	Type of referenced object
REFERENCED_LINK_NAME	VARCHAR2 (30)	* Name of dblink if this is a remote object

ALL_ERRORS	*	Current errors on stored objects that user is allowed to create
OWNER	VARCHAR2 (30)	Owner of the object
NAME	VARCHAR2 (30)	Name of the object
TYPE	VARCHAR2 (12)	Type of object: "VIEW", "PROCEDURE", "FUNCTION", "PACKAGE" or "PACKAGE BODY"

SEQUENCE	NUMBER	Sequence number used for ordering purposes
LINE	NUMBER	Line number at which this error occurs
POSITION	NUMBER	Position in the line at which this error occurs
TEXT	VARCHAR2 (200)	Text of the error

ALL_INDEXES		Descriptions of indexes on tables accessible to the user
OWNER	VARCHAR2 (30)	Username of the owner of the index
INDEX_NAME	VARCHAR2 (30)	Name of the index
TABLE_OWNER	VARCHAR2 (30)	Owner of the indexed object
TABLE_NAME	VARCHAR2 (30)	Name of the indexed object
TABLE_TYPE	VARCHAR2 (11)	Type of the indexed object
UNIQUENESS	VARCHAR2 (9)	Uniqueness status of the index: "UNIQUE" or "NONUNIQUE"
TABLESPACE_NAME	VARCHAR2 (30)	Name of the tablespace containing the index
INI_TRANS	NUMBER	* Initial number of transactions
MAX_TRANS	NUMBER	* Maximum number of transactions
INITIAL_EXTENT	NUMBER	* Size of the initial extent
NEXT_EXTENT	NUMBER	* Size of secondary extents
MIN_EXTENTS	NUMBER	* Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER	* Maximum number of extents allowed in the segment
PCT_INCREASE	NUMBER	* Percentage increase in extent size
PCT_FREE	NUMBER	* Minimum percentage of free space in a block
BLEVEL	NUMBER	* B*-Tree level
LEAF_BLOCKS	NUMBER	* The number of leaf blocks in the index
DISTINCT_KEYS	NUMBER	The number of distinct keys in the index
AVG_LEAF_BLOCKS_ PER_KEY	NUMBER	* The average number of leaf blocks per key
AVG_DATA_BLOCKS_ PER_KEY	NUMBER	* The average number of data blocks per key
CLUSTERING_FACTOR	NUMBER	* A measurement of the amount of (dis)order of the table this index is for
STATUS	VARCHAR2 (11)	Whether index is in Direct Load State or not

ALL_IND_COLUMNS		COLUMNs comprising INDEXes on accessible TABLES
INDEX_OWNER	VARCHAR2 (30)	Index owner
INDEX_NAME	VARCHAR2 (30)	Index name
TABLE_OWNER	VARCHAR2 (30)	Table or cluster owner
TABLE_NAME	VARCHAR2 (30)	Table or cluster name
COLUMN_NAME	VARCHAR2 (30)	Column name
COLUMN_POSITION	NUMBER	Position of column within index
COLUMN_LENGTH	NUMBER	Indexed length of the column

ALL_OBJECTS		Objects accessible to the user
OWNER	VARCHAR2 (30)	Username of the owner of the object
OBJECT_NAME	VARCHAR2 (30)	Name of the object
OBJECT_ID	RAW (8)	Object number of the object
OBJECT_TYPE	VARCHAR2 (11)	Type of the object
CREATED	DATE	Timestamp for the creation of the object
LAST_DDL_TIME	DATE	Timestamp for the last DDL change (including GRANT and REVOKE) to the object
TIMESTAMP	VARCHAR2 (75)	Timestamp for the specification of the object
STATUS	VARCHAR2 (7)	Status of the object

ALL_SEQUENCES		Description of SEQUENCEs accessible to the user
SEQUENCE_OWNER	VARCHAR2 (30)	Name of the owner of the sequence
SEQUENCE_NAME	VARCHAR2 (30)	SEQUENCE name
MIN_VALUE	NUMBER	Minimum value of the sequence
MAX_VALUE	NUMBER	Maximum value of the sequence
INCREMENT_BY	NUMBER	Value by which sequence is incremented
CYCLE_FLAG	VARCHAR2 (1)	Does sequence wrap around on reaching limit?
ORDER_FLAG	VARCHAR2 (1)	Are sequence numbers generated in order?
CACHE_SIZE	NUMBER	Number of sequence numbers to cache
LAST_NUMBER	NUMBER	Last sequence number written to disk

ALL_SNAPSHOTS		Snapshots the user can look at
OWNER	VARCHAR2 (30)	Owner of the snapshot
NAME	VARCHAR2 (30)	The view used by users and applications for viewing the snapshot
TABLE_NAME	VARCHAR2 (30)	* Table the snapshot is stored in -- has an extra column for the master rowid
MASTER_VIEW	VARCHAR2 (30)	* View of the master table, owned by the snapshot owner, used for refreshes
MASTER_OWNER	VARCHAR2 (30)	Owner of the master table
MASTER	VARCHAR2 (30)	Name of the master table that this snapshot is a copy of
MASTER_LINK	VARCHAR2 (30)	* Database link name to the master site
CAN_USE_LOG	VARCHAR2 (3)	If NO, this snapshot is complex and will never use a log
LAST_REFRESH	DATE	* SYSDATE from the master site at the time of the last refresh
ERROR	NUMBER	* The error returned last time an automatic refresh was attempted
TYPE	VARCHAR2 (8)	The type of refresh (complete,fast,force) for all automatic refreshes
NEXT	VARCHAR2 (254)	* The date function used to compute next refresh dates
START_WITH	DATE	* The date function used to compute next refresh dates
QUERY	LONG	The original query that this snapshot is an instantiation of

ALL_SOURCE	*	Current source on stored objects that user is allowed to create
OWNER	VARCHAR2 (30)	Owner of the object
NAME	VARCHAR2 (30)	Name of the object
TYPE	VARCHAR2 (11)	Type of the object: "PROCEDURE", "FUNCTION", "PACKAGE" or "PACKAGE BODY"
LINE	NUMBER	Line number of this line of source
TEXT	VARCHAR2 (200)	Source text
ALL_SYNONYMS		All synonyms accessible to the user
OWNER	VARCHAR2 (30)	Owner of the synonym
SYNONYM_NAME	VARCHAR2 (30)	Name of the synonym
TABLE_OWNER	VARCHAR2 (30)	Owner of the object referenced by the synonym
TABLE_NAME	VARCHAR2 (30)	Name of the object referenced by the synonym
DB_LINK	VARCHAR2 (30)	* Name of the database link referenced in a remote synonym
ALL_TABLES		Description of tables accessible to the user
OWNER	VARCHAR2 (30)	Owner of the table
TABLE_NAME	VARCHAR2 (30)	Name of the table
TABLESPACE_NAME	VARCHAR2 (30)	Name of the tablespace containing the table
CLUSTER_NAME	VARCHAR2 (30)	* Name of the cluster, if any, to which the table belongs
PCT_FREE	NUMBER	* Minimum percentage of free space in a block
PCT_USED	NUMBER	* Minimum percentage of used space in a block
INI_TRANS	NUMBER	* Initial number of transactions
MAX_TRANS	NUMBER	* Maximum number of transactions
INITIAL_EXTENT	NUMBER	* Size of the initial extent in bytes
NEXT_EXTENT	NUMBER	* Size of secondary extents in bytes
MIN_EXTENTS	NUMBER	* Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER	* Maximum number of extents allowed in the segment
PCT_INCREASE	NUMBER	* Percentage increase in extent size
BACKED_UP	VARCHAR2 (1)	* Has table been backed up since last modification?
NUM_ROWS	NUMBER	* The number of rows in the table
BLOCKS	NUMBER	* The number of used blocks in the table
EMPTY_BLOCKS	NUMBER	* The number of empty (never used) blocks in the table
AVG_SPACE	NUMBER	* The average available free space in the table
CHAIN_CNT	NUMBER	* The number of chained rows in the table
AVG_ROW_LEN	NUMBER	* The average row length, including row overhead

ALL_TAB_COLUMNS		Columns of all tables, views and clusters
OWNER	VARCHAR2 (30)	Owner of the table, view or cluster
TABLE_NAME	VARCHAR2 (30)	Table, view or cluster name
COLUMN_NAME	VARCHAR2 (30)	Column name
DATA_TYPE	VARCHAR2 (9)	Datatype of the column
DATA_LENGTH	NUMBER	Length of the column in bytes
DATA_PRECISION	NUMBER	Length: decimal digits (NUMBER) or binary digits (FLOAT)
DATA_SCALE	NUMBER	Digits to right of decimal point in a number
NULLABLE	VARCHAR2 (1)	Does column allow NULL values?
COLUMN_ID	NUMBER	Sequence number of the column as created
DEFAULT_LENGTH	NUMBER	Length of default value for the column
DATA_DEFAULT	VARCHAR2 (254)	Default value for the column
NUM_DISTINCT	NUMBER	The number of distinct values for the column
LOW_VALUE	RAW (1)	* The second smallest value for the column
HIGH_VALUE	RAW (1)	* The second highest value for the column
DENSITY	NUMBER	* The density of the column

ALL_TAB_COMMENTS		Comments on tables and views accessible to the user
OWNER	VARCHAR2 (30)	Owner of the object
TABLE_NAME	VARCHAR2 (30)	Name of the object
TABLE_TYPE	VARCHAR2 (11)	Type of the object
COMMENTS	VARCHAR2 (254)	Comment on the object

ALL_TAB_PRIVS		Grants on objects for which the user is the grantor, grantee, owner, or an enabled role or PUBLIC is the grantee
GRANTOR	VARCHAR2 (30)	Name of the user who performed the grant
GRANTEE	VARCHAR2 (30)	Name of the user to whom access was granted
TABLE_SCHEMA	VARCHAR2 (30)	Schema of the object
TABLE_NAME	VARCHAR2 (30)	Name of the object
PRIVILEGE	VARCHAR2 (30)	Table Privilege
GRANTABLE	VARCHAR2 (3)	Privilege is grantable

ALL_TAB_PRIVS_MADE		User's grants and grants on user's objects
GRANTEE	VARCHAR2 (30)	Name of the user to whom access was granted
OWNER	VARCHAR2 (30)	Owner of the object
TABLE_NAME	VARCHAR2 (30)	Name of the object
GRANTOR	VARCHAR2 (30)	Name of the user who performed the grant
PRIVILEGE	VARCHAR2 (30)	Table Privilege

GRANTABLE	VARCHAR2 (3)	Privilege is grantable
ALL_TAB_PRIVS_RECD		Grants on objects for which the user, PUBLIC or enabled role is the grantee
GRANTEE	VARCHAR2 (30)	Name of the user to whom access was granted
OWNER	VARCHAR2 (30)	Owner of the object
TABLE_NAME	VARCHAR2 (30)	Name of the object
GRANTOR	VARCHAR2 (30)	Name of the user who performed the grant
PRIVILEGE	VARCHAR2 (30)	Table Privilege
GRANTABLE	VARCHAR2 (3)	Privilege is grantable
ALL_TRIGGERS		Triggers accessible to the current user
OWNER	VARCHAR2 (30)	Owner of the trigger
TRIGGER_NAME	VARCHAR2 (30)	Name of the trigger
TRIGGER_TYPE	VARCHAR2 (16)	When the trigger fires - BEFORE/AFTER and STATEMENT/ROW
TRIGGERING_EVENT	VARCHAR2 (26)	Statement that will fire the trigger - INSERT, UPDATE and/or DELETE
TABLE_OWNER	VARCHAR2 (30)	Owner of the table that this trigger is associated with
TABLE_NAME	VARCHAR2 (30)	Name of the table that this trigger is associated with
REFERENCING_NAMES	VARCHAR2 (87)	* Names used for referencing to OLD and NEW values within the trigger
WHEN_CLAUSE	VARCHAR2 (200)	* WHEN clause must evaluate to true in order for triggering body to execute
STATUS	VARCHAR2 (8)	If DISABLED then trigger will not fire
DESCRIPTION	VARCHAR2 (254)	Trigger description, useful for re-creating trigger creation statement
TRIGGER_BODY	RAW (1)	* Action taken by this trigger when it fires
ALL_TRIGGER_COLS	*	Column usage in user's triggers or in triggers on user's tables
TRIGGER_OWNER	VARCHAR2 (30)	Owner of the trigger
TRIGGER_NAME	VARCHAR2 (30)	Name of the trigger
TABLE_OWNER	VARCHAR2 (30)	Owner of the table
TABLE_NAME	VARCHAR2 (30)	Name of the table on which the trigger is defined
COLUMN_NAME	VARCHAR2 (30)	Name of the column used in trigger definition
COLUMN_LIST	VARCHAR2 (3)	Is column specified in UPDATE OF clause?
COLUMN_USAGE	VARCHAR2 (17)	Usage of column within trigger body
ALL_USERS		Information about all users of the database
USERNAME	VARCHAR2 (30)	Name of the user
USER_ID	NUMBER	ID number of the user
CREATED	DATE	User creation date

ALL_VIEWS		Text of views accessible to the user
OWNER	VARCHAR2 (30)	Owner of the view
VIEW_NAME	VARCHAR2 (30)	Name of the view
TEXT_LENGTH	NUMBER	Length of the view text
TEXT	LONG	View text
AUDIT_ACTIONS		Description table for audit trail action type codes. Maps action type numbers to action type names
ACTION	FLOAT (22)	Numeric audit trail action type code
NAME	VARCHAR2 (27)	Name of the type of audit trail action
COLUMN_PRIVILEGES		Grants on columns for which the user is the grantor, grantee, owner, or an enabled role or PUBLIC is the grantee
GRANTEE	VARCHAR2 (30)	Name of the user to whom access was granted
OWNER	VARCHAR2 (30)	Username of the object's owner
TABLE_NAME	VARCHAR2 (30)	Name of the object
COLUMN_NAME	VARCHAR2 (30)	Name of the column
GRANTOR	VARCHAR2 (30)	Name of the user who performed the grant
INSERT_PRIV	VARCHAR2 (1)	Permission to insert into the column
UPDATE_PRIV	VARCHAR2 (1)	Permission to update the column
REFERENCES_PRIV	VARCHAR2 (1)	Permission to reference the column
CREATED	DATE	Timestamp for the grant
DBA_2PC_NEIGHBORS	*	information about incoming and outgoing connections for pending transactions
LOCAL_TRAN_ID	VARCHAR2 (22)	Transaction ID on the local database in the format n.n.n
IN_OUT	VARCHAR2 (3)	"in" for incoming connections, "out" for outgoing
DATABASE	VARCHAR2 (128)	in: client database name; out: outgoing db link
DBUSER_OWNER	VARCHAR2 (30)	in: name of local user; out: owner of db link
INTERFACE	VARCHAR2 (1)	"C" for request commit, else "N" for prepare or request readonly commit
DBID	VARCHAR2 (16)	the database id at the other end of the connection
SESS#	NUMBER	session number at this database of the connection
BRANCH	VARCHAR2 (128)	transaction branch ID at this database of the connection
DBA_2PC_PENDING	*	info about distributed transactions awaiting recovery
LOCAL_TRAN_ID	VARCHAR2 (22)	string of form: n.n.n, n a number
GLOBAL_TRAN_ID	VARCHAR2 (169)	globally unique transaction id
STATE	VARCHAR2 (16)	collecting, prepared, committed, forced

		commit, or forced rollback
MIXED	VARCHAR2 (3)	yes => part of the transaction committed and part rolled back (commit or rollback with the FORCE option was used)
ADVICE	VARCHAR2 (1)	C for commit, R for rollback, else null
TRAN_COMMENT	VARCHAR2 (254)	text for "commit work comment <text>"
FAIL_TIME	DATE	value of SYSDATE when the row was inserted (tx or system recovery)
FORCE_TIME	DATE	time of manual force decision (null if not forced locally)
RETRY_TIME	DATE	time automatic recovery (RECO) last tried to recover the transaction
OS_USER	VARCHAR2 (254)	operating system specific name for the end-user
OS_TERMINAL	VARCHAR2 (254)	operating system specific name for the end-user terminal
HOST	VARCHAR2 (254)	name of the host machine for the end-user
DB_USER	VARCHAR2 (30)	Oracle user name of the end-user at the topmost database
COMMIT#	VARCHAR2 (16)	global commit number for committed transactions

DBA_AUDIT_EXISTS	*	All audit trail entries
OS_USERNAME	VARCHAR2 (254)	Operating System logon user name of the user whose actions were audited
USERNAME	VARCHAR2 (30)	Name (not ID number) of the user whose actions were audited
USERHOST	VARCHAR2 (254)	Numeric instance ID for the Oracle instance from which the user is accessing the database. Used only in environments with distributed file systems and shared database files (e.g., clustered Oracle on DEC VAX/VMS clusters)
TERMINAL	VARCHAR2 (254)	Identifier for the user's terminal
TIMESTAMP	DATE	Timestamp for the creation of the audit trail entry (Timestamp for the user's logon for entries created by AUDIT SESSION)
OWNER	VARCHAR2 (30)	Creator of object affected by the action
OBJ_NAME	VARCHAR2 (128)	Name of the object affected by the action
ACTION_NAME	VARCHAR2 (27)	Name of the action type corresponding to the numeric code in ACTION
NEW_OWNER	VARCHAR2 (30)	The owner of the object named in the NEW_NAME column
NEW_NAME	VARCHAR2 (128)	New name of object after RENAME, or name of underlying object (e.g. CREATE INDEX owner.obj_name ON new_owner.new_name)
OBJ_PRIVILEGE	VARCHAR2 (16)	Object privileges granted/revoked by a GRANT/REVOKE statement
SYS_PRIVILEGE	VARCHAR2 (40)	System privileges granted/revoked by a GRANT/REVOKE statement
GRANTEE	VARCHAR2 (30)	The name of the grantee specified in a GRANT/REVOKE statement
SESSIONID	NUMBER	Numeric ID for each Oracle session

ENTRYID	NUMBER	Numeric ID for each audit trail entry in the session
STATEMENTID	NUMBER	Numeric ID for each statement run (a statement may cause many actions)
RETURNCODE	NUMBER	Oracle error code generated by the action. Zero if the action succeeded
DBA_AUDIT_OBJECT	*	Audit trail records for statements concerning objects, specifically: table, cluster, view, index, sequence, [public] database link, [public] synonym, procedure, trigger, rollback segment, tablespace, role, user
OS_USERNAME	VARCHAR2 (254)	Operating System logon user name of the user whose actions were audited
USERNAME	VARCHAR2 (30)	Name (not ID number) of the user whose actions were audited
USERHOST	VARCHAR2 (254)	Numeric instance ID for the Oracle instance from which the user is accessing the database. Used only in environments with distributed file systems and shared database files (e.g., clustered Oracle on DEC VAX/VMS clusters)
TERMINAL	VARCHAR2 (254)	Identifier for the user's terminal
TIMESTAMP	DATE	Timestamp for the creation of the audit trail entry (Timestamp for the user's logon for entries created by AUDIT SESSION)
OWNER	VARCHAR2 (30)	Creator of object affected by the action
OBJ_NAME	VARCHAR2 (128)	Name of the object affected by the action
ACTION_NAME	VARCHAR2 (27)	Name of the action type corresponding to the numeric code in ACTION
NEW_OWNER	VARCHAR2 (30)	The owner of the object named in the NEW_NAME column
NEW_NAME	VARCHAR2 (128)	New name of object after RENAME, or name of underlying object (e.g. CREATE INDEX owner.obj_name ON new_owner.new_name)
SES_ACTIONS	VARCHAR2 (16)	Session summary. A string of 11 characters, one for each action type, in this order: Alter, Audit, Comment, Delete, Grant, Index, Insert, Lock, Rename, Select, Update. Values: "-" = None, "S" = Success, "F" = Failure, "B" = Both
COMMENT_TEXT	VARCHAR2 (254)	Text comment on the audit trail entry
SESSIONID	NUMBER	Numeric ID for each Oracle session
ENTRYID	NUMBER	Numeric ID for each audit trail entry in the session
STATEMENTID	NUMBER	Numeric ID for each statement run (a statement may cause many actions)
RETURNCODE	NUMBER	Oracle error code generated by the action. Zero if the action succeeded
PRIV_USED	VARCHAR2 (40)	System privilege used to execute the action
OBJECT_LABEL	VARCHAR2 (1)	Optional Trusted ORACLE label associated with object being audited

SESSION_LABEL	VARCHAR2 (1)	Trusted ORACLE label associated with user session
DBA_AUDIT_SESSION	*	All audit trail entries concerning connect and disconnect
OS_USERNAME	VARCHAR2 (254)	Operating System logon user name of the user whose actions were audited
USERNAME	VARCHAR2 (30)	Name (not ID number) of the user whose actions were audited
USERHOST	VARCHAR2 (254)	Numeric instance ID for the Oracle instance from which the user is accessing the database. Used only in environments with distributed file systems and shared database files (e.g., clustered Oracle on DEC VAX/VMS clusters)
TERMINAL	VARCHAR2 (254)	Identifier for the user's terminal
TIMESTAMP	DATE	Timestamp for the creation of the audit trail entry (Timestamp for the user's logon for entries created by AUDIT SESSION)
ACTION_NAME	VARCHAR2 (27)	Name of the action type corresponding to the numeric code in ACTION
LOGOFF_TIME	DATE	Timestamp for user logoff
LOGOFF_LREAD	NUMBER	Logical reads for the session
LOGOFF_PREAD	NUMBER	Physical reads for the session
LOGOFF_LWRITE	NUMBER	Logical writes for the session
LOGOFF_DLOCK	VARCHAR2 (40)	Deadlocks detected during the session
SESSIONID	NUMBER	Numeric ID for each Oracle session
RETURNCODE	NUMBER	Oracle error code generated by the action. Zero if the action succeeded
SESSION_LABEL	VARCHAR2 (1)	Trusted ORACLE label associated with user session
DBA_AUDIT_STATEMENT	*	Audit trail records concerning grant, revoke, audit, noaudit and alter system
OS_USERNAME	VARCHAR2 (254)	Operating System logon user name of the user whose actions were audited
USERNAME	VARCHAR2 (30)	Name (not ID number) of the user whose actions were audited
USERHOST	VARCHAR2 (254)	Numeric instance ID for the Oracle instance from which the user is accessing the database. Used only in environments with distributed file systems and shared database files (e.g., clustered Oracle on DEC VAX/VMS clusters)
TERMINAL	VARCHAR2 (254)	Identifier for the user's terminal
TIMESTAMP	DATE	Timestamp for the creation of the audit trail entry (Timestamp for the user's logon for entries created by AUDIT SESSION)
OWNER	VARCHAR2 (30)	Creator of object affected by the action
OBJ_NAME	VARCHAR2 (128)	Name of the object affected by the action
ACTION_NAME	VARCHAR2 (27)	Name of the action type corresponding to the numeric code in ACTION
NEW_NAME	VARCHAR2 (128)	New name of object after RENAME, or name of underlying object (e.g. CREATE

		INDEX owner.obj_name ON new_owner.new_name)
OBJ_PRIVILEGE	VARCHAR2 (16)	Object privileges granted/revoked by a GRANT/REVOKE statement
SYS_PRIVILEGE	VARCHAR2 (40)	System privileges granted/revoked by a GRANT/REVOKE statement
ADMIN_OPTION	VARCHAR2 (1)	If role/sys_priv was granted WITH ADMIN OPTION, A/-
GRANTEE	VARCHAR2 (30)	The name of the grantee specified in a GRANT/REVOKE statement
AUDIT_OPTION	VARCHAR2 (40)	Auditing option set with the audit statement
SES_ACTIONS	VARCHAR2 (16)	Session summary. A string of 11 characters, one for each action type, in this order: Alter, Audit, Comment, Delete, Grant, Index, Insert, Lock, Rename, Select, Update. Values: "-" = None, "S" = Success, "F" = Failure, "B" = Both
COMMENT_TEXT	VARCHAR2 (254)	Text comment on the audit trail entry
SESSIONID	NUMBER	Numeric ID for each Oracle session
ENTRYID	NUMBER	Numeric ID for each audit trail entry in the session
STATEMENTID	NUMBER	Numeric ID for each statement run (a statement may cause many actions)
RETURNCODE	NUMBER	Oracle error code generated by the action. Zero if the action succeeded
PRIV_USED	VARCHAR2 (40)	System privilege used to execute the action
SESSION_LABEL	VARCHAR2 (1)	Trusted ORACLE label associated with user session
DBA_AUDIT_TRAIL	*	All audit trail entries
OS_USERNAME	VARCHAR2 (254)	Operating System logon user name of the user whose actions were audited
USERNAME	VARCHAR2 (30)	Name (not ID number) of the user whose actions were audited
USERHOST	VARCHAR2 (254)	Numeric instance ID for the Oracle instance from which the user is accessing the database. Used only in environments with distributed file systems and shared database files (e.g., clustered Oracle on DEC VAX/VMS clusters)
TERMINAL	VARCHAR2 (254)	Identifier for the user's terminal
TIMESTAMP	DATE	Timestamp for the creation of the audit trail entry (Timestamp for the user's logon for entries created by AUDIT SESSION)
OWNER	VARCHAR2 (30)	Creator of object affected by the action
OBJ_NAME	VARCHAR2 (128)	Name of the object affected by the action
ACTION	NUMBER	Numeric action type code. The corresponding name of the action type (CREATE TABLE, INSERT, etc.) is in the column ACTION_NAME
ACTION_NAME	VARCHAR2 (27)	Name of the action type corresponding to the numeric code in ACTION
NEW_OWNER	VARCHAR2 (30)	The owner of the object named in the

		NEW_NAME column
NEW_NAME	VARCHAR2 (128)	New name of object after RENAME, or name of underlying object (e.g. CREATE INDEX owner.obj_name ON new_owner.new_name)
OBJ_PRIVILEGE	VARCHAR2 (16)	Object privileges granted/revoked by a GRANT/REVOKE statement
SYS_PRIVILEGE	VARCHAR2 (40)	System privileges granted/revoked by a GRANT/REVOKE statement
ADMIN_OPTION	VARCHAR2 (1)	If role/sys_priv was granted WITH ADMIN OPTION, A/-
GRANTEE	VARCHAR2 (30)	The name of the grantee specified in a GRANT/REVOKE statement
AUDIT_OPTION	VARCHAR2 (40)	Auditing option set with the audit statement
SES_ACTIONS	VARCHAR2 (16)	Session summary. A string of 11 characters, one for each action type, in this order: Alter, Audit, Comment, Delete, Grant, Index, Insert, Lock, Rename, Select, Update. Values: "-" = None, "S" = Success, "F" = Failure, "B" = Both
LOGOFF_TIME	DATE	Timestamp for user logoff
LOGOFF_LREAD	NUMBER	Logical reads for the session
LOGOFF_PREAD	NUMBER	Physical reads for the session
LOGOFF_LWRITE	NUMBER	Logical writes for the session
LOGOFF_DLOCK	VARCHAR2 (40)	Deadlocks detected during the session
COMMENT_TEXT	VARCHAR2 (254)	Text comment on the audit trail entry
SESSIONID	NUMBER	Numeric ID for each Oracle session
ENTRYID	NUMBER	Numeric ID for each audit trail entry in the session
STATEMENTID	NUMBER	Numeric ID for each statement run (a statement may cause many actions)
RETURNCODE	NUMBER	Oracle error code generated by the action. Zero if the action succeeded
PRIV_USED	VARCHAR2 (40)	System privilege used to execute the action
OBJECT_LABEL	VARCHAR2 (1)	Optional Trusted ORACLE label associated with object being audited
SESSION_LABEL	VARCHAR2 (1)	Trusted ORACLE label associated with user session

DBA_BLOCKERS	*	All sessions that have someone waiting on a lock they hold that are not themselves waiting on a lock
SESSION_ID	NUMBER	Session holding a lock

DBA_CATALOG		All database Tables, Views, Synonyms, Sequences

The structure of the system table dba_catalog and the meaning of its columns correspond to those of the system table all_catalog.

DBA_CLUSTERS	*	Description of all clusters in the database
OWNER	VARCHAR2 (30)	Owner of the cluster
CLUSTER_NAME	VARCHAR2 (30)	Name of the cluster
TABLESPACE_NAME	VARCHAR2 (30)	Name of the tablespace containing the cluster
PCT_FREE	NUMBER	Minimum percentage of free space in a block
PCT_USED	NUMBER	Minimum percentage of used space in a block
KEY_SIZE	NUMBER	Estimated size of cluster key plus associated rows
INI_TRANS	NUMBER	Initial number of transactions
MAX_TRANS	NUMBER	Maximum number of transactions
INITIAL_EXTENT	NUMBER	Size of the initial extent in bytes
NEXT_EXTENT	NUMBER	Size of secondary extents in bytes
MIN_EXTENTS	NUMBER	Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER	Maximum number of extents allowed in the segment
PCT_INCREASE	NUMBER	Percentage increase in extent size
AVG_BLOCKS_PER_KEY	NUMBER	Average number of blocks containing rows with a given cluster key
CLUSTER_TYPE	VARCHAR2 (5)	Type of cluster: b-tree index or hash
FUNCTION	VARCHAR2 (7)	If a hash cluster, the hash function
HASHKEYS	NUMBER	If a hash cluster, the number of hash keys (hash buckets)

DBA_CLU_COLUMNS	*	Mapping of table columns to cluster columns
OWNER	VARCHAR2 (30)	Owner of the cluster
CLUSTER_NAME	VARCHAR2 (30)	Cluster name
CLU_COLUMN_NAME	VARCHAR2 (30)	Key column in the cluster
TABLE_NAME	VARCHAR2 (30)	Clustered table name
TAB_COLUMN_NAME	VARCHAR2 (30)	Key column in the table

DBA_COL_COMMENTS		Comments on columns of all tables and views

The structure of the system table dba_col_comments and the meaning of its columns correspond to those of the system table all_col_comments.

DBA_COL_PRIVS		All grants on columns in the database
GRANTEE	VARCHAR2 (30)	Name of the user to whom access was granted
OWNER	VARCHAR2 (30)	Username of the owner of the object
TABLE_NAME	VARCHAR2 (30)	Name of the object
COLUMN_NAME	VARCHAR2 (30)	Name of the column
GRANTOR	VARCHAR2 (30)	Name of the user who performed the grant
PRIVILEGE	VARCHAR2 (30)	Column Privilege
GRANTABLE	VARCHAR2 (3)	Privilege is grantable

DBA_CONSTRAINTS	Constraint definitions on all tables

The structure of the system table dba_constraints and the meaning of its columns correspond to those of the system table all_constraints.

DBA_CONS_COLUMNS	Information about accessible columns in constraint definitions

The structure of the system table dba_cons_columns and the meaning of its columns correspond to those of the system table all_cons_columns.

DBA_DATA_FILES	*	Information about database files
FILE_NAME	VARCHAR2 (72)	Name of the database file
FILE_ID	NUMBER	ID of the database file
TABLESPACE_NAME	VARCHAR2 (30)	Name of the tablespace to which the file belongs
BYTES	NUMBER	Size of the file in bytes
BLOCKS	NUMBER	Size of the file in ORACLE blocks
STATUS	VARCHAR2 (9)	File status: "INVALID" or "AVAILABLE"

DBA_DB_LINKS	*	All database links in the database
OWNER	VARCHAR2 (30)	Owner of the database link
DB_LINK	VARCHAR2 (30)	Name of the database link
USERNAME	VARCHAR2 (30)	Name of user to log on as
PASSWORD	VARCHAR2 (30)	Password for logon
HOST	VARCHAR2 (254)	SQL*Net string for connect
CREATED	DATE	Creation time of the database link

DBA_DDL_LOCKS	*	All DDL locks held in the database and all outstanding requests for a DDL lock
SESSION_ID	NUMBER	Session identifier
OWNER	VARCHAR2 (30)	Owner of the lock
NAME	VARCHAR2 (1)	Name of the lock
TYPE	VARCHAR2 (1)	Lock type: Cursor, Table/Procedure, Body, Trigger, Index, Cluster
MODE_HELD	VARCHAR2 (10)	Lock mode: None, Null, Share, Exclusive
MODE_REQUESTED	VARCHAR2 (10)	Lock request type: None, Null, Share, Exclusive

DBA_DEPENDENCIES	Dependencies to and from objects

The structure of the system table dba_dependencies and the meaning of its columns correspond to those of the system table all_dependencies.

DBA_DML_LOCKS	*	All DML locks held in the database and all outstanding requests for a DML lock
SESSION_ID	NUMBER	Session holding or aquiring the lock
OWNER	VARCHAR2 (30)	Owner of the lock
NAME	VARCHAR2 (1)	Name of the lock
MODE_HELD	VARCHAR2 (10)	Lock mode

MODE_REQUESTED	VARCHAR2 (10)	Lock request type: None, Null, Share, Exclusive
DBA_ERRORS	*	Current errors on all stored objects in the database

The structure of the system table dba_errors and the meaning of its columns correspond to those of the system table all_errors.

DBA_EXP_FILES	*	Description of export files
EXP_VERSION	NUMBER	Version number of the export session
EXP_TYPE	VARCHAR2 (11)	Type of export file (full, cumulative, or incremental)
FILE_NAME	VARCHAR2 (72)	Name of the export file
USER_NAME	VARCHAR2 (30)	Name of user who executed export
TIMESTAMP	DATE	Timestamp of the export session
DBA_EXP_OBJECTS	*	Objects that have been incrementally exported
OWNER	VARCHAR2 (30)	Owner of exported object
OBJECT_NAME	VARCHAR2 (30)	Name of exported object
OBJECT_TYPE	VARCHAR2 (12)	Type of exported object
CUMULATIVE	DATE	Timestamp of last cumulative export
INCREMENTAL	DATE	Timestamp of last incremental export
EXPORT_VERSION	NUMBER	The id of the export session
DBA_EXP_VERSION	*	Version number of the last export session
EXP_VERSION	NUMBER	Version number of the last export session
DBA_EXTENTS	*	Extents comprising all segments in the database
OWNER	VARCHAR2 (30)	Owner of the segment associated with the extent
SEGMENT_NAME	VARCHAR2 (72)	Name of the segment associated with the extent
SEGMENT_TYPE	VARCHAR2 (17)	Type of the segment
TABLESPACE_NAME	VARCHAR2 (30)	Name of the tablespace containing the extent
EXTENT_ID	NUMBER	Extent number in the segment
FILE_ID	NUMBER	Name of the file containing the extent
BLOCK_ID	NUMBER	Starting block number of the extent
BYTES	NUMBER	Size of the extent in bytes
BLOCKS	NUMBER	Size of the extent in ORACLE blocks
DBA_FREE_SPACE	*	Free extents in all tablespaces
TABLESPACE_NAME	VARCHAR2 (30)	Name of the tablespace containing the extent
FILE_ID	NUMBER	ID number of the file containing the extent
BLOCK_ID	NUMBER	Starting block number of the extent

BYTES	NUMBER	Size of the extent in bytes
BLOCKS	NUMBER	Size of the extent in ORACLE blocks

DBA_INDEXES		Description for all indexes in the database

The structure of the system table dba_indexes and the meaning of its columns correspond to those of the system table all_indexes.

DBA_IND_COLUMNS		COLUMNs comprising INDEXes on all TABLEs and CLUSTERs

The structure of the system table dba_ind_columns and the meaning of its columns correspond to those of the system table all_ind_columns.

DBA_LOCKS	*	All locks or latch held in the database, and all outstanding requests for a lock or latch. This view includes DML locks and DDL locks
SESSION_ID	NUMBER	Session holding or aquiring the lock
TYPE	VARCHAR2 (2)	Lock type
MODE_HELD	VARCHAR2 (4)	Lock mode
MODE_REQUESTED	VARCHAR2 (4)	Lock request type
LOCK_ID1	NUMBER	Type-specific lock identifier, part 1
LOCK_ID2	NUMBER	Type-specific lock identifier, part 2

DBA_OBJECTS		All objects in the database

The structure of the system table dba_objects and the meaning of its columns correspond to those of the system table all_objects.

DBA_OBJECT_SIZE	*	Sizes, in bytes, of various pl/sql objects
OWNER	VARCHAR2 (30)	Owner of the object
NAME	VARCHAR2 (30)	Name of the object
TYPE	VARCHAR2 (12)	Type of the object: "TABLE", "VIEW", "SYNONYM", "SEQUENCE", "PROCEDURE", "FUNCTION", "PACKAGE" or "PACKAGE BODY"
SOURCE_SIZE	NUMBER	Size of the source, in bytes. Must be in memory during compilation, or dynamic recompilation
PARSED_SIZE	NUMBER	Size of the parsed form of the object, in bytes. Must be in memory when an object is being compiled that references this object
CODE_SIZE	NUMBER	Code size, in bytes. Must be in memory when this object is executing
ERROR_SIZE	NUMBER	Size of error messages, in bytes. In memory during the compilation of the object when there are compilation errors

DBA_OBJ_AUDIT_OPTS		Auditing options for all tables and views
OWNER	VARCHAR2 (30)	Owner of the object
OBJECT_NAME	VARCHAR2 (30)	Name of the object
OBJECT_TYPE	VARCHAR2 (11)	Type of the object
ALT	VARCHAR2 (3)	Auditing ALTER WHENEVER SUCCESSFUL / UNSUCCESSFUL
AUD	VARCHAR2 (3)	Auditing AUDIT WHENEVER SUCCESSFUL / UNSUCCESSFUL
COM	VARCHAR2 (3)	Auditing COMMENT WHENEVER SUCCESSFUL / UNSUCCESSFUL
DEL	VARCHAR2 (3)	Auditing DELETE WHENEVER SUCCESSFUL / UNSUCCESSFUL
GRA	VARCHAR2 (3)	Auditing GRANT WHENEVER SUCCESSFUL / UNSUCCESSFUL
IND	VARCHAR2 (3)	Auditing INDEX WHENEVER SUCCESSFUL / UNSUCCESSFUL
INS	VARCHAR2 (3)	Auditing INSERT WHENEVER SUCCESSFUL / UNSUCCESSFUL
LOC	VARCHAR2 (3)	Auditing LOCK WHENEVER SUCCESSFUL / UNSUCCESSFUL
REN	VARCHAR2 (3)	Auditing RENAME WHENEVER SUCCESSFUL / UNSUCCESSFUL
SEL	VARCHAR2 (3)	Auditing SELECT WHENEVER SUCCESSFUL / UNSUCCESSFUL
UPD	VARCHAR2 (3)	Auditing UPDATE WHENEVER SUCCESSFUL / UNSUCCESSFUL
REF	VARCHAR2 (3)	Auditing REFERENCES WHENEVER SUCCESSFUL / UNSUCCESSFUL (not used)
EXE	VARCHAR2 (3)	Auditing EXECUTE WHENEVER SUCCESSFUL / UNSUCCESSFUL
DBA_PRIV_AUDIT_OPTS	*	Describes current system privileges being audited across the system and by user
USER_NAME	VARCHAR2 (30)	User name if by user auditing, else null for system wide auditing
PRIVILEGE	VARCHAR2 (40)	Name of the system privilege being audited
SUCCESS	VARCHAR2 (10)	Mode for WHENEVER SUCCESSFUL system auditing
FAILURE	VARCHAR2 (10)	Mode for WHENEVER NOT SUCCESSFUL system auditing
DBA_PROFILES	*	Display all profiles and their limits
PROFILE	VARCHAR2 (30)	Profile name
RESOURCE_NAME	VARCHAR2 (32)	Resource name
LIMIT	VARCHAR2 (40)	Limit placed on this resource for this profile
DBA_ROLES	*	All Roles which exist in the database
ROLE	VARCHAR2 (30)	Role Name

PASSWORD_REQUIRED	VARCHAR2 (8)	Indicates if the role requires a password to be enabled
DBA_ROLE_PRIVS	*	Roles granted to users and roles
GRANTEE	VARCHAR2 (30)	Grantee Name, User or Role receiving the grant
GRANTED_ROLE	VARCHAR2 (30)	Granted role name
ADMIN_OPTION	VARCHAR2 (3)	Grant was with the ADMIN option
DEFAULT_ROLE	VARCHAR2 (3)	Role is designated as a DEFAULT ROLE for the user
DBA_ROLLBACK_SEGS	*	Description of rollback segments
SEGMENT_NAME	VARCHAR2 (30)	Name of the rollback segment
OWNER	VARCHAR2 (30)	Owner of the rollback segment
TABLESPACE_NAME	VARCHAR2 (30)	Name of the tablespace containing the rollback segment
SEGMENT_ID	NUMBER	ID number of the rollback segment
FILE_ID	NUMBER	ID number of the file containing the segment header
BLOCK_ID	NUMBER	ID number of the block containing the segment header
INITIAL_EXTENT	NUMBER	Initial extent size in bytes
NEXT_EXTENT	NUMBER	Secondary extent size in bytes
MIN_EXTENTS	NUMBER	Minimum number of extents
MAX_EXTENTS	NUMBER	Maximum number of extents
PCT_INCREASE	NUMBER	Percent increase for extent size
STATUS	VARCHAR2 (1)	Rollback segment status
INSTANCE_NUM	NUMBER	Rollback segment owning parallel server instance number
DBA_SEGMENTS		Storage allocated for all database segments
OWNER	VARCHAR2 (30)	Username of the segment owner
SEGMENT_NAME	VARCHAR2 (81)	Name, if any, of the segment
SEGMENT_TYPE	VARCHAR2 (17)	Type of segment: "TABLE", "CLUSTER", "INDEX", "ROLLBACK", "DEFERRED ROLLBACK", "TEMPORARY", or "CACHE"
TABLESPACE_NAME	VARCHAR2 (30)	Name of the tablespace containing the segment
HEADER_FILE	NUMBER	* ID of the file containing the segment header
HEADER_BLOCK	NUMBER	* ID of the block containing the segment header
BYTES	NUMBER	* Size, in bytes, of the segment
BLOCKS	NUMBER	* Size, in Oracle blocks, of the segment
EXTENTS	NUMBER	* Number of extents allocated to the segment
INITIAL_EXTENT	NUMBER	* Size, in Oracle blocks, of the initial extent of the segment
NEXT_EXTENT	NUMBER	* Size, in Oracle blocks, of the next extent to be allocated to the segment
MIN_EXTENTS	NUMBER	* Minimum number of extents allowed in

		the segment
MAX_EXTENTS	NUMBER	* Maximum number of extents allowed in the segment
PCT_INCREASE	NUMBER	* Percent by which to increase the size of the next extent to be allocated
FREELISTS	NUMBER	* Number of process freelists allocated in this segment
FREELIST_GROUPS	NUMBER	* Number of freelist groups allocated in this segment

DBA_SEQUENCES		Description of all SEQUENCEs in the database

The structure of the system table dba_seqences and the meaning of its columns correspond to those of the system table all_sequences.

DBA_SNAPSHOTS		All snapshots in the database

The structure of the system table dba_snapshots and the meaning of its columns correspond to those of the system table all_snapshots.

DBA_SNAPSHOT_LOGS		All snapshot logs in the database
LOG_OWNER	VARCHAR2 (30)	Owner of the snapshot log
MASTER	VARCHAR2 (30)	Name of the master table which the log logs changes of
LOG_TABLE	VARCHAR2 (30)	Log table that holds the rowids and timestamps of rows that changed in the master table
LOG_TRIGGER	VARCHAR2 (30)	* An after-row trigger on the master which inserts rows into the log
CURRENT_SNAPSHOTS	DATE	* One date per snapshot -- the date the snapshot of the master last refreshed

DBA_SOURCE	*	Source of all stored objects in the database

The structure of the system table dba_source and the meaning of its columns correspond to those of the system table all_source.

DBA_STMT_AUDIT_OPTS	*	Describes current system auditing options across the system and by user
USER_NAME	VARCHAR2 (30)	User name if by user auditing, else null for system wide auditing
AUDIT_OPTION	VARCHAR2 (3)	Name of the system auditing option
SUCCESS	VARCHAR2 (3)	Mode for WHENEVER SUCCESSFUL system auditing
FAILURE	VARCHAR2 (3)	Mode for WHENEVER NOT SUCCESSFUL system auditing

DBA_SYNONYMS		All synonyms in the database

The structure of the system table dba_synonyms and the meaning of its columns correspond to those of the system table all_synonyms.

DBA_SYS_PRIVS	*	System privileges granted to users and roles
GRANTEE	VARCHAR2 (30)	Grantee Name, User or Role receiving the grant
PRIVILEGE	VARCHAR2 (40)	System privilege
ADMIN_OPTION	VARCHAR2 (3)	Grant was with the ADMIN option

DBA_TABLES		Description of all tables in the database

The structure of the system table dba_tables and the meaning of its columns correspond to those of the system table all_tables.

DBA_TABLESPACES	*	Description of all tablespaces
TABLESPACE_NAME	VARCHAR2 (30)	Tablespace name
INITIAL_EXTENT	NUMBER	Default initial extent size
NEXT_EXTENT	NUMBER	Default incremental extent size
MIN_EXTENTS	NUMBER	Default minimum number of extents
MAX_EXTENTS	NUMBER	Default maximum number of extents
PCT_INCREASE	NUMBER	Default percent increase for extent size
STATUS	VARCHAR2 (9)	Tablespace status: "ONLINE" or "OFFLINE"

DBA_TAB_COLUMNS		Columns of all tables, views and clusters

The structure of the system table dba_tab_columns and the meaning of its columns correspond to those of the system table all_tab_columns.

DBA_TAB_COMMENTS		Comments on all tables and views in the database

The structure of the system table dba_tab_comments and the meaning of its columns correspond to those of the system table all_tab_comments.

DBA_TAB_PRIVS		All grants on objects in the database
GRANTEE	VARCHAR2 (30)	User to whom access was granted
OWNER	VARCHAR2 (30)	Owner of the object
TABLE_NAME	VARCHAR2 (30)	Name of the object
GRANTOR	VARCHAR2 (30)	Name of the user who performed the grant
PRIVILEGE	VARCHAR2 (30)	Table Privilege
GRANTABLE	VARCHAR2 (3)	Privilege is grantable

DBA_TRIGGERS		All triggers in the database

The structure of the system table dba_triggers and the meaning of its columns correspond to those of the system table all_triggers.

DBA_TRIGGER_COLS	*	Column usage in all triggers

The structure of the system table dba_trigger_cols and the meaning of its columns correspond to those of the system table all_trigger_cols.

DBA_TS_QUOTAS	*	Tablespace quotas for all users
TABLESPACE_NAME	VARCHAR2 (30)	Tablespace name
USERNAME	VARCHAR2 (30)	User with resource rights on the tablespace
BYTES	NUMBER	Number of bytes charged to the user
MAX_BYTES	NUMBER	User's quota in bytes. NULL if no limit
BLOCKS	NUMBER	Number of ORACLE blocks charged to the user
MAX_BLOCKS	NUMBER	User's quota in ORACLE blocks. NULL if no limit

DBA_USERS		Information about all users of the database
USERNAME	VARCHAR2 (30)	Name of the user
USER_ID	NUMBER	ID number of the user
PASSWORD	VARCHAR2 (30)	Encrypted password
DEFAULT_TABLESPACE	VARCHAR2 (30)	Default tablespace for data
TEMPORARY_TABLESPACE	VARCHAR2 (30)	Default tablespace for temporary tables
CREATED	DATE	User creation date
PROFILE	VARCHAR2 (30)	* User resource profile name

DBA_VIEWS		Text of all views in the database
OWNER	VARCHAR2 (30)	Owner of the view
VIEW_NAME	VARCHAR2 (30)	Name of the view
TEXT_LENGTH	NUMBER	Length of the view text
TEXT	LONG	View text

DBA_WAITERS	*	All locks or latch held in the database, and all outstanding requests for a lock or latch. This view includes DML locks and DDL locks
WAITING_SESSION	NUMBER	Session waiting for a lock
HOLDING_SESSION	NUMBER	Session holding a lock
TYPE	VARCHAR2 (2)	Lock type
MODE_HELD	VARCHAR2 (9)	Lock mode: Row-S (SS), Row-X (SX), Share, S/Row-X(SSX), Exclusive
MODE_REQUESTED	VARCHAR2 (9)	Lock request type: Null, Row-S (SS), Row-X (SX), Share, S/Row-X(SSX), Exclusive
LOCK_ID1	NUMBER	Type-specific lock identifier, part 1
LOCK_ID2	NUMBER	Type-specific lock identifier, part 2

DICTIONARY		Description of data dictionary tables and

		views
TABLE_NAME	VARCHAR2 (30)	Name of the object
COMMENTS	VARCHAR2 (254)	Text comment on the object
DICT_COLUMNS		Description of columns in data dictionary tables and views
TABLE_NAME	VARCHAR2 (30)	Name of the object that contains the column
COLUMN_NAME	VARCHAR2 (30)	Name of the column
COMMENTS	VARCHAR2 (254)	Text comment on the object
LOBAL_NAME		Global database name
GLOBAL_NAME	VARCHAR2 (83)	Global name of the database
PUBLIC_DEPENDENCY	*	Dependencies to and from objects, by object number
OBJECT_ID	RAW (8)	Object number
REFERENCED_OBJECT_ID	RAW (8)	Referenced object (the parent object)
RESOURCE_COST	*	Cost for each resource
RESOURCE_NAME	VARCHAR2 (32)	Name of the resource
UNIT_COST	NUMBER	Cost of the resource
ROLE_ROLE_PRIVS	*	Roles which are granted to roles
ROLE	VARCHAR2 (30)	Name of the role
GRANTED_ROLE	VARCHAR2 (30)	Role that was granted
ADMIN_OPTION	VARCHAR2 (3)	Signifies that the role was granted with ADMIN option
ROLE_SYS_PRIVS	*	System privileges granted to roles
ROLE	VARCHAR2 (30)	Name of the role
PRIVILEGE	VARCHAR2 (40)	System privilege granted to the role
ADMIN_OPTION	VARCHAR2 (3)	Signifies the grant was with the ADMIN option
ROLE_TAB_PRIVS	*	Table privileges granted to roles
ROLE	VARCHAR2 (30)	Name of the role
OWNER	VARCHAR2 (30)	Owner of the object
TABLE_NAME	VARCHAR2 (30)	Name of the object
COLUMN_NAME	VARCHAR2 (30)	Name of the column, if applicable
PRIVILEGE	VARCHAR2 (30)	Object privilege granted to the role
GRANTABLE	VARCHAR2 (3)	YES if the role was granted with ADMIN OPTION, otherwise NO
SESSION_PRIVS	*	Privileges which the user currently has set
PRIVILEGE	VARCHAR2 (40)	Name of the privilege

SESSION_ROLES	*	Roles which the user currently has enabled
ROLE	VARCHAR2 (30)	Name of the role

TABLE_PRIVILEGES		Grants on objects for which the user is the grantor, grantee, owner, or an enabled role or PUBLIC is the grantee
GRANTEE	VARCHAR2 (30)	Name of the user to whom access was granted
OWNER	VARCHAR2 (30)	Username of the object's owner
TABLE_NAME	VARCHAR2 (30)	Name of the object
GRANTOR	VARCHAR2 (30)	Name of the user who performed the grant
SELECT_PRIV	VARCHAR2 (1)	Permission to select from the object
INSERT_PRIV	VARCHAR2 (1)	Permission to insert into the column
DELETE_PRIV	VARCHAR2 (1)	Permission to update the object
UPDATE_PRIV	VARCHAR2 (1)	Permission to update the column
REFERENCES_PRIV	VARCHAR2 (1)	Permission to reference the column
ALTER_PRIV	VARCHAR2 (1)	Permission to alter the object
INDEX_PRIV	VARCHAR2 (1)	Permission to create or drop an index on the object
CREATED	DATE	Timestamp for the grant

USER_AUDIT_OBJECT	*	Audit trail records for statements concerning objects, specifically: table, cluster, view, index, sequence, [public] database link, [public] synonym, procedure, trigger, rollback segment, tablespace, role, user

The structure of the system table user_audit_object and the meaning of its columns correspond to those of the system table DBA_audit_object.

USER_AUDIT_SESSION	*	All audit trail entries concerning connect and disconnect

The structure of the system table user_audit_session and the meaning of its columns correspond to those of the system table DBA_audit_session.

USER_AUDIT_STATEMENT		Audit trail records concerning grant, revoke, audit, noaudit and alter system

The structure of the system table user_audit_statement and the meaning of its columns correspond to those of the system table DBA_audit_statement.

USER_AUDIT_TRAIL	*	Audit trail entries relevant to the user

The structure of the system table user_audit_trail and the meaning of its columns correspond to those of the system table DBA_audit_trail.

USER_CATALOG		Tables, Views, Synonyms and Sequences owned by the user

The structure of the system table user_catalog and the meaning of its columns correspond to those of the system table ALL_catalog. In contrast to all_catalog, user_catalog does not contain the column OWNER.

USER_CLUSTERS	*	Descriptions of user's own clusters

The structure of the system table user_clusters and the meaning of its columns correspond to those of the system table DBA_clusters. In contrast to DBA_clusters, user_clusters does not contain the column OWNER.

USER_CLU_COLUMNS	*	Mapping of table columns to cluster columns

The structure of the system table user_clu_columns and the meaning of its columns correspond to those of the system table DBA_clu_columns. In contrast to DBA_clu_columns, user_clu_columns does not contain the column owner.

USER_COL_COMMENTS		Comments on columns of user's tables and views

The structure of the system table user_col_comments and the meaning of its columns correspond to those of the system table all_col_comments. In contrast to all_col_comments, user_col_comments does not contain the column OWNer.

USER_COL_PRIVS		Grants on columns for which the user is the owner, grantor or grantee

The structure of the system table user_col_privs and the meaning of its columns correspond to those of the system table DBA_col_privs.

USER_COL_PRIVS_MADE		All grants on columns of objects owned by the user

The structure of the system table user_col_privs_made and the meaning of its columns correspond to those of the system table all_col_privs_made. In contrast to all_col_privs_made, user_col_privs_made does not contain the column owner.

USER_COL_PRIVS_RECD		Grants on columns for which the user is the grantee

The structure of the system table user_col_privs_recd and the meaning of its columns correspond to those of the system table all_col_privs_recd. In contrast to All-col_privs_recd, user_col_privs_recd does not contain the column grantee.

USER_CONSTRAINTS		Constraint definitions on user's own

		tables

The structure of the system table user_constraints and the meaning of its columns correspond to those of the system table all_constraints.

USER_CONS_COLUMNS		Information about accessible columns in constraint definitions

The structure of the system table user_cons_columns and the meaning of its columns correspond to those of the system table all_cons_columns.

USER_DB_LINKS	*	Database links owned by the user

The structure of the system table user_db_links and the meaning of its columns correspond to those of the system table DBA_db_links.

USER_DEPENDENCIES		Dependencies to and from a users objects

The structure of the system table user_dependencies and the meaning of its columns correspond to those of the system table all_dependencies. In contrast to all_dependencies, user_dependencies does not contain the column owner.

USER_ERRORS	*	Current errors on stored objects owned by the user

The structure of the system table user_errors and the meaning of its columns correspond to those of the system table all_errors. In contrast to all_errors, user_errors does not contain the column owner.

USER_EXTENTS	*	Extents comprising segments owned by the user
SEGMENT_NAME	VARCHAR2 (81)	Name of the segment associated with the extent
SEGMENT_TYPE	VARCHAR2 (17)	Type of the segment
TABLESPACE_NAME	VARCHAR2 (30)	Name of the tablespace containing the extent
EXTENT_ID	NUMBER	Extent number in the segment
BYTES	NUMBER	Size of the extent in bytes
BLOCKS	NUMBER	Size of the extent in ORACLE blocks

USER_FREE_SPACE	*	Free extents in tablespaces accessible to the user
TABLESPACE_NAME	VARCHAR2 (30)	Name of the tablespace containing the extent
FILE_ID	NUMBER	ID number of the file containing the extent
BLOCK_ID	NUMBER	Starting block number of the extent
BYTES	NUMBER	Size of the extent in bytes
BLOCKS	NUMBER	Size of the extent in ORACLE blocks

USER_INDEXES		Description of the user's own indexes

The structure of the system table user_indexes and the meaning of its columns correspond to those of the system table all_indexes. In contrast to ALL_INDEXES, USER_INDEXES does not contain the column OWNER.

USER_IND_COLUMNS		COLUMNs comprising user's INDEXes or on user's TABLES

The structure of the system table user_ind_columns and the meaning of its columns correspond to those of the system table all_ind_columns. In contrast to all_ind_columns, user_ind_columns does not contain the column index_owner.

USER_OBJECTS		Objects owned by the user

The structure of the system table user_objects and the meaning of its columns correspond to those of the system table all_objects. In contrast to all_objects, user_objects does not contain the column owner.

USER_OBJECT_SIZE	*	Sizes, in bytes, of various pl/sql objects

The structure of the system table user_objects_size and the meaning of its columns correspond to those of the system table DBA_objects_size. In contrast to DBA_object_size, user_object_size does not contain the column owner.

USER_OBJ_AUDIT_OPTS		Auditing options for user's own tables and views

The structure of the system table user_obj_audit_opts and the meaning of its columns correspond to those of the system table DBA_Obj_audit_opts. In contrast to DBA_obj_audit_opts, user_obj_audit_opts does not contain the column owner.

USER_RESOURCE_LIMITS	*	Display resource limit of the user
RESOURCE_NAME	VARCHAR2 (32)	Resource name
LIMIT	VARCHAR2 (40)	Limit placed on this resource

USER_ROLE_PRIVS	*	Roles granted to current user
USERNAME	VARCHAR2 (30)	User Name or PUBLIC
GRANTED_ROLE	VARCHAR2 (30)	Granted role name
ADMIN_OPTION	VARCHAR2 (3)	Grant was with the ADMIN option
DEFAULT_ROLE	VARCHAR2 (3)	Role is designated as a DEFAULT ROLE for the user
OS_GRANTED	VARCHAR2 (3)	Role is granted via the operating system (using OS_ROLES = TRUE)

USER_SEGMENTS		Storage allocated for all database segments

The structure of the system table user_segments and the meaning of its columns correspond to those of the system table DBA_segments. In contrast to DBA_segments, user_segments does not contain the columns owner, header_file and header_block.

USER_SEQUENCES		Description of the user's own SEQUENCEs

The structure of the system table user_sequences and the meaning of its columns correspond to those of the system table all_sequences. In contrast to all_sequences, user_sequences does not contain the column owner.

USER_SNAPSHOTS		Snapshots the user can look at

The structure of the system table user_snapshots and the meaning of its columns correspond to those of the system table all_snapshots.

USER_SNAPSHOT_LOGS		All snapshot logs owned by the user

The structure of the system table user_snapshot_logs and the meaning of its columns correspond to those of the system table DBA_snapshot_logs.

USER_SOURCE	*	Source of stored objects accessible to the user

The structure of the system table user_source and the meaning of its columns correspond to those of the system table all_source. In contrast to all_source, user_source does not contain the column owner.

USER_SYNONYMS		The user's private synonyms

The structure of the system table user_synonyms and the meaning of its columns correspond to those of the system table all_synonyms. In contrast to all_synonyms, user_synonyms does not contain the column owner.

USER_SYS_PRIVS	*	System privileges granted to current user
USERNAME	VARCHAR2 (30)	User Name or PUBLIC
PRIVILEGE	VARCHAR2 (40)	System privilege
ADMIN_OPTION	VARCHAR2 (3)	Grant was with the ADMIN option

USER_TABLES		Description of the user's own tables

The structure of the system table user_tables and the meaning of its columns correspond to those of the system table all_tables. In contrast to all_tables, user_tables does not contain the column owner.

USER_TABLESPACES	*	Description of accessible tablespaces

The structure of the system table user_tablespaces and the meaning of its columns correspond to those of the system table DBA_tablespaces.

USER_TAB_COLUMNS		Columns of user's tables, views and clusters

The structure of the system table user_tab_columns and the meaning of its columns correspond to those of the system table all_tab_columns. In contrast to all_tab_columns, user_tab_columns does not contain the column owner.

USER_TAB_COMMENTS		Comments on the tables and views owned by the user

The structure of the system table user_tab_comments and the meaning of its columns correspond to those of the system table all_tab_comments. In contrast to all_tab_comments, user_tab_comments does not contain the column owner.

USER_TAB_PRIVS		Grants on objects for which the user is the owner, grantor or grantee

The structure of the system table user_tab_privs and the meaning of its columns correspond to those of the system table DBA_tab_privs.

USER_TAB_PRIVS_MADE		All grants on objects owned by the user
GRANTEE	VARCHAR2 (30)	Name of the user to whom access was granted
TABLE_NAME	VARCHAR2 (30)	Name of the object
GRANTOR	VARCHAR2 (30)	Name of the user who performed the grant
PRIVILEGE	VARCHAR2 (30)	Table Privilege
GRANTABLE	VARCHAR2 (3)	Privilege is grantable

USER_TAB_PRIVS_RECD		Grants on objects for which the user is the grantee
OWNER	VARCHAR2 (30)	Owner of the object
TABLE_NAME	VARCHAR2 (30)	Name of the object
GRANTOR	VARCHAR2 (30)	Name of the user who performed the grant
PRIVILEGE	VARCHAR2 (30)	Table Privilege
GRANTABLE	VARCHAR2 (3)	Privilege is grantable

USER_TRIGGERS		Triggers owned by the user

The structure of the system table user_triggers and the meaning of its columns correspond to those of the system table all_triggers. In contrast to all_triggers, user_triggers does not contain the column owner.

USER_TRIGGER_COLS	*	Column usage in user's triggers

The structure of the system table user_trigger_cols and the meaning of its columns correspond to those of the system table all_trigger_cols.

USER_TS_QUOTAS	*	Tablespace quotas for the user

The structure of the system table user_ts_quotas and the meaning of its columns correspond to those of the system table DBA_ts_quotas. In contrast to DBA_ts_quotas, user_ts_quotas does not contain the column username.

USER_USERS		Information about the current user

The structure of the system table user_users and the meaning of its columns correspond to those of the system table DBA_users. In contrast to DBA_users, user_users does not contain the columns Password and profile.

USER_VIEWS		Text of views owned by the user
VIEW_NAME	VARCHAR2 (30)	Name of the view
TEXT_LENGTH	NUMBER	Length of the view text
TEXT	LONG	View text

Restrictions

Maximum values :

Number of tables	unlimited
Length of an identifier	18 characters
Internal length of a table row	4047 characters
Length of a LONG column	2147483647 characters
Columns per table (with KEY)	255 columns
Columns per table (without KEY)	254 columns
Number of key columns	127 columns
Precision of numeric values	18 digits
Sum of internal lengths of all key columns	255 characters
Sum of internal lengths of all columns belonging to an index	255 characters
Length of sort columns in SELECT	250 characters
Number of result columns	254 columns
Number of join tables in SELECT	16 tables
Number of join conditions in a WHERE clause of a SELECT	64
Number of named indexes per table	256
Number of correlated columns in an SQL statement	64
Number of correlated tables in an SQL statement	16
Sequences per SERVERDB	8
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

Introduction

1. This chapter lists the differences between the syntax and semantics in ADABAS and the definition of ORACLE7.

Program-relevant Differences

1. If a <table name> is specified without an <owner> and if there is no private table of the user with the name <table name>, the behavior of ADABAS differs from that of ORACLE.

2. In ADABAS, numbers are not implicitly converted into date values or vice versa. Character strings are converted into numeric values, or vice versa, in application programs for comparisons or when variables are used. The arguments of functions are not implicitly converted.

3. When two character strings are compared in ADABAS, the shorter one is filled with blanks (code attribute ASCII or EBCDIC) or binary zeros (code attribute BYTE) up to the length of the longer character string. This is not done in ORACLE.

4. The range of values of numbers in ADABAS is between -9.99999999999999999E+62 and -1E-64 and between +1E-64 and +9.99999999999999999E+62. In ORACLE, the range comprises values between 0.999E-128 and 0.999E126.

5. ADABAS does not know a National Language Support; i.e., it does not know the corresponding functions nor the corresponding parameters in other functions.

6. The NULL value and the empty <string literal>, or <string literal>s only containing blanks are different in ADABAS.
7. In ADABAS, numbers cannot end with a 'K' or 'M'.

8. Date values in ORACLE are between 01-01-4712 BC and 12-31-4712 AC. In ADABAS, date values can be between 01-01-0001 and 12-31-9999.

9. The pseudo column LEVEL is not supported in ADABAS.

10. The functions:

    NLSSORT
    CHARTOROWID
    ROWIDTOCHAR
    CONVERT
    DUMP
    USERENV
    NLS_INITCAP
    NLS_LOWER
    NLS_UPPER
    TO_MULTI_BYTE
    TO_SINGLE_BYTE
    INSTRB
    LENGTHB
    SUBSTRB

    are not provided in ADABAS.

11. In ADABAS, the pseudo column ROWNUM cannot be used in <update columns and values> of the <update statement>.

12. In ADABAS, a correlated subquery is not allowed for a <subquery> in a <set update clause>.

13. A <set update clause> of the format <column name>,... = <subquery> is not available in ADABAS.

14. The pseudo column ROWID is not supported in ADABAS.

15. The pseudo column ROWNUM is entered into the <select column> after sorting the result table.

16. The specification of CONNECT BY <search condition>
[START WITH <search condition>] is not available in ADABAS.

17. The specifications for the <connect statement> to be made in addition to the user name and password differ in ADABAS and ORACLE.

18. The effects of the <lock spec> in the <lock statement> partially differ in ORACLE and ADABAS.

19. In contrast to ORACLE, read and write operations performed on a table in the ISOLATION LEVELs 1, 2, and 3 can collide in ADABAS.

20. In ADABAS, locks are not released by a <rollback to statement>.

21. FORCE or COMMENT cannot be specified with the <connect statement> or <rollback statement>.

Differences in the Database Management

1. In ADABAS, there are usergroups, so that it may happen that the <owner> of an object is not a user but a usergroup.

2. In ADABAS, a user can only create and delete database objects for himself or for his usergroup, and he can create database objects which are not assigned to a particular user (e.g., indexes). It is not possible to create database objects for other users.

3. In contrast to ORACLE, tables of a user are in a namespace. Sequences, procedures and functions build separate namespaces. Indexes and constraints in ADABAS must have unique names within a table.

4. In contrast to ORACLE, DBAs in ADABAS are also subject to the privileges concept and cannot execute any DDL statements (e.g., CREATE INDEX) or DML statements (e.g., DELETE) for which they have not received the corresponding privileges.

5. Numbers have only 18 significant digits in ADABAS, instead of 38.

6. While ORACLE distinguishes names of up to 30 characters in length, only 18 characters are significant in ADABAS.

7. The indication of a database link for a <table name> specification is neither possible nor required in ADABAS.

8. The following key words are not reserved in ADABAS:

   ACCESS
   COMPRESS
   ELSE
   FILE
   IF IMMEDIATE INCREMENT INITIAL
   MAXEXTENTS
   NOAUDIT NOCOMPRESS
   OFFLINE ONLINE
   PRIOR
   SESSION SIZE START SUCCESSFUL
   THEN
   VALIDATE

9. In contrast to ORACLE, a <fixed point literal> in ADABAS can begin or end with a '.'

10. In ADABAS, LONG columns can be specified in the <null predicate>.

11. The following <sql statement>s are not available in ADABAS:

    ALTER / CREATE / DROP CLUSTER

    CREATE CONTROLFILE

    ALTER / CREATE DATABASE

    CREATE / DROP DATABASE LINK

ALTER / CREATE / DROP FUNCTION

ALTER INDEX
VALIDATE INDEX

ALTER / CREATE / DROP PACKAGE

ALTER / CREATE / DROP PROCEDURE

ALTER / CREATE / DROP PROFILE

ALTER RESOURCE COST

ALTER / CREATE / DROP / SET ROLE

ALTER / DROP ROLLBACK SEGMENT

ALTER SEQUENCE

ALTER SESSION

ALTER SNAPSHOT

ALTER SNAPSHOT LOG

ALTER SYSTEM

ALTER / DROP TABLESPACE

ALTER / CREATE / DROP TRIGGER

ALTER USER

ALTER VIEW

ANALYZE

AUDIT
NOAUDIT

EXPLAIN PLAN

RENAME

SET TRANSACTION

12. Columns with the data type CHAR and with a length less than or equal to 30 characters are stored with a fixed length in ADABAS, not with a variable length as in ORACLE.

13. In ADABAS, the code attribute of a column of the data type CHAR, VARCHAR or LONG is determined by the code attribute specified during the configuration which should, but need not, correspond to the character set of the computer.

14. In ADABAS, data types cannot be defined for numbers with a negative scale. It is not possible to specify a scale that is greater than the number of significant digits.

15. In ADABAS, the possible maximum number of significant digits is always used for the data type FLOAT.

16. In ADABAS, a <constraint definition> defined in a <column definition> can also refer to several columns of the table.

17. In ADABAS, constraints cannot be deactivated temporarily.

18. The specification of the <oracle option>s in the <create table statement> and <create index statement> has no effect in ADABAS.

19. In contrast to ORACLE, tables for which no PRIMARY KEY was defined contain the key column SYSKEY in ADABAS. This column can be specified in a <select column> and in the <search condition> and its value cannot be changed. User-defined key columns can be changed in ADABAS.

20. For a <drop table statement> or <drop view statement>, all objects, synonyms, and view tables belonging to this table are deleted in ADABAS, not only marked as not being usable.

21. A <create synonym statement> is not possible for sequences, procedures or functions. The object for which the synonym is defined must exist and the current user must have at least one privilege for it.

22. In ADABAS, a time for an automatic refresh cannot be specified during the definition of snapshot table. An automatic refresh of a snapshot table is not possible.

23. In ADABAS, an index cannot be defined across a cluster.

24. In ADABAS, an index in descending order is created when DESC has been specified.

25. In ADABAS, an <index name> may only be unique together with a <table name>. In contrast to ORACLE, non-unique <index name>s require the specification of the corresponding <table name> in the <drop index statement>.

26. PUBLIC synonyms cannot be created and deleted in ADABAS.

27. The <sql statement>s

    ```
    CREATE TABLESPACE
    CREATE [ PUBLIC ] ROLLBACK SEGMENT
    ```

    are accepted by ADABAS but have no effect.

28. The <grant statement> in ADABAS corresponds to the ORACLE GRANT (object privileges). The other types of the GRANT are not available in ADABAS. The same is valid for REVOKE.

29. In contrast to ORACLE, it is not possible in ADABAS to grant a user a privilege of two different, other users.

Error Messages

The following error messages have been adopted :

```
-   1     DUPLICATE SECONDARY KEY

-  51     LOCK REQUEST TIMEOUT

-  54     LOCK COLLISION

- 602     SYSTEM ERROR

- 942     UNKNOWN TABLE NAME

- 955     DUPLICATE NAME

-1402     INTEGRITY VIOLATION

-1556     DATA SPACE FULL (EMERGENCY SHUTDOWN)
```

All the other error messages have not been adopted and are specific for the database system ADABAS. The only message with a positive value is 100 ROW NOT FOUND.

Syntax

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

<alias name> ::=
     <identifier>

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

<alter table statement> ::=
     ALTER TABLE <table name> <add definition>
   | ALTER TABLE <table name> <modify definition>

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

<arithmetic function> ::=
     TRUNC   ( <expression>[, <expression>] )
   | ROUND   ( <expression>[, <expression>] )
   | CEIL    ( <expression> )
   | FLOOR   ( <expression> )
   | SIGN    ( <expression> )
   | ABS     ( <expression> )
   | POWER   ( <expression>, <expression> )
   | EXP     ( <expression> )
   | SQRT    ( <expression> )
   | LN      ( <expression> )
   | LOG     ( <expression>, <expression> )
   | MOD     ( <expression>, <expression> )
   | LENGTH  ( <expression> )
   | VSIZE   ( <expression> )
   | INSTR   ( <string spec>, <string spec>
               [,<expression>[, <expression>] ] )
   | ASCII   ( <expression> )

<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>

<cache spec> ::=
       CACHE
     | NOCACHE
<character> ::=
     <digit>
   | <letter>
   | <extended letter>
   | <hex digit>
   | <language specific character>
   | <special character>

<check expression> ::=
     <expression>

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

<column attributes> ::=
     [<default spec>]
     [ NULL        [CONSTRAINT <constraint name>] ]
     [ NOT NULL    [CONSTRAINT <constraint name>] ]
     [ UNIQUE      [CONSTRAINT <constraint name>] ]
     [ PRIMARY KEY [CONSTRAINT <constraint name>] ]
     [ REFERENCES <table name> [(<column name>)] ]
     [<constraint definition>]

<column comment> ::=
     <column name> IS <string literal>

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

<column name> ::=
     <identifier>

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


<comment statement> ::=
     COMMENT ON TABLE <table name> IS <string literal>
   | COMMENT ON COLUMN
           <table name>.<column name> IS <string literal>
   | COMMENT ON <table name> ( <column comment>,... )

<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>
   | <expression list> <equal or not> (<expression list>)
   | <expression list> <equal or not> <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> ::=
     [CONSTRAINT <constraint name>] CHECK (<search condition>)

<constraint name> ::=
     <identifier>

<conversion function> ::=
     TO_NUMBER ( <string spec>[, <number format> ] )
   | CHR       ( <expression> )
   | RAWTOHEX  ( <expression> )
   | HEXTORAW  ( <expression> )
   | TO_CHAR   ( <expression>[, <date or number format> ] )
   | TO_DATE   ( <expression>[, <date format> ] )

<create index statement> ::=
     CREATE [UNIQUE] INDEX <index spec>
     [<oracle option>]


<create sequence statement> ::=
     CREATE SEQUENCE [<owner>.]<sequence name>
     [INCREMENT BY <integer>]
     [START WITH <integer>]
     [<maxvalue spec>]
     [<minvalue spec]
     [<cycle spec]
     [<cache spec>]
     [<order spec>]

<create snapshot log statement> ::=
     CREATE SNAPSHOT LOG ON <table name> [<oracle option>]

<create snapshot statement> ::=
     CREATE SNAPSHOT <table name>
     [<oracle snapshot options>] [<refresh spec>]
     AS <query expression>

<create synonym statement> ::=
     CREATE SYNONYM [<owner>.]<synonym name> FOR <table name>

<create table statement> ::=
     CREATE TABLE <table name> [(<table description element>,...)]
           [<oracle option> ...] [AS <query expression>]

<create user statement> ::=
     CREATE USER <user name> IDENTIFIED BY <password>
     [<oracle user option> ...]

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

<cycle spec> ::=
       CYCLE
     | NOCYCLE

<data type> ::=
     CHAR    [(<unsigned integer>)]
   | VARCHAR [(<unsigned integer>)]
   | LONG [RAW]
   | NUMBER  [(<unsigned integer> [,<unsigned integer>])]
   | NUMBER  [(* [,<unsigned integer>])]
   | DATE
   | RAW     [(<unsigned integer>)]

<date and time expression> ::=
     <expression>

<date and time function> ::=
     ADD_MONTHS     ( <date and time expression> , <expression> )
   | MONTHS_BETWEEN ( <date and time expression> ,
                      <date and time expression> )
   | LAST_DAY       ( <date and time expression> )
   | NEXT_DAY       ( <date and time expression>, <string spec> )
   | NEW_TIME       ( <date and time expression>,
                      <source timezone spec>,
                      <dest timezone spec> )
   | ROUND          ( <date and time expression>
                      [, <trunc and round format> ] )
   | TRUNC          ( <date and time expression>
                      [, <trunc and round format> ] )

<date format> ::=
     see <conversion function> in chapter 0

<date or number format> ::=
     <number format>
   | <date format>

<datetime expression> ::=
     <datetime primary>
   | <datetime expression> + <factor>
   | <datetime expression> - <factor>
   | <factor> + <datetime expression>

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

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

<default expression> ::=
     <expression>

<default spec> ::=
     DEFAULT <expression>

<delete statement> ::=
     DELETE FROM <table name> [<reference name>]
            [WHERE <search condition>]
   | DELETE FROM <table name> [<reference name>]
            WHERE CURRENT OF <result table name>


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

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

<dest timezone spec> ::=
     <timezone spec>

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

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

<distinct spec> ::=
     DISTINCT
   | ALL

<double quotes> ::=
     "

<drop index statement> ::=
     DROP INDEX <index name> [ON <table name>]

<drop sequence statement> ::=
     DROP SEQUENCE [<owner>.]<sequence name>

<drop snapshot statement> ::=
     DROP SNAPSHOT <table name>

<drop snapshot log statement> ::=
     DROP SNAPSHOT LOG ON <table name>

<drop synonym statement> ::=
     DROP SYNONYM [<owner>.]<synonym name>

<drop table statement> ::=
     DROP TABLE <table name> [CASCADE CONSTRAINTS]

<drop user statement> ::=
     DROP USER <user name> [<cascade option>]

<drop view statement> ::=
     DROP VIEW <table name>


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

<exists predicate> ::=
     EXISTS <subquery>

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

<expression> ::=
     <arithmetic expression>
   | <datetime expression>

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

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

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

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

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

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

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

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

<from clause> ::=
     FROM <table spec>,...

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

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

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

<group clause> ::=
     GROUP BY <expression>,...

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

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

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

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

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

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

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

<index clause> ::=
     <column name> [<order spec>]

<index name> ::=
     <identifier>

<index spec> ::=
     <index name> ON <table name> ( <index clause>,... )

<indicator name> ::=
     <parameter name>

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

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

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

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

<key definition> ::=
     [CONSTRAINT <constraint name>] PRIMARY KEY (<column name>,...)
     [USING INDEX <oracle option>

<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>

<lock spec> ::=
     SHARE
   | ROW SHARE
   | SHARE UPDATE
   | EXCLUSIVE
   | ROW EXCLUSIVE
   | SHARE ROW EXCLUSIVE

<lock statement> ::=
     LOCK TABLE <table name>,... IN <lock spec> MODE [NOWAIT]

<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'

<maxvalue spec> ::=
       MAXVALUE <integer>
     | NOMAXVALUE


<minvalue spec> ::=
       MINVALUE <integer>
     | NOMINVALUE

<modify definition> ::=
     MODIFY (<column name> <data type>)

<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

<number format> ::=
     see <conversion function> in chapter 0

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

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

<oracle ddl statement> ::=
     <create tablespace statement>
   | <create rollback segment statement>
   | <create public rollback segment statement>


<oracle option> ::=
     PCTFREE <unsigned integer>
   | PCTUSED <unsigned integer>
   | INITTRANS <unsigned integer>
   | MAXTRANS <unsigned integer>
   | TABLESPACE <identifier>
   | STORAGE <storage clause>

<oracle snapshot options> ::=
       <oracle option>
     | CLUSTER <identifier> (<column name>,...)

<oracle user option> ::=
     DEFAULT TABLESPACE <tablespace>
   | TEMPORARY TABLESPACE <tablespace>
   | QUOTA <unsigned integer [ K | M ] ON <tablespace>
   | QUOTA UNLIMITED
   | PROFILE <identifier>

<order and update clause> ::=
     <order clause> [<update clause>]
   | <update clause> [<order clause>]

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

<order sequence spec> ::=
       ORDER
     | NOORDER

<order spec> ::=
     ASC
   | DESC

<outer join indicator> ::=
     (+)

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

<parameter name> ::=
     :<identifier>

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

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

<password spec> ::=
     <parameter name>

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

<predicate> ::=
     <between predicate>
   | <comparison predicate>
   | <exists predicate>
   | <in predicate>
   | <join predicate>
   | <like predicate>
   | <null predicate>
   | <quantified predicate>
   | <rownum predicate>

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

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

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

<quantifier> ::=
     ALL
   | <some>

<query expression> ::=
     <query primary>
   | <query expression> UNION     <query primary>
   | <query expression> INTERSECT <query primary>
   | <query expression> MINUS     <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> ::=
     [CONSTRAINT <constraint name>]
     FOREIGN KEY (<referencing column>,...)
     REFERENCES <referenced table> [(<referenced column>,...)]
     [ON DELETE CASCADE]

<referential constraint name> ::=
     <identifier>

<refresh kind> ::=
     FAST
   | COMPLETE
   | FORCE

<refresh spec> ::=
     REFRESH [<refresh kind>]
     [START WITH <date and time expression>]
     [NEXT <date and time expression>]

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


<release statement> ::=
     COMMIT   [WORK] RELEASE
   | ROLLBACK [WORK] RELEASE

<reserved key word> ::=
   ADD          ALL          ALTER        AND          ANY
   AS           ASC          AUDIT

   BETWEEN      BY

   CHAR         CHECK        CLUSTER      COLUMN       COMMENT
   CONNECT      CREATE       CURRENT

   DATE         DECIMAL      DEFAULT      DELETE       DESC
   DISTINCT     DROP

   EXCLUSIVE    EXISTS

   FLOAT        FOR          FROM

   GRANT        GROUP

   HAVING

   IDENTIFIED   IN           INDEX        INSERT       INTEGER
   INTERSECT    INTO         IS

   LEVEL        LIKE         LOCK         LONG

   MINUS        MODE         MODIFY

   NOT          NOWAIT       NULL         NUMBER

   OF           ON           OPTION       OR           ORDER

   PCTFREE      PRIVILEGES   PUBLIC

   RAW          RENAME       RESOURCE     REVOKE       ROW
   ROWID        ROWNUM       ROWS

   SELECT       SET          SHARE        SMALLINT     SYNONYM
   SYSDATE

   TABLE        TO           TRIGGER

   UID          UNION        UNIQUE       UPDATE       USER

   VALUES       VARCHAR      VARCHAR2     VIEW

   WHENEVER     WHERE        WITH

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

   BEGIN        BINARY       BIT          BOOLEAN      BOTH
   BUFFERPOOL   BYTE

   CASCADE      CAST         CATALOG      CEIL         CEILING
   CHARACTER    CHR          CLOSE        COMMIT       CONCAT
   CONNECTED    CONSTRAINT   COS          COSH         COT
   COUNT        CURDATE      CURRENT_DATE CURRENT_TIME CURSOR
   CURTIME

   DATABASE     DATEDIFF     DAY          DAYNAME      DAYOFMONTH
   DAYOFWEEK    DAYOFYEAR    DBYTE        DEC          DECLARE
   DECODE       DEGREES      DESCRIBE     DIGITS       DIRECT
   DISCONNECT   DOMAIN       DOUBLE

   EBCDIC       EDITPROC     END          ENTRY        ENTRYDEF
   ESCAPE       EXCEPT       EXECUTE      EXP          EXPAND
   EXTRACT

   FALSE        FETCH        FIRST        FIXED        FLOOR
   FOREIGN
   GET          GRAPHIC      GREATEST

   HEX          HOUR

   IFNULL       IGNORE       INDICATOR    INITCAP      INNER
   INT          ISOLATION

   JOIN

   KEY

   LANGUAGE     LAST         LCASE        LEADING      LEAST
   LEFT         LENGTH       LFILL        LINK         LIST
   LN           LOCAL        LOCALSYSDBA  LOG          LOG10
   LOWER        LPAD         LTRIM

   MAKEDATE     MAKETIME     MAPCHAR      MAX          MICROSECOND
   MIN          MINUTE       MONTH        MONTHNAME

   NATURAL      NEXT         NO           NOROUND      NOW
   NUM          NUMERIC

   OBID         OBJECT       ONLY         OPEN         OPTIMIZE
   OUTER

   PACKED       PI           POWER        PRECISION    PREV
   PRIMARY      PROCEDURE

   RADIANS      READ         REAL         REFERENCED   REFERENCES
   REJECT       RELEASE      REPLACE      RESTRICT     RFILL
   RIGHT        ROLLBACK     ROUND        ROWNO        RPAD
   RTRIM

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

   TABLESPACE   TAN          TANH         TIME         TIMEDIFF
   TIMESTAMP    TIMEZONE     TOIDENTIFIER TRAILING     TRANSACTION
   TRANSLATE    TRIM         TRUE         TRUNC        TRUNCATE

   UCASE        UNKNOWN      UPPER        USAGE        USERGROUP
   USING

   VALIDPROC    VALUE        VARGRAPHIC   VARIANCE     VARYING

   WEEKOFYEAR   WORK         WRITE

   YEAR

   ZONED

<result column name> ::=
     <identifier>

<result expression> ::=
     <expression>

<result table name> ::=
     <identifier>

<revoke statement> ::=
     REVOKE <table privileges> ON <table name> FROM <grantee>,...

<rollback statement> ::=
     ROLLBACK [WORK]

<rollback to statement> ::=
     ROLLBACK TO [SAVEPOINT] <savepoint name>

<rownum column> ::=
     ROWNUM [<result column name>]


<rownum predicate> ::=
     ROWNUM <  <rownum spec>
   | ROWNUM <= <rownum spec>

<rownum spec> ::=
     <unsigned integer>
   | <parameter spec>

<savepoint name> ::=
     <identifier>

<savepoint statement> ::=
     SAVEPOINT <savepoint name>

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

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

<search expression> ::=
     <expression>

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

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

<sequence name> ::=
     <identifier>

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

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

<set update clause> ::=
     <column name> = <expression>
   | <column name> = <subquery>
   | <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>]
   | <expression> [<sort option>]

<source timezone spec> ::=
     <timezone spec>

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

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

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

<special identifier character> ::=
     Any character.

<sql statement> ::=
     <create table statement>
   | <drop table statement>
   | <alter table statement>
   | <create synonym statement>
   | <drop synonym statement>
   | <create snapshot statement>
   | <drop snapshot statement>
   | <create snapshot log statement>
   | <drop snapshot log statement>
   | <create view statement>
   | <drop view statement>
   | <create index statement>
   | <drop index statement>
   | <create sequence statement>
   | <drop sequence statement>
   | <oracle ddl statement>
   | <comment statement>

   | <create user statement>
   | <drop user statement>
   | <grant statement>
   | <revoke statement>

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

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

   | <connect statement>
   | <commit statement>
   | <rollback statement>
   | <rollback to statement>
   | <savepoint statement>
   | <lock statement>
   | <release statement>


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

<storage clause> ::=
     ([INITIAL <unsigned integer>] [NEXT <unsigned integer>]
     [MINEXTENTS <unsigned integer>]
     [MAXEXTENTS <unsigned integer>]
     [PCTINCREASE <unsigned integer>])

<storage reuse spec> ::=
     DROP STORAGE
   | REUSE STORAGE

<string function> ::=
     <string spec> || <string spec>
   | CONCAT    ( <string spec>, <string spec> )
   | SUBSTR    ( <string spec>, <expression>[, <expression>] )
   | LPAD      ( <string spec>, <unsigned integer>
                 [, <string literal> ] )
   | RPAD      ( <string spec>, <unsigned integer>
                 [, <string literal> ] )
   | LTRIM     ( <string spec>[, <string spec> ] )
   | RTRIM     ( <string spec>[, <string spec> ] )
   | UPPER     ( <string spec> )
   | LOWER     ( <string spec> )
   | INITCAP   ( <string spec> )
   | REPLACE   ( <string spec>, <string spec>[, <string spec> ] )
   | TRANSLATE ( <string spec>, <string spec>, <string spec> )
   | SOUNDEX   ( <string spec> )

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

 <string spec> ::=
     <expression>

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

<synonym name> ::=
     <identifier>


<table columns> ::=
     *
   | <table name>.*
   | <reference name>.*

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

<table expression> ::=
     <from clause>
     [<where clause>]
     [<group clause> [<having clause>] ]

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

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

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

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

<termchar set name> ::=
     <identifier>

<timezone spec> ::=
     'AST'
   | 'ADT'
   | 'BST'
   | 'BDT'
   | 'CST'
   | 'CDT'
   | 'EST'
   | 'EDT'
   | 'GMT'
   | 'HST'
   | 'HDT'
   | 'MST'
   | 'MDT'
   | 'NST'
   | 'PST'
   | 'PDT'
   | 'YST'
   | 'YDT'

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

<trigonometric function> ::=
     COS     ( <expression> )
   | SIN     ( <expression> )
   | TAN     ( <expression> )
   | COSH    ( <expression> )
   | SINH    ( <expression> )
   | TANH    ( <expression> )

<trunc and round format> ::=
     see <date and time function> in chapter 0

<truncate statement> ::=
     TRUNCATE <table name> [<storage reuse spec>]

<underscore> ::=
     _

<unique definition> ::=
     [CONSTRAINT <constraint name>] UNIQUE (<column name>,...)
     [USING INDEX <oracle option>]

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

<update clause> ::=
     FOR UPDATE OF <column name>,... [NOWAIT]

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

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

<user name> ::=
     <identifier>

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

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

```
<usergroup name> ::=
     <identifier>

<value spec> ::=
     <literal>
   | <parameter spec>
   | NULL
   | USER
   | [<owner>.]<sequence name>.NEXTVAL
   | [<owner>.]<sequence name>.CURRVAL
   | SYSDATE
   | UID

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